

---

# TEAM LANGUAGE FINAL REPORT

---

May 12, 2021

Wenlu (Lulu) Zheng: [lulu.zheng@tufts.edu](mailto:lulu.zheng@tufts.edu)  
Yingjie Ling: [yingjie.ling@tufts.edu](mailto:yingjie.ling@tufts.edu)  
Saurav Gyawali: [saurav.gyawali@tufts.edu](mailto:saurav.gyawali@tufts.edu)  
Naoki Okada: [naoki.okada@tufts.edu](mailto:naoki.okada@tufts.edu)

# Contents

<b>1</b>	<b>Introduction to TEAM</b>	<b>4</b>
<b>2</b>	<b>Language Tutorial</b>	<b>5</b>
2.1	Environment Setup . . . . .	5
2.2	Compiler Instructions . . . . .	5
2.3	Quick Start . . . . .	5
2.3.1	Print . . . . .	5
2.3.2	Variables . . . . .	6
2.3.3	Lists . . . . .	6
2.3.4	Functions . . . . .	6
2.3.5	File I/O . . . . .	7
2.4	Advanced Features . . . . .	7
2.4.1	Special Operators . . . . .	7
2.4.2	Higher Order Functions . . . . .	8
<b>3</b>	<b>Language Reference Manual</b>	<b>9</b>
3.1	LRM Reading Directions . . . . .	9
3.2	Lexical Elements . . . . .	9
3.2.1	Identifiers . . . . .	9
3.2.2	Keywords . . . . .	9
3.2.3	Constants . . . . .	9
3.2.4	Operators . . . . .	9
3.2.5	Separators . . . . .	10
3.2.6	Comments . . . . .	10
3.3	Data Types . . . . .	10
3.3.1	Primitive Types . . . . .	10
3.3.2	Collection Data Types . . . . .	10
3.3.3	Function Types . . . . .	11
3.4	Expressions and Operators . . . . .	11
3.4.1	Arithmetic Operators . . . . .	13
3.4.2	Assignment Operators . . . . .	13
3.4.3	Relational Operators . . . . .	14
3.4.4	Logical Operators . . . . .	15
3.4.5	Range Operator . . . . .	15
3.4.6	Indexing . . . . .	15
3.4.7	Slicing . . . . .	16
3.4.8	Function Call . . . . .	16
3.4.9	Literals . . . . .	17
3.4.10	Variables . . . . .	17
3.5	Operator Precedence . . . . .	18
3.6	Statements . . . . .	19
3.6.1	Expression Statement . . . . .	20
3.6.2	Variable Declaration and Initialization . . . . .	20
3.6.3	if statement . . . . .	20
3.6.4	while statement . . . . .	21
3.6.5	for statement . . . . .	21

3.6.6	break statement . . . . .	22
3.6.7	continue statement . . . . .	22
3.6.8	return statement . . . . .	22
3.7	Functions . . . . .	22
3.7.1	Function Definitions . . . . .	22
3.7.2	Calling Functions . . . . .	23
3.7.3	Function Parameters . . . . .	23
3.7.4	Higher order functions . . . . .	24
3.8	Scope . . . . .	24
3.9	Program Structure . . . . .	25
<b>4</b>	<b>Built-in Functions</b>	<b>26</b>
4.1	List . . . . .	26
4.2	Print, Formatted Strings, and File I/O . . . . .	27
<b>5</b>	<b>Standard Library Functions</b>	<b>29</b>
5.1	List Library . . . . .	29
5.2	String/Char Library . . . . .	30
<b>6</b>	<b>Regular Expressions</b>	<b>32</b>
6.1	Meta characters . . . . .	32
6.2	Special sequences . . . . .	32
6.3	Sets . . . . .	32
6.4	Regular Expression Examples . . . . .	33
6.5	Built-in Functions . . . . .	33
<b>7</b>	<b>Project Plan</b>	<b>35</b>
7.1	Planning, Specification, and Development Process . . . . .	35
7.2	Project Timeline . . . . .	35
7.3	Roles and Responsibilities . . . . .	36
7.4	Development Tools . . . . .	36
7.5	Project Log . . . . .	36
<b>8</b>	<b>Architectural Design</b>	<b>37</b>
8.1	Overview . . . . .	37
8.2	Standard Library . . . . .	37
8.3	Scanner . . . . .	38
8.4	Parser . . . . .	38
8.5	Semantic Analyzer . . . . .	38
8.6	Type Resolver . . . . .	38
8.7	Code Generator . . . . .	38
8.8	Builtin C Library . . . . .	38
8.9	Workflow . . . . .	38
<b>9</b>	<b>Test Plan</b>	<b>40</b>
9.1	Test Suite/Automation . . . . .	40
9.2	Test Scripts . . . . .	40
9.2.1	Makefile . . . . .	41
9.2.2	runtests.py . . . . .	42

9.3	Examples of Tests . . . . .	45
9.3.1	max_profit.tm . . . . .	45
9.3.2	maximum_subarray.tm . . . . .	119
9.4	Breakdown of Responsibilities . . . . .	126
<b>10</b>	<b>Lessons Learned</b>	<b>127</b>
10.1	Lulu . . . . .	127
10.2	Yingjie . . . . .	127
10.3	Naoki . . . . .	127
10.4	Saurav . . . . .	127
<b>11</b>	<b>Appendix</b>	<b>129</b>
11.1	Git Log . . . . .	129
11.2	Complete Code Listing . . . . .	142
11.2.1	standard_library/list.tm . . . . .	142
11.2.2	standard_library/string.tm . . . . .	145
11.2.3	src/scanner.mll . . . . .	147
11.2.4	src/parser.mly . . . . .	149
11.2.5	src/ast.ml . . . . .	152
11.2.6	src/sast.ml . . . . .	155
11.2.7	src/exceptions.ml . . . . .	158
11.2.8	src/semant.ml . . . . .	165
11.2.9	src/resolve.ml . . . . .	176
11.2.10	src/codegen.ml . . . . .	182
11.2.11	src/team.ml . . . . .	211
11.2.12	scripts/compile.sh . . . . .	213
11.2.13	c_library/fileio.c . . . . .	214
11.2.14	c_library/regex.c . . . . .	214
11.2.15	README & Makefile . . . . .	228
11.2.16	Standard Library LLVM . . . . .	231

# 1 Introduction to TEAM

TEAM (Text Extraction And Manipulation) is a domain specific programming language designed for text processing, data extraction, and report generation. With its straightforward syntax and various built-in functions, TEAM offers a clean layer of abstraction for users to perform tasks that are often cumbersome to do in general purpose languages.

TEAM is imperative and mostly statically typed except for lists. Like many other modern languages, TEAM supports six primitive types: `int`, `float`, `string`, `bool`, `char`, and `file`. Moreover, TEAM has container type `list`. For easy file I/O, we introduce the `file` type to our language. Control flow is carried out by `for` loops, `while` loops, and `if` statements.

The various built-in functions along with the support for regular expressions and file I/O free the users from the ordeal of repetitive tasks such as file parsing and string manipulation. Additionally, TEAM recognizes functions as first-class citizens, which allows users to expand language features for more complicated tasks with minimal effort. TEAM also provides a standard library for basic operations on strings and lists.

This document will present a language tutorial, provide an in-depth language reference manual and discuss the implementation process and our lessons learned.

## 2 Language Tutorial

### 2.1 Environment Setup

This compiler is written in OCaml and uses the OCaml llvm library. The OCaml llvm library is most easily installed using opam. Install LLVM and its development libraries, the m4 macro preprocessor, and opam, then use opam to install llvm. The version of the OCaml llvm library must match the version of the LLVM system installed on your own system.

TEAM compiler also needs gcc to be installed as a dependency. Make sure the appropriate gcc is installed in your system.

After installing OCaml, the OCaml llvm library and gcc, follow the below steps to setup and install the TEAM compiler:

- Install the PCRE2 (Perl Compatible Regular Expressions) library using the following command while in the root directory of the compiler:  

```
cd pcre2-10.36 && ./configure && make && make install && cd ..
```

  
In case the PCRE2 library fails to install, go to <https://ftp.pcre.org/pub/pcre/> and install the pcre2-10.36 manually.
- Build the compiler using make to create an executable compiler which will be named team.native

### 2.2 Compiler Instructions

To compile a TEAM program named `test.tm`, do:

```
sh scripts/compile.sh test.tm run
```

This command generates `test.s`, `test.ll`, and `test.exe`, to remove them, do:

```
sh scripts/compile.sh test.tm clean
```

To remove the `.o` files generated with the make command, do:

```
make clean
```

### 2.3 Quick Start

A TEAM program is executed from the top to the bottom line by line and does not require a main function.

#### 2.3.1 Print

Here is a simple program that prints a string in TEAM:

```
1 print("Hello World");
```

Compiling and executing the above code outputs:

```
1 Hello World;
```

### 2.3.2 Variables

Variables in TEAM are typed. The primitive types in TEAM are int, char, bool, float, string and file. Below are examples of how you would use them:

```
1 int sum = 3 + 5;
2 sum = sum / 2;
3 string name = "Team";
4 float fl = 3.222;
5 bool feelingGood = true;
6 print("%d\n", sum);
7 print("%s\n", name);
8 print("%f\n", fl);
9 print("%s\n", feelingGood);
```

Compiling and executing the above code outputs:

```
1 4
2 Team
3 3.222000
4 true
```

### 2.3.3 Lists

TEAM has a native list type. When you declare a list, you do not specify the type of its elements. You can also declare an empty list. Note that the type of a list cannot be list. The following code demonstrates some examples:

```
1 list nums = [4,5,6,7,9];
2 print("%d\n", num[0]);
3 nums[0] = 19;
4 print("%d\n", num[0]);
```

Compiling and executing the above code outputs:

```
1 4
2 19
```

### 2.3.4 Functions

When you write a function in TEAM, you specify the return type and the parameters (type and name) in the function signature and add statements in the body:

```
1 int calcSum(list data):
2     int count = 0;
3     for num in data:
4         count += num;
5     end
6     return count;
7 end
8
9 list nums = [4,6,10,11,7,3];
10 int sum = calcSum(nums);
11 print("%d\n", sum);
```

Compiling and executing the above code outputs:

```
1 41
```

### 2.3.5 File I/O

Here is a simple program to read in from a file:

```
1 file input = open("example.txt", "r");
2 string currline;
3 while ((currline = readline(input)) != ""):
4     print(currline);
5 end
6
7 close(input);
```

Here is a simple program to write to a file:

```
1 file input = open("example.txt", "r");
2 list roster = ["Lulu", "Yingjie", "Saurav", "Naoki"];
3 for name in roster:
4     write(output, name + "\n");
5 end
6
7 close(input);
```

## 2.4 Advanced Features

### 2.4.1 Special Operators

1. The addition operator supports the following operands combinations:
  - *int* + *int* (result type: int)
  - *int* + *float* (result type: float)
  - *float* + *float* (result type: float)
  - *string* + *string* (result type: string)
  - *char* + *char* (result type: string)
  - *string* + *char* (result type: string)
  - *char* + *string* (result type: string)
2. The range operator returns a sequence of integer numbers in a given range (start included, end excluded). The sequence is represented as a list of integers and follows the syntax below:

`<list> x = start..end`

3. String and list slicing:
  - A character in a string and an element in a list can be accessed using the syntax below:



`x[index]`

where `x` can be of `string` or `list` type.

- A substring in a string or a sublist of a list can be obtained using the syntax below:

`x[start:end]`

where `x` can be of `string` or `list` type. Both *start* and *end* are optional; when omitted, *start* defaults to 0 and *end* to `length(x)`. The code above returns a list from *start* to *end* - 1.

- A single element in the list can be overwritten using the syntax below:

`x[index] = y`

where `x` is a list containing elements of the same type as `y`.

- A sublist can be overwritten using the syntax below:

`x[start:end] = y`

where both `x` and `y` are lists containing same type of elements. *start* and *end* are also optional here, following the same rule as list accessing (second bullet point above). The list from *start* to *end* - 1 is overwritten.

### 2.4.2 Higher Order Functions

Standard library and user defined functions can be passed to other functions as arguments and returned as results. However, it is worth noting that TEAM does not support nested function definition. For a more detailed discussion of higher order function, please refer to section 3.7.4. The type of a function is expressed using the syntax below:

`(type1, ...) -> ret_type`

where the parameter types are specified in the parenthesis and *ret\_type* is the return type. There may be 0 or more parameter types. For example, a function that returns an integer and takes another function that takes two integers and returns an integer looks like this:

`int function1 ((int,int)->int function2)`

where `function1` returns an integer and takes a function (`function2`), which takes two integers and returns an integer.

## 3 Language Reference Manual

### 3.1 LRM Reading Directions

Different font sizes and styles are used throughout this LRM for a clear presentation of its content. In general, prose will be written in plain font like this: "Lorem ipsum dolor sit...". Short snippets and inline codes appear in this font: `int x = 5`. Sometimes, we use italics to denote grammatical placeholders like: *type*, *expression*, and *statement*. Finally, large code blocks are formatted with special syntax highlighting like this:

```
1 string language = "TEAM";
```

Whenever a function is described in detail, the types of the parameters and the return values are explicitly stated in parentheses.

### 3.2 Lexical Elements

There are six kinds of tokens: identifiers, keywords, constants, strings, operators and other separators. Blank spaces, newlines and comments are generally ignored.

#### 3.2.1 Identifiers

Identifiers must begin with a lowercase or uppercase letter followed by letters, digits, and `_`.

#### 3.2.2 Keywords

The following are reserved keywords: `if`, `else`, `elif`, `while`, `for`, `end`, `int`, `bool`, `char`, `string`, `float`, `list`, `file`, `void`, `and`, `or`, `not`, `in`, `return`, `true`, `false`, `break`, `continue`.

#### 3.2.3 Constants

There are five types of constants:

1. Integer constant: a sequence of digits where the leftmost digit is a number between 1 and 9, and every other digit is an integer between 0 to 9.
2. Floating constant: consists of an integer part (a sequence of digits with a non-zero leading digit), a dot denoting a decimal point, and a fraction part (a sequence of digits), which are all required.
3. Char constant: consists of a character surrounded by single quotation marks.
4. Bool constant: either `true` or `false`
5. String constant: consists of a sequence of characters surrounded by double quotation marks.

#### 3.2.4 Operators

Operators include: `+`, `-`, `*`, `/`, `%`, `^`, `==`, `!=`, `<`, `<=`, `>`, `>=`, `and`, `or`, `not`, `..`, `=`, `+=`, `-=`, `*=`, `/=`, `%=`,

More details for each operator are given below in section 3.4.

### 3.2.5 Separators

Semicolons are used to denote the end of statements.

### 3.2.6 Comments

Single-line comments are followed by `//`, and block comments are surrounded by `/*` and `*/`, that is, text after `//` in the same line, and any text between `/*` and `*/` are ignored. This means that any single-line comments inside block comments are ignored, allowing nested comments.

## 3.3 Data Types

### 3.3.1 Primitive Types

TEAM supports the following six primitive data types:

1. `int`: A 32-bit signed integer.
2. `float`: A 64-bit floating point number.
3. `string`: A sequence of characters.
4. `bool`: A 1-bit boolean.
5. `char`: A character.
6. `file`: A file handle for file reading and writing.

### 3.3.2 Collection Data Types

To provide easy and intuitive access to a collection of data, TEAM offers `list` as a container type. Lists are ordered and store zero or more elements of the same type. The size of a list is dynamically determined, and the type of the elements a list is resolved at compile-time. Note that the type of a list can be integer, string, char, bool, float, file but cannot be list or function. We can declare a list using the following syntax:

```
list var_name;
```

Initialization follows by assigning a list of expressions that evaluate to the same type:

```
list var_name = [expr1, expr2, ..., exprn];
```

More concretely:

```
list var_name = [1, 2, 3];
```

Note that a list can be initialized with zero or more elements in it, as long as all the elements are of the same type. An element of a list at index  $i$  can be easily retrieved with the following syntax:

```

1 // Note that TEAM uses zero-based indexing
2 list arr = [0, 1, 2, 3, 4, 5];
3 int first_element = arr[0]; // => 0

```

Team also supports list slicing with the following syntax:

```

1 // Note that the right bound is not included
2 list arr = [0, 1, 2, 3, 4, 5];
3 list first_three_elements = arr[0:3]; // => [0, 1, 2]

```

We can define nested lists by recursively defining lists of lists. For example:

```
list var_name = [[1, 2], [3, 4], [5, 6]];
```

We also provide both built-in and standard library operations to manipulate lists, which is discussed in sections 4.2 and 5.1 respectively.

### 3.3.3 Function Types

In TEAM, a function can be passed into another function as an argument, and can also be returned from another function. The type of a function as a parameter to another function consists of a list of parameters types of the parameter function, and the return type of the parameter function. The list of parameters types of the parameter function are comma separated and surrounded by parenthesis, followed by an arrow and the return type of the parameter function: `(type1, type2, ...) -> (type)`

```

1 // Function that calculates the average of a list of integers, after
  removing outliers
2 int computeAverage((list)->int average, list data):
3     list validData = []
4     for num in data:
5         if (num < 95):
6             validData = append(validData, num);
7         end
8     end
9     return average(validData);
10 end
11
12 // Assume the function average takes in a list and returns an int
13 list data = [40, 50, 34, 89, 99, 57];
14 computeAverage(average, data)

```

## 3.4 Expressions and Operators

An expression is made up of one or more operands and zero or more operators. Operands can be literal values, a variable to a value, function calls, or a combination of these. Parentheses are used to group subexpressions as part of a bigger expression. A semicolon terminates an expression. Here are some examples:

```

1 35;
2 -42 + 6;
3 (9 - 2) / square(3);

```

Here is the grammar for all expressions:

```

1  expr -> unary_expr
2  expr -> expr + expr
3  expr -> expr - expr
4  expr -> expr * expr
5  expr -> expr / expr
6  expr -> expr ^ expr
7  expr -> expr % expr
8  expr -> expr == expr
9  expr -> expr != expr
10 expr -> expr < expr
11 expr -> expr <= expr
12 expr -> expr > expr
13 expr -> expr >= expr
14 expr -> expr and expr
15 expr -> expr or expr
16 expr -> not expr
17 expr -> expr .. expr
18 expr -> expr = expr
19 expr -> expr += expr
20 expr -> expr -= expr
21 expr -> expr *= expr
22 expr -> expr /= expr
23 expr -> expr %= expr
24
25 unary_expr -> bracket_expr
26 unary_expr -> - unary_expr
27 unary_expr -> not unary_expr
28
29 bracket_expr -> primary
30 bracket_expr -> bracket_expr [ index ]
31 bracket_expr -> bracket_expr ( args_opt )
32
33 primary -> LITERAL (integer literal)
34 primary -> BLIT    (boolean literal)
35 primary -> FLIT    (float literal)
36 primary -> CLIT    (char literal)
37 primary -> SLIT    (string literal)
38 primary -> ID
39 primary -> [ list_literal ]
40 primary -> ( expr )
41
42 ID -> ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_' ]*
43
44 index -> expr
45 index -> expr : expr
46 index -> : expr
47 index -> expr :
48 index -> :
49
50 list_literal -> NOTHING
51 list_literal -> expr
52 list_literal -> list_literal : expr
53
54 args_opt ->
55 args_opt -> args_list
56
57 args_list -> expr
58 args_list -> args_list , expr

```

### 3.4.1 Arithmetic Operators

Usage of arithmetic operators is straightforward. All of the following operators group left-to-right except for when `-` is used for negation, which groups right-to-left. Here is the list of arithmetic operators supported in TEAM:

1. `+` is used for addition. If both operands are `int`, the result is `int`. If both are `float`, the result is `float`. If one is `int` and one is `float`, the former is converted to a `float` and the result is `float`. `+` can also be used on strings to concatenate two strings and lists to concatenate two lists. `+` can also be used to combine a character and string and vice versa.
2. `-` is used for subtraction or negation. When used for subtraction, if both operands are `int`, the result is `int`. If both are `float`, the result is `float`. If one is `int` and one is `float`, the former is converted to a `float` and the result is `float`. When used for negation, the result is the negative of the expression, and has the same type. The type of the expression must be `int` or `float`.
3. `*` is used for multiplication. It has the same type considerations as `-` when `-` is used as a binary operator.
4. `/` is used for division. It has the same type considerations as `-` when `-` is used as a binary operator.
5. `%` is used for modular division. Both operands must be an `int`, and the result is `int`.
6. `^` is used for exponentiation. Both operands must either be `float` or `int`.

The grammar for the above arithmetic operators is:

```
1 expr -> expr + expr
2 expr -> expr - expr
3 expr -> expr * expr
4 expr -> expr / expr
5 expr -> expr ^ expr
6 expr -> expr % expr
```

### 3.4.2 Assignment Operators

Assignment operators store the value of the right operand into the variable specified by the left operand.

The left operand is any identifier or list element (e.g. `lst[0]`) and the right operand is any expression. The left operand cannot be a literal value. All assignments group right-to-left. The type of an assignment expression is that of its left operand.

The `=` operator assigns the value of the right operand to the left operand. The assign operator can be used to store values of primitive types and the list type.

```
1 int x;
2 x = 5;
3 float y = 5.9;
4 list stringList = [];
```

Compound assignment operators combine the standard assign operator with another binary operator. The type rules for compound assignment operators follow their corresponding arithmetic operators, which are discussed in the section above. TEAM supports the following compound assignment operators:

1. `+=` adds the value of the right operand to the value of the left operand and assigns the result to the left operand.
2. `-=` subtracts the value of the right operand from the left operand and assigns the result to the left operand.
3. `*=` multiplies the values of the two operands and assigns the result to the left operand.
4. `/=` divides the value of the left operand by the right operand and assigns the result to the left operand.
5. `%=` performs modular division on the two operands and assigns the result to the left operand.

The grammar for compound assignment operators is:

```
1 expr -> expr += expr
2 expr -> expr -= expr
3 expr -> expr *= expr
4 expr -> expr /= expr
5 expr -> expr %= expr
```

### 3.4.3 Relational Operators

The relational operators compares two operands structurally and returns a Boolean value. All relational operators group left-to-right. The relational operators supported by TEAM are listed below:

1. `==` tests the operands for equality. `==` is supported on all values of primitive types, but not on lists. The only mixed type comparison `==` allows is between `int` and `float`.
2. `!=` tests the operands for inequality. The `==` and `!=` operators are lower in precedence than the other relational operators. The mixed typing rules for `==` also applies for `!=`.
3. `<` tests if the left operand is less than the right. The mixed typing rules for `==` also applies for `<`.
4. `>` tests if the left operand is greater than the right. The mixed typing rules for `==` also applies for `>`.
5. `<=` tests if the left operand is less than or equal to the right. The mixed typing rules for `==` also applies for `<=`.
6. `>=` tests if the left operand is greater than or equal to the right. The mixed typing rules for `==` also applies for `>=`.

The above operators are supported for the following operands types

Operators	Left Operand	Right Operand
==, !=, <, >, <=, >=	int	int
==, !=, <, >, <=, >=	int	float
==, !=, <, >, <=, >=	float	int
==, !=, <, >, <=, >=	float	float
==, !=	bool	bool
==, !=	char	char

The grammar for the above relational operators is:

```

1 expr -> expr == expr
2 expr -> expr != expr
3 expr -> expr < expr
4 expr -> expr <= expr
5 expr -> expr > expr
6 expr -> expr >= expr

```

### 3.4.4 Logical Operators

Logical operators test the Boolean value of a pair of operands. These operators are applicable only to `bool` typed expressions. TEAM supports the ones below:

1. `and` is used to evaluate logical and. Groups left-to-right.
2. `or` is used to evaluate logical or. Groups left-to-right.
3. `not` flips the Boolean value. Groups right-to-left.

The grammar for the above logical operators is:

```

1 expr -> expr and expr
2 expr -> expr or expr
3 expr -> not expr

```

### 3.4.5 Range Operator

The range operator (`..`) creates a list of successive integers. The left boundary is included while the right is not. The range operator is non-associative.

Here is an example of how the range operator is used:

```

1 // prints 0, 1, 2, 3, 4
2 for int i in 0..5:
3     print("%d\n", i);
4 end

```

The grammar for the range operator is:

```

1 expr -> expr .. expr

```

### 3.4.6 Indexing

Indexing is used to extract a character from a string or an element from a list. A string or list is followed by `[index]`, where *index* is an integer that denotes the index of the character or element to be extracted. An index expression returns a `char` in case of a string, and a variable of the inner type in case of a list.



```

1 string greeting = "hello";
2 print("%s", greeting[0]); // Prints "h"
3
4 list classes = ["Compilers", "Security", "Algorithms", "Entrepreneurship", "
    Nutrition"];
5 print("%s", classes[0]); // Prints "Compilers"

```

Here is the grammar for index expressions:

```

1 bracket_expr -> bracket_expr [ index ]
2
3 index -> expr

```

### 3.4.7 Slicing

Slicing is used to extract a substring from a string or a sublist from a list. A string or list is followed by [*index1*:*index2*], where *index1* and *index2* are integers that denote indices. A slice expression will return a substring/sublist that consists of elements starting at *index1* and ending at *index2* - 1. If *index1* and *index2* are left blank, it is implied that *index1* is 0 and *index2* is the length of string or list. Below are some examples that demonstrate how slicing works:

```

1 string greeting = "hello";
2 print("%s", greeting[1:5]); // Prints "ello"
3
4 string state = "Massachusetts";
5 print("%s", state[:4]); // Prints "Mass"
6
7 list classes = ["Compilers", "Security", "Algorithms", "Entrepreneurship", "
    Nutrition"];
8 list csClasses = classes[:3];
9
10 for class in csClasses:
11     print("%s ", class); // Prints "Compilers Security Algorithms"
12 end

```

Here is the grammar for slice expressions:

```

1 expr -> unary_expr
2
3 unary_expr -> bracket_expr
4
5 bracket_expr -> bracket_expr [ index ]
6
7 index -> expr : expr
8 index -> : expr
9 index -> expr :
10 index -> :

```

### 3.4.8 Function Call

A function call can be invoked to make a call to a specific function with zero or more arguments. One can also call a function using the name of the variable the function was assigned to in the current scope.

Below are some examples that demonstrate how a function call works:

```

1 list data = [3,4,1,8,10,24,17];
2 int avg = computeAverage(data);

```

Here is the grammar for function call:

```

1 expr -> unary_expr
2
3 unary_expr -> bracket_expr
4
5 bracket_expr -> bracket_expr ( args_opt )
6
7 args_opt ->
8 args_opt -> args_list
9
10 args_list -> expr
11 args_list -> args_list , expr

```

### 3.4.9 Literals

An expression can also be a literal. Below are some examples that demonstrate how literals can be assigned and used.

```

1 int my_favorite_number = 43;
2 string university = "Tufts";

```

Here is the grammar for literals:

```

1 expr -> unary_expr
2
3 unary_expr -> bracket_expr
4
5 bracket_expr -> primary
6
7 primary -> LITERAL (integer literal)
8 primary -> BLIT (boolean literal)
9 primary -> FLIT (float literal)
10 primary -> CLIT (char literal)
11 primary -> SLIT (string literal)
12 primary -> ID
13 primary -> [ list_literal ]
14
15 ID -> [ 'a'-'z' 'A'-'Z' ] [ 'a'-'z' 'A'-'Z' '0'-'9' '_' ] *

```

### 3.4.10 Variables

An expression can be a variable. Below are some examples that demonstrate how variables can be declared and initialized:

```

1 bool graduated;
2 string class = "Compilers";

```

Here is the grammar for variables:

```

1 expr -> unary_expr
2
3 unary_expr -> bracket_expr
4
5 bracket_expr -> primary

```

```
6  
7 primary -> expr
```

### 3.5 Operator Precedence

The rules of precedence determine the order of evaluation of terms in expressions that contain multiple operators. Expressions with multiple of the same operators are evaluated based on the operators' associativity. The following is a set of operator precedence rules, presented in order of highest precedence to lowest precedence. Operators that are shown together on a line have the same precedence.

1. not
2. ^
3. \*, /, %
4. +, -
5. ..
6. >, <, >=, <=
7. ==, !=
8. and
9. or
10. =, +=, -=, \*=, /=, %=

The associativity of the operators are shown below:

Operators	Associativity
=	right
+=	right
-=	right
*=	right
/=	right
%=	right
or	left
and	left
==	left
!=	left
<	left
>	left
<=	left
>=	left
..	non-associative
+	left
-	left
*	left
/	left
%	left
^	left
not	right

### 3.6 Statements

Statements are executed in sequence from top to bottom. The grammar for statements is:

```

1 stmt -> vdecl ;
2 stmt -> expr ;
3 stmt -> return expr_opt ;
4 stmt -> if internal_if
5 stmt -> for ID in expr : stmt_list end
6 stmt -> while expr : stmt_list end
7 stmt -> break ;
8 stmt -> continue ;
9
10 vdecl -> typ ID = expr
11 vdecl -> typ ID
12
13 internal_if -> expr : stmt_list else_list end
14
15 stmt_list -> NOTHING
16 stmt_list -> stmt_list stmt
17
18 expr_opt -> NOTHING
19 expr_opt -> expr
20
21 else_list -> NOTHING
22 else_list -> elif expr : stmt_list else_list
23 else_list -> else : stmt_list

```

```

24
25 ID -> ['a'-'z' 'A'-'Z'] ['a'-'z' 'A'-'Z' '0'-'9' '_']*

```

### 3.6.1 Expression Statement

Most statements have the form:

*expression*;

A semicolon is used to indicate the end of a statement.

### 3.6.2 Variable Declaration and Initialization

Variable declaration has the following form:

1. `typ ID;`

In this case, variable `ID` of type `typ` is declared but not initialized.

2. `typ ID = expr;`

In this case, variable `ID` of type `typ` is declared and initialized to the value of `expr`.

### 3.6.3 if statement

The `if` statement is a conditional construct that evaluates statements based on an expression that evaluates to Boolean. `if` statements can have the following three forms:

1. 

```
if expression:
    stmt_list
end
```

In this case, if *expression* evaluates to Boolean literal `true`, *stmt\_list* gets evaluated from top to bottom.

2. 

```
if expression:
    stmt_list1
else:
    stmt_list2
end
```

In this case, if *expression* evaluates to Boolean literal `true`, *stmt\_list1* will be evaluated; otherwise, *stmt\_list2* will be evaluated.

```

3.          if expression:
               stmt_list
          elif expression:
               stmt_list
          (more elif statements)
          else:
               stmt_list
          end

```

In this case, the expressions are evaluated sequentially from the top until one is found to be true, at which point the corresponding *stmt\_list* is executed. If none of the expressions are evaluated to be true, then the *stmt\_list* corresponding to else is executed. Note that the elif and else are optional.

### 3.6.4 while statement

The while statement is a looping construct that continuously evaluates a list of statements conditionally based on an expression that evaluates to a Boolean value.

while statement has the following form:

```

          while expression:
               stmt_list
          end

```

While *expression* evaluates to true, the statements represented by *stmt\_list* keep getting evaluated. Once *expression* evaluates to false, the statements stop being evaluated.

### 3.6.5 for statement

A for statement is a looping construct that iterates over a list. It has the following form:

```

          for var_name in expression:
               stmt_list
          end

```

The *expression*, which has the type list, is evaluated first. *stmt\_list* is evaluated once for every item in *expression*. On the first iteration, *var\_name* is assigned the first element of the list and *stmt\_list* is evaluated. At each subsequent  $i^{\text{th}}$  iteration, the variable is assigned the  $i^{\text{th}}$  element of the list and *stmt\_list* is evaluated.

### 3.6.6 **break** statement

The `break` statement is used inside a looping construct to break from the loop regardless of the condition of the loop. It has the following form:

```
break;
```

When the `break` statement gets evaluated inside a loop, all the statements after it inside the loop are skipped, and the execution moves out of the looping construct.

### 3.6.7 **continue** statement

The `continue` statement is used inside a looping construct to skip the current iteration of the loop. It has the following form:

```
continue;
```

After the `continue` statement gets evaluated, all the statements after it inside the loop are skipped, and the execution moves on to the next iteration of the loop.

### 3.6.8 **return** statement

The `return` statement is used inside a function to exit the function. It either returns an expression or nothing. It has the following forms:

1. 

```
return expression;
```

In this case, *expression* gets evaluated first, the execution exits the current function, and the result is returned.

2. 

```
return;
```

This `return` statement is used in `void` functions, and it simply exits the current function without returning anything.

## 3.7 **Functions**

Functions allow you to separate parts of your program into distinct snippets of code that can be called subsequently. Since functions are treated as first class citizens, they can be assigned as values to identifiers, passed as arguments to other functions, and returned as values from other functions. A function is globally-scoped and can only be defined on the top level. Nested functions are not allowed in TEAM.

### 3.7.1 **Function Definitions**

A function definition consists of information regarding the function's name, return type, types and names of parameters, and the body of the function. The function body consists of a series of statements and is terminated by the keyword `end`. The types of the parameters and the return type are required in the signature of the function definition. If a function does not return anything, use the keyword `void` in place of the return type.

Here is the general form of a function definition:

```
type foo (type p1, type p2, ...):  
    statements  
end
```

Below defines a simple function that adds two numbers.

```
1 int add(int x, int y):  
2     return x + y;  
3 end
```

### 3.7.2 Calling Functions

A function is called by using its name and supplying any parameter it requires. Furthermore it can also be called using the name of the variable the function was assigned to in the current scope. Here is an example that calls the add function defined above:

```
1 add(1, 2);
```

A function call is an expression. A call to a non-void function evaluates to the return value from the function. This can be used as part of other expressions and statements. However, the evaluation of a call expression to a void function cannot be a part of other expressions.

Here is an example where the returned value is assigned to a variable:

```
1 int x = add(1, 2);
```

More details on the grammar for function calls can be found in section 3.4.8.

### 3.7.3 Function Parameters

Function parameters can be any expression of any type except for void. TEAM functions use call-by-value semantics for parameter passing for all types except for list and file, which are passed by reference in TEAM functions.

Here is an example to demonstrate parameters of all types except list are passed by value:

```
1 void add_one(int x)  
2     x += 1;  
3 end  
4  
5 int x = 2;  
6 add_one(x);  
7 print("%d", x); // 2
```

Here is another example to demonstrate that a list is changed if it is modified inside a function where it is passed:

```
1 void change_first_element(list x)  
2     x[0] = 2;  
3 end  
4  
5 list x = [1];  
6 change_first_element(x);  
7 print("%d", x[0]); // 2
```



Once the function definition has been written, the number of arguments stay fixed. A call to the function is only recognized if the number of arguments matches the number of parameters in the function definition.

### 3.7.4 Higher order functions

Functions are first class values in TEAM. They can be passed as argument to other functions and assigned to a variable of the same function type. The code snippet below demonstrates an example of higher order functions in TEAM.

```
1 string shout(string text):
2     string res = "";
3     for c in text:
4         res += upper(c);
5     end
6
7     return res;
8 end
9
10 string whisper(string text):
11     string res = "";
12     for c in text:
13         res += lower(c);
14     end
15
16     return res;
17 end
18
19 /*
20 greet takes a function that takes a string
21 and returns a string (string->string)
22 and returns nothing (void)
23 */
24 void greet((string)->string func):
25     string greeting = func("Hello");
26     print("%s", greeting);
27 end
28
29 /*
30 shout function is assigned to a function
31 variable named shout_var
32 */
33 (string)->string shout_var = shout;
34 greet(shout_var); // prints "HELLO"
35 greet(whisper); // prints "hello"
```

## 3.8 Scope

TEAM is statically scoped. Variables can be defined on the top level, inside functions and as formal parameters to functions. Here are the scoping rules in TEAM:

- Variable declared on the top level is visible to all top level statements that come after its declaration and everywhere within functions

```
1 int y = 5;
2 print("%d", y); // y is visible here
3
4 void local():
```

```

5     print("%d", y); // y is also visible here
6 end

```

- Variable declared within a loop is only visible to statements inside the loop that come after its declaration

```

1 for i in 1..10:
2     int j = 1;
3     print("%d", j); // j is visible here
4 end
5 // j is not visible here

```

- Variable declared within a function is only visible to statements inside the function that come after the declaration

```

1 void local():
2     int j = 1;
3     print("%d", j); // j is visible here
4 end
5 // j is not visible here

```

- Formal parameters are only visible inside its respective function

```

1 void local(int j):
2     print("%d", j); // j is visible here
3 end
4 // j is not visible here

```

- When a variable is accessed, it resolves to the variable with the same name in the same space or the closest enclosing space where it is defined

```

1 int j = 1;
2 void local():
3     int j = 2;
4     for i in 1..10:
5         int j = 3;
6         print("%d", j); // j will be 3 here
7     end
8 end

```

### 3.9 Program Structure

A program in TEAM is a list of statements on the top-level and functions. TEAM does not require a main function. Statements on the top-level are evaluated from the top. However, functions are declared in no particular order and have access to all the variables declared on the top level regardless of the position of the variable declaration.

## 4 Built-in Functions

All function parameters will be specified by the name of the parameter followed by the type of the parameter wrapped in parenthesis. The special italicized keyword *type* indicates that the type of the parameter can be of any type except for `void`. The type of the return value is specified similarly. The following are built-in functions supported by TEAM:

### 4.1 List

TEAM offers the following built-in functions for list:

- `length(l)`: determines the length of a list.
  - Parameter:
    - \* `l (list)`: a list
  - Returns: length of the list (`int`)
- `append(l, ele)`: appends a new element to the end of a list. Note that lists in TEAM are immutable, so the original list is unmodified in the end of the function call.
  - Parameter:
    - \* `l (list)`: a list of type *type*
    - \* `ele (type)`: the element to be appended
  - Returns: a new list (`list`)
- `insert(l, ele, i)`: appends a new element to a list at index `i`. Note that lists in TEAM are immutable, so the original list is unmodified in the end of the function call.
  - Parameter:
    - \* `l (list)`: a list of type *type*
    - \* `ele (type)`: the element to be appended
    - \* `i (int)`: the position where the new element is inserted
  - Returns: a new list (`list`)
- `+`: concatenates two lists. The original lists are unmodified. `+` is an infix function that can operate on two lists.
  - Parameter:
    - \* `arr1 (list)`: a list
    - \* `arr2 (list)`: a list
  - Returns: a new list (`list`) with two lists concatenated.

The following code snippet demonstrates the usage of the aforementioned list related functions.

```
1 list courses_taken = [11, 15, 40, 61];
2 list courses_to_take = [105, 160, 170];
3
4 if contains(courses_taken, 160):
5     print("This kid knows algorithms\n");
6 else:
7     print("This kid does know algorithms\n");
8 end
9
10 list courses_required = courses_taken + courses_to_take;
11 // courses_required is [11, 15, 40, 61, 105, 160, 170]
12 // courses_taken is still [11, 15, 40, 61]
13 // courses_to_take is still [105, 160, 170]
14
```

```

15 list more_cs = append(courses_taken, 107);
16 // more_cs is [11, 15, 40, 61, 107];
17
18 list more_math = insert(courses_taken, 34, 0);
19 // more_math is [34, 11, 15, 40, 61, 107];

```

## 4.2 Print, Formatted Strings, and File I/O

TEAM offers a print function with support of C-like formatted strings. The `print` function can be used to print any number of arguments. It accepts a string as the first argument, and allows any subsequent arguments to be printed according to the formatted string and C specifications. However, there is one exception that does not align with conventional C standards. Booleans use the `%s` format specifier. Instead of using 1 and 0 like in C, TEAM print outputs `true` for a true value and `false` for a false value.

Here are a few examples of how outputs can be formatted:

```

1 int i = 5;
2 float f = 3.14159;
3 char c = 'a';
4 print("%d", i); // displays 5
5 print("%f", f); // displays 3.14
6 print("%c", c); // displays a
7 print("%s", true); // displays true
8 print("Hello!"); // strings do not require an explicit formatter

```

It is straightforward to work with files in TEAM. To read from a file, a variable of file type should be defined first; this variable serves as a buffer for the file. Then the content of the file is transferred to the buffer with the built-in `open` function. Quick access to the file content can be achieved with the `readline` function.

- `open(file_name, mode)`: creates a new file or opens an existing file
  - Parameter:
    - \* `file_name` (string): the path of the file to be opened
    - \* `mode` (string): the access mode
  - Returns: a file handle
  - Note: If mode is not specified, the file is opened in read mode by default. Other modes can be explicitly mentioned:
    1. `r`: opens a file for reading only. The file must already exist for the program to behave as expected.
    2. `r+`: opens a file for reading and writing. The stream is positioned at the beginning of the file. The file must already exist for the program to behave as expected.
    3. `w`: opens a file for writing only and overwrite any existing file with the same name. If the file doesn't exist, a new file is created.
    4. `w+`: opens a file for reading and writing. The stream is positioned at the beginning of the file. The file is created if it does not exist, otherwise it is truncated.
    5. `a`: opens a file for appending new information to it. If the file doesn't exist, a new file is created.
- `readline(file_handle)`: reads an entire line from the given file specified by handle `file_handle`.
  - Parameter:

- \* `file_handle (file)`: the file handle that refers to the file
- Returns: the current line of the file, which is null-terminated and includes the newline character, if one was found.
- `write(file_handle, content)`: writes the given content to the file specified by `file_handle`
  - Parameter:
    - \* `file_handle (file)`: the file handle that refers to the file
    - \* `content (string)`: the string to be written
  - Returns: nothing (void)
- `close(file_handle)`: closes the opened file specified by `file_handle`. If the file was never opened, then the program might exhibit undefined behavior.
  - Parameter:
    - \* `file_handle (file)`: the file handle that refers to the file
  - Returns: nothing (void)

The following code snippet demonstrates the usage of the aforementioned file I/O related functions.

```

1 list roster = ["Lulu", "Yingjie", "Saurav", "Naoki"];
2
3 file output = open("output.txt", "w");
4
5 for name in roster:
6     write(output, name + "\n");
7 end
8
9 close(output);
10
11 file input = open("output.txt", "r");
12 string currline;
13
14 while ((currline = readline(input)) != ""):
15     print(currline);
16 end
17
18 close(input);

```

## 5 Standard Library Functions

The following standard library functions are all implemented in TEAM. These functions demonstrate how users can expand the power of TEAM using well-constructed small blocks of built-in functions. These standard library functions also provide users with an interface to carry out essential tasks with lists, strings, and chars.

### 5.1 List Library

TEAM comes with the following list standard library functions.

- `contains(l1, l2)`: determines if the elements within `l2` exist in `l1`.
  - Parameter:
    - \* `l1 (list)`: a list
    - \* `l2 (list)`: another list
  - Returns: a boolean value (`bool`) that indicates the state of existence. Only returns `true` if all elements within `l2` exist in `l1`.
- `remove_int(l, ele, all)`: removes an integer from `l`. The original list is unmodified.
  - Parameter:
    - \* `l (list)`: a list
    - \* `ele (int)`: the element to be removed
    - \* `all (bool)`: when `all` is `true`, the entire list is traversed, and all duplicates are removed; otherwise, only the first occurrence is removed. In either case, the original list is unmodified.
  - Returns: a new list (`list`)
- `remove_float(l, ele, all)`: removes a float from `l`. The original list is unmodified.
  - Parameter:
    - \* `l (list)`: a list
    - \* `ele (float)`: the element to be removed
    - \* `all (bool)`: when `all` is `true`, the entire list is traversed, and all duplicates are removed; otherwise, only the first occurrence is removed. In either case, the original list is unmodified.
  - Returns: a new list (`list`)
- `remove_bool(l, ele, all)`: removes a boolean from `l`. The original list is unmodified.
  - Parameter:
    - \* `l (list)`: a list
    - \* `ele (bool)`: the element to be removed
    - \* `all (bool)`: when `all` is `true`, the entire list is traversed, and all duplicates are removed; otherwise, only the first occurrence is removed. In either case, the original list is unmodified.
  - Returns: a new list (`list`)
- `remove_char(l, ele, all)`: removes a character from `l`. The original list is unmodified.
  - Parameter:
    - \* `l (list)`: a list
    - \* `ele (char)`: the element to be removed

- \* `all (bool)`: when `all` is `true`, the entire list is traversed, and all duplicates are removed; otherwise, only the first occurrence is removed. In either case, the original list is unmodified.
- Returns: a new list (`list`)
- `remove_string(l, ele, all)`: removes a string from `l`. The original list is unmodified.
  - Parameter:
    - \* `l (list)`: a list
    - \* `ele (string)`: the element to be removed
    - \* `all (bool)`: when `all` is `true`, the entire list is traversed, and all duplicates are removed; otherwise, only the first occurrence is removed. In either case, the original list is unmodified.
  - Returns: a new list (`list`)

The following code snippet demonstrates the usage of some of the aforementioned list related functions.

```

1 list courses_taken = [11, 15, 40, 61];
2 list courses_to_take = [105, 160, 170];
3 list courses_required = courses_taken + courses_to_take;
4
5 if contains(courses_taken, [160]):
6     print("This kid knows algorithms\n");
7 else:
8     print("This kid does not know algorithms\n");
9 end
10
11 list cs_minor = remove_int(remove_int(courses_required, 160, true), 105,
12                               true);
13 // courses_required is still [11, 15, 40, 61, 105, 160, 170]
14 // cs_minor is [11, 15, 40, 61, 170]
```

## 5.2 String/Char Library

The string/char library provides essential functions to perform string and char manipulation. TEAM comes with the following string/char library functions.

- `split(str, delim)`: takes in a string and a delimiter and returns a list of substrings divided by the delimiter.
    - Parameter:
      - \* `str (string)`: a string
      - \* `delim (string)`: a string
    - Returns: a list (`list`) of strings that are separated by the delimiter
- ```

1 list l = split("Hello,World", ",");
2 // l = ["Hello", "World"]
```
- `join(str_list, delim)`: takes in a list of strings and combines them into one string joined by `delim`.
    - Parameter:
      - \* `str_list (list)`: a list of strings
      - \* `delim (string)`: a string

- Returns: a new string (string)

```
1 list l = ["Jack", "likes", "fishing"];
2 string s = join(l, " ");
3 // s = "Jack likes fishing"
```

- `startswith(s, c)`: takes in a string and a char returns true if the string starts with the char and false if otherwise.

- Parameter:

- \* `s` (string): a string

- \* `c` (char): a char

- Returns: a boolean (bool)

```
1 bool b = startswith("hello world", 'h');
2 // b = true
3 bool c = startswith("hello world", 'i');
4 // c = false
```

- `endswith(s, c)`: takes in a string and a char returns true if the string ends with the char and false if otherwise.

- Parameter:

- \* `s` (string): a string

- \* `c` (char): a char

- Returns: a boolean (bool)

```
1 bool b = endswith("chocolate", 'e');
2 // b = true
3 bool b = endswith("chocolate", 'm');
4 // b = false
```

- `lower(c)`: takes in a char and returns the lower case of `c`. Note that the original char is not changed after the function call.

- Parameter:

- \* `c` (char): a char

- Returns: the lower case version of the char (char)

```
1 char c = lower('H');
2 // c = 'h';
```

- `upper(c)`: takes in a char and returns the capitalized version of the `c`. Note that the original char is not changed after the function call.

- Parameter:

- \* `c` (char): a char

- Returns: the capitalized version of the char (char)

```
1 char c = upper('h');
2 // c = 'H'
```



## 6 Regular Expressions

Regular expressions describe patterns that are used to determine if some other target string has characteristics specified by the pattern. TEAM provides robust built-in features for using regular expressions. Our features offer the ability to match strings to regular expressions and replace matches with another string. A regular expression is passed into a function as a string but is interpreted as a regular expression using the rules described below.

### 6.1 Meta characters

- `\`: indicates a special sequence
- `^`: matches the beginning of a string
- `$`: matches the end of a string
- `|`: alternation, allows the target to be matched over a number of patterns
- `()`: grouping, allows the target to be matched over a number of patterns specified in the parenthesis
- `.`: any character except newline character
- `*`: zero or more occurrences
- `+`: one or more occurrences
- `?`: zero or one occurrence(s)
- `[]`: a set of characters
- `{ }`: denotes number of occurrences expected

### 6.2 Special sequences

- `\d`: matches if the target string contains a digit
- `\D`: matches if the target string contains a non-digit character
- `\w`: matches any single letter, number or underscore
- `\W`: matches any single character that doesn't match by
- `\s`: matches if the target string contains a whitespace
- `\S`: matches if the target string contains no whitespace
- `\b`: matches the specified characters to the beginning of a word in the target string
- `\B`: matches the specified characters somewhere in the target string, except at the beginning of a word
- `\t`: matches a tab character in the target string
- `\n`: matches a newline character in the target string
- `\0`: matches a null character in the target string

### 6.3 Sets

The following rules can be used to define a set (`[]`).

- `[num1-num2]`: matches any digit between *num1* and *num2*, inclusive
- `[letter1-letter2]`: matches any upper or lower case letter between *letter1* and *letter2*, inclusive
- `[digit1, digit2, ...]`: matches any digit included in the sequence
- `[letter1, letter2, ...]`: matches any letter included in the sequence

## 6.4 Regular Expression Examples

The code snippet below demonstrates some of the useful regular expressions that follows the rules described above.

```
1 // One or more digits between 0 and 9, inclusive
2 string digits = "[0-9]+";
3
4 // Any string that starts with "chachacha"
5 string echos = "^cha{3}[.]*";
6
7 // Any string that ends with "a" or "b"
8 string ab = "(a|b)$";
```

## 6.5 Built-in Functions

- `match(target, regex)`: takes a target string and a regular expression and returns `true` if target matches the regex and `false` if otherwise.

– Parameter:

\* `target (string)`: a target string that will be matched against the given regular expression

\* `regex (string)`: a string that is interpreted as a regular expression

– Returns: a Boolean value (`bool`) that indicates if the target string matches the regex

```
1 bool b = match("guru99 get", "(g\\w+)\\W(g\\w+)");
2 // b = true
```

- `find(target, regex)`: takes a target string and a regular expression and returns the first substring of the string that matches the regular expression.

– Parameter:

\* `target (string)`: a target string that will be matched against the given regular expression

\* `regex (string)`: a string that is interpreted as a regular expression

– Returns: If no match is found, return the empty string. Otherwise, return the first match (`string`) found.

```
1 string s = find("guru99 get", "(g\\w+)\\W(g\\w+)");
2 // s = "guru99"
```

- `findall(target, regex)`: takes a target string and a regular expression and returns a list of all matches.

– Parameter:

\* `target (string)`: a target string that will be matched against the given regular expression

\* `regex (string)`: a string that is interpreted as a regular expression

– Returns: If no match is found, return the empty list. Otherwise, return a list (`list`) of all matches found.

```
1 list l = findall("2+2 3*3 4-4 5+5 6*6", "\\d+[\\+\\-x\\*]\\d+");
2 // l = ["2+2", "3*3", "4-4", "5+5", "6*6"]
```

- `replaceall(target, regex, replc)`: takes a target string, a regular expression, and a replacement string and returns a string that is constructed by replacing all matches in target with `replc`. Note that the original string (`target`) is not modified.

- Parameter:
  - \* `target (string)`: a target string that will be matched against the given regular expression
  - \* `regex (string)`: a string that is interpreted as a regular expression
  - \* `replc (string)`: a string that will replace the matches found
- Returns: a string (`string`) constructed from replacing all matches in `target` with `replc`.

```
1 string a = replaceall("u", "a", "jumbo tufts");
2 // a = "jambo tafts"
```

- `replace(target, regex, replc, count)`: takes a target string, a regular expression, replacement string, and count number and returns a string that is constructed by replacing the first count number of matches in `target` with `replc`. Note that the original string (`target`) is not modified.

- Parameter:
  - \* `target (string)`: a target string that will be matched against the given regular expression
  - \* `regex (string)`: a string that is interpreted as a regular expression
  - \* `replc (string)`: a string that will replace the matches found
  - \* `count (int)`: an integer that indicates how many matches should be replaced by `replc`
- Returns: a string (`string`) that is constructed by replacing `n` matches in `target` with `replc`, where `n` is specified by `count`.

```
1 string google = "www.google.com";
2 string facebook = replace("google", google, "facebook", 1);
3 // facebook = "www.facebook.com"
```

## 7 Project Plan

### 7.1 Planning, Specification, and Development Process

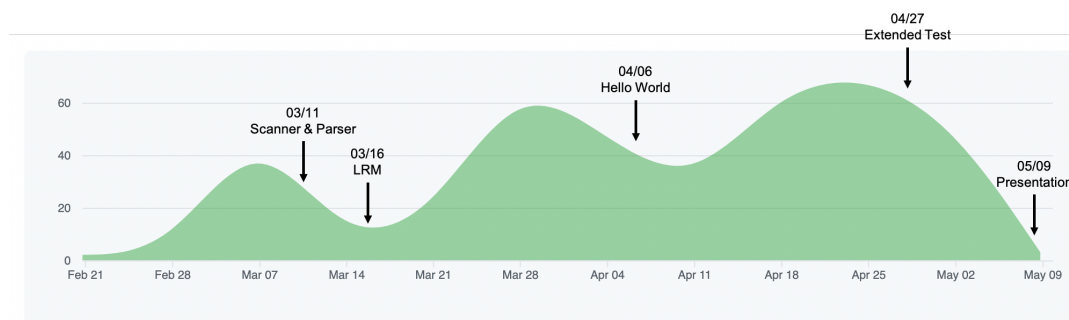
At the very beginning of the project, we each came up with our own language and voted on the one the one we thought was the best to implement. We picked the area of text manipulation because we realized how cumbersome it is to perform some simple text related tasks with general purpose languages. After we had a better idea of what our language would specialize in, we decided to follow Python as a guide to design our syntax because of its simplicity. We then drafted our language reference manual where we wrote down some simple programs in our language. We worked on the lexer, parser, and semantic phase together as a group, but later split the work. We did not have a programming style guide but instead conducted group code review to ensure our code is clear and well-documented.

### 7.2 Project Timeline

For earlier deliverables, we followed the class deadlines. For later ones, we split up the work and implemented specific features at our own pace. We strove to finish the work before the deadline to give us enough time for testing and code cleaning. Here is an approximate order in which we implemented the major components of our project, with approximate dates.

| Feature/Deliverables                                | Finished Around |
|-----------------------------------------------------|-----------------|
| General Brainstorming                               | 02/13           |
| Language Proposal                                   | 02/19           |
| Scanner                                             | 02/27           |
| Parser                                              | 03/09           |
| Basic Test Suite for Scanner and Parser             | 03/11           |
| LRM                                                 | 03/16           |
| Implementation of Basic Language Features           | 03/24           |
| Hello World Deliverable                             | 04/06           |
| List Implementation, Slicing, and Indexing          | 04/11           |
| Regex Built-In Functions and Higher Order Functions | 04/20           |
| List Built-In Functions                             | 04/23           |
| Extended Test Suite                                 | 04/27           |
| String Standard Library Functions                   | 04/28           |
| List Standard Library Functions                     | 05/01           |
| File I/O                                            | 05/04           |
| Demo and Presentation                               | 05/07           |
| Final Report                                        | 05/11           |

The following git commit graph gives a visual representation of our timeline. Note that the commit peaks are usually a few days before the deadline.



### 7.3 Roles and Responsibilities

We did not assign a specific role for each group member as we believe it is more efficient to split the language features and have individuals be responsible for a few components of the language. In this way, each individual becomes familiar with the majority of the code base can help others debug. Here is a list of the parts each individual has worked on:

- Lulu: List polymorphism, List standard library functions, Regular expressions
- Saurav: List implementation
- Yingjie: List built-in functions, String standard library functions, Test script
- Naoki: File IO, print

All group members are involved in the implementation of lexer, parser, semantic checker, and code generator.

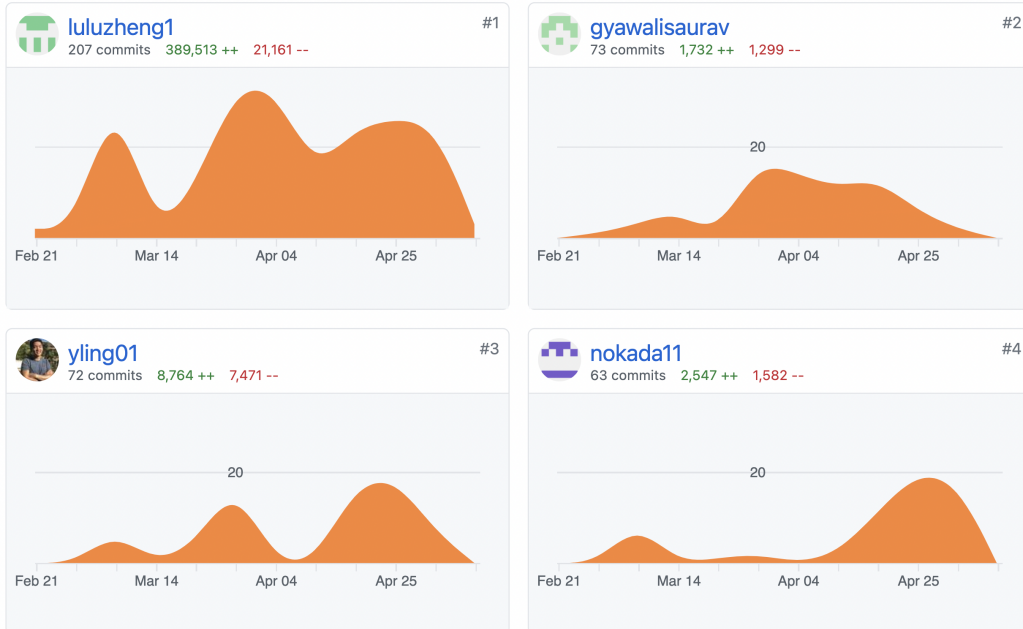
### 7.4 Development Tools

The IDE used by all group members was VSCode. The majority of this project written in OCaml. Built-in functions for file IO and regular expressions are written in C. The test script is written in Python. Below is a list of the tools and languages we used for development.

- Ocaml 4.11.1
- Opam 2.0.7
- LLVM 11.1.0
- Python 3.x
- C compiler/Make utility

### 7.5 Project Log

The entire commit history is included in the first section of the appendix. Each group member's contribution is summarized below.

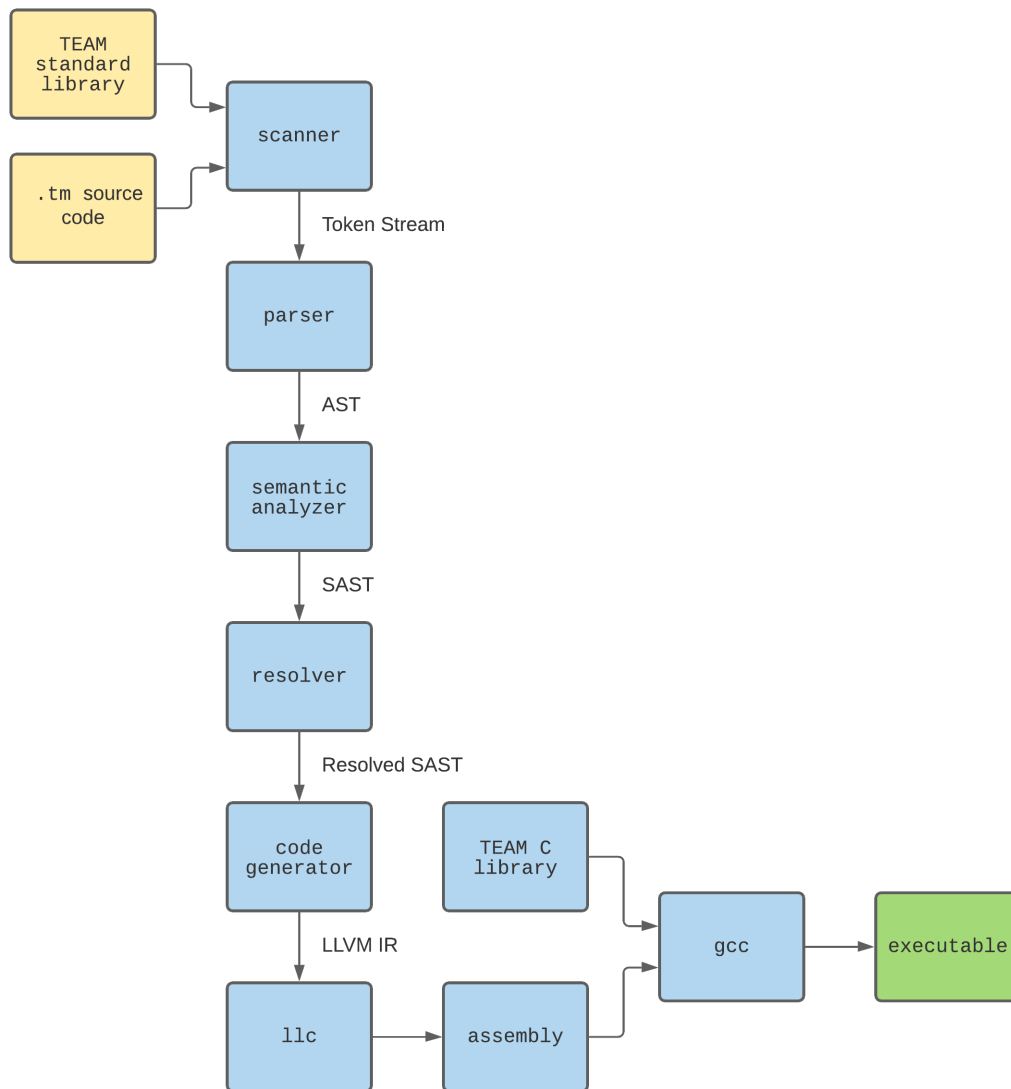


## 8 Architectural Design

### 8.1 Overview

The architecture of the TEAM compiler follows the conventional compiler design. It contains a scanner, parser, semantic analyzer, and code generator. We have added one additional translation pass to the architectural structure of our language: type resolving, which is dedicated to resolve any unspecialized list types in a program.

This section will describe each component of the compiler as well as point out some key implementation details. The architecture of the TEAM compiler is presented in the block diagram below.



### 8.2 Standard Library

TEAM provides two standard libraries: one for string manipulation and another for list manipulation. Both standard libraries include a collection of useful functions that is accessible to every TEAM program. For more information, a list of our standard library functions is provided

in the LRM. The standard libraries are scanned and parsed into a separate AST that is later prepended to the AST of the program before they are passed to the semantic analyzer.

### 8.3 Scanner

The scanner is the first step of compilation. It takes as input a .tm file and performs lexical analysis to process the file content into a series of tokens. It then passes the tokens to the parser. Any input that is not recognized by the scanner will result in an exception thrown pointing to the specific line where an illegal character was found.

### 8.4 Parser

The parser takes the tokens generated from the scanner and generates an abstract syntax tree based on the grammar rules of TEAM. Any input that does not conform to the grammar will result in an exception thrown pointing to the specific line where a syntax error was found.

### 8.5 Semantic Analyzer

The semantic analyzer receives an abstract syntax tree from the parser and produces a semantically checked abstract syntax tree, where the type of every expression and statement is known with the exception of ones that may contain any lists. It will check the types are well formed and infer the types of every expression that contains any lists. In TEAM, the type of lists are inferred if not already known. The semantic analyzer traverses the AST once and performs best-effort type inference, passing the semi-type inferred SAST to the resolver. Exceptions will be thrown if a semantic error is encountered.

### 8.6 Type Resolver

The type resolver receives the SAST generated from the semantic analyzer and outputs a resolved and semantically checked abstract syntax tree, where the type of every expression and statement is known. It performs a second round of type inference on expressions that contain lists. The bulk of the type inference is performed in variable assignments, variable declarations, function calls, and function declarations. When it is finished, the resolver passes the type resolved SAST to the code generator.

### 8.7 Code Generator

The code generator takes in a resolved SAST and produces an equivalent LLVM module, which can then be compiled by llc, gcc and down to executable code. It generates LLVM code for many key features of TEAM, including lists, strings, and slice/string slicing and indexing. Moreover, the code generator also interfaces with C code to implement some of the more complicated parts of our language.

### 8.8 Builtin C Library

The builtin library, internal to TEAM, is written in C and contains functions that handle operations that deal with regular expressions and file I/O, such as opening, reading, writing, and closing a file. When compiling a TEAM program into executable binary, the builtin library is compiled to LLVM by gcc and linked with the assembly code of the user program that is created from the generated LLVM module.

### 8.9 Workflow

- Scanner, Parser: All members
- Semantic Analyzer: All members

- Resolver: Lulu
- Code Generator: All members
- Standard Library:
  - String: Yingjie
  - List: Lulu
- Builtin C Library:
  - Regular Expression: Lulu
  - File I/O: Naoki



## 9 Test Plan

We strove to write unit tests immediately after a specific language feature or a phase (i.e. `ast`, `sast`, `codegen`) is implemented. After a significant portion of the language features were implemented, we started to writing tests that depend on multiple components of the language. We provide an automated test script written in Python that runs all of the tests and compares the generated results to the expected ones. The details are described below.

### 9.1 Test Suite/Automation

Our test suite includes both positive tests (tests that are expected to generate output) and negative tests (tests that cause the compiler to throw an error). The negative tests are prefixed with ‘bad’. In either case, the workflow is as below:

- Run our compiler against the input files to generate output. Depending on the mode of the test script is running (described below), the output can be an abstract syntax tree (`ast`), a semantically checked `ast` (`sast`), or expected results from a program.
- Compare the raw output from the compiler to the expected output files using the `diff` command.
- If a test fails, the name of the test is printed along with the difference between raw and expected output.

The automated test script is located in the `scripts/` folder; the test suite is located in the `tests/` folder; and the expected output is located in the `ref/` folder. Inside of `test/` and `ref/`, tests and references are divided into different phases. To run our automated test script at **root directory**, do:

```
python scripts/runtest.py -m mode
```

where *mode* can be `ast`, `sast`, `codegen`, `extended`, and `all`. When the mode flag is omitted, the mode defaults to `extended`.

### 9.2 Test Scripts

Here is a listing of our test scripts.

### 9.2.1 Makefile

The Makefile is required by `runtests.py` to compile the top level.

```
1 all: team.native fileio regex
2 team.native : ./src/parser.mly ./src/scanner.mll ./src/codegen.ml ./src/
   semant.ml ./src/resolve.ml ./src/team.ml
3   opam config exec -- \
4   ocamlbuild -use-ocamlfind ./src/team.native
5 # For built-in functions
6 .PHONY: fileio
7 fileio: ./c_library/fileio.c
8   gcc -c -Wall -g ./c_library/fileio.c
9   gcc -g -o fileio -DBUILD_TEST ./c_library/fileio.c
10
11 .PHONY: regex
12 regex : ./c_library/regex.c
13   gcc -c -Wall -g ./c_library/regex.c
14   gcc -g -lpcposix -lpcr2-8 -o regex -DBUILD_TEST ./c_library/regex.c
15
16 .PHONY : clean
17 clean :
18   ocamlbuild -clean
19   rm *.o
20   rm regex
21   rm fileio
22   rm -r *.dSYM/
```

### 9.2.2 runtests.py

This python file does the automated testing following the workflow discussed in 9.1.

```
1 #!/usr/bin/env python3
2 import sys
3 import os
4 import subprocess
5 import optparse
6
7 class bcolors:
8     OKGREEN = '\033[92m'
9     FAIL = '\033[91m'
10    ENDC = '\033[0m'
11    WARNING = '\033[93m'
12    UNDERLINE = '\033[4m'
13
14    SCANNER_PARSER_DIR = ("tests/ast_tests", "ref/ast_ref")
15    SEMANT_DIR = ("tests/sast_tests", "ref/sast_ref")
16    CODEGEN_DIR = ("tests/codegen_tests", "ref/codegen_ref")
17    EXTENDED_DIR = ("tests/extended_tests", "ref/extended_ref")
18
19    def getCompiledFiles():
20        fileList = [f for f in os.listdir(".") if ".o" in f]
21        return " ".join(fileList)
22
23    def sortingKey(fileName):
24        return fileName.split(".")[0]
25
26    def runTests(testMode):
27        if testMode == "sast":
28            dirTuple = SEMANT_DIR
29        elif testMode == "ast":
30            dirTuple = SCANNER_PARSER_DIR
31        elif testMode == "codegen":
32            dirTuple = CODEGEN_DIR
33        elif testMode == "extended":
34            dirTuple = EXTENDED_DIR
35        else:
36            print(bcolors.FAIL + "Test mode: {} not supported".format(testMode)
37                  + bcolors.ENDC)
38            sys.exit()
39
40        testFileDir, refFileDir = dirTuple
41        testFiles = list(map(lambda x: "/".join((testFileDir, x)), sorted(filter(
42            lambda x: "tm" in x, os.listdir(testFileDir)), key=sortingKey)))
43        refFiles = list(map(lambda x: "/".join((refFileDir, x)), sorted(filter(
44            lambda x: "log" in x, os.listdir(refFileDir)), key=sortingKey)))
45
46        assert len(testFiles) == len(refFiles)
47
48        for i in range(len(testFiles)):
49            testFile, refFile = testFiles[i], refFiles[i]
50
51            logFile = runFile(testFile, testMode)
52            checkResults(logFile, refFile)
53
54    def printFailedTestMessage(f_generated, f_reference):
55        tmFile = f_generated.split("/")[-1].split(".")[0]
```

```

53     print(bcolors.FAIL + "{:20s} -- FAILED!\n".format(tmFile + ".tm") +
bcolors.ENDC)
54     print("+" * 100)
55     print(bcolors.WARNING + bcolors.UNDERLINE + "Running diff...\n" +
bcolors.ENDC)
56     process = subprocess.Popen(["diff", "-y", f_generated, f_reference],
57                                stdout=subprocess.PIPE,
58                                stderr=subprocess.PIPE)
59     stdout, stderr = process.communicate()
60     print(bcolors.WARNING +
61           "command: diff -y {} (output) {} (standard)\n".format(
f_generated, f_reference) +
62           bcolors.ENDC)
63     print(bcolors.WARNING + stdout.decode("utf-8") + bcolors.ENDC)
64     print("+" * 100)
65
66 def printSuccessTestMessage(logFile):
67     tmFile = logFile.split("/")[-1].split(".")[0]
68     print(bcolors.OKGREEN + "{:20s} -- OK!\n".format(tmFile + ".tm") +
bcolors.ENDC)
69
70 def checkResults(f_generated, f_reference):
71     generated = [line for line in open(f_generated)]
72     reference = [line for line in open(f_reference)]
73
74     if len(generated) != len(reference):
75         printFailedTestMessage(f_generated, f_reference)
76         return
77     for index, astGenerateLine in enumerate(generated):
78         if astGenerateLine != reference[index]:
79             printFailedTestMessage(f_generated, f_reference)
80             return
81     printSuccessTestMessage(f_generated)
82
83 def runFile(fileName, testMode, userInput=False):
84     if testMode == "ast":
85         flag = "-a"
86     elif testMode == "sast":
87         flag = "-s"
88     elif testMode == "codegen":
89         flag = "-l"
90     elif testMode == "extended":
91         flag = "-l"
92     else:
93         print(bcolors.FAIL + "Test mode: {} not supported".format(testMode)
+ bcolors.ENDC)
94         sys.exit()
95
96     if testMode not in ["codegen", "extended"] or "bad" in fileName:
97         command = ['./team.native', flag, fileName]
98     else:
99         command = ['./scripts/compile.sh', fileName, "run"]
100     process = subprocess.Popen(command,
101                                stdout=subprocess.PIPE,
102                                stderr=subprocess.PIPE)
103     stdout, stderr = process.communicate()
104     clean = ['./scripts/compile.sh', fileName, "clean"]
105     process = subprocess.Popen(clean,
106                                stdout=subprocess.PIPE,

```

```

107         stderr=subprocess.PIPE)
108
109     if userInput:
110         dir_prefix = "user_log"
111     else:
112         dir_prefix = "log/" + "_".join((testMode, "log"))
113
114     if not os.path.exists(dir_prefix):
115         os.makedirs(dir_prefix)
116     filename = '{}/{}.log'.format(dir_prefix, fileName.split('/')[1].split(
117         '.')[0])
118     with open(filename, 'w+') as fo:
119         toWrite = stdout if stdout else stderr
120         fo.write(toWrite.decode('utf-8'))
121     return filename
122
123 def interpretCommand(command):
124     return True if command[0].upper() == "T" else False
125
126 def compile(topLevel):
127     process = subprocess.Popen(['ocamlbuild', '-use-ocamlfind', topLevel],
128                                stdout=subprocess.PIPE,
129                                stderr=subprocess.PIPE)
130     stdout, stderr = process.communicate()
131     if 'failed' in stdout.decode('utf-8') or "Error" in stdout.decode('utf-8'):
132         print(bcolors.FAIL + "Error detected when compiling {}. See message
133             below.".format(topLevel) + bcolors.ENDC)
134         print(stdout.decode('utf-8'))
135         sys.exit()
136
137 def clean(target):
138     if target == "ocaml":
139         process = subprocess.Popen(['ocamlbuild', '-clean'],
140                                    stdout=subprocess.PIPE,
141                                    stderr=subprocess.PIPE)
142
143 if __name__ == "__main__":
144     parser = optparse.OptionParser()
145     parser.add_option('-c', '--recompile',
146                       dest='recompile', default='False',
147                       help='True to recompile top level.')
148     parser.add_option('-t', '--testFile',
149                       dest='testFile', default='',
150                       help='Specify one test file. AST is written.')
151     parser.add_option('-r', '--reference',
152                       dest='reference', default='',
153                       help='The reference to compare generated AST.')
154     parser.add_option('-l', '--topLevel',
155                       dest='topLevel', default='team.native',
156                       help='Name of the top level')
157     parser.add_option('-m', '--testMode',
158                       dest='testMode', default='extended',
159                       help='ast, sast, codegen, extended, all')
160
161     options, args = parser.parse_args()
162     recompile = interpretCommand(options.recompile)
163     testFile = options.testFile

```

```

163     topLevel = options.topLevel
164     reference = options.reference
165     testMode = options.testMode
166
167     if reference != '' and testFile == '':
168         print("Error detected!\nReference specified with no test file
169         specified.\nExiting...\n")
170         sys.exit()
171
172     if recompile or topLevel not in os.listdir('/'):
173         compile(topLevel)
174
175     if testFile != '':
176         if testMode == "all":
177             print("Error detected!\n Test mode cannot be all when testing a
178             single file")
179             sys.exit()
180             logFile = runFile(testFile, testMode, True)
181             checkResults(logFile, reference)
182         else:
183             testMode = [testMode] if testMode != "all" else ["ast", "sast", "
184             codegen", "extended"]
185             for m in testMode:
186                 print(bcolors.WARNING + bcolors.UNDERLINE + "\nTest mode: {} \n".
187                 format(m) + bcolors.ENDC)
188                 runTests(m)

```

## 9.3 Examples of Tests

We show some of the test programs, the LLVM generated by the compiler, and the output when running the executable. Note that because the standard library functions are prepended to every .tm file, the actual LLVM is very long. Therefore, only the LLVM relevant to the current test program is shown here. For a complete standard library functions in TEAM, see appendix.

### 9.3.1 max\_profit.tm

Below is the TEAM source code:

```

1  /* This program maximizes profit by choosing a single day to buy
2  * one stock and choosing a different day in the future to sell
3  * that stock.
4  * Return the maximum profit one can achieve from this transaction.
5  * If no profits can be achieved, return 0. */
6
7  int max_profit(list stock_prices):
8      int min_price = stock_prices[0];
9      int max_profit = 0;
10
11     for i in 1..length(stock_prices)-1:
12         int current_price = stock_prices[i];
13         int next_price = stock_prices[i+1];
14         if current_price > next_price:
15             int potential_profit = current_price - min_price;
16             max_profit += potential_profit;
17             min_price = stock_prices[i+1];
18         else:
19             if min_price > current_price:
20                 min_price = current_price;

```

```

21         end
22     end
23 end
24 return max_profit;
25 end
26
27 print("%d\n", max_profit([5,2,6,9,3]));

```

The program is expected to have the following output:

```

1 7

```

Below is the LLVM generated by the compiler:

```

1 ; ModuleID = 'TEAM'
2 source_filename = "TEAM"
3
4 %list_item = type <{ i8*, %list_item* }>
5
6 @ASCII = global %list_item** null
7 @string = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
8 @string.1 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
9 @string.2 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
10 @string.3 = private unnamed_addr constant [6 x i8] c"wolrd\00", align 1
11 @string.4 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
12 @string.5 = private unnamed_addr constant [5 x i8] c"what\00", align 1
13 @string.6 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
14 @string.7 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
15 @string.8 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
16 @string.9 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
17 @string.10 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
18 @string.11 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
19
20 declare i32 @printf(i8*, ...)
21
22 declare double @pow(double, double)
23
24 declare i8* @fopen(i8*, i8*)
25
26 declare i32 @close(i8*)
27
28 declare i8* @readline(i8*)
29
30 declare i8* @write(i8*, i8*)
31
32 declare i1 @match(i8*, i8*)
33
34 declare i8* @find(i8*, i8*)
35
36 declare i8* @replace(i8*, i8*, i8*, i32)
37
38 declare i8* @replace_all(i8*, i8*, i8*)
39
40 declare %list_item** @find_all(i8*, i8*)
41
42 define i32 @main() {
43 entry:
44     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
45         i1** null, i32 1) to i32))
46     %list = bitcast i8* %mallocall to %list_item**

```

```

46 %allocacll1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
47 %list_item = bitcast i8* %allocacll1 to %list_item*
48 store %list_item zeroinitializer, %list_item* %list_item, align 1
49 %copied = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
50 store i8 90, i8* %copied, align 1
51 %data_ptr_container = getelementptr inbounds %list_item, %list_item* %
    list_item, i32 0, i32 0
52 store i8* %copied, i8** %data_ptr_container, align 8
53 %next = getelementptr inbounds %list_item, %list_item* %list_item, i32 0,
    i32 1
54 store %list_item* null, %list_item** %next, align 8
55 %allocacll3 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
56 %list_item4 = bitcast i8* %allocacll3 to %list_item*
57 store %list_item zeroinitializer, %list_item* %list_item4, align 1
58 %copied6 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
59 store i8 89, i8* %copied6, align 1
60 %data_ptr_container7 = getelementptr inbounds %list_item, %list_item* %
    list_item4, i32 0, i32 0
61 store i8* %copied6, i8** %data_ptr_container7, align 8
62 %next8 = getelementptr inbounds %list_item, %list_item* %list_item4, i32
    0, i32 1
63 store %list_item* %list_item, %list_item** %next8, align 8
64 %allocacll9 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
65 %list_item10 = bitcast i8* %allocacll9 to %list_item*
66 store %list_item zeroinitializer, %list_item* %list_item10, align 1
67 %copied12 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
68 store i8 88, i8* %copied12, align 1
69 %data_ptr_container13 = getelementptr inbounds %list_item, %list_item* %
    list_item10, i32 0, i32 0
70 store i8* %copied12, i8** %data_ptr_container13, align 8
71 %next14 = getelementptr inbounds %list_item, %list_item* %list_item10, i32
    0, i32 1
72 store %list_item* %list_item4, %list_item** %next14, align 8
73 %allocacll15 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
74 %list_item16 = bitcast i8* %allocacll15 to %list_item*
75 store %list_item zeroinitializer, %list_item* %list_item16, align 1
76 %copied18 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
77 store i8 87, i8* %copied18, align 1
78 %data_ptr_container19 = getelementptr inbounds %list_item, %list_item* %
    list_item16, i32 0, i32 0
79 store i8* %copied18, i8** %data_ptr_container19, align 8
80 %next20 = getelementptr inbounds %list_item, %list_item* %list_item16, i32
    0, i32 1
81 store %list_item* %list_item10, %list_item** %next20, align 8
82 %allocacll21 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
83 %list_item22 = bitcast i8* %allocacll21 to %list_item*
84 store %list_item zeroinitializer, %list_item* %list_item22, align 1
85 %copied24 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
86 store i8 86, i8* %copied24, align 1

```



```

87 %data_ptr_container25 = getelementptr inbounds %list_item, %list_item* %
    list_item22, i32 0, i32 0
88 store i8* %copied24, i8** %data_ptr_container25, align 8
89 %next26 = getelementptr inbounds %list_item, %list_item* %list_item22, i32
    0, i32 1
90 store %list_item* %list_item16, %list_item** %next26, align 8
91 %allocaall127 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
92 %list_item28 = bitcast i8* %allocaall127 to %list_item*
93 store %list_item zeroinitializer, %list_item* %list_item28, align 1
94 %copied30 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
95 store i8 85, i8* %copied30, align 1
96 %data_ptr_container31 = getelementptr inbounds %list_item, %list_item* %
    list_item28, i32 0, i32 0
97 store i8* %copied30, i8** %data_ptr_container31, align 8
98 %next32 = getelementptr inbounds %list_item, %list_item* %list_item28, i32
    0, i32 1
99 store %list_item* %list_item22, %list_item** %next32, align 8
100 %allocaall133 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
101 %list_item34 = bitcast i8* %allocaall133 to %list_item*
102 store %list_item zeroinitializer, %list_item* %list_item34, align 1
103 %copied36 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
104 store i8 84, i8* %copied36, align 1
105 %data_ptr_container37 = getelementptr inbounds %list_item, %list_item* %
    list_item34, i32 0, i32 0
106 store i8* %copied36, i8** %data_ptr_container37, align 8
107 %next38 = getelementptr inbounds %list_item, %list_item* %list_item34, i32
    0, i32 1
108 store %list_item* %list_item28, %list_item** %next38, align 8
109 %allocaall139 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
110 %list_item40 = bitcast i8* %allocaall139 to %list_item*
111 store %list_item zeroinitializer, %list_item* %list_item40, align 1
112 %copied42 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
113 store i8 83, i8* %copied42, align 1
114 %data_ptr_container43 = getelementptr inbounds %list_item, %list_item* %
    list_item40, i32 0, i32 0
115 store i8* %copied42, i8** %data_ptr_container43, align 8
116 %next44 = getelementptr inbounds %list_item, %list_item* %list_item40, i32
    0, i32 1
117 store %list_item* %list_item34, %list_item** %next44, align 8
118 %allocaall145 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
119 %list_item46 = bitcast i8* %allocaall145 to %list_item*
120 store %list_item zeroinitializer, %list_item* %list_item46, align 1
121 %copied48 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
122 store i8 82, i8* %copied48, align 1
123 %data_ptr_container49 = getelementptr inbounds %list_item, %list_item* %
    list_item46, i32 0, i32 0
124 store i8* %copied48, i8** %data_ptr_container49, align 8
125 %next50 = getelementptr inbounds %list_item, %list_item* %list_item46, i32
    0, i32 1
126 store %list_item* %list_item40, %list_item** %next50, align 8
127 %allocaall151 = tail call i8* @malloc(i32 ptrtoint (%list_item*

```

```

    getelementptr (%list_item, %list_item* null, i32 1) to i32))
128 %list_item52 = bitcast i8* %malloccall151 to %list_item*
129 store %list_item zeroinitializer, %list_item* %list_item52, align 1
130 %copied54 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
131 store i8 81, i8* %copied54, align 1
132 %data_ptr_container55 = getelementptr inbounds %list_item, %list_item* %
    list_item52, i32 0, i32 0
133 store i8* %copied54, i8** %data_ptr_container55, align 8
134 %next56 = getelementptr inbounds %list_item, %list_item* %list_item52, i32
    0, i32 1
135 store %list_item* %list_item46, %list_item** %next56, align 8
136 %malloccall157 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
137 %list_item58 = bitcast i8* %malloccall157 to %list_item*
138 store %list_item zeroinitializer, %list_item* %list_item58, align 1
139 %copied60 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
140 store i8 80, i8* %copied60, align 1
141 %data_ptr_container61 = getelementptr inbounds %list_item, %list_item* %
    list_item58, i32 0, i32 0
142 store i8* %copied60, i8** %data_ptr_container61, align 8
143 %next62 = getelementptr inbounds %list_item, %list_item* %list_item58, i32
    0, i32 1
144 store %list_item* %list_item52, %list_item** %next62, align 8
145 %malloccall163 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
146 %list_item64 = bitcast i8* %malloccall163 to %list_item*
147 store %list_item zeroinitializer, %list_item* %list_item64, align 1
148 %copied66 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
149 store i8 79, i8* %copied66, align 1
150 %data_ptr_container67 = getelementptr inbounds %list_item, %list_item* %
    list_item64, i32 0, i32 0
151 store i8* %copied66, i8** %data_ptr_container67, align 8
152 %next68 = getelementptr inbounds %list_item, %list_item* %list_item64, i32
    0, i32 1
153 store %list_item* %list_item58, %list_item** %next68, align 8
154 %malloccall169 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
155 %list_item70 = bitcast i8* %malloccall169 to %list_item*
156 store %list_item zeroinitializer, %list_item* %list_item70, align 1
157 %copied72 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
158 store i8 78, i8* %copied72, align 1
159 %data_ptr_container73 = getelementptr inbounds %list_item, %list_item* %
    list_item70, i32 0, i32 0
160 store i8* %copied72, i8** %data_ptr_container73, align 8
161 %next74 = getelementptr inbounds %list_item, %list_item* %list_item70, i32
    0, i32 1
162 store %list_item* %list_item64, %list_item** %next74, align 8
163 %malloccall175 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
164 %list_item76 = bitcast i8* %malloccall175 to %list_item*
165 store %list_item zeroinitializer, %list_item* %list_item76, align 1
166 %copied78 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
167 store i8 77, i8* %copied78, align 1
168 %data_ptr_container79 = getelementptr inbounds %list_item, %list_item* %

```

```

    list_item76, i32 0, i32 0
169 store i8* %copied78, i8** %data_ptr_container79, align 8
170 %next80 = getelementptr inbounds %list_item, %list_item* %list_item76, i32
    0, i32 1
171 store %list_item* %list_item70, %list_item** %next80, align 8
172 %malloccall181 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
173 %list_item82 = bitcast i8* %malloccall181 to %list_item*
174 store %list_item zeroinitializer, %list_item* %list_item82, align 1
175 %copied84 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
176 store i8 76, i8* %copied84, align 1
177 %data_ptr_container85 = getelementptr inbounds %list_item, %list_item* %
    list_item82, i32 0, i32 0
178 store i8* %copied84, i8** %data_ptr_container85, align 8
179 %next86 = getelementptr inbounds %list_item, %list_item* %list_item82, i32
    0, i32 1
180 store %list_item* %list_item76, %list_item** %next86, align 8
181 %malloccall187 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
182 %list_item88 = bitcast i8* %malloccall187 to %list_item*
183 store %list_item zeroinitializer, %list_item* %list_item88, align 1
184 %copied90 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
185 store i8 75, i8* %copied90, align 1
186 %data_ptr_container91 = getelementptr inbounds %list_item, %list_item* %
    list_item88, i32 0, i32 0
187 store i8* %copied90, i8** %data_ptr_container91, align 8
188 %next92 = getelementptr inbounds %list_item, %list_item* %list_item88, i32
    0, i32 1
189 store %list_item* %list_item82, %list_item** %next92, align 8
190 %malloccall193 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
191 %list_item94 = bitcast i8* %malloccall193 to %list_item*
192 store %list_item zeroinitializer, %list_item* %list_item94, align 1
193 %copied96 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
194 store i8 74, i8* %copied96, align 1
195 %data_ptr_container97 = getelementptr inbounds %list_item, %list_item* %
    list_item94, i32 0, i32 0
196 store i8* %copied96, i8** %data_ptr_container97, align 8
197 %next98 = getelementptr inbounds %list_item, %list_item* %list_item94, i32
    0, i32 1
198 store %list_item* %list_item88, %list_item** %next98, align 8
199 %malloccall199 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
200 %list_item100 = bitcast i8* %malloccall199 to %list_item*
201 store %list_item zeroinitializer, %list_item* %list_item100, align 1
202 %copied102 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
203 store i8 73, i8* %copied102, align 1
204 %data_ptr_container103 = getelementptr inbounds %list_item, %list_item* %
    list_item100, i32 0, i32 0
205 store i8* %copied102, i8** %data_ptr_container103, align 8
206 %next104 = getelementptr inbounds %list_item, %list_item* %list_item100,
    i32 0, i32 1
207 store %list_item* %list_item94, %list_item** %next104, align 8
208 %malloccall1105 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))

```

```

209 %list_item106 = bitcast i8* %allocacll105 to %list_item*
210 store %list_item zeroinitializer, %list_item* %list_item106, align 1
211 %copied108 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
212 store i8 72, i8* %copied108, align 1
213 %data_ptr_container109 = getelementptr inbounds %list_item, %list_item* %
    list_item106, i32 0, i32 0
214 store i8* %copied108, i8** %data_ptr_container109, align 8
215 %next110 = getelementptr inbounds %list_item, %list_item* %list_item106,
    i32 0, i32 1
216 store %list_item* %list_item100, %list_item** %next110, align 8
217 %allocacll111 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
218 %list_item112 = bitcast i8* %allocacll111 to %list_item*
219 store %list_item zeroinitializer, %list_item* %list_item112, align 1
220 %copied114 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
221 store i8 71, i8* %copied114, align 1
222 %data_ptr_container115 = getelementptr inbounds %list_item, %list_item* %
    list_item112, i32 0, i32 0
223 store i8* %copied114, i8** %data_ptr_container115, align 8
224 %next116 = getelementptr inbounds %list_item, %list_item* %list_item112,
    i32 0, i32 1
225 store %list_item* %list_item106, %list_item** %next116, align 8
226 %allocacll117 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
227 %list_item118 = bitcast i8* %allocacll117 to %list_item*
228 store %list_item zeroinitializer, %list_item* %list_item118, align 1
229 %copied120 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
230 store i8 70, i8* %copied120, align 1
231 %data_ptr_container121 = getelementptr inbounds %list_item, %list_item* %
    list_item118, i32 0, i32 0
232 store i8* %copied120, i8** %data_ptr_container121, align 8
233 %next122 = getelementptr inbounds %list_item, %list_item* %list_item118,
    i32 0, i32 1
234 store %list_item* %list_item112, %list_item** %next122, align 8
235 %allocacll123 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
236 %list_item124 = bitcast i8* %allocacll123 to %list_item*
237 store %list_item zeroinitializer, %list_item* %list_item124, align 1
238 %copied126 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
239 store i8 69, i8* %copied126, align 1
240 %data_ptr_container127 = getelementptr inbounds %list_item, %list_item* %
    list_item124, i32 0, i32 0
241 store i8* %copied126, i8** %data_ptr_container127, align 8
242 %next128 = getelementptr inbounds %list_item, %list_item* %list_item124,
    i32 0, i32 1
243 store %list_item* %list_item118, %list_item** %next128, align 8
244 %allocacll129 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
245 %list_item130 = bitcast i8* %allocacll129 to %list_item*
246 store %list_item zeroinitializer, %list_item* %list_item130, align 1
247 %copied132 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
248 store i8 68, i8* %copied132, align 1
249 %data_ptr_container133 = getelementptr inbounds %list_item, %list_item* %
    list_item130, i32 0, i32 0

```

```

250 store i8* %copied132, i8** %data_ptr_container133, align 8
251 %next134 = getelementptr inbounds %list_item, %list_item* %list_item130,
    i32 0, i32 1
252 store %list_item* %list_item124, %list_item** %next134, align 8
253 %malloccall1135 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
254 %list_item136 = bitcast i8* %malloccall1135 to %list_item*
255 store %list_item zeroinitializer, %list_item* %list_item136, align 1
256 %copied138 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
257 store i8 67, i8* %copied138, align 1
258 %data_ptr_container139 = getelementptr inbounds %list_item, %list_item* %
    list_item136, i32 0, i32 0
259 store i8* %copied138, i8** %data_ptr_container139, align 8
260 %next140 = getelementptr inbounds %list_item, %list_item* %list_item136,
    i32 0, i32 1
261 store %list_item* %list_item130, %list_item** %next140, align 8
262 %malloccall1141 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
263 %list_item142 = bitcast i8* %malloccall1141 to %list_item*
264 store %list_item zeroinitializer, %list_item* %list_item142, align 1
265 %copied144 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
266 store i8 66, i8* %copied144, align 1
267 %data_ptr_container145 = getelementptr inbounds %list_item, %list_item* %
    list_item142, i32 0, i32 0
268 store i8* %copied144, i8** %data_ptr_container145, align 8
269 %next146 = getelementptr inbounds %list_item, %list_item* %list_item142,
    i32 0, i32 1
270 store %list_item* %list_item136, %list_item** %next146, align 8
271 %malloccall1147 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
272 %list_item148 = bitcast i8* %malloccall1147 to %list_item*
273 store %list_item zeroinitializer, %list_item* %list_item148, align 1
274 %copied150 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
275 store i8 65, i8* %copied150, align 1
276 %data_ptr_container151 = getelementptr inbounds %list_item, %list_item* %
    list_item148, i32 0, i32 0
277 store i8* %copied150, i8** %data_ptr_container151, align 8
278 %next152 = getelementptr inbounds %list_item, %list_item* %list_item148,
    i32 0, i32 1
279 store %list_item* %list_item142, %list_item** %next152, align 8
280 %malloccall1153 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
281 %list_item154 = bitcast i8* %malloccall1153 to %list_item*
282 store %list_item zeroinitializer, %list_item* %list_item154, align 1
283 %copied156 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
284 store i8 122, i8* %copied156, align 1
285 %data_ptr_container157 = getelementptr inbounds %list_item, %list_item* %
    list_item154, i32 0, i32 0
286 store i8* %copied156, i8** %data_ptr_container157, align 8
287 %next158 = getelementptr inbounds %list_item, %list_item* %list_item154,
    i32 0, i32 1
288 store %list_item* %list_item148, %list_item** %next158, align 8
289 %malloccall1159 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
290 %list_item160 = bitcast i8* %malloccall1159 to %list_item*

```

```

291 store %list_item zeroinitializer, %list_item* %list_iteml60, align 1
292 %copiedl62 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
293 store i8 121, i8* %copiedl62, align 1
294 %data_ptr_containerl63 = getelementptr inbounds %list_item, %list_item* %
    list_iteml60, i32 0, i32 0
295 store i8* %copiedl62, i8** %data_ptr_containerl63, align 8
296 %nextl64 = getelementptr inbounds %list_item, %list_item* %list_iteml60,
    i32 0, i32 1
297 store %list_item* %list_iteml54, %list_item** %nextl64, align 8
298 %malloccall165 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
299 %list_iteml66 = bitcast i8* %malloccall165 to %list_item*
300 store %list_item zeroinitializer, %list_item* %list_iteml66, align 1
301 %copiedl68 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
302 store i8 120, i8* %copiedl68, align 1
303 %data_ptr_containerl69 = getelementptr inbounds %list_item, %list_item* %
    list_iteml66, i32 0, i32 0
304 store i8* %copiedl68, i8** %data_ptr_containerl69, align 8
305 %nextl70 = getelementptr inbounds %list_item, %list_item* %list_iteml66,
    i32 0, i32 1
306 store %list_item* %list_iteml60, %list_item** %nextl70, align 8
307 %malloccall171 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
308 %list_iteml72 = bitcast i8* %malloccall171 to %list_item*
309 store %list_item zeroinitializer, %list_item* %list_iteml72, align 1
310 %copiedl74 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
311 store i8 119, i8* %copiedl74, align 1
312 %data_ptr_containerl75 = getelementptr inbounds %list_item, %list_item* %
    list_iteml72, i32 0, i32 0
313 store i8* %copiedl74, i8** %data_ptr_containerl75, align 8
314 %nextl76 = getelementptr inbounds %list_item, %list_item* %list_iteml72,
    i32 0, i32 1
315 store %list_item* %list_iteml66, %list_item** %nextl76, align 8
316 %malloccall177 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
317 %list_iteml78 = bitcast i8* %malloccall177 to %list_item*
318 store %list_item zeroinitializer, %list_item* %list_iteml78, align 1
319 %copiedl80 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
320 store i8 118, i8* %copiedl80, align 1
321 %data_ptr_containerl81 = getelementptr inbounds %list_item, %list_item* %
    list_iteml78, i32 0, i32 0
322 store i8* %copiedl80, i8** %data_ptr_containerl81, align 8
323 %nextl82 = getelementptr inbounds %list_item, %list_item* %list_iteml78,
    i32 0, i32 1
324 store %list_item* %list_iteml72, %list_item** %nextl82, align 8
325 %malloccall183 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
326 %list_iteml84 = bitcast i8* %malloccall183 to %list_item*
327 store %list_item zeroinitializer, %list_item* %list_iteml84, align 1
328 %copiedl86 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
329 store i8 117, i8* %copiedl86, align 1
330 %data_ptr_containerl87 = getelementptr inbounds %list_item, %list_item* %
    list_iteml84, i32 0, i32 0
331 store i8* %copiedl86, i8** %data_ptr_containerl87, align 8

```

```

332 %next188 = getelementptr inbounds %list_item, %list_item* %list_item184,
    i32 0, i32 1
333 store %list_item* %list_item178, %list_item** %next188, align 8
334 %allocaall189 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
335 %list_item190 = bitcast i8* %allocaall189 to %list_item*
336 store %list_item zeroinitializer, %list_item* %list_item190, align 1
337 %copied192 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
338 store i8 116, i8* %copied192, align 1
339 %data_ptr_container193 = getelementptr inbounds %list_item, %list_item* %
    list_item190, i32 0, i32 0
340 store i8* %copied192, i8** %data_ptr_container193, align 8
341 %next194 = getelementptr inbounds %list_item, %list_item* %list_item190,
    i32 0, i32 1
342 store %list_item* %list_item184, %list_item** %next194, align 8
343 %allocaall195 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
344 %list_item196 = bitcast i8* %allocaall195 to %list_item*
345 store %list_item zeroinitializer, %list_item* %list_item196, align 1
346 %copied198 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
347 store i8 115, i8* %copied198, align 1
348 %data_ptr_container199 = getelementptr inbounds %list_item, %list_item* %
    list_item196, i32 0, i32 0
349 store i8* %copied198, i8** %data_ptr_container199, align 8
350 %next200 = getelementptr inbounds %list_item, %list_item* %list_item196,
    i32 0, i32 1
351 store %list_item* %list_item190, %list_item** %next200, align 8
352 %allocaall201 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
353 %list_item202 = bitcast i8* %allocaall201 to %list_item*
354 store %list_item zeroinitializer, %list_item* %list_item202, align 1
355 %copied204 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
356 store i8 114, i8* %copied204, align 1
357 %data_ptr_container205 = getelementptr inbounds %list_item, %list_item* %
    list_item202, i32 0, i32 0
358 store i8* %copied204, i8** %data_ptr_container205, align 8
359 %next206 = getelementptr inbounds %list_item, %list_item* %list_item202,
    i32 0, i32 1
360 store %list_item* %list_item196, %list_item** %next206, align 8
361 %allocaall207 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
362 %list_item208 = bitcast i8* %allocaall207 to %list_item*
363 store %list_item zeroinitializer, %list_item* %list_item208, align 1
364 %copied210 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
365 store i8 113, i8* %copied210, align 1
366 %data_ptr_container211 = getelementptr inbounds %list_item, %list_item* %
    list_item208, i32 0, i32 0
367 store i8* %copied210, i8** %data_ptr_container211, align 8
368 %next212 = getelementptr inbounds %list_item, %list_item* %list_item208,
    i32 0, i32 1
369 store %list_item* %list_item202, %list_item** %next212, align 8
370 %allocaall213 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
371 %list_item214 = bitcast i8* %allocaall213 to %list_item*
372 store %list_item zeroinitializer, %list_item* %list_item214, align 1

```



```

373 %copied216 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
374 store i8 112, i8* %copied216, align 1
375 %data_ptr_container217 = getelementptr inbounds %list_item, %list_item* %
    list_item214, i32 0, i32 0
376 store i8* %copied216, i8** %data_ptr_container217, align 8
377 %next218 = getelementptr inbounds %list_item, %list_item* %list_item214,
    i32 0, i32 1
378 store %list_item* %list_item208, %list_item** %next218, align 8
379 %malloccall1219 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
380 %list_item220 = bitcast i8* %malloccall1219 to %list_item*
381 store %list_item zeroinitializer, %list_item* %list_item220, align 1
382 %copied222 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
383 store i8 111, i8* %copied222, align 1
384 %data_ptr_container223 = getelementptr inbounds %list_item, %list_item* %
    list_item220, i32 0, i32 0
385 store i8* %copied222, i8** %data_ptr_container223, align 8
386 %next224 = getelementptr inbounds %list_item, %list_item* %list_item220,
    i32 0, i32 1
387 store %list_item* %list_item214, %list_item** %next224, align 8
388 %malloccall1225 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
389 %list_item226 = bitcast i8* %malloccall1225 to %list_item*
390 store %list_item zeroinitializer, %list_item* %list_item226, align 1
391 %copied228 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
392 store i8 110, i8* %copied228, align 1
393 %data_ptr_container229 = getelementptr inbounds %list_item, %list_item* %
    list_item226, i32 0, i32 0
394 store i8* %copied228, i8** %data_ptr_container229, align 8
395 %next230 = getelementptr inbounds %list_item, %list_item* %list_item226,
    i32 0, i32 1
396 store %list_item* %list_item220, %list_item** %next230, align 8
397 %malloccall1231 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
398 %list_item232 = bitcast i8* %malloccall1231 to %list_item*
399 store %list_item zeroinitializer, %list_item* %list_item232, align 1
400 %copied234 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
401 store i8 109, i8* %copied234, align 1
402 %data_ptr_container235 = getelementptr inbounds %list_item, %list_item* %
    list_item232, i32 0, i32 0
403 store i8* %copied234, i8** %data_ptr_container235, align 8
404 %next236 = getelementptr inbounds %list_item, %list_item* %list_item232,
    i32 0, i32 1
405 store %list_item* %list_item226, %list_item** %next236, align 8
406 %malloccall1237 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
407 %list_item238 = bitcast i8* %malloccall1237 to %list_item*
408 store %list_item zeroinitializer, %list_item* %list_item238, align 1
409 %copied240 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
410 store i8 108, i8* %copied240, align 1
411 %data_ptr_container241 = getelementptr inbounds %list_item, %list_item* %
    list_item238, i32 0, i32 0
412 store i8* %copied240, i8** %data_ptr_container241, align 8
413 %next242 = getelementptr inbounds %list_item, %list_item* %list_item238,

```



```

    i32 0, i32 1
414 store %list_item* %list_item232, %list_item** %next242, align 8
415 %mallocall243 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
416 %list_item244 = bitcast i8* %mallocall243 to %list_item*
417 store %list_item zeroinitializer, %list_item* %list_item244, align 1
418 %copied246 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
419 store i8 107, i8* %copied246, align 1
420 %data_ptr_container247 = getelementptr inbounds %list_item, %list_item* %
    list_item244, i32 0, i32 0
421 store i8* %copied246, i8** %data_ptr_container247, align 8
422 %next248 = getelementptr inbounds %list_item, %list_item* %list_item244,
    i32 0, i32 1
423 store %list_item* %list_item238, %list_item** %next248, align 8
424 %mallocall249 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
425 %list_item250 = bitcast i8* %mallocall249 to %list_item*
426 store %list_item zeroinitializer, %list_item* %list_item250, align 1
427 %copied252 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
428 store i8 106, i8* %copied252, align 1
429 %data_ptr_container253 = getelementptr inbounds %list_item, %list_item* %
    list_item250, i32 0, i32 0
430 store i8* %copied252, i8** %data_ptr_container253, align 8
431 %next254 = getelementptr inbounds %list_item, %list_item* %list_item250,
    i32 0, i32 1
432 store %list_item* %list_item244, %list_item** %next254, align 8
433 %mallocall255 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
434 %list_item256 = bitcast i8* %mallocall255 to %list_item*
435 store %list_item zeroinitializer, %list_item* %list_item256, align 1
436 %copied258 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
437 store i8 105, i8* %copied258, align 1
438 %data_ptr_container259 = getelementptr inbounds %list_item, %list_item* %
    list_item256, i32 0, i32 0
439 store i8* %copied258, i8** %data_ptr_container259, align 8
440 %next260 = getelementptr inbounds %list_item, %list_item* %list_item256,
    i32 0, i32 1
441 store %list_item* %list_item250, %list_item** %next260, align 8
442 %mallocall261 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
443 %list_item262 = bitcast i8* %mallocall261 to %list_item*
444 store %list_item zeroinitializer, %list_item* %list_item262, align 1
445 %copied264 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
446 store i8 104, i8* %copied264, align 1
447 %data_ptr_container265 = getelementptr inbounds %list_item, %list_item* %
    list_item262, i32 0, i32 0
448 store i8* %copied264, i8** %data_ptr_container265, align 8
449 %next266 = getelementptr inbounds %list_item, %list_item* %list_item262,
    i32 0, i32 1
450 store %list_item* %list_item256, %list_item** %next266, align 8
451 %mallocall267 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
452 %list_item268 = bitcast i8* %mallocall267 to %list_item*
453 store %list_item zeroinitializer, %list_item* %list_item268, align 1
454 %copied270 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8

```

```

    * null, i32 1) to i32))
455 store i8 103, i8* %copied270, align 1
456 %data_ptr_container271 = getelementptr inbounds %list_item, %list_item* %
    list_item268, i32 0, i32 0
457 store i8* %copied270, i8** %data_ptr_container271, align 8
458 %next272 = getelementptr inbounds %list_item, %list_item* %list_item268,
    i32 0, i32 1
459 store %list_item* %list_item262, %list_item** %next272, align 8
460 %mallocall1273 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
461 %list_item274 = bitcast i8* %mallocall1273 to %list_item*
462 store %list_item zeroinitializer, %list_item* %list_item274, align 1
463 %copied276 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
464 store i8 102, i8* %copied276, align 1
465 %data_ptr_container277 = getelementptr inbounds %list_item, %list_item* %
    list_item274, i32 0, i32 0
466 store i8* %copied276, i8** %data_ptr_container277, align 8
467 %next278 = getelementptr inbounds %list_item, %list_item* %list_item274,
    i32 0, i32 1
468 store %list_item* %list_item268, %list_item** %next278, align 8
469 %mallocall1279 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
470 %list_item280 = bitcast i8* %mallocall1279 to %list_item*
471 store %list_item zeroinitializer, %list_item* %list_item280, align 1
472 %copied282 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
473 store i8 101, i8* %copied282, align 1
474 %data_ptr_container283 = getelementptr inbounds %list_item, %list_item* %
    list_item280, i32 0, i32 0
475 store i8* %copied282, i8** %data_ptr_container283, align 8
476 %next284 = getelementptr inbounds %list_item, %list_item* %list_item280,
    i32 0, i32 1
477 store %list_item* %list_item274, %list_item** %next284, align 8
478 %mallocall1285 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
479 %list_item286 = bitcast i8* %mallocall1285 to %list_item*
480 store %list_item zeroinitializer, %list_item* %list_item286, align 1
481 %copied288 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
482 store i8 100, i8* %copied288, align 1
483 %data_ptr_container289 = getelementptr inbounds %list_item, %list_item* %
    list_item286, i32 0, i32 0
484 store i8* %copied288, i8** %data_ptr_container289, align 8
485 %next290 = getelementptr inbounds %list_item, %list_item* %list_item286,
    i32 0, i32 1
486 store %list_item* %list_item280, %list_item** %next290, align 8
487 %mallocall1291 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
488 %list_item292 = bitcast i8* %mallocall1291 to %list_item*
489 store %list_item zeroinitializer, %list_item* %list_item292, align 1
490 %copied294 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
491 store i8 99, i8* %copied294, align 1
492 %data_ptr_container295 = getelementptr inbounds %list_item, %list_item* %
    list_item292, i32 0, i32 0
493 store i8* %copied294, i8** %data_ptr_container295, align 8
494 %next296 = getelementptr inbounds %list_item, %list_item* %list_item292,
    i32 0, i32 1

```

```

495 store %list_item* %list_item286, %list_item** %next296, align 8
496 %malloccall1297 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
497 %list_item298 = bitcast i8* %malloccall1297 to %list_item*
498 store %list_item zeroinitializer, %list_item* %list_item298, align 1
499 %copied300 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
500 store i8 98, i8* %copied300, align 1
501 %data_ptr_container301 = getelementptr inbounds %list_item, %list_item* %
    list_item298, i32 0, i32 0
502 store i8* %copied300, i8** %data_ptr_container301, align 8
503 %next302 = getelementptr inbounds %list_item, %list_item* %list_item298,
    i32 0, i32 1
504 store %list_item* %list_item292, %list_item** %next302, align 8
505 %malloccall1303 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
506 %list_item304 = bitcast i8* %malloccall1303 to %list_item*
507 store %list_item zeroinitializer, %list_item* %list_item304, align 1
508 %copied306 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
509 store i8 97, i8* %copied306, align 1
510 %data_ptr_container307 = getelementptr inbounds %list_item, %list_item* %
    list_item304, i32 0, i32 0
511 store i8* %copied306, i8** %data_ptr_container307, align 8
512 %next308 = getelementptr inbounds %list_item, %list_item* %list_item304,
    i32 0, i32 1
513 store %list_item* %list_item298, %list_item** %next308, align 8
514 store %list_item* %list_item304, %list_item** %list, align 8
515 store %list_item** %list, %list_item*** @ASCII, align 8
516 %malloccall1309 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
517 %list310 = bitcast i8* %malloccall1309 to %list_item**
518 %malloccall1311 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
519 %list_item312 = bitcast i8* %malloccall1311 to %list_item*
520 store %list_item zeroinitializer, %list_item* %list_item312, align 1
521 %malloccall1313 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
522 %copied314 = bitcast i8* %malloccall1313 to i8**
523 store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.1, i32 0,
    i32 0), i8** %copied314, align 8
524 %cast_ptr = bitcast i8** %copied314 to i8*
525 %data_ptr_container315 = getelementptr inbounds %list_item, %list_item* %
    list_item312, i32 0, i32 0
526 store i8* %cast_ptr, i8** %data_ptr_container315, align 8
527 %next316 = getelementptr inbounds %list_item, %list_item* %list_item312,
    i32 0, i32 1
528 store %list_item* null, %list_item** %next316, align 8
529 store %list_item* %list_item312, %list_item** %list310, align 8
530 %_result = call i8* @join_string_string(%list_item** %list310, i8*
    getelementptr inbounds ([1 x i8], [1 x i8]* @string, i32 0, i32 0))
531 %malloccall1317 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
532 %list318 = bitcast i8* %malloccall1317 to %list_item**
533 %malloccall1319 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
534 %list_item320 = bitcast i8* %malloccall1319 to %list_item*
535 store %list_item zeroinitializer, %list_item* %list_item320, align 1
536 %malloccall1321 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (

```

```

    i32, i32* null, i32 1) to i32))
537 %copied322 = bitcast i8* %allocacll321 to i32*
538 store i32 1, i32* %copied322, align 4
539 %cast_ptr323 = bitcast i32* %copied322 to i8*
540 %data_ptr_container324 = getelementptr inbounds %list_item, %list_item* %
    list_item320, i32 0, i32 0
541 store i8* %cast_ptr323, i8** %data_ptr_container324, align 8
542 %next325 = getelementptr inbounds %list_item, %list_item* %list_item320,
    i32 0, i32 1
543 store %list_item* null, %list_item** %next325, align 8
544 store %list_item* %list_item320, %list_item** %list318, align 8
545 %allocacll326 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
546 %list327 = bitcast i8* %allocacll326 to %list_item**
547 %allocacll328 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
548 %list_item329 = bitcast i8* %allocacll328 to %list_item*
549 store %list_item zeroinitializer, %list_item* %list_item329, align 1
550 %allocacll330 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
551 %copied331 = bitcast i8* %allocacll330 to i32*
552 store i32 2, i32* %copied331, align 4
553 %cast_ptr332 = bitcast i32* %copied331 to i8*
554 %data_ptr_container333 = getelementptr inbounds %list_item, %list_item* %
    list_item329, i32 0, i32 0
555 store i8* %cast_ptr332, i8** %data_ptr_container333, align 8
556 %next334 = getelementptr inbounds %list_item, %list_item* %list_item329,
    i32 0, i32 1
557 store %list_item* null, %list_item** %next334, align 8
558 %allocacll335 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
559 %list_item336 = bitcast i8* %allocacll335 to %list_item*
560 store %list_item zeroinitializer, %list_item* %list_item336, align 1
561 %allocacll337 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
562 %copied338 = bitcast i8* %allocacll337 to i32*
563 store i32 1, i32* %copied338, align 4
564 %cast_ptr339 = bitcast i32* %copied338 to i8*
565 %data_ptr_container340 = getelementptr inbounds %list_item, %list_item* %
    list_item336, i32 0, i32 0
566 store i8* %cast_ptr339, i8** %data_ptr_container340, align 8
567 %next341 = getelementptr inbounds %list_item, %list_item* %list_item336,
    i32 0, i32 1
568 store %list_item* %list_item329, %list_item** %next341, align 8
569 store %list_item* %list_item336, %list_item** %list327, align 8
570 %_result342 = call i1 @contains_int_int(%list_item** %list327, %list_item
    ** %list318)
571 %allocacll343 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
572 %list344 = bitcast i8* %allocacll343 to %list_item**
573 %allocacll345 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
574 %list_item346 = bitcast i8* %allocacll345 to %list_item*
575 store %list_item zeroinitializer, %list_item* %list_item346, align 1
576 %allocacll347 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
577 %copied348 = bitcast i8* %allocacll347 to i8**
578 store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.2, i32 0,
    i32 0), i8** %copied348, align 8

```

```

579 %cast_ptr349 = bitcast i8** %copied348 to i8*
580 %data_ptr_container350 = getelementptr inbounds %list_item, %list_item* %
    list_item346, i32 0, i32 0
581 store i8* %cast_ptr349, i8** %data_ptr_container350, align 8
582 %next351 = getelementptr inbounds %list_item, %list_item* %list_item346,
    i32 0, i32 1
583 store %list_item* null, %list_item** %next351, align 8
584 store %list_item* %list_item346, %list_item** %list344, align 8
585 %alloca1352 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
586 %list353 = bitcast i8* %alloca1352 to %list_item**
587 %alloca1354 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
588 %list_item355 = bitcast i8* %alloca1354 to %list_item*
589 store %list_item zeroinitializer, %list_item* %list_item355, align 1
590 %alloca1356 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
591 %copied357 = bitcast i8* %alloca1356 to i8**
592 store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.3, i32 0,
    i32 0), i8** %copied357, align 8
593 %cast_ptr358 = bitcast i8** %copied357 to i8*
594 %data_ptr_container359 = getelementptr inbounds %list_item, %list_item* %
    list_item355, i32 0, i32 0
595 store i8* %cast_ptr358, i8** %data_ptr_container359, align 8
596 %next360 = getelementptr inbounds %list_item, %list_item* %list_item355,
    i32 0, i32 1
597 store %list_item* null, %list_item** %next360, align 8
598 %alloca1361 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
599 %list_item362 = bitcast i8* %alloca1361 to %list_item*
600 store %list_item zeroinitializer, %list_item* %list_item362, align 1
601 %alloca1363 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
602 %copied364 = bitcast i8* %alloca1363 to i8**
603 store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.4, i32 0,
    i32 0), i8** %copied364, align 8
604 %cast_ptr365 = bitcast i8** %copied364 to i8*
605 %data_ptr_container366 = getelementptr inbounds %list_item, %list_item* %
    list_item362, i32 0, i32 0
606 store i8* %cast_ptr365, i8** %data_ptr_container366, align 8
607 %next367 = getelementptr inbounds %list_item, %list_item* %list_item362,
    i32 0, i32 1
608 store %list_item* %list_item355, %list_item** %next367, align 8
609 store %list_item* %list_item362, %list_item** %list353, align 8
610 %_result368 = call i1 @contains_string_string(%list_item** %list353, %
    list_item** %list344)
611 %alloca1369 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
612 %list370 = bitcast i8* %alloca1369 to %list_item**
613 %alloca1371 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
614 %list_item372 = bitcast i8* %alloca1371 to %list_item*
615 store %list_item zeroinitializer, %list_item* %list_item372, align 1
616 %copied374 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
617 store i8 104, i8* %copied374, align 1
618 %data_ptr_container375 = getelementptr inbounds %list_item, %list_item* %
    list_item372, i32 0, i32 0
619 store i8* %copied374, i8** %data_ptr_container375, align 8

```

```

620 %next376 = getelementptr inbounds %list_item, %list_item* %list_item372,
    i32 0, i32 1
621 store %list_item* null, %list_item** %next376, align 8
622 store %list_item* %list_item372, %list_item** %list370, align 8
623 %malloccall377 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
624 %list378 = bitcast i8* %malloccall377 to %list_item**
625 %malloccall379 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
626 %list_item380 = bitcast i8* %malloccall379 to %list_item*
627 store %list_item zeroinitializer, %list_item* %list_item380, align 1
628 %copied382 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
629 store i8 101, i8* %copied382, align 1
630 %data_ptr_container383 = getelementptr inbounds %list_item, %list_item* %
    list_item380, i32 0, i32 0
631 store i8* %copied382, i8** %data_ptr_container383, align 8
632 %next384 = getelementptr inbounds %list_item, %list_item* %list_item380,
    i32 0, i32 1
633 store %list_item* null, %list_item** %next384, align 8
634 %malloccall385 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
635 %list_item386 = bitcast i8* %malloccall385 to %list_item*
636 store %list_item zeroinitializer, %list_item* %list_item386, align 1
637 %copied388 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
638 store i8 104, i8* %copied388, align 1
639 %data_ptr_container389 = getelementptr inbounds %list_item, %list_item* %
    list_item386, i32 0, i32 0
640 store i8* %copied388, i8** %data_ptr_container389, align 8
641 %next390 = getelementptr inbounds %list_item, %list_item* %list_item386,
    i32 0, i32 1
642 store %list_item* %list_item380, %list_item** %next390, align 8
643 store %list_item* %list_item386, %list_item** %list378, align 8
644 %_result391 = call i1 @contains_char_char(%list_item** %list378, %
    list_item** %list370)
645 %malloccall392 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
646 %list393 = bitcast i8* %malloccall392 to %list_item**
647 %malloccall394 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
648 %list_item395 = bitcast i8* %malloccall394 to %list_item*
649 store %list_item zeroinitializer, %list_item* %list_item395, align 1
650 %malloccall396 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
651 %copied397 = bitcast i8* %malloccall396 to i1*
652 store i1 true, i1* %copied397, align 1
653 %cast_ptr398 = bitcast i1* %copied397 to i8*
654 %data_ptr_container399 = getelementptr inbounds %list_item, %list_item* %
    list_item395, i32 0, i32 0
655 store i8* %cast_ptr398, i8** %data_ptr_container399, align 8
656 %next400 = getelementptr inbounds %list_item, %list_item* %list_item395,
    i32 0, i32 1
657 store %list_item* null, %list_item** %next400, align 8
658 store %list_item* %list_item395, %list_item** %list393, align 8
659 %malloccall401 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
660 %list402 = bitcast i8* %malloccall401 to %list_item**
661 %malloccall403 = tail call i8* @malloc(i32 ptrtoint (%list_item*

```

```

    getelementptr (%list_item, %list_item* null, i32 1) to i32))
662 %list_item404 = bitcast i8* %alloca1403 to %list_item*
663 store %list_item zeroinitializer, %list_item* %list_item404, align 1
664 %alloca1405 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
665 %copied406 = bitcast i8* %alloca1405 to i1*
666 store i1 false, i1* %copied406, align 1
667 %cast_ptr407 = bitcast i1* %copied406 to i8*
668 %data_ptr_container408 = getelementptr inbounds %list_item, %list_item* %
    list_item404, i32 0, i32 0
669 store i8* %cast_ptr407, i8** %data_ptr_container408, align 8
670 %next409 = getelementptr inbounds %list_item, %list_item* %list_item404,
    i32 0, i32 1
671 store %list_item* null, %list_item** %next409, align 8
672 %alloca1410 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
673 %list_item411 = bitcast i8* %alloca1410 to %list_item*
674 store %list_item zeroinitializer, %list_item* %list_item411, align 1
675 %alloca1412 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
676 %copied413 = bitcast i8* %alloca1412 to i1*
677 store i1 true, i1* %copied413, align 1
678 %cast_ptr414 = bitcast i1* %copied413 to i8*
679 %data_ptr_container415 = getelementptr inbounds %list_item, %list_item* %
    list_item411, i32 0, i32 0
680 store i8* %cast_ptr414, i8** %data_ptr_container415, align 8
681 %next416 = getelementptr inbounds %list_item, %list_item* %list_item411,
    i32 0, i32 1
682 store %list_item* %list_item404, %list_item** %next416, align 8
683 store %list_item* %list_item411, %list_item** %list402, align 8
684 %result417 = call i1 @contains_bool_bool(%list_item** %list402, %
    list_item** %list393)
685 %alloca1418 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
686 %list419 = bitcast i8* %alloca1418 to %list_item**
687 %alloca1420 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
688 %list_item421 = bitcast i8* %alloca1420 to %list_item*
689 store %list_item zeroinitializer, %list_item* %list_item421, align 1
690 %alloca1422 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
    (double, double* null, i32 1) to i32))
691 %copied423 = bitcast i8* %alloca1422 to double*
692 store double 1.100000e+00, double* %copied423, align 8
693 %cast_ptr424 = bitcast double* %copied423 to i8*
694 %data_ptr_container425 = getelementptr inbounds %list_item, %list_item* %
    list_item421, i32 0, i32 0
695 store i8* %cast_ptr424, i8** %data_ptr_container425, align 8
696 %next426 = getelementptr inbounds %list_item, %list_item* %list_item421,
    i32 0, i32 1
697 store %list_item* null, %list_item** %next426, align 8
698 store %list_item* %list_item421, %list_item** %list419, align 8
699 %alloca1427 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
700 %list428 = bitcast i8* %alloca1427 to %list_item**
701 %alloca1429 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
702 %list_item430 = bitcast i8* %alloca1429 to %list_item*
703 store %list_item zeroinitializer, %list_item* %list_item430, align 1
704 %alloca1431 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr

```



```

    (double, double* null, i32 1) to i32))
705 %copied432 = bitcast i8* %allocacll431 to double*
706 store double 1.200000e+00, double* %copied432, align 8
707 %cast_ptr433 = bitcast double* %copied432 to i8*
708 %data_ptr_container434 = getelementptr inbounds %list_item, %list_item* %
    list_item430, i32 0, i32 0
709 store i8* %cast_ptr433, i8** %data_ptr_container434, align 8
710 %next435 = getelementptr inbounds %list_item, %list_item* %list_item430,
    i32 0, i32 1
711 store %list_item* null, %list_item** %next435, align 8
712 %allocacll436 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
713 %list_item437 = bitcast i8* %allocacll436 to %list_item*
714 store %list_item zeroinitializer, %list_item* %list_item437, align 1
715 %allocacll438 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
    (double, double* null, i32 1) to i32))
716 %copied439 = bitcast i8* %allocacll438 to double*
717 store double 1.100000e+00, double* %copied439, align 8
718 %cast_ptr440 = bitcast double* %copied439 to i8*
719 %data_ptr_container441 = getelementptr inbounds %list_item, %list_item* %
    list_item437, i32 0, i32 0
720 store i8* %cast_ptr440, i8** %data_ptr_container441, align 8
721 %next442 = getelementptr inbounds %list_item, %list_item* %list_item437,
    i32 0, i32 1
722 store %list_item* %list_item430, %list_item** %next442, align 8
723 store %list_item* %list_item437, %list_item** %list428, align 8
724 %_result443 = call i1 @contains_float_float(%list_item** %list428, %
    list_item** %list419)
725 %allocacll444 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
726 %list445 = bitcast i8* %allocacll444 to %list_item**
727 %allocacll446 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
728 %list_item447 = bitcast i8* %allocacll446 to %list_item*
729 store %list_item zeroinitializer, %list_item* %list_item447, align 1
730 %allocacll448 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
731 %copied449 = bitcast i8* %allocacll448 to i32*
732 store i32 1, i32* %copied449, align 4
733 %cast_ptr450 = bitcast i32* %copied449 to i8*
734 %data_ptr_container451 = getelementptr inbounds %list_item, %list_item* %
    list_item447, i32 0, i32 0
735 store i8* %cast_ptr450, i8** %data_ptr_container451, align 8
736 %next452 = getelementptr inbounds %list_item, %list_item* %list_item447,
    i32 0, i32 1
737 store %list_item* null, %list_item** %next452, align 8
738 store %list_item* %list_item447, %list_item** %list445, align 8
739 %_result453 = call %list_item** @remove_int_int_int_bool(%list_item** %
    list445, i32 1, i1 true)
740 %allocacll454 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
741 %list455 = bitcast i8* %allocacll454 to %list_item**
742 %allocacll456 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
743 %list_item457 = bitcast i8* %allocacll456 to %list_item*
744 store %list_item zeroinitializer, %list_item* %list_item457, align 1
745 %allocacll458 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
    (double, double* null, i32 1) to i32))
746 %copied459 = bitcast i8* %allocacll458 to double*

```



```

747 store double 1.500000e+00, double* %copied459, align 8
748 %cast_ptr460 = bitcast double* %copied459 to i8*
749 %data_ptr_container461 = getelementptr inbounds %list_item, %list_item* %
    list_item457, i32 0, i32 0
750 store i8* %cast_ptr460, i8** %data_ptr_container461, align 8
751 %next462 = getelementptr inbounds %list_item, %list_item* %list_item457,
    i32 0, i32 1
752 store %list_item* null, %list_item** %next462, align 8
753 store %list_item* %list_item457, %list_item** %list455, align 8
754 %_result463 = call %list_item** @remove_float_float_float_bool(%list_item
    ** %list455, double 1.500000e+00, i1 false)
755 %malloccall464 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
756 %list465 = bitcast i8* %malloccall464 to %list_item**
757 %malloccall466 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
758 %list_item467 = bitcast i8* %malloccall466 to %list_item*
759 store %list_item zeroinitializer, %list_item* %list_item467, align 1
760 %malloccall468 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
761 %copied469 = bitcast i8* %malloccall468 to i1*
762 store i1 true, i1* %copied469, align 1
763 %cast_ptr470 = bitcast i1* %copied469 to i8*
764 %data_ptr_container471 = getelementptr inbounds %list_item, %list_item* %
    list_item467, i32 0, i32 0
765 store i8* %cast_ptr470, i8** %data_ptr_container471, align 8
766 %next472 = getelementptr inbounds %list_item, %list_item* %list_item467,
    i32 0, i32 1
767 store %list_item* null, %list_item** %next472, align 8
768 store %list_item* %list_item467, %list_item** %list465, align 8
769 %_result473 = call %list_item** @remove_bool_bool_bool_bool(%list_item** %
    list465, i1 true, i1 true)
770 %malloccall474 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
771 %list475 = bitcast i8* %malloccall474 to %list_item**
772 %malloccall476 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
773 %list_item477 = bitcast i8* %malloccall476 to %list_item*
774 store %list_item zeroinitializer, %list_item* %list_item477, align 1
775 %copied479 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
776 store i8 97, i8* %copied479, align 1
777 %data_ptr_container480 = getelementptr inbounds %list_item, %list_item* %
    list_item477, i32 0, i32 0
778 store i8* %copied479, i8** %data_ptr_container480, align 8
779 %next481 = getelementptr inbounds %list_item, %list_item* %list_item477,
    i32 0, i32 1
780 store %list_item* null, %list_item** %next481, align 8
781 store %list_item* %list_item477, %list_item** %list475, align 8
782 %_result482 = call %list_item** @remove_char_char_char_bool(%list_item** %
    list475, i8 97, i1 true)
783 %malloccall483 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
784 %list484 = bitcast i8* %malloccall483 to %list_item**
785 %malloccall485 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
786 %list_item486 = bitcast i8* %malloccall485 to %list_item*
787 store %list_item zeroinitializer, %list_item* %list_item486, align 1
788 %malloccall487 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (

```

```

    i1*, i1** null, i32 1) to i32))
789 %copied488 = bitcast i8* %mallocall487 to i8**
790 store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.6, i32 0,
    i32 0), i8** %copied488, align 8
791 %cast_ptr489 = bitcast i8** %copied488 to i8*
792 %data_ptr_container490 = getelementptr inbounds %list_item, %list_item* %
    list_item486, i32 0, i32 0
793 store i8* %cast_ptr489, i8** %data_ptr_container490, align 8
794 %next491 = getelementptr inbounds %list_item, %list_item* %list_item486,
    i32 0, i32 1
795 store %list_item* null, %list_item** %next491, align 8
796 store %list_item* %list_item486, %list_item** %list484, align 8
797 %_result492 = call %list_item** @remove_string_string_string_bool(%
    list_item** %list484, i8* getelementptr inbounds ([5 x i8], [5 x i8]*
    @string.5, i32 0, i32 0), i1 true)
798 %mallocall493 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
799 %list494 = bitcast i8* %mallocall493 to %list_item**
800 %mallocall495 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
801 %list_item496 = bitcast i8* %mallocall495 to %list_item*
802 store %list_item zeroinitializer, %list_item* %list_item496, align 1
803 %mallocall497 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
804 %copied498 = bitcast i8* %mallocall497 to i32*
805 store i32 3, i32* %copied498, align 4
806 %cast_ptr499 = bitcast i32* %copied498 to i8*
807 %data_ptr_container500 = getelementptr inbounds %list_item, %list_item* %
    list_item496, i32 0, i32 0
808 store i8* %cast_ptr499, i8** %data_ptr_container500, align 8
809 %next501 = getelementptr inbounds %list_item, %list_item* %list_item496,
    i32 0, i32 1
810 store %list_item* null, %list_item** %next501, align 8
811 %mallocall502 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
812 %list_item503 = bitcast i8* %mallocall502 to %list_item*
813 store %list_item zeroinitializer, %list_item* %list_item503, align 1
814 %mallocall504 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
815 %copied505 = bitcast i8* %mallocall504 to i32*
816 store i32 9, i32* %copied505, align 4
817 %cast_ptr506 = bitcast i32* %copied505 to i8*
818 %data_ptr_container507 = getelementptr inbounds %list_item, %list_item* %
    list_item503, i32 0, i32 0
819 store i8* %cast_ptr506, i8** %data_ptr_container507, align 8
820 %next508 = getelementptr inbounds %list_item, %list_item* %list_item503,
    i32 0, i32 1
821 store %list_item* %list_item496, %list_item** %next508, align 8
822 %mallocall509 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
823 %list_item510 = bitcast i8* %mallocall509 to %list_item*
824 store %list_item zeroinitializer, %list_item* %list_item510, align 1
825 %mallocall511 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
826 %copied512 = bitcast i8* %mallocall511 to i32*
827 store i32 6, i32* %copied512, align 4
828 %cast_ptr513 = bitcast i32* %copied512 to i8*
829 %data_ptr_container514 = getelementptr inbounds %list_item, %list_item* %
    list_item510, i32 0, i32 0

```

```

830 store i8* %cast_ptr513, i8** %data_ptr_container514, align 8
831 %next515 = getelementptr inbounds %list_item, %list_item* %list_item510,
      i32 0, i32 1
832 store %list_item* %list_item503, %list_item** %next515, align 8
833 %alloca1516 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
834 %list_item517 = bitcast i8* %alloca1516 to %list_item*
835 store %list_item zeroinitializer, %list_item* %list_item517, align 1
836 %alloca1518 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
      i32, i32* null, i32 1) to i32))
837 %copied519 = bitcast i8* %alloca1518 to i32*
838 store i32 2, i32* %copied519, align 4
839 %cast_ptr520 = bitcast i32* %copied519 to i8*
840 %data_ptr_container521 = getelementptr inbounds %list_item, %list_item* %
      list_item517, i32 0, i32 0
841 store i8* %cast_ptr520, i8** %data_ptr_container521, align 8
842 %next522 = getelementptr inbounds %list_item, %list_item* %list_item517,
      i32 0, i32 1
843 store %list_item* %list_item510, %list_item** %next522, align 8
844 %alloca1523 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
845 %list_item524 = bitcast i8* %alloca1523 to %list_item*
846 store %list_item zeroinitializer, %list_item* %list_item524, align 1
847 %alloca1525 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
      i32, i32* null, i32 1) to i32))
848 %copied526 = bitcast i8* %alloca1525 to i32*
849 store i32 5, i32* %copied526, align 4
850 %cast_ptr527 = bitcast i32* %copied526 to i8*
851 %data_ptr_container528 = getelementptr inbounds %list_item, %list_item* %
      list_item524, i32 0, i32 0
852 store i8* %cast_ptr527, i8** %data_ptr_container528, align 8
853 %next529 = getelementptr inbounds %list_item, %list_item* %list_item524,
      i32 0, i32 1
854 store %list_item* %list_item517, %list_item** %next529, align 8
855 store %list_item* %list_item524, %list_item** %list494, align 8
856 %_result530 = call i32 @max_profit_int(%list_item** %list494)
857 %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8
      ], [4 x i8]* @string.7, i32 0, i32 0), i32 %_result530)
858 ret i32 0
859 }
860
861 define i32 @max_profit_int(%list_item** %stock_prices) {
862 entry:
863   %potential_profit = alloca i32, align 4
864   %next_price = alloca i32, align 4
865   %current_price = alloca i32, align 4
866   %i = alloca i32, align 4
867   %for_index = alloca i32, align 4
868   %max_profit = alloca i32, align 4
869   %min_price = alloca i32, align 4
870   %stock_prices1 = alloca %list_item**, align 8
871   store %list_item** %stock_prices, %list_item*** %stock_prices1, align 8
872   %stock_prices2 = load %list_item**, %list_item*** %stock_prices1, align 8
873   %i1ist = load %list_item*, %list_item** %stock_prices2, align 8
874   %_result = call %list_item* @list_access(%list_item* %i1ist, i32 0)
875   %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
      i32 0, i32 0
876   %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
877   %cast_data_ptr = bitcast i8* %data_ptr to i32*

```

```

878 %data = load i32, i32* %cast_data_ptr, align 4
879 store i32 %data, i32* %min_price, align 4
880 store i32 0, i32* %max_profit, align 4
881 store i32 0, i32* %for_index, align 4
882 store i32 0, i32* %i, align 4
883 br label %while
884
885 while:                                     ; preds = %merge34, %entry
886 %for_index57 = load i32, i32* %for_index, align 4
887 %stock_prices58 = load %list_item**, %list_item** %stock_prices1, align 8
888 %ilist59 = load %list_item**, %list_item** %stock_prices58, align 8
889 %length60 = call i32 @list_length(%list_item* %ilist59, i32 0)
890 %tmp61 = sub i32 %length60, 1
891 %mallocall62 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    *, i1** null, i32 1) to i32))
892 %head_ptr_ptr63 = bitcast i8* %mallocall62 to %list_item**
893 store %list_item* null, %list_item** %head_ptr_ptr63, align 8
894 %range_list64 = call %list_item** @range_function(i32 1, i32 %tmp61, %
    list_item** %head_ptr_ptr63, i32 0)
895 %ilist65 = load %list_item**, %list_item** %range_list64, align 8
896 %length66 = call i32 @list_length(%list_item* %ilist65, i32 0)
897 %tmp67 = icmp slt i32 %for_index57, %length66
898 br i1 %tmp67, label %while_body, label %merge
899
900 merge:                                     ; preds = %while
901 %max_profit68 = load i32, i32* %max_profit, align 4
902 ret i32 %max_profit68
903
904 while_body:                               ; preds = %while
905 %stock_prices3 = load %list_item**, %list_item** %stock_prices1, align 8
906 %ilist4 = load %list_item**, %list_item** %stock_prices3, align 8
907 %length = call i32 @list_length(%list_item* %ilist4, i32 0)
908 %tmp = sub i32 %length, 1
909 %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
910 %head_ptr_ptr = bitcast i8* %mallocall to %list_item**
911 store %list_item* null, %list_item** %head_ptr_ptr, align 8
912 %range_list = call %list_item** @range_function(i32 1, i32 %tmp, %
    list_item** %head_ptr_ptr, i32 0)
913 %ilist5 = load %list_item**, %list_item** %range_list, align 8
914 %for_index6 = load i32, i32* %for_index, align 4
915 %_result7 = call %list_item* @list_access(%list_item* %ilist5, i32 %
    for_index6)
916 %data_ptr_ptr8 = getelementptr inbounds %list_item, %list_item* %_result7,
    i32 0, i32 0
917 %data_ptr9 = load i8*, i8** %data_ptr_ptr8, align 8
918 %cast_data_ptr10 = bitcast i8* %data_ptr9 to i32*
919 %data11 = load i32, i32* %cast_data_ptr10, align 4
920 store i32 %data11, i32* %i, align 4
921 %for_index12 = load i32, i32* %for_index, align 4
922 %tmp13 = add i32 %for_index12, 1
923 store i32 %tmp13, i32* %for_index, align 4
924 %stock_prices14 = load %list_item**, %list_item** %stock_prices1, align 8
925 %ilist15 = load %list_item**, %list_item** %stock_prices14, align 8
926 %i16 = load i32, i32* %i, align 4
927 %_result17 = call %list_item* @list_access(%list_item* %ilist15, i32 %i16)
928 %data_ptr_ptr18 = getelementptr inbounds %list_item, %list_item* %
    _result17, i32 0, i32 0
929 %data_ptr19 = load i8*, i8** %data_ptr_ptr18, align 8

```

```

930 %cast_data_ptr20 = bitcast i8* %data_ptr19 to i32*
931 %data21 = load i32, i32* %cast_data_ptr20, align 4
932 store i32 %data21, i32* %current_price, align 4
933 %stock_prices22 = load %list_item**, %list_item** %stock_prices1, align 8
934 %ilist23 = load %list_item*, %list_item** %stock_prices22, align 8
935 %i24 = load i32, i32* %i, align 4
936 %tmp25 = add i32 %i24, 1
937 %_result26 = call %list_item* @list_access(%list_item* %ilist23, i32 %
    tmp25)
938 %data_ptr_ptr27 = getelementptr inbounds %list_item, %list_item* %
    _result26, i32 0, i32 0
939 %data_ptr28 = load i8*, i8** %data_ptr_ptr27, align 8
940 %cast_data_ptr29 = bitcast i8* %data_ptr28 to i32*
941 %data30 = load i32, i32* %cast_data_ptr29, align 4
942 store i32 %data30, i32* %next_price, align 4
943 %current_price31 = load i32, i32* %current_price, align 4
944 %next_price32 = load i32, i32* %next_price, align 4
945 %tmp33 = icmp sgt i32 %current_price31, %next_price32
946 br i1 %tmp33, label %then, label %else
947
948 merge34:                                ; preds = %merge53, %then
949     br label %while
950
951 then:                                    ; preds = %while_body
952     %current_price35 = load i32, i32* %current_price, align 4
953     %min_price36 = load i32, i32* %min_price, align 4
954     %tmp37 = sub i32 %current_price35, %min_price36
955     store i32 %tmp37, i32* %potential_profit, align 4
956     %max_profit38 = load i32, i32* %max_profit, align 4
957     %potential_profit39 = load i32, i32* %potential_profit, align 4
958     %tmp40 = add i32 %max_profit38, %potential_profit39
959     store i32 %tmp40, i32* %max_profit, align 4
960     %stock_prices41 = load %list_item**, %list_item** %stock_prices1, align 8
961     %ilist42 = load %list_item*, %list_item** %stock_prices41, align 8
962     %i43 = load i32, i32* %i, align 4
963     %tmp44 = add i32 %i43, 1
964     %_result45 = call %list_item* @list_access(%list_item* %ilist42, i32 %
        tmp44)
965     %data_ptr_ptr46 = getelementptr inbounds %list_item, %list_item* %
        _result45, i32 0, i32 0
966     %data_ptr47 = load i8*, i8** %data_ptr_ptr46, align 8
967     %cast_data_ptr48 = bitcast i8* %data_ptr47 to i32*
968     %data49 = load i32, i32* %cast_data_ptr48, align 4
969     store i32 %data49, i32* %min_price, align 4
970     br label %merge34
971
972 else:                                    ; preds = %while_body
973     %min_price50 = load i32, i32* %min_price, align 4
974     %current_price51 = load i32, i32* %current_price, align 4
975     %tmp52 = icmp sgt i32 %min_price50, %current_price51
976     br i1 %tmp52, label %then54, label %else56
977
978 merge53:                                ; preds = %else56, %then54
979     br label %merge34
980
981 then54:                                  ; preds = %else
982     %current_price55 = load i32, i32* %current_price, align 4
983     store i32 %current_price55, i32* %min_price, align 4
984     br label %merge53

```

```

985
986 else56:                                ; preds = %else
987     br label %merge53
988 }
989
990 define %list_item** @remove_string_string_string_bool(%list_item** %l, i8* %
    elem, i1 %all) {
991 entry:
992     %index = alloca i32, align 4
993     %for_index = alloca i32, align 4
994     %remove_index = alloca i32, align 4
995     %len = alloca i32, align 4
996     %i = alloca i32, align 4
997     %retlist = alloca %list_item**, align 8
998     %l1 = alloca %list_item**, align 8
999     store %list_item** %l, %list_item*** %l1, align 8
1000     %elem2 = alloca i8*, align 8
1001     store i8* %elem, i8** %elem2, align 8
1002     %all3 = alloca i1, align 1
1003     store i1 %all, i1* %all3, align 1
1004     %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))
1005     %list = bitcast i8* %malloccall to %list_item**
1006     store %list_item* null, %list_item** %list, align 8
1007     store %list_item** %list, %list_item*** %retlist, align 8
1008     store i32 0, i32* %i, align 4
1009     %l4 = load %list_item**, %list_item*** %l1, align 8
1010     %ilist = load %list_item*, %list_item** %l4, align 8
1011     %length = call i32 @list_length(%list_item* %ilist, i32 0)
1012     store i32 %length, i32* %len, align 4
1013     store i32 0, i32* %remove_index, align 4
1014     %all5 = load i1, i1* %all3, align 1
1015     br i1 %all5, label %then, label %else30
1016
1017 merge:                                ; preds = %merge68, %
    merge6
1018     %retlist109 = load %list_item**, %list_item*** %retlist, align 8
1019     ret %list_item** %retlist109
1020
1021 then:                                ; preds = %entry
1022     br label %while
1023
1024 while:                                ; preds = %merge12, %
    then13, %then
1025     %i27 = load i32, i32* %i, align 4
1026     %len28 = load i32, i32* %len, align 4
1027     %tmp29 = icmp slt i32 %i27, %len28
1028     br i1 %tmp29, label %while_body, label %merge6
1029
1030 merge6:                                ; preds = %while
1031     br label %merge
1032
1033 while_body:                            ; preds = %while
1034     %elem7 = load i8*, i8** %elem2, align 8
1035     %l8 = load %list_item**, %list_item*** %l1, align 8
1036     %ilist9 = load %list_item*, %list_item** %l8, align 8
1037     %i10 = load i32, i32* %i, align 4
1038     %_result = call %list_item* @list_access(%list_item* %ilist9, i32 %i10)
1039     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,

```

```

1040     i32 0, i32 0
1041     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1042     %cast_data_ptr = bitcast i8* %data_ptr to i8**
1043     %data = load i8*, i8** %cast_data_ptr, align 8
1044     %_result11 = call i1 @strcmp(i8* %data, i8* %elem7)
1045     br i1 %_result11, label %then13, label %else
1046 merge12:                                     ; preds = %else
1047     %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1048     %list_ptr = load %list_item**, %list_item** %retlist15, align 8
1049     %l16 = load %list_item**, %list_item*** %l1, align 8
1050     %ilist17 = load %list_item**, %list_item** %l16, align 8
1051     %i18 = load i32, i32* %i, align 4
1052     %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
1053     %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
        _result19, i32 0, i32 0
1054     %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
1055     %cast_data_ptr22 = bitcast i8* %data_ptr21 to i8**
1056     %data23 = load i8*, i8** %cast_data_ptr22, align 8
1057     %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1058     %list_ptr_ptr = call %list_item** @insert_string(%list_item** %retlist15,
        i8* %data23, i32 %length24)
1059     store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1060     %i25 = load i32, i32* %i, align 4
1061     %tmp26 = add i32 %i25, 1
1062     store i32 %tmp26, i32* %i, align 4
1063     br label %while
1064
1065 then13:                                     ; preds = %while_body
1066     %i14 = load i32, i32* %i, align 4
1067     %tmp = add i32 %i14, 1
1068     store i32 %tmp, i32* %i, align 4
1069     br label %while
1070
1071 else:                                       ; preds = %while_body
1072     br label %merge12
1073
1074 else30:                                   ; preds = %entry
1075     br label %while31
1076
1077 while31:                                   ; preds = %merge44, %
        else30
1078     %i62 = load i32, i32* %i, align 4
1079     %len63 = load i32, i32* %len, align 4
1080     %tmp64 = icmp slt i32 %i62, %len63
1081     br i1 %tmp64, label %while_body33, label %merge32
1082
1083 merge32:                                   ; preds = %while31, %
        then45
1084     %i65 = load i32, i32* %i, align 4
1085     %len66 = load i32, i32* %len, align 4
1086     %tmp67 = icmp ne i32 %i65, %len66
1087     br i1 %tmp67, label %then69, label %else108
1088
1089 while_body33:                             ; preds = %while31
1090     %elem34 = load i8*, i8** %elem2, align 8
1091     %l35 = load %list_item**, %list_item*** %l1, align 8
1092     %ilist36 = load %list_item**, %list_item** %l35, align 8
1093     %i37 = load i32, i32* %i, align 4

```

```

1094 %_result38 = call %list_item* @list_access(%list_item* %ilist36, i32 %i37)
1095 %data_ptr_ptr39 = getelementptr inbounds %list_item, %list_item* %
    _result38, i32 0, i32 0
1096 %data_ptr40 = load i8*, i8** %data_ptr_ptr39, align 8
1097 %cast_data_ptr41 = bitcast i8* %data_ptr40 to i8**
1098 %data42 = load i8*, i8** %cast_data_ptr41, align 8
1099 %_result43 = call i1 @strcmp(i8* %data42, i8* %elem34)
1100 br i1 %_result43, label %then45, label %else47
1101
1102 merge44:                                ; preds = %else47
1103 %retlist48 = load %list_item**, %list_item*** %retlist, align 8
1104 %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
1105 %l50 = load %list_item**, %list_item*** %l1, align 8
1106 %ilist51 = load %list_item*, %list_item** %l50, align 8
1107 %i52 = load i32, i32* %i, align 4
1108 %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
1109 %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
    _result53, i32 0, i32 0
1110 %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
1111 %cast_data_ptr56 = bitcast i8* %data_ptr55 to i8**
1112 %data57 = load i8*, i8** %cast_data_ptr56, align 8
1113 %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
1114 %list_ptr_ptr59 = call %list_item** @insert_string(%list_item** %retlist48
    , i8* %data57, i32 %length58)
1115 store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
1116 %i60 = load i32, i32* %i, align 4
1117 %tmp61 = add i32 %i60, 1
1118 store i32 %tmp61, i32* %i, align 4
1119 br label %while31
1120
1121 then45:                                ; preds = %while_body33
1122 %i46 = load i32, i32* %i, align 4
1123 store i32 %i46, i32* %remove_index, align 4
1124 br label %merge32
1125
1126 else47:                                ; preds = %while_body33
1127 br label %merge44
1128
1129 merge68:                                ; preds = %else108, %
    merge71
1130 br label %merge
1131
1132 then69:                                ; preds = %merge32
1133 store i32 0, i32* %for_index, align 4
1134 store i32 0, i32* %index, align 4
1135 br label %while70
1136
1137 while70:                                ; preds = %while_body72, %
    then69
1138 %for_index98 = load i32, i32* %for_index, align 4
1139 %remove_index99 = load i32, i32* %remove_index, align 4
1140 %tmp100 = add i32 %remove_index99, 1
1141 %len101 = load i32, i32* %len, align 4
1142 %alloca1102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
1143 %head_ptr_ptr103 = bitcast i8* %alloca1102 to %list_item**
1144 store %list_item* null, %list_item** %head_ptr_ptr103, align 8
1145 %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)

```



```

1146 %ilist105 = load %list_item*, %list_item** %range_list104, align 8
1147 %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
1148 %tmp107 = icmp slt i32 %for_index98, %length106
1149 br i1 %tmp107, label %while_body72, label %merge71
1150
1151 merge71:                                ; preds = %while70
1152   br label %merge68
1153
1154 while_body72:                            ; preds = %while70
1155   %remove_index73 = load i32, i32* %remove_index, align 4
1156   %tmp74 = add i32 %remove_index73, 1
1157   %len75 = load i32, i32* %len, align 4
1158   %mallocall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
1159     *, i1** null, i32 1) to i32))
1159   %head_ptr_ptr = bitcast i8* %mallocall76 to %list_item**
1160   store %list_item* null, %list_item** %head_ptr_ptr, align 8
1161   %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
1162     list_item** %head_ptr_ptr, i32 0)
1162   %ilist77 = load %list_item*, %list_item** %range_list, align 8
1163   %for_index78 = load i32, i32* %for_index, align 4
1164   %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
1165     for_index78)
1165   %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
1166     _result79, i32 0, i32 0
1166   %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
1167   %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
1168   %data83 = load i32, i32* %cast_data_ptr82, align 4
1169   store i32 %data83, i32* %index, align 4
1170   %for_index84 = load i32, i32* %for_index, align 4
1171   %tmp85 = add i32 %for_index84, 1
1172   store i32 %tmp85, i32* %for_index, align 4
1173   %retlist86 = load %list_item**, %list_item*** %retlist, align 8
1174   %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
1175   %l88 = load %list_item**, %list_item*** %l1, align 8
1176   %ilist89 = load %list_item*, %list_item** %l88, align 8
1177   %index90 = load i32, i32* %index, align 4
1178   %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
1179     index90)
1179   %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
1180     _result91, i32 0, i32 0
1180   %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
1181   %cast_data_ptr94 = bitcast i8* %data_ptr93 to i8**
1182   %data95 = load i8*, i8** %cast_data_ptr94, align 8
1183   %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
1184   %list_ptr_ptr97 = call %list_item** @insert_string(%list_item** %retlist86
1185     , i8* %data95, i32 %length96)
1185   store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
1186   br label %while70
1187
1188 else108:                                ; preds = %merge32
1189   br label %merge68
1190 }
1191
1192 define %list_item** @remove_char_char_char_bool(%list_item** %l, i8 %elem,
1193   i1 %all) {
1193 entry:
1194   %index = alloca i32, align 4
1195   %for_index = alloca i32, align 4
1196   %remove_index = alloca i32, align 4

```

```

1197 %len = alloca i32, align 4
1198 %i = alloca i32, align 4
1199 %retlist = alloca %list_item**, align 8
1200 %l1 = alloca %list_item**, align 8
1201 store %list_item** %l, %list_item*** %l1, align 8
1202 %elem2 = alloca i8, align 1
1203 store i8 %elem, i8* %elem2, align 1
1204 %all3 = alloca i1, align 1
1205 store i1 %all, i1* %all3, align 1
1206 %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
1207 %list = bitcast i8* %malloccall to %list_item**
1208 store %list_item* null, %list_item** %list, align 8
1209 store %list_item** %list, %list_item*** %retlist, align 8
1210 store i32 0, i32* %i, align 4
1211 %l4 = load %list_item**, %list_item*** %l1, align 8
1212 %ilist = load %list_item*, %list_item** %l4, align 8
1213 %length = call i32 @list_length(%list_item* %ilist, i32 0)
1214 store i32 %length, i32* %len, align 4
1215 store i32 0, i32* %remove_index, align 4
1216 %all5 = load i1, i1* %all3, align 1
1217 br i1 %all5, label %then, label %else29
1218
1219 merge:                                ; preds = %merge65, %
    merge6
1220 %retlist104 = load %list_item**, %list_item*** %retlist, align 8
1221 ret %list_item** %retlist104
1222
1223 then:                                ; preds = %entry
    br label %while
1224
1225
1226 while:                                ; preds = %merge11, %
    then12, %then
1227 %i26 = load i32, i32* %i, align 4
1228 %len27 = load i32, i32* %len, align 4
1229 %tmp28 = icmp slt i32 %i26, %len27
1230 br i1 %tmp28, label %while_body, label %merge6
1231
1232 merge6:                                ; preds = %while
    br label %merge
1233
1234
1235 while_body:                            ; preds = %while
    %l7 = load %list_item**, %list_item*** %l1, align 8
1236 %ilist8 = load %list_item*, %list_item** %l7, align 8
1237 %i9 = load i32, i32* %i, align 4
1238 %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1239 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
1240 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1241 %data = load i8, i8* %data_ptr, align 1
1242 %elem10 = load i8, i8* %elem2, align 1
1243 %tmp = icmp eq i8 %data, %elem10
1244 br i1 %tmp, label %then12, label %else
1245
1246
1247 merge11:                              ; preds = %else
    %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1248 %list_ptr = load %list_item*, %list_item** %retlist15, align 8
1249 %l16 = load %list_item**, %list_item*** %l1, align 8
1250 %ilist17 = load %list_item*, %list_item** %l16, align 8

```

```

1252 %i18 = load i32, i32* %i, align 4
1253 %_result19 = call %list_item* @list_access(%list_item* %i18, i32 %i18)
1254 %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
1255 %data_ptr21 = load i8*, i8* %data_ptr_ptr20, align 8
1256 %data22 = load i8, i8* %data_ptr21, align 1
1257 %length23 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1258 %list_ptr_ptr = call %list_item** @insert_char(%list_item** %retlist15, i8
    %data22, i32 %length23)
1259 store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1260 %i24 = load i32, i32* %i, align 4
1261 %tmp25 = add i32 %i24, 1
1262 store i32 %tmp25, i32* %i, align 4
1263 br label %while
1264
1265 then12:                                ; preds = %while_body
1266 %i13 = load i32, i32* %i, align 4
1267 %tmp14 = add i32 %i13, 1
1268 store i32 %tmp14, i32* %i, align 4
1269 br label %while
1270
1271 else:                                    ; preds = %while_body
1272 br label %merge11
1273
1274 else29:                                  ; preds = %entry
1275 br label %while30
1276
1277 while30:                                 ; preds = %merge42, %
    else29
1278 %i59 = load i32, i32* %i, align 4
1279 %len60 = load i32, i32* %len, align 4
1280 %tmp61 = icmp slt i32 %i59, %len60
1281 br i1 %tmp61, label %while_body32, label %merge31
1282
1283 merge31:                                 ; preds = %while30, %
    then43
1284 %i62 = load i32, i32* %i, align 4
1285 %len63 = load i32, i32* %len, align 4
1286 %tmp64 = icmp ne i32 %i62, %len63
1287 br i1 %tmp64, label %then66, label %else103
1288
1289 while_body32:                             ; preds = %while30
1290 %l33 = load %list_item**, %list_item*** %l1, align 8
1291 %i134 = load %list_item*, %list_item** %l33, align 8
1292 %i35 = load i32, i32* %i, align 4
1293 %_result36 = call %list_item* @list_access(%list_item* %i134, i32 %i35)
1294 %data_ptr_ptr37 = getelementptr inbounds %list_item, %list_item* %
    _result36, i32 0, i32 0
1295 %data_ptr38 = load i8*, i8* %data_ptr_ptr37, align 8
1296 %data39 = load i8, i8* %data_ptr38, align 1
1297 %elem40 = load i8, i8* %elem2, align 1
1298 %tmp41 = icmp eq i8 %data39, %elem40
1299 br i1 %tmp41, label %then43, label %else45
1300
1301 merge42:                                 ; preds = %else45
1302 %retlist46 = load %list_item**, %list_item*** %retlist, align 8
1303 %list_ptr47 = load %list_item*, %list_item** %retlist46, align 8
1304 %l48 = load %list_item**, %list_item*** %l1, align 8
1305 %i149 = load %list_item*, %list_item** %l48, align 8

```

```

1306 %i50 = load i32, i32* %i, align 4
1307 %_result51 = call %list_item* @list_access(%list_item* %ilist49, i32 %i50)
1308 %data_ptr_ptr52 = getelementptr inbounds %list_item, %list_item* %
    _result51, i32 0, i32 0
1309 %data_ptr53 = load i8*, i8* %data_ptr_ptr52, align 8
1310 %data54 = load i8, i8* %data_ptr53, align 1
1311 %length55 = call i32 @list_length(%list_item* %list_ptr47, i32 0)
1312 %list_ptr_ptr56 = call %list_item** @insert_char(%list_item** %retlist46,
    i8 %data54, i32 %length55)
1313 store %list_item** %list_ptr_ptr56, %list_item*** %retlist, align 8
1314 %i57 = load i32, i32* %i, align 4
1315 %tmp58 = add i32 %i57, 1
1316 store i32 %tmp58, i32* %i, align 4
1317 br label %while30
1318
1319 then43:                                     ; preds = %while_body32
1320 %i44 = load i32, i32* %i, align 4
1321 store i32 %i44, i32* %remove_index, align 4
1322 br label %merge31
1323
1324 else45:                                     ; preds = %while_body32
1325 br label %merge42
1326
1327 merge65:                                   ; preds = %else103, %
    merge68
1328 br label %merge
1329
1330 then66:                                     ; preds = %merge31
1331 store i32 0, i32* %for_index, align 4
1332 store i32 0, i32* %index, align 4
1333 br label %while67
1334
1335 while67:                                   ; preds = %while_body69, %
    then66
1336 %for_index93 = load i32, i32* %for_index, align 4
1337 %remove_index94 = load i32, i32* %remove_index, align 4
1338 %tmp95 = add i32 %remove_index94, 1
1339 %len96 = load i32, i32* %len, align 4
1340 %mallocall97 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
1341 %head_ptr_ptr98 = bitcast i8* %mallocall97 to %list_item**
1342 store %list_item* null, %list_item** %head_ptr_ptr98, align 8
1343 %range_list99 = call %list_item** @range_function(i32 %tmp95, i32 %len96,
    %list_item** %head_ptr_ptr98, i32 0)
1344 %ilist100 = load %list_item*, %list_item** %range_list99, align 8
1345 %length101 = call i32 @list_length(%list_item* %ilist100, i32 0)
1346 %tmp102 = icmp slt i32 %for_index93, %length101
1347 br i1 %tmp102, label %while_body69, label %merge68
1348
1349 merge68:                                   ; preds = %while67
1350 br label %merge65
1351
1352 while_body69:                             ; preds = %while67
1353 %remove_index70 = load i32, i32* %remove_index, align 4
1354 %tmp71 = add i32 %remove_index70, 1
1355 %len72 = load i32, i32* %len, align 4
1356 %mallocall73 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
1357 %head_ptr_ptr = bitcast i8* %mallocall73 to %list_item**

```

```

1358 store %list_item* null, %list_item** %head_ptr_ptr, align 8
1359 %range_list = call %list_item** @range_function(i32 %tmp71, i32 %len72, %
    list_item** %head_ptr_ptr, i32 0)
1360 %ilist74 = load %list_item*, %list_item** %range_list, align 8
1361 %for_index75 = load i32, i32* %for_index, align 4
1362 %_result76 = call %list_item* @list_access(%list_item* %ilist74, i32 %
    for_index75)
1363 %data_ptr_ptr77 = getelementptr inbounds %list_item, %list_item* %
    _result76, i32 0, i32 0
1364 %data_ptr78 = load i8*, i8** %data_ptr_ptr77, align 8
1365 %cast_data_ptr = bitcast i8* %data_ptr78 to i32*
1366 %data79 = load i32, i32* %cast_data_ptr, align 4
1367 store i32 %data79, i32* %index, align 4
1368 %for_index80 = load i32, i32* %for_index, align 4
1369 %tmp81 = add i32 %for_index80, 1
1370 store i32 %tmp81, i32* %for_index, align 4
1371 %retlist82 = load %list_item**, %list_item*** %retlist, align 8
1372 %list_ptr83 = load %list_item*, %list_item** %retlist82, align 8
1373 %l84 = load %list_item**, %list_item*** %l1, align 8
1374 %ilist85 = load %list_item*, %list_item** %l84, align 8
1375 %index86 = load i32, i32* %index, align 4
1376 %_result87 = call %list_item* @list_access(%list_item* %ilist85, i32 %
    index86)
1377 %data_ptr_ptr88 = getelementptr inbounds %list_item, %list_item* %
    _result87, i32 0, i32 0
1378 %data_ptr89 = load i8*, i8** %data_ptr_ptr88, align 8
1379 %data90 = load i8, i8* %data_ptr89, align 1
1380 %length91 = call i32 @list_length(%list_item* %list_ptr83, i32 0)
1381 %list_ptr_ptr92 = call %list_item** @insert_char(%list_item** %retlist82,
    i8 %data90, i32 %length91)
1382 store %list_item** %list_ptr_ptr92, %list_item*** %retlist, align 8
1383 br label %while67
1384
1385 else103:                                ; preds = %merge31
1386 br label %merge65
1387 }
1388
1389 define %list_item** @remove_bool_bool_bool_bool(%list_item** %l, i1 %elem,
    i1 %all) {
1390 entry:
1391 %index = alloca i32, align 4
1392 %for_index = alloca i32, align 4
1393 %remove_index = alloca i32, align 4
1394 %len = alloca i32, align 4
1395 %i = alloca i32, align 4
1396 %retlist = alloca %list_item**, align 8
1397 %l1 = alloca %list_item**, align 8
1398 store %list_item** %l, %list_item*** %l1, align 8
1399 %elem2 = alloca i1, align 1
1400 store i1 %elem, i1* %elem2, align 1
1401 %all3 = alloca i1, align 1
1402 store i1 %all, i1* %all3, align 1
1403 %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
1404 %list = bitcast i8* %mallocall to %list_item**
1405 store %list_item* null, %list_item** %list, align 8
1406 store %list_item** %list, %list_item*** %retlist, align 8
1407 store i32 0, i32* %i, align 4
1408 %l4 = load %list_item**, %list_item*** %l1, align 8

```

```

1409 %ilist = load %list_item*, %list_item** %l4, align 8
1410 %length = call i32 @list_length(%list_item* %ilist, i32 0)
1411 store i32 %length, i32* %len, align 4
1412 store i32 0, i32* %remove_index, align 4
1413 %all5 = load i1, i1* %all3, align 1
1414 br i1 %all5, label %then, label %else30
1415
1416 merge:                                     ; preds = %merge68, %
      merge6
1417 %retlist109 = load %list_item**, %list_item*** %retlist, align 8
1418 ret %list_item** %retlist109
1419
1420 then:                                     ; preds = %entry
      br label %while
1421
1422
1423 while:                                     ; preds = %merge11, %
      then12, %then
1424 %i27 = load i32, i32* %i, align 4
1425 %len28 = load i32, i32* %len, align 4
1426 %tmp29 = icmp slt i32 %i27, %len28
1427 br i1 %tmp29, label %while_body, label %merge6
1428
1429 merge6:                                     ; preds = %while
      br label %merge
1430
1431
1432 while_body:                               ; preds = %while
1433 %l7 = load %list_item**, %list_item*** %l1, align 8
1434 %ilist8 = load %list_item*, %list_item** %l7, align 8
1435 %i9 = load i32, i32* %i, align 4
1436 %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1437 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
      i32 0, i32 0
1438 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1439 %cast_data_ptr = bitcast i8* %data_ptr to i1*
1440 %data = load i1, i1* %cast_data_ptr, align 1
1441 %elem10 = load i1, i1* %elem2, align 1
1442 %tmp = icmp eq i1 %data, %elem10
1443 br i1 %tmp, label %then12, label %else
1444
1445 merge11:                                   ; preds = %else
1446 %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1447 %list_ptr = load %list_item*, %list_item** %retlist15, align 8
1448 %l16 = load %list_item**, %list_item*** %l1, align 8
1449 %ilist17 = load %list_item*, %list_item** %l16, align 8
1450 %i18 = load i32, i32* %i, align 4
1451 %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
1452 %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
      _result19, i32 0, i32 0
1453 %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
1454 %cast_data_ptr22 = bitcast i8* %data_ptr21 to i1*
1455 %data23 = load i1, i1* %cast_data_ptr22, align 1
1456 %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1457 %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %retlist15, i1
      %data23, i32 %length24)
1458 store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1459 %i25 = load i32, i32* %i, align 4
1460 %tmp26 = add i32 %i25, 1
1461 store i32 %tmp26, i32* %i, align 4
1462 br label %while

```

```

1463
1464 then12:                                     ; preds = %while_body
1465     %i13 = load i32, i32* %i, align 4
1466     %tmp14 = add i32 %i13, 1
1467     store i32 %tmp14, i32* %i, align 4
1468     br label %while
1469
1470 else:   ; preds = %while_body
1471     br label %merge11
1472
1473 else30:                                       ; preds = %entry
1474     br label %while31
1475
1476 while31:                                     ; preds = %merge44, %
1477     else30
1478     %i62 = load i32, i32* %i, align 4
1479     %len63 = load i32, i32* %len, align 4
1480     %tmp64 = icmp slt i32 %i62, %len63
1481     br i1 %tmp64, label %while_body33, label %merge32
1482
1483 merge32:                                     ; preds = %while31, %
1484     then45
1485     %i65 = load i32, i32* %i, align 4
1486     %len66 = load i32, i32* %len, align 4
1487     %tmp67 = icmp ne i32 %i65, %len66
1488     br i1 %tmp67, label %then69, label %else108
1489
1490 while_body33:                               ; preds = %while31
1491     %l34 = load %list_item**, %list_item*** %l1, align 8
1492     %i135 = load %list_item*, %list_item** %l34, align 8
1493     %i36 = load i32, i32* %i, align 4
1494     %_result37 = call @list_item* @list_access(%list_item* %i135, i32 %i36)
1495     %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
1496         _result37, i32 0, i32 0
1497     %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
1498     %cast_data_ptr40 = bitcast i8* %data_ptr39 to i1*
1499     %data41 = load i1, i1* %cast_data_ptr40, align 1
1500     %elem42 = load i1, i1* %elem2, align 1
1501     %tmp43 = icmp eq i1 %data41, %elem42
1502     br i1 %tmp43, label %then45, label %else47
1503
1504 merge44:                                     ; preds = %else47
1505     %retlist48 = load %list_item**, %list_item*** %retlist, align 8
1506     %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
1507     %l50 = load %list_item**, %list_item*** %l1, align 8
1508     %i151 = load %list_item*, %list_item** %l50, align 8
1509     %i52 = load i32, i32* %i, align 4
1510     %_result53 = call %list_item* @list_access(%list_item* %i151, i32 %i52)
1511     %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
1512         _result53, i32 0, i32 0
1513     %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
1514     %cast_data_ptr56 = bitcast i8* %data_ptr55 to i1*
1515     %data57 = load i1, i1* %cast_data_ptr56, align 1
1516     %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
1517     %list_ptr_ptr59 = call %list_item** @insert_bool(%list_item** %retlist48,
1518         i1 %data57, i32 %length58)
1519     store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
1520     %i60 = load i32, i32* %i, align 4
1521     %tmp61 = add i32 %i60, 1

```

```

1517     store i32 %tmp61, i32* %i, align 4
1518     br label %while31
1519
1520 then45:                                     ; preds = %while_body33
1521     %i46 = load i32, i32* %i, align 4
1522     store i32 %i46, i32* %remove_index, align 4
1523     br label %merge32
1524
1525 else47:                                     ; preds = %while_body33
1526     br label %merge44
1527
1528 merge68:                                   ; preds = %else108, %
1529     merge71
1530     br label %merge
1531
1532 then69:                                     ; preds = %merge32
1533     store i32 0, i32* %for_index, align 4
1534     store i32 0, i32* %index, align 4
1535     br label %while70
1536
1537 while70:                                   ; preds = %while_body72, %
1538     then69
1539     %for_index98 = load i32, i32* %for_index, align 4
1540     %remove_index99 = load i32, i32* %remove_index, align 4
1541     %tmp100 = add i32 %remove_index99, 1
1542     %len101 = load i32, i32* %len, align 4
1543     %mallocall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
1544         i1*, i1** null, i32 1) to i32))
1545     %head_ptr_ptr103 = bitcast i8* %mallocall102 to %list_item**
1546     store %list_item* null, %list_item** %head_ptr_ptr103, align 8
1547     %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
1548         len101, %list_item** %head_ptr_ptr103, i32 0)
1549     %ilist105 = load %list_item*, %list_item** %range_list104, align 8
1550     %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
1551     %tmp107 = icmp slt i32 %for_index98, %length106
1552     br i1 %tmp107, label %while_body72, label %merge71
1553
1554 merge71:                                   ; preds = %while70
1555     br label %merge68
1556
1557 while_body72:                               ; preds = %while70
1558     %remove_index73 = load i32, i32* %remove_index, align 4
1559     %tmp74 = add i32 %remove_index73, 1
1560     %len75 = load i32, i32* %len, align 4
1561     %mallocall176 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
1562         *, i1** null, i32 1) to i32))
1563     %head_ptr_ptr = bitcast i8* %mallocall176 to %list_item**
1564     store %list_item* null, %list_item** %head_ptr_ptr, align 8
1565     %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
1566         list_item** %head_ptr_ptr, i32 0)
1567     %ilist77 = load %list_item*, %list_item** %range_list, align 8
1568     %for_index78 = load i32, i32* %for_index, align 4
1569     %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
1570         for_index78)
1571     %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
1572         _result79, i32 0, i32 0
1573     %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
1574     %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
1575     %data83 = load i32, i32* %cast_data_ptr82, align 4

```



```

1568 store i32 %data83, i32* %index, align 4
1569 %for_index84 = load i32, i32* %for_index, align 4
1570 %tmp85 = add i32 %for_index84, 1
1571 store i32 %tmp85, i32* %for_index, align 4
1572 %retlist86 = load %list_item**, %list_item*** %retlist, align 8
1573 %list_ptr87 = load %list_item**, %list_item*** %retlist86, align 8
1574 %l88 = load %list_item**, %list_item*** %l1, align 8
1575 %ilist89 = load %list_item*, %list_item** %l88, align 8
1576 %index90 = load i32, i32* %index, align 4
1577 %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
1578 %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
1579 %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
1580 %cast_data_ptr94 = bitcast i8* %data_ptr93 to i1*
1581 %data95 = load i1, i1* %cast_data_ptr94, align 1
1582 %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
1583 %list_ptr_ptr97 = call %list_item** @insert_bool(%list_item** %retlist86,
    i1 %data95, i32 %length96)
1584 store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
1585 br label %while70
1586
1587 else108:                                ; preds = %merge32
1588 br label %merge68
1589 }
1590
1591 define %list_item** @remove_float_float_float_bool(%list_item** %l, double %
    elem, i1 %all) {
1592 entry:
1593 %index = alloca i32, align 4
1594 %for_index = alloca i32, align 4
1595 %remove_index = alloca i32, align 4
1596 %len = alloca i32, align 4
1597 %i = alloca i32, align 4
1598 %retlist = alloca %list_item**, align 8
1599 %l1 = alloca %list_item**, align 8
1600 store %list_item** %l, %list_item*** %l1, align 8
1601 %elem2 = alloca double, align 8
1602 store double %elem, double* %elem2, align 8
1603 %all3 = alloca i1, align 1
1604 store i1 %all, i1* %all3, align 1
1605 %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
1606 %list = bitcast i8* %malloccall to %list_item**
1607 store %list_item* null, %list_item** %list, align 8
1608 store %list_item** %list, %list_item*** %retlist, align 8
1609 store i32 0, i32* %i, align 4
1610 %l4 = load %list_item**, %list_item*** %l1, align 8
1611 %ilist = load %list_item*, %list_item** %l4, align 8
1612 %length = call i32 @list_length(%list_item* %ilist, i32 0)
1613 store i32 %length, i32* %len, align 4
1614 store i32 0, i32* %remove_index, align 4
1615 %all5 = load i1, i1* %all3, align 1
1616 br i1 %all5, label %then, label %else30
1617
1618 merge:                                ; preds = %merge68, %
    merge6
1619 %retlist109 = load %list_item**, %list_item*** %retlist, align 8
1620 ret %list_item** %retlist109

```

```

1621
1622 then:                                     ; preds = %entry
1623     br label %while
1624
1625 while:                                     ; preds = %merge11, %
    then12, %then
1626     %i27 = load i32, i32* %i, align 4
1627     %len28 = load i32, i32* %len, align 4
1628     %tmp29 = icmp slt i32 %i27, %len28
1629     br i1 %tmp29, label %while_body, label %merge6
1630
1631 merge6:                                    ; preds = %while
1632     br label %merge
1633
1634 while_body:                               ; preds = %while
1635     %l7 = load %list_item**, %list_item*** %l1, align 8
1636     %ilist8 = load %list_item**, %list_item** %l7, align 8
1637     %i9 = load i32, i32* %i, align 4
1638     @_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1639     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* @_result,
        i32 0, i32 0
1640     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1641     %cast_data_ptr = bitcast i8* %data_ptr to double*
1642     %data = load double, double* %cast_data_ptr, align 8
1643     %elem10 = load double, double* %elem2, align 8
1644     %tmp = fcmp oeq double %data, %elem10
1645     br i1 %tmp, label %then12, label %else
1646
1647 merge11:                                   ; preds = %else
1648     %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1649     %list_ptr = load %list_item**, %list_item** %retlist15, align 8
1650     %l16 = load %list_item**, %list_item*** %l1, align 8
1651     %ilist17 = load %list_item**, %list_item** %l16, align 8
1652     %i18 = load i32, i32* %i, align 4
1653     @_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
1654     %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
        _result19, i32 0, i32 0
1655     %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
1656     %cast_data_ptr22 = bitcast i8* %data_ptr21 to double*
1657     %data23 = load double, double* %cast_data_ptr22, align 8
1658     %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1659     %list_ptr_ptr = call %list_item** @insert_float(%list_item** %retlist15,
        double %data23, i32 %length24)
1660     store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1661     %i25 = load i32, i32* %i, align 4
1662     %tmp26 = add i32 %i25, 1
1663     store i32 %tmp26, i32* %i, align 4
1664     br label %while
1665
1666 then12:                                    ; preds = %while_body
1667     %i13 = load i32, i32* %i, align 4
1668     %tmp14 = add i32 %i13, 1
1669     store i32 %tmp14, i32* %i, align 4
1670     br label %while
1671
1672 else:                                       ; preds = %while_body
1673     br label %merge11
1674
1675 else30:                                    ; preds = %entry

```

```

1676     br label %while31
1677
1678 while31:                                     ; preds = %merge44, %
    else30
1679     %i62 = load i32, i32* %i, align 4
1680     %len63 = load i32, i32* %len, align 4
1681     %tmp64 = icmp slt i32 %i62, %len63
1682     br i1 %tmp64, label %while_body33, label %merge32
1683
1684 merge32:                                     ; preds = %while31, %
    then45
1685     %i65 = load i32, i32* %i, align 4
1686     %len66 = load i32, i32* %len, align 4
1687     %tmp67 = icmp ne i32 %i65, %len66
1688     br i1 %tmp67, label %then69, label %else108
1689
1690 while_body33:                               ; preds = %while31
1691     %l34 = load %list_item**, %list_item*** %l1, align 8
1692     %ilist35 = load %list_item*, %list_item** %l34, align 8
1693     %i36 = load i32, i32* %i, align 4
1694     %_result37 = call %list_item* @list_access(%list_item* %ilist35, i32 %i36)
1695     %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
        _result37, i32 0, i32 0
1696     %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
1697     %cast_data_ptr40 = bitcast i8* %data_ptr39 to double*
1698     %data41 = load double, double* %cast_data_ptr40, align 8
1699     %elem42 = load double, double* %elem2, align 8
1700     %tmp43 = fcmp oeq double %data41, %elem42
1701     br i1 %tmp43, label %then45, label %else47
1702
1703 merge44:                                     ; preds = %else47
1704     %retlist48 = load %list_item**, %list_item*** %retlist, align 8
1705     %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
1706     %l50 = load %list_item**, %list_item*** %l1, align 8
1707     %ilist51 = load %list_item*, %list_item** %l50, align 8
1708     %i52 = load i32, i32* %i, align 4
1709     %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
1710     %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
        _result53, i32 0, i32 0
1711     %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
1712     %cast_data_ptr56 = bitcast i8* %data_ptr55 to double*
1713     %data57 = load double, double* %cast_data_ptr56, align 8
1714     %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
1715     %list_ptr_ptr59 = call %list_item** @insert_float(%list_item** %retlist48,
        double %data57, i32 %length58)
1716     store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
1717     %i60 = load i32, i32* %i, align 4
1718     %tmp61 = add i32 %i60, 1
1719     store i32 %tmp61, i32* %i, align 4
1720     br label %while31
1721
1722 then45:                                     ; preds = %while_body33
1723     %i46 = load i32, i32* %i, align 4
1724     store i32 %i46, i32* %remove_index, align 4
1725     br label %merge32
1726
1727 else47:                                     ; preds = %while_body33
1728     br label %merge44
1729

```

```

1730 merge68:                                     ; preds = %else108, %
      merge71
1731   br label %merge
1732
1733 then69:   ; preds = %merge32
1734   store i32 0, i32* %for_index, align 4
1735   store i32 0, i32* %index, align 4
1736   br label %while70
1737
1738 while70:                                       ; preds = %while_body72, %
      then69
1739   %for_index98 = load i32, i32* %for_index, align 4
1740   %remove_index99 = load i32, i32* %remove_index, align 4
1741   %tmp100 = add i32 %remove_index99, 1
1742   %len101 = load i32, i32* %len, align 4
1743   %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
      i1*, i1** null, i32 1) to i32))
1744   %head_ptr_ptr103 = bitcast i8* %malloccall102 to %list_item**
1745   store %list_item* null, %list_item** %head_ptr_ptr103, align 8
1746   %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
      len101, %list_item** %head_ptr_ptr103, i32 0)
1747   %i105 = load %list_item*, %list_item** %range_list104, align 8
1748   %length106 = call i32 @list_length(%list_item* %i105, i32 0)
1749   %tmp107 = icmp slt i32 %for_index98, %length106
1750   br i1 %tmp107, label %while_body72, label %merge71
1751
1752 merge71:                                       ; preds = %while70
1753   br label %merge68
1754
1755 while_body72:                                  ; preds = %while70
1756   %remove_index73 = load i32, i32* %remove_index, align 4
1757   %tmp74 = add i32 %remove_index73, 1
1758   %len75 = load i32, i32* %len, align 4
1759   %malloccall176 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
      *, i1** null, i32 1) to i32))
1760   %head_ptr_ptr = bitcast i8* %malloccall176 to %list_item**
1761   store %list_item* null, %list_item** %head_ptr_ptr, align 8
1762   %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
      list_item** %head_ptr_ptr, i32 0)
1763   %i177 = load %list_item*, %list_item** %range_list, align 8
1764   %for_index78 = load i32, i32* %for_index, align 4
1765   %_result79 = call %list_item* @list_access(%list_item* %i177, i32 %
      for_index78)
1766   %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
      _result79, i32 0, i32 0
1767   %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
1768   %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
1769   %data83 = load i32, i32* %cast_data_ptr82, align 4
1770   store i32 %data83, i32* %index, align 4
1771   %for_index84 = load i32, i32* %for_index, align 4
1772   %tmp85 = add i32 %for_index84, 1
1773   store i32 %tmp85, i32* %for_index, align 4
1774   %retlist86 = load %list_item**, %list_item** %retlist, align 8
1775   %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
1776   %l88 = load %list_item**, %list_item** %l1, align 8
1777   %i189 = load %list_item*, %list_item** %l88, align 8
1778   %index90 = load i32, i32* %index, align 4
1779   %_result91 = call %list_item* @list_access(%list_item* %i189, i32 %
      index90)

```

```

1780 %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
1781 %data_ptr93 = load i8*, i8* %data_ptr_ptr92, align 8
1782 %cast_data_ptr94 = bitcast i8* %data_ptr93 to double*
1783 %data95 = load double, double* %cast_data_ptr94, align 8
1784 %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
1785 %list_ptr_ptr97 = call %list_item** @insert_float(%list_item** %retlist86,
    double %data95, i32 %length96)
1786 store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
1787 br label %while70
1788
1789 else108:                                ; preds = %merge32
1790 br label %merge68
1791 }
1792
1793 define %list_item** @remove_int_int_int_bool(%list_item** %l, i32 %elem, i1
    %all) {
1794 entry:
1795 %index = alloca i32, align 4
1796 %for_index = alloca i32, align 4
1797 %remove_index = alloca i32, align 4
1798 %len = alloca i32, align 4
1799 %i = alloca i32, align 4
1800 %retlist = alloca %list_item**, align 8
1801 %l1 = alloca %list_item**, align 8
1802 store %list_item** %l, %list_item*** %l1, align 8
1803 %elem2 = alloca i32, align 4
1804 store i32 %elem, i32* %elem2, align 4
1805 %all3 = alloca i1, align 1
1806 store i1 %all, i1* %all3, align 1
1807 %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
1808 %list = bitcast i8* %malloccall to %list_item**
1809 store %list_item* null, %list_item** %list, align 8
1810 store %list_item** %list, %list_item*** %retlist, align 8
1811 store i32 0, i32* %i, align 4
1812 %l4 = load %list_item**, %list_item*** %l1, align 8
1813 %ilist = load %list_item*, %list_item** %l4, align 8
1814 %length = call i32 @list_length(%list_item* %ilist, i32 0)
1815 store i32 %length, i32* %len, align 4
1816 store i32 0, i32* %remove_index, align 4
1817 %all5 = load i1, i1* %all3, align 1
1818 br i1 %all5, label %then, label %else30
1819
1820 merge:                                ; preds = %merge68, %
    merge6
1821 %retlist109 = load %list_item**, %list_item*** %retlist, align 8
1822 ret %list_item** %retlist109
1823
1824 then:                                ; preds = %entry
1825 br label %while
1826
1827 while:                                ; preds = %merge11, %
    then12, %then
1828 %i27 = load i32, i32* %i, align 4
1829 %len28 = load i32, i32* %len, align 4
1830 %tmp29 = icmp slt i32 %i27, %len28
1831 br i1 %tmp29, label %while_body, label %merge6
1832

```

```

1833 merge6:                                     ; preds = %while
1834     br label %merge
1835
1836 while_body:                                   ; preds = %while
1837     %l7 = load %list_item**, %list_item*** %l1, align 8
1838     %ilist8 = load %list_item*, %list_item** %l7, align 8
1839     %i9 = load i32, i32* %i, align 4
1840     %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1841     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
1842         i32 0, i32 0
1843     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1844     %cast_data_ptr = bitcast i8* %data_ptr to i32*
1845     %data = load i32, i32* %cast_data_ptr, align 4
1846     %elem10 = load i32, i32* %elem2, align 4
1847     %tmp = icmp eq i32 %data, %elem10
1848     br i1 %tmp, label %then12, label %else
1849
1850 merge11:                                     ; preds = %else
1851     %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1852     %list_ptr = load %list_item*, %list_item** %retlist15, align 8
1853     %l16 = load %list_item**, %list_item*** %l1, align 8
1854     %ilist17 = load %list_item*, %list_item** %l16, align 8
1855     %i18 = load i32, i32* %i, align 4
1856     %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
1857     %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
1858         _result19, i32 0, i32 0
1859     %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
1860     %cast_data_ptr22 = bitcast i8* %data_ptr21 to i32*
1861     %data23 = load i32, i32* %cast_data_ptr22, align 4
1862     %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1863     %list_ptr_ptr = call %list_item** @insert_int(%list_item** %retlist15, i32
1864         %data23, i32 %length24)
1865     store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1866     %i25 = load i32, i32* %i, align 4
1867     %tmp26 = add i32 %i25, 1
1868     store i32 %tmp26, i32* %i, align 4
1869     br label %while
1870
1871 then12:                                       ; preds = %while_body
1872     %i13 = load i32, i32* %i, align 4
1873     %tmp14 = add i32 %i13, 1
1874     store i32 %tmp14, i32* %i, align 4
1875     br label %while
1876
1877 else:   ; preds = %while_body
1878     br label %merge11
1879
1880 else30:                                       ; preds = %entry
1881     br label %while31
1882
1883 while31:                                     ; preds = %merge44, %
1884     else30
1885     %i62 = load i32, i32* %i, align 4
1886     %len63 = load i32, i32* %len, align 4
1887     %tmp64 = icmp slt i32 %i62, %len63
1888     br i1 %tmp64, label %while_body33, label %merge32
1889
1890 merge32:                                     ; preds = %while31, %
1891     then45

```

```

1887 %i65 = load i32, i32* %i, align 4
1888 %len66 = load i32, i32* %len, align 4
1889 %tmp67 = icmp ne i32 %i65, %len66
1890 br i1 %tmp67, label %then69, label %else108
1891
1892 while_body33:                                ; preds = %while31
1893 %l34 = load %list_item**, %list_item*** %l1, align 8
1894 %ilist35 = load %list_item*, %list_item** %l34, align 8
1895 %i36 = load i32, i32* %i, align 4
1896 %_result37 = call %list_item* @list_access(%list_item* %ilist35, i32 %i36)
1897 %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
    _result37, i32 0, i32 0
1898 %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
1899 %cast_data_ptr40 = bitcast i8* %data_ptr39 to i32*
1900 %data41 = load i32, i32* %cast_data_ptr40, align 4
1901 %elem42 = load i32, i32* %elem2, align 4
1902 %tmp43 = icmp eq i32 %data41, %elem42
1903 br i1 %tmp43, label %then45, label %else47
1904
1905 merge44:                                    ; preds = %else47
1906 %retlist48 = load %list_item**, %list_item*** %retlist, align 8
1907 %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
1908 %l50 = load %list_item**, %list_item*** %l1, align 8
1909 %ilist51 = load %list_item*, %list_item** %l50, align 8
1910 %i52 = load i32, i32* %i, align 4
1911 %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
1912 %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
    _result53, i32 0, i32 0
1913 %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
1914 %cast_data_ptr56 = bitcast i8* %data_ptr55 to i32*
1915 %data57 = load i32, i32* %cast_data_ptr56, align 4
1916 %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
1917 %list_ptr_ptr59 = call %list_item** @insert_int(%list_item** %retlist48,
    i32 %data57, i32 %length58)
1918 store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
1919 %i60 = load i32, i32* %i, align 4
1920 %tmp61 = add i32 %i60, 1
1921 store i32 %tmp61, i32* %i, align 4
1922 br label %while31
1923
1924 then45:                                    ; preds = %while_body33
1925 %i46 = load i32, i32* %i, align 4
1926 store i32 %i46, i32* %remove_index, align 4
1927 br label %merge32
1928
1929 else47:                                    ; preds = %while_body33
1930 br label %merge44
1931
1932 merge68:                                    ; preds = %else108, %
    merge71
1933 br label %merge
1934
1935 then69:                                    ; preds = %merge32
1936 store i32 0, i32* %for_index, align 4
1937 store i32 0, i32* %index, align 4
1938 br label %while70
1939
1940 while70:                                    ; preds = %while_body72, %
    then69

```

```

1941 %for_index98 = load i32, i32* %for_index, align 4
1942 %remove_index99 = load i32, i32* %remove_index, align 4
1943 %tmp100 = add i32 %remove_index99, 1
1944 %len101 = load i32, i32* %len, align 4
1945 %mallocall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
1946 %head_ptr_ptr103 = bitcast i8* %mallocall102 to %list_item**
1947 store %list_item* null, %list_item** %head_ptr_ptr103, align 8
1948 %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
1949 %ilist105 = load %list_item*, %list_item** %range_list104, align 8
1950 %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
1951 %tmp107 = icmp slt i32 %for_index98, %length106
1952 br i1 %tmp107, label %while_body72, label %merge71
1953
1954 merge71:                                ; preds = %while70
1955 br label %merge68
1956
1957 while_body72:                            ; preds = %while70
1958 %remove_index73 = load i32, i32* %remove_index, align 4
1959 %tmp74 = add i32 %remove_index73, 1
1960 %len75 = load i32, i32* %len, align 4
1961 %mallocall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
1962 %head_ptr_ptr = bitcast i8* %mallocall76 to %list_item**
1963 store %list_item* null, %list_item** %head_ptr_ptr, align 8
1964 %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
1965 %ilist77 = load %list_item*, %list_item** %range_list, align 8
1966 %for_index78 = load i32, i32* %for_index, align 4
1967 %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
    for_index78)
1968 %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
1969 %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
1970 %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
1971 %data83 = load i32, i32* %cast_data_ptr82, align 4
1972 store i32 %data83, i32* %index, align 4
1973 %for_index84 = load i32, i32* %for_index, align 4
1974 %tmp85 = add i32 %for_index84, 1
1975 store i32 %tmp85, i32* %for_index, align 4
1976 %retlist86 = load %list_item**, %list_item*** %retlist, align 8
1977 %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
1978 %l88 = load %list_item**, %list_item*** %l1, align 8
1979 %ilist89 = load %list_item*, %list_item** %l88, align 8
1980 %index90 = load i32, i32* %index, align 4
1981 %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
1982 %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
1983 %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
1984 %cast_data_ptr94 = bitcast i8* %data_ptr93 to i32*
1985 %data95 = load i32, i32* %cast_data_ptr94, align 4
1986 %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
1987 %list_ptr_ptr97 = call %list_item** @insert_int(%list_item** %retlist86,
    i32 %data95, i32 %length96)
1988 store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
1989 br label %while70
1990

```



```

1991 else108:                                     ; preds = %merge32
1992     br label %merge68
1993 }
1994
1995 define i1 @contains_float_float(%list_item** %l, %list_item** %items) {
1996 entry:
1997     %r = alloca i1, align 1
1998     %for_index42 = alloca i32, align 4
1999     %ref = alloca double, align 8
2000     %for_index7 = alloca i32, align 4
2001     %item = alloca double, align 8
2002     %for_index = alloca i32, align 4
2003     %res = alloca %list_item**, align 8
2004     %l1 = alloca %list_item**, align 8
2005     store %list_item** %l, %list_item*** %l1, align 8
2006     %items2 = alloca %list_item**, align 8
2007     store %list_item** %items, %list_item*** %items2, align 8
2008     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
2009         i1** null, i32 1) to i32))
2009     %list = bitcast i8* %mallocall to %list_item**
2010     store %list_item** null, %list_item** %list, align 8
2011     store %list_item** %list, %list_item*** %res, align 8
2012     store i32 0, i32* %for_index, align 4
2013     store double 0.000000e+00, double* %item, align 8
2014     br label %while
2015
2016 while:   ; preds = %merge9, %entry
2017     %for_index37 = load i32, i32* %for_index, align 4
2018     %items38 = load %list_item**, %list_item*** %items2, align 8
2019     %ilist39 = load %list_item*, %list_item** %items38, align 8
2020     %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
2021     %tmp41 = icmp slt i32 %for_index37, %length40
2022     br i1 %tmp41, label %while_body, label %merge
2023
2024 merge:                                       ; preds = %while
2025     store i32 0, i32* %for_index42, align 4
2026     store i1 false, i1* %r, align 1
2027     br label %while43
2028
2029 while_body:                                ; preds = %while
2030     %items3 = load %list_item**, %list_item*** %items2, align 8
2031     %ilist = load %list_item*, %list_item** %items3, align 8
2032     %for_index4 = load i32, i32* %for_index, align 4
2033     %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
2034         for_index4)
2035     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
2036         i32 0, i32 0
2037     %data_ptr = load i8*, i8* %data_ptr_ptr, align 8
2038     %cast_data_ptr = bitcast i8* %data_ptr to double*
2039     %data = load double, double* %cast_data_ptr, align 8
2040     store double %data, double* %item, align 8
2041     %for_index5 = load i32, i32* %for_index, align 4
2042     %tmp = add i32 %for_index5, 1
2043     store i32 %tmp, i32* %for_index, align 4
2044     %res6 = load %list_item**, %list_item*** %res, align 8
2045     %list_ptr = load %list_item*, %list_item** %res6, align 8
2046     %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2047     %list_ptr_ptr = call %list_item* @insert_bool(%list_item** %res6, i1
2048         false, i32 %length)

```

```

2046 store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
2047 store i32 0, i32* %for_index7, align 4
2048 store double 0.000000e+00, double* %ref, align 8
2049 br label %while8
2050
2051 while8:                                     ; preds = %merge24, %
    while_body
2052 %for_index32 = load i32, i32* %for_index7, align 4
2053 %l33 = load %list_item**, %list_item*** %l1, align 8
2054 %ilist34 = load %list_item*, %list_item** %l33, align 8
2055 %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
2056 %tmp36 = icmp slt i32 %for_index32, %length35
2057 br i1 %tmp36, label %while_body10, label %merge9
2058
2059 merge9:                                     ; preds = %while8
    br label %while
2060
2061
2062 while_body10:                               ; preds = %while8
    %l11 = load %list_item**, %list_item*** %l1, align 8
2063 %ilist12 = load %list_item*, %list_item** %l11, align 8
2064 %for_index13 = load i32, i32* %for_index7, align 4
2065 %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
2066 %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
2067 %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
2068 %cast_data_ptr17 = bitcast i8* %data_ptr16 to double*
2069 %data18 = load double, double* %cast_data_ptr17, align 8
2070 store double %data18, double* %ref, align 8
2071 %for_index19 = load i32, i32* %for_index7, align 4
2072 %tmp20 = add i32 %for_index19, 1
2073 store i32 %tmp20, i32* %for_index7, align 4
2074 %ref21 = load double, double* %ref, align 8
2075 %item22 = load double, double* %item, align 8
2076 %tmp23 = fcmp oeq double %ref21, %item22
2077 br i1 %tmp23, label %then, label %else
2078
2079
2080 merge24:                                   ; preds = %else, %then
    br label %while8
2081
2082
2083 then:                                       ; preds = %while_body10
    %res25 = load %list_item**, %list_item*** %res, align 8
2084 %ilist26 = load %list_item*, %list_item** %res25, align 8
2085 %res27 = load %list_item**, %list_item*** %res, align 8
2086 %ilist28 = load %list_item*, %list_item** %res27, align 8
2087 %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
2088 %tmp30 = sub i32 %length29, 1
2089 %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
2090 %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
2091 %mallocall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
    i1* null, i32 1) to i32))
2092 %copy_ptr = bitcast i8* %mallocall31 to i1*
2093 store i1 true, i1* %copy_ptr, align 1
2094 %ccopy = bitcast i1* %copy_ptr to i8*
2095 store i8* %ccopy, i8** %data_ptpt, align 8
2096 br label %merge24
2097
2098
2099 else:                                       ; preds = %while_body10

```

```

2100     br label %merge24
2101
2102 while43:                                     ; preds = %merge58, %merge
2103     %for_index61 = load i32, i32* %for_index42, align 4
2104     %res62 = load %list_item**, %list_item*** %res, align 8
2105     %ilist63 = load %list_item*, %list_item** %res62, align 8
2106     %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
2107     %tmp65 = icmp slt i32 %for_index61, %length64
2108     br i1 %tmp65, label %while_body45, label %merge44
2109
2110 merge44:                                     ; preds = %while43
2111     ret i1 true
2112
2113 while_body45:                               ; preds = %while43
2114     %res46 = load %list_item**, %list_item*** %res, align 8
2115     %ilist47 = load %list_item*, %list_item** %res46, align 8
2116     %for_index48 = load i32, i32* %for_index42, align 4
2117     %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
        for_index48)
2118     %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
        _result49, i32 0, i32 0
2119     %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
2120     %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
2121     %data53 = load i1, i1* %cast_data_ptr52, align 1
2122     store i1 %data53, i1* %r, align 1
2123     %for_index54 = load i32, i32* %for_index42, align 4
2124     %tmp55 = add i32 %for_index54, 1
2125     store i32 %tmp55, i32* %for_index42, align 4
2126     %r56 = load i1, i1* %r, align 1
2127     %tmp57 = xor i1 %r56, true
2128     br i1 %tmp57, label %then59, label %else60
2129
2130 merge58:                                     ; preds = %else60
2131     br label %while43
2132
2133 then59:                                     ; preds = %while_body45
2134     ret i1 false
2135
2136 else60:                                     ; preds = %while_body45
2137     br label %merge58
2138 }
2139
2140 define i1 @contains_bool_bool(%list_item** %l, %list_item** %items) {
2141 entry:
2142     %r = alloca i1, align 1
2143     %for_index42 = alloca i32, align 4
2144     %ref = alloca i1, align 1
2145     %for_index7 = alloca i32, align 4
2146     %item = alloca i1, align 1
2147     %for_index = alloca i32, align 4
2148     %res = alloca %list_item**, align 8
2149     %l1 = alloca %list_item**, align 8
2150     store %list_item** %l, %list_item*** %l1, align 8
2151     %items2 = alloca %list_item**, align 8
2152     store %list_item** %items, %list_item*** %items2, align 8
2153     %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))
2154     %list = bitcast i8* %malloccall to %list_item**
2155     store %list_item* null, %list_item** %list, align 8

```

```

2156 store %list_item** %list, %list_item*** %res, align 8
2157 store i32 0, i32* %for_index, align 4
2158 store i1 false, i1* %item, align 1
2159 br label %while
2160
2161 while: ; preds = %merge9, %entry
2162 %for_index37 = load i32, i32* %for_index, align 4
2163 %items38 = load %list_item**, %list_item*** %items2, align 8
2164 %ilist39 = load %list_item*, %list_item** %items38, align 8
2165 %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
2166 %tmp41 = icmp slt i32 %for_index37, %length40
2167 br i1 %tmp41, label %while_body, label %merge
2168
2169 merge: ; preds = %while
2170 store i32 0, i32* %for_index42, align 4
2171 store i1 false, i1* %r, align 1
2172 br label %while43
2173
2174 while_body: ; preds = %while
2175 %items3 = load %list_item**, %list_item*** %items2, align 8
2176 %ilist = load %list_item*, %list_item** %items3, align 8
2177 %for_index4 = load i32, i32* %for_index, align 4
2178 %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
2179 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
2180 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2181 %cast_data_ptr = bitcast i8* %data_ptr to i1*
2182 %data = load i1, i1* %cast_data_ptr, align 1
2183 store i1 %data, i1* %item, align 1
2184 %for_index5 = load i32, i32* %for_index, align 4
2185 %tmp = add i32 %for_index5, 1
2186 store i32 %tmp, i32* %for_index, align 4
2187 %res6 = load %list_item**, %list_item*** %res, align 8
2188 %list_ptr = load %list_item*, %list_item** %res6, align 8
2189 %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2190 %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
2191 store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
2192 store i32 0, i32* %for_index7, align 4
2193 store i1 false, i1* %ref, align 1
2194 br label %while8
2195
2196 while8: ; preds = %merge24, %
    while_body
2197 %for_index32 = load i32, i32* %for_index7, align 4
2198 %l33 = load %list_item**, %list_item*** %l1, align 8
2199 %ilist34 = load %list_item*, %list_item** %l33, align 8
2200 %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
2201 %tmp36 = icmp slt i32 %for_index32, %length35
2202 br i1 %tmp36, label %while_body10, label %merge9
2203
2204 merge9: ; preds = %while8
2205 br label %while
2206
2207 while_body10: ; preds = %while8
2208 %l11 = load %list_item**, %list_item*** %l1, align 8
2209 %ilist12 = load %list_item*, %list_item** %l11, align 8
2210 %for_index13 = load i32, i32* %for_index7, align 4

```

```

2211 %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
      for_index13)
2212 %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
      _result14, i32 0, i32 0
2213 %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
2214 %cast_data_ptr17 = bitcast i8* %data_ptr16 to i1*
2215 %data18 = load i1, i1* %cast_data_ptr17, align 1
2216 store i1 %data18, i1* %ref, align 1
2217 %for_index19 = load i32, i32* %for_index7, align 4
2218 %tmp20 = add i32 %for_index19, 1
2219 store i32 %tmp20, i32* %for_index7, align 4
2220 %ref21 = load i1, i1* %ref, align 1
2221 %item22 = load i1, i1* %item, align 1
2222 %tmp23 = icmp eq i1 %ref21, %item22
2223 br i1 %tmp23, label %then, label %else
2224
2225 merge24:                                ; preds = %else, %then
2226     br label %while8
2227
2228 then:                                    ; preds = %while_body10
2229 %res25 = load %list_item**, %list_item*** %res, align 8
2230 %ilist26 = load %list_item*, %list_item** %res25, align 8
2231 %res27 = load %list_item**, %list_item*** %res, align 8
2232 %ilist28 = load %list_item*, %list_item** %res27, align 8
2233 %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
2234 %tmp30 = sub i32 %length29, 1
2235 %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
2236 %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
      0, i32 0
2237 %alloca131 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
      i1* null, i32 1) to i32))
2238 %copy_ptr = bitcast i8* %alloca131 to i1*
2239 store i1 true, i1* %copy_ptr, align 1
2240 %ccopy = bitcast i1* %copy_ptr to i8*
2241 store i8* %ccopy, i8** %data_ptpt, align 8
2242 br label %merge24
2243
2244 else:                                    ; preds = %while_body10
2245     br label %merge24
2246
2247 while43:                                ; preds = %merge58, %merge
2248 %for_index61 = load i32, i32* %for_index42, align 4
2249 %res62 = load %list_item**, %list_item*** %res, align 8
2250 %ilist63 = load %list_item*, %list_item** %res62, align 8
2251 %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
2252 %tmp65 = icmp slt i32 %for_index61, %length64
2253 br i1 %tmp65, label %while_body45, label %merge44
2254
2255 merge44:                                ; preds = %while43
2256     ret i1 true
2257
2258 while_body45:                            ; preds = %while43
2259 %res46 = load %list_item**, %list_item*** %res, align 8
2260 %ilist47 = load %list_item*, %list_item** %res46, align 8
2261 %for_index48 = load i32, i32* %for_index42, align 4
2262 %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
      for_index48)
2263 %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
      _result49, i32 0, i32 0

```

```

2264 %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
2265 %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
2266 %data53 = load i1, i1* %cast_data_ptr52, align 1
2267 store i1 %data53, i1* %r, align 1
2268 %for_index54 = load i32, i32* %for_index42, align 4
2269 %tmp55 = add i32 %for_index54, 1
2270 store i32 %tmp55, i32* %for_index42, align 4
2271 %r56 = load i1, i1* %r, align 1
2272 %tmp57 = xor i1 %r56, true
2273 br i1 %tmp57, label %then59, label %else60
2274
2275 merge58:                                ; preds = %else60
2276     br label %while43
2277
2278 then59:                                ; preds = %while_body45
2279     ret i1 false
2280
2281 else60:                                ; preds = %while_body45
2282     br label %merge58
2283 }
2284
2285 define i1 @contains_char_char(%list_item** %l, %list_item** %items) {
2286 entry:
2287     %r = alloca i1, align 1
2288     %for_index41 = alloca i32, align 4
2289     %ref = alloca i8, align 1
2290     %for_index7 = alloca i32, align 4
2291     %item = alloca i8, align 1
2292     %for_index = alloca i32, align 4
2293     %res = alloca %list_item**, align 8
2294     %l1 = alloca %list_item**, align 8
2295     store %list_item** %l, %list_item*** %l1, align 8
2296     %items2 = alloca %list_item**, align 8
2297     store %list_item** %items, %list_item*** %items2, align 8
2298     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
2299         i1** null, i32 1) to i32))
2300     %list = bitcast i8* %mallocall to %list_item**
2301     store %list_item* null, %list_item** %list, align 8
2302     store %list_item** %list, %list_item*** %res, align 8
2303     store i32 0, i32* %for_index, align 4
2304     store i8 0, i8* %item, align 1
2305     br label %while
2306
2307 while:                                ; preds = %merge9, %entry
2308     %for_index36 = load i32, i32* %for_index, align 4
2309     %items37 = load %list_item**, %list_item*** %items2, align 8
2310     %i1ist38 = load %list_item*, %list_item** %items37, align 8
2311     %length39 = call i32 @list_length(%list_item* %i1ist38, i32 0)
2312     %tmp40 = icmp slt i32 %for_index36, %length39
2313     br i1 %tmp40, label %while_body45, label %merge
2314
2315 merge:                                ; preds = %while
2316     store i32 0, i32* %for_index41, align 4
2317     store i1 false, i1* %r, align 1
2318     br label %while42
2319
2320 while_body:                            ; preds = %while
2321     %items3 = load %list_item**, %list_item*** %items2, align 8
2322     %i1ist = load %list_item*, %list_item** %items3, align 8

```

```

2322 %for_index4 = load i32, i32* %for_index, align 4
2323 %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
      for_index4)
2324 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
      i32 0, i32 0
2325 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2326 %data = load i8, i8* %data_ptr, align 1
2327 store i8 %data, i8* %item, align 1
2328 %for_index5 = load i32, i32* %for_index, align 4
2329 %tmp = add i32 %for_index5, 1
2330 store i32 %tmp, i32* %for_index, align 4
2331 %res6 = load %list_item**, %list_item*** %res, align 8
2332 %list_ptr = load %list_item*, %list_item** %res6, align 8
2333 %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2334 %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
      false, i32 %length)
2335 store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
2336 store i32 0, i32* %for_index7, align 4
2337 store i8 0, i8* %ref, align 1
2338 br label %while8
2339
2340 while8:                                ; preds = %merge23, %
      while_body
2341 %for_index31 = load i32, i32* %for_index7, align 4
2342 %l32 = load %list_item**, %list_item*** %l1, align 8
2343 %ilist33 = load %list_item*, %list_item** %l32, align 8
2344 %length34 = call i32 @list_length(%list_item* %ilist33, i32 0)
2345 %tmp35 = icmp slt i32 %for_index31, %length34
2346 br i1 %tmp35, label %while_body10, label %merge9
2347
2348 merge9:                                ; preds = %while8
      br label %while
2349
2350
2351 while_body10:                           ; preds = %while8
2352 %l11 = load %list_item**, %list_item*** %l1, align 8
2353 %ilist12 = load %list_item*, %list_item** %l11, align 8
2354 %for_index13 = load i32, i32* %for_index7, align 4
2355 %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
      for_index13)
2356 %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
      _result14, i32 0, i32 0
2357 %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
2358 %data17 = load i8, i8* %data_ptr16, align 1
2359 store i8 %data17, i8* %ref, align 1
2360 %for_index18 = load i32, i32* %for_index7, align 4
2361 %tmp19 = add i32 %for_index18, 1
2362 store i32 %tmp19, i32* %for_index7, align 4
2363 %ref20 = load i8, i8* %ref, align 1
2364 %item21 = load i8, i8* %item, align 1
2365 %tmp22 = icmp eq i8 %ref20, %item21
2366 br i1 %tmp22, label %then, label %else
2367
2368 merge23:                                ; preds = %else, %then
      br label %while8
2369
2370
2371 then:                                    ; preds = %while_body10
2372 %res24 = load %list_item**, %list_item*** %res, align 8
2373 %ilist25 = load %list_item*, %list_item** %res24, align 8
2374 %res26 = load %list_item**, %list_item*** %res, align 8

```

```

2375 %ilist27 = load %list_item*, %list_item** %res26, align 8
2376 %length28 = call i32 @list_length(%list_item* %ilist27, i32 0)
2377 %tmp29 = sub i32 %length28, 1
2378 %result = call %list_item* @list_access(%list_item* %ilist25, i32 %tmp29)
2379 %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
2380 %allocaall30 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
    i1* null, i32 1) to i32))
2381 %copy_ptr = bitcast i8* %allocaall30 to i1*
2382 store i1 true, i1* %copy_ptr, align 1
2383 %ccopy = bitcast i1* %copy_ptr to i8*
2384 store i8* %ccopy, i8** %data_ptpt, align 8
2385 br label %merge23
2386
2387 else:                                     ; preds = %while_body10
2388     br label %merge23
2389
2390 while42:                                   ; preds = %merge56, %merge
2391     %for_index59 = load i32, i32* %for_index41, align 4
2392     %res60 = load %list_item**, %list_item*** %res, align 8
2393     %ilist61 = load %list_item*, %list_item** %res60, align 8
2394     %length62 = call i32 @list_length(%list_item* %ilist61, i32 0)
2395     %tmp63 = icmp slt i32 %for_index59, %length62
2396     br i1 %tmp63, label %while_body44, label %merge43
2397
2398 merge43:                                   ; preds = %while42
2399     ret i1 true
2400
2401 while_body44:                               ; preds = %while42
2402     %res45 = load %list_item**, %list_item*** %res, align 8
2403     %ilist46 = load %list_item*, %list_item** %res45, align 8
2404     %for_index47 = load i32, i32* %for_index41, align 4
2405     %_result48 = call %list_item* @list_access(%list_item* %ilist46, i32 %
        for_index47)
2406     %data_ptr_ptr49 = getelementptr inbounds %list_item, %list_item* %
        _result48, i32 0, i32 0
2407     %data_ptr50 = load i8*, i8** %data_ptr_ptr49, align 8
2408     %cast_data_ptr = bitcast i8* %data_ptr50 to i1*
2409     %data51 = load i1, i1* %cast_data_ptr, align 1
2410     store i1 %data51, i1* %r, align 1
2411     %for_index52 = load i32, i32* %for_index41, align 4
2412     %tmp53 = add i32 %for_index52, 1
2413     store i32 %tmp53, i32* %for_index41, align 4
2414     %r54 = load i1, i1* %r, align 1
2415     %tmp55 = xor i1 %r54, true
2416     br i1 %tmp55, label %then57, label %else58
2417
2418 merge56:                                   ; preds = %else58
2419     br label %while42
2420
2421 then57:                                     ; preds = %while_body44
2422     ret i1 false
2423
2424 else58:                                     ; preds = %while_body44
2425     br label %merge56
2426 }
2427
2428 define i1 @contains_string_string(%list_item** %l, %list_item** %items) {
2429 entry:

```



```

2430 %r = alloca i1, align 1
2431 %for_index41 = alloca i32, align 4
2432 %ref = alloca i8*, align 8
2433 %for_index7 = alloca i32, align 4
2434 %item = alloca i8*, align 8
2435 %for_index = alloca i32, align 4
2436 %res = alloca %list_item**, align 8
2437 %l1 = alloca %list_item**, align 8
2438 store %list_item** %l, %list_item*** %l1, align 8
2439 %items2 = alloca %list_item**, align 8
2440 store %list_item** %items, %list_item*** %items2, align 8
2441 %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
2442 %list = bitcast i8* %mallocall to %list_item**
2443 store %list_item* null, %list_item** %list, align 8
2444 store %list_item** %list, %list_item*** %res, align 8
2445 store i32 0, i32* %for_index, align 4
2446 store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.8, i32 0,
    i32 0), i8** %item, align 8
2447 br label %while
2448
2449 while:                                     ; preds = %merge9, %entry
2450 %for_index36 = load i32, i32* %for_index, align 4
2451 %items37 = load %list_item**, %list_item*** %items2, align 8
2452 %ilist38 = load %list_item*, %list_item** %items37, align 8
2453 %length39 = call i32 @list_length(%list_item* %ilist38, i32 0)
2454 %tmp40 = icmp slt i32 %for_index36, %length39
2455 br i1 %tmp40, label %while_body, label %merge
2456
2457 merge:                                     ; preds = %while
2458 store i32 0, i32* %for_index41, align 4
2459 store i1 false, i1* %r, align 1
2460 br label %while42
2461
2462 while_body:                               ; preds = %while
2463 %items3 = load %list_item**, %list_item*** %items2, align 8
2464 %ilist = load %list_item*, %list_item** %items3, align 8
2465 %for_index4 = load i32, i32* %for_index, align 4
2466 %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
2467 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
2468 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2469 %cast_data_ptr = bitcast i8* %data_ptr to i8**
2470 %data = load i8*, i8** %cast_data_ptr, align 8
2471 store i8* %data, i8** %item, align 8
2472 %for_index5 = load i32, i32* %for_index, align 4
2473 %tmp = add i32 %for_index5, 1
2474 store i32 %tmp, i32* %for_index, align 4
2475 %res6 = load %list_item**, %list_item*** %res, align 8
2476 %list_ptr = load %list_item*, %list_item** %res6, align 8
2477 %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2478 %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
2479 store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
2480 store i32 0, i32* %for_index7, align 4
2481 store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.9, i32 0,
    i32 0), i8** %ref, align 8
2482 br label %while8

```

```

2483
2484 while8:                                     ; preds = %merge23, %
    while_body
2485     %for_index31 = load i32, i32* %for_index7, align 4
2486     %l32 = load %list_item**, %list_item*** %l1, align 8
2487     %ilist33 = load %list_item*, %list_item** %l32, align 8
2488     %length34 = call i32 @list_length(%list_item* %ilist33, i32 0)
2489     %tmp35 = icmp slt i32 %for_index31, %length34
2490     br i1 %tmp35, label %while_body10, label %merge9
2491
2492 merge9:                                     ; preds = %while8
2493     br label %while
2494
2495 while_body10:                             ; preds = %while8
2496     %l11 = load %list_item**, %list_item*** %l1, align 8
2497     %ilist12 = load %list_item*, %list_item** %l11, align 8
2498     %for_index13 = load i32, i32* %for_index7, align 4
2499     %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
        for_index13)
2500     %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
        _result14, i32 0, i32 0
2501     %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
2502     %cast_data_ptr17 = bitcast i8* %data_ptr16 to i8**
2503     %data18 = load i8*, i8** %cast_data_ptr17, align 8
2504     store i8* %data18, i8** %ref, align 8
2505     %for_index19 = load i32, i32* %for_index7, align 4
2506     %tmp20 = add i32 %for_index19, 1
2507     store i32 %tmp20, i32* %for_index7, align 4
2508     %ref21 = load i8*, i8** %ref, align 8
2509     %item22 = load i8*, i8** %item, align 8
2510     %strcmp_eq = call i1 @strcmp_function(i8* %ref21, i8* %item22)
2511     br i1 %strcmp_eq, label %then, label %else
2512
2513 merge23:                                   ; preds = %else, %then
2514     br label %while8
2515
2516 then:                                     ; preds = %while_body10
2517     %res24 = load %list_item**, %list_item*** %res, align 8
2518     %ilist25 = load %list_item*, %list_item** %res24, align 8
2519     %res26 = load %list_item**, %list_item*** %res, align 8
2520     %ilist27 = load %list_item*, %list_item** %res26, align 8
2521     %length28 = call i32 @list_length(%list_item* %ilist27, i32 0)
2522     %tmp29 = sub i32 %length28, 1
2523     %result = call %list_item* @list_access(%list_item* %ilist25, i32 %tmp29)
2524     %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
        0, i32 0
2525     %mallocall30 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
        i1* null, i32 1) to i32))
2526     %copy_ptr = bitcast i8* %mallocall30 to i1*
2527     store i1 true, i1* %copy_ptr, align 1
2528     %ccopy = bitcast i1* %copy_ptr to i8*
2529     store i8* %ccopy, i8** %data_ptpt, align 8
2530     br label %merge23
2531
2532 else:                                     ; preds = %while_body10
2533     br label %merge23
2534
2535 while42:                                   ; preds = %merge57, %merge
2536     %for_index60 = load i32, i32* %for_index41, align 4

```

```

2537 %res61 = load %list_item**, %list_item*** %res, align 8
2538 %ilist62 = load %list_item*, %list_item** %res61, align 8
2539 %length63 = call i32 @list_length(%list_item* %ilist62, i32 0)
2540 %tmp64 = icmp slt i32 %for_index60, %length63
2541 br i1 %tmp64, label %while_body44, label %merge43
2542
2543 merge43:                                ; preds = %while42
2544     ret i1 true
2545
2546 while_body44:                            ; preds = %while42
2547 %res45 = load %list_item**, %list_item*** %res, align 8
2548 %ilist46 = load %list_item*, %list_item** %res45, align 8
2549 %for_index47 = load i32, i32* %for_index41, align 4
2550 %_result48 = call %list_item* @list_access(%list_item* %ilist46, i32 %
    for_index47)
2551 %data_ptr_ptr49 = getelementptr inbounds %list_item, %list_item* %
    _result48, i32 0, i32 0
2552 %data_ptr50 = load i8*, i8** %data_ptr_ptr49, align 8
2553 %cast_data_ptr51 = bitcast i8* %data_ptr50 to i1*
2554 %data52 = load i1, i1* %cast_data_ptr51, align 1
2555 store i1 %data52, i1* %r, align 1
2556 %for_index53 = load i32, i32* %for_index41, align 4
2557 %tmp54 = add i32 %for_index53, 1
2558 store i32 %tmp54, i32* %for_index41, align 4
2559 %r55 = load i1, i1* %r, align 1
2560 %tmp56 = xor i1 %r55, true
2561 br i1 %tmp56, label %then58, label %else59
2562
2563 merge57:                                ; preds = %else59
2564     br label %while42
2565
2566 then58:                                  ; preds = %while_body44
2567     ret i1 false
2568
2569 else59:                                  ; preds = %while_body44
2570     br label %merge57
2571 }
2572
2573 define i1 @contains_int_int(%list_item** %l, %list_item** %items) {
2574 entry:
2575     %r = alloca i1, align 1
2576     %for_index42 = alloca i32, align 4
2577     %ref = alloca i32, align 4
2578     %for_index7 = alloca i32, align 4
2579     %item = alloca i32, align 4
2580     %for_index = alloca i32, align 4
2581     %res = alloca %list_item**, align 8
2582     %l1 = alloca %list_item**, align 8
2583     store %list_item** %l, %list_item*** %l1, align 8
2584     %items2 = alloca %list_item**, align 8
2585     store %list_item** %items, %list_item*** %items2, align 8
2586     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))
2587     %list = bitcast i8* %mallocall to %list_item**
2588     store %list_item* null, %list_item** %list, align 8
2589     store %list_item** %list, %list_item*** %res, align 8
2590     store i32 0, i32* %for_index, align 4
2591     store i32 0, i32* %item, align 4
2592     br label %while

```

```

2593
2594 while:                                     ; preds = %merge9, %entry
2595     %for_index37 = load i32, i32* %for_index, align 4
2596     %items38 = load %list_item**, %list_item*** %items2, align 8
2597     %ilist39 = load %list_item*, %list_item** %items38, align 8
2598     %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
2599     %tmp41 = icmp slt i32 %for_index37, %length40
2600     br i1 %tmp41, label %while_body, label %merge
2601
2602 merge:                                     ; preds = %while
2603     store i32 0, i32* %for_index42, align 4
2604     store i1 false, i1* %r, align 1
2605     br label %while43
2606
2607 while_body:                               ; preds = %while
2608     %items3 = load %list_item**, %list_item*** %items2, align 8
2609     %ilist = load %list_item*, %list_item** %items3, align 8
2610     %for_index4 = load i32, i32* %for_index, align 4
2611     %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
        for_index4)
2612     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
        i32 0, i32 0
2613     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2614     %cast_data_ptr = bitcast i8* %data_ptr to i32*
2615     %data = load i32, i32* %cast_data_ptr, align 4
2616     store i32 %data, i32* %item, align 4
2617     %for_index5 = load i32, i32* %for_index, align 4
2618     %tmp = add i32 %for_index5, 1
2619     store i32 %tmp, i32* %for_index, align 4
2620     %res6 = load %list_item**, %list_item*** %res, align 8
2621     %list_ptr = load %list_item*, %list_item** %res6, align 8
2622     %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2623     %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
        false, i32 %length)
2624     store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
2625     store i32 0, i32* %for_index7, align 4
2626     store i32 0, i32* %ref, align 4
2627     br label %while8
2628
2629 while8:                                   ; preds = %merge24, %
        while_body
2630     %for_index32 = load i32, i32* %for_index7, align 4
2631     %l33 = load %list_item**, %list_item*** %l1, align 8
2632     %ilist34 = load %list_item*, %list_item** %l33, align 8
2633     %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
2634     %tmp36 = icmp slt i32 %for_index32, %length35
2635     br i1 %tmp36, label %while_body10, label %merge9
2636
2637 merge9:                                   ; preds = %while8
2638     br label %while
2639
2640 while_body10:                             ; preds = %while8
2641     %l11 = load %list_item**, %list_item*** %l1, align 8
2642     %ilist12 = load %list_item*, %list_item** %l11, align 8
2643     %for_index13 = load i32, i32* %for_index7, align 4
2644     %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
        for_index13)
2645     %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
        _result14, i32 0, i32 0

```

```

2646 %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
2647 %cast_data_ptr17 = bitcast i8* %data_ptr16 to i32*
2648 %data18 = load i32, i32* %cast_data_ptr17, align 4
2649 store i32 %data18, i32* %ref, align 4
2650 %for_index19 = load i32, i32* %for_index7, align 4
2651 %tmp20 = add i32 %for_index19, 1
2652 store i32 %tmp20, i32* %for_index7, align 4
2653 %ref21 = load i32, i32* %ref, align 4
2654 %item22 = load i32, i32* %item, align 4
2655 %tmp23 = icmp eq i32 %ref21, %item22
2656 br i1 %tmp23, label %then, label %else
2657
2658 merge24:                                ; preds = %else, %then
2659     br label %while8
2660
2661 then:                                    ; preds = %while_body10
2662     %res25 = load %list_item**, %list_item*** %res, align 8
2663     %ilist26 = load %list_item**, %list_item** %res25, align 8
2664     %res27 = load %list_item**, %list_item*** %res, align 8
2665     %ilist28 = load %list_item**, %list_item** %res27, align 8
2666     %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
2667     %tmp30 = sub i32 %length29, 1
2668     %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
2669     %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
        0, i32 0
2670     %mallocall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
        i1* null, i32 1) to i32))
2671     %copy_ptr = bitcast i8* %mallocall31 to i1*
2672     store i1 true, i1* %copy_ptr, align 1
2673     %ccopy = bitcast i1* %copy_ptr to i8*
2674     store i8* %ccopy, i8** %data_ptpt, align 8
2675     br label %merge24
2676
2677 else:                                    ; preds = %while_body10
2678     br label %merge24
2679
2680 while43:                                ; preds = %merge58, %merge
2681     %for_index61 = load i32, i32* %for_index42, align 4
2682     %res62 = load %list_item**, %list_item*** %res, align 8
2683     %ilist63 = load %list_item**, %list_item** %res62, align 8
2684     %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
2685     %tmp65 = icmp slt i32 %for_index61, %length64
2686     br i1 %tmp65, label %while_body45, label %merge44
2687
2688 merge44:                                ; preds = %while43
2689     ret i1 true
2690
2691 while_body45:                            ; preds = %while43
2692     %res46 = load %list_item**, %list_item*** %res, align 8
2693     %ilist47 = load %list_item**, %list_item** %res46, align 8
2694     %for_index48 = load i32, i32* %for_index42, align 4
2695     %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
        for_index48)
2696     %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
        _result49, i32 0, i32 0
2697     %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
2698     %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
2699     %data53 = load i1, i1* %cast_data_ptr52, align 1
2700     store i1 %data53, i1* %r, align 1

```

```

2701 %for_index54 = load i32, i32* %for_index42, align 4
2702 %tmp55 = add i32 %for_index54, 1
2703 store i32 %tmp55, i32* %for_index42, align 4
2704 %r56 = load i1, i1* %r, align 1
2705 %tmp57 = xor i1 %r56, true
2706 br i1 %tmp57, label %then59, label %else60
2707
2708 merge58:                                ; preds = %else60
2709     br label %while43
2710
2711 then59:                                ; preds = %while_body45
2712     ret i1 false
2713
2714 else60:                                ; preds = %while_body45
2715     br label %merge58
2716 }
2717
2718 define i8* @join_string_string(%list_item** %text_list, i8* %connector) {
2719 entry:
2720     %index = alloca i32, align 4
2721     %for_index = alloca i32, align 4
2722     %list_length = alloca i32, align 4
2723     %res = alloca i8*, align 8
2724     %text_list1 = alloca %list_item**, align 8
2725     store %list_item** %text_list, %list_item*** %text_list1, align 8
2726     %connector2 = alloca i8*, align 8
2727     store i8* %connector, i8** %connector2, align 8
2728     store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.10, i32 0,
2729         i32 0), i8** %res, align 8
2729     %text_list3 = load %list_item**, %list_item*** %text_list1, align 8
2730     %ilist = load %list_item**, %list_item*** %text_list3, align 8
2731     %length = call i32 @list_length(%list_item* %ilist, i32 0)
2732     store i32 %length, i32* %list_length, align 4
2733     store i32 0, i32* %for_index, align 4
2734     store i32 0, i32* %index, align 4
2735     br label %while
2736
2737 while:                                ; preds = %while_body, %
2738     entry
2739     %for_index31 = load i32, i32* %for_index, align 4
2740     %list_length32 = load i32, i32* %list_length, align 4
2741     %tmp33 = sub i32 %list_length32, 1
2742     %mallocall34 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
2743         *, i1** null, i32 1) to i32))
2744     %head_ptr_ptr35 = bitcast i8* %mallocall34 to %list_item**
2745     store %list_item* null, %list_item** %head_ptr_ptr35, align 8
2746     %range_list36 = call %list_item** @range_function(i32 0, i32 %tmp33, %
2747         list_item** %head_ptr_ptr35, i32 0)
2748     %ilist37 = load %list_item**, %list_item*** %range_list36, align 8
2749     %length38 = call i32 @list_length(%list_item* %ilist37, i32 0)
2750     %tmp39 = icmp slt i32 %for_index31, %length38
2751     br i1 %tmp39, label %while_body, label %merge
2752
2753 merge:                                ; preds = %while
2754     %res40 = load i8*, i8** %res, align 8
2755     %text_list41 = load %list_item**, %list_item*** %text_list1, align 8
2756     %ilist42 = load %list_item**, %list_item*** %text_list41, align 8
2757     %list_length43 = load i32, i32* %list_length, align 4
2758     %tmp44 = sub i32 %list_length43, 1

```

```

2756 %_result45 = call %list_item* @list_access(%list_item* %ilist42, i32 %
      tmp44)
2757 %data_ptr_ptr46 = getelementptr inbounds %list_item, %list_item* %
      _result45, i32 0, i32 0
2758 %data_ptr47 = load i8*, i8** %data_ptr_ptr46, align 8
2759 %cast_data_ptr48 = bitcast i8* %data_ptr47 to i8**
2760 %data49 = load i8*, i8** %cast_data_ptr48, align 8
2761 %length50 = call i32 @string_length(i8* %res40, i32 0)
2762 %length51 = call i32 @string_length(i8* %data49, i32 0)
2763 %new_length52 = add i32 %length50, %length51
2764 %new_length_nul53 = add i32 %new_length52, 1
2765 %mallocsize54 = mul i32 %new_length_nul53, ptrtoint (i8* getelementptr (i8
      , i8* null, i32 1) to i32)
2766 %new_string56 = tail call i8* @malloc(i32 %mallocsize54)
2767 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string56, i8* %res40, i32 %
      length50, i1 true)
2768 %new_spot57 = getelementptr i8, i8* %new_string56, i32 %length50
2769 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot57, i8* %data49, i32 %
      length51, i1 true)
2770 %string_term58 = getelementptr i8, i8* %new_string56, i32 %new_length52
2771 store i8 0, i8* %string_term58, align 1
2772 store i8* %new_string56, i8** %res, align 8
2773 %res59 = load i8*, i8** %res, align 8
2774 ret i8* %res59
2775
2776 while_body:                                ; preds = %while
2777 %list_length4 = load i32, i32* %list_length, align 4
2778 %tmp = sub i32 %list_length4, 1
2779 %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
      i1** null, i32 1) to i32))
2780 %head_ptr_ptr = bitcast i8* %malloccall to %list_item**
2781 store %list_item* null, %list_item** %head_ptr_ptr, align 8
2782 %range_list = call %list_item** @range_function(i32 0, i32 %tmp, %
      list_item** %head_ptr_ptr, i32 0)
2783 %ilist5 = load %list_item*, %list_item** %range_list, align 8
2784 %for_index6 = load i32, i32* %for_index, align 4
2785 %_result = call %list_item* @list_access(%list_item* %ilist5, i32 %
      for_index6)
2786 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
      i32 0, i32 0
2787 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2788 %cast_data_ptr = bitcast i8* %data_ptr to i32*
2789 %data = load i32, i32* %cast_data_ptr, align 4
2790 store i32 %data, i32* %index, align 4
2791 %for_index7 = load i32, i32* %for_index, align 4
2792 %tmp8 = add i32 %for_index7, 1
2793 store i32 %tmp8, i32* %for_index, align 4
2794 %res9 = load i8*, i8** %res, align 8
2795 %text_list10 = load %list_item**, %list_item** %text_list1, align 8
2796 %ilist11 = load %list_item*, %list_item** %text_list10, align 8
2797 %index12 = load i32, i32* %index, align 4
2798 %_result13 = call %list_item* @list_access(%list_item* %ilist11, i32 %
      index12)
2799 %data_ptr_ptr14 = getelementptr inbounds %list_item, %list_item* %
      _result13, i32 0, i32 0
2800 %data_ptr15 = load i8*, i8** %data_ptr_ptr14, align 8
2801 %cast_data_ptr16 = bitcast i8* %data_ptr15 to i8**
2802 %data17 = load i8*, i8** %cast_data_ptr16, align 8
2803 %length18 = call i32 @string_length(i8* %res9, i32 0)

```

```

2804 %length19 = call i32 @string_length(i8* %data17, i32 0)
2805 %new_length = add i32 %length18, %length19
2806 %new_length_nul = add i32 %new_length, 1
2807 %mallocsize = mul i32 %new_length_nul, ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32)
2808 %new_string = tail call i8* @malloc(i32 %mallocsize)
2809 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %res9, i32 %
    length18, i1 true)
2810 %new_spot = getelementptr i8, i8* %new_string, i32 %length18
2811 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot, i8* %data17, i32 %
    length19, i1 true)
2812 %string_term = getelementptr i8, i8* %new_string, i32 %new_length
2813 store i8 0, i8* %string_term, align 1
2814 %connector21 = load i8*, i8** %connector2, align 8
2815 %length22 = call i32 @string_length(i8* %new_string, i32 0)
2816 %length23 = call i32 @string_length(i8* %connector21, i32 0)
2817 %new_length24 = add i32 %length22, %length23
2818 %new_length_nul25 = add i32 %new_length24, 1
2819 %mallocsize26 = mul i32 %new_length_nul25, ptrtoint (i8* getelementptr (i8
    , i8* null, i32 1) to i32)
2820 %new_string28 = tail call i8* @malloc(i32 %mallocsize26)
2821 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string28, i8* %new_string,
    i32 %length22, i1 true)
2822 %new_spot29 = getelementptr i8, i8* %new_string28, i32 %length22
2823 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot29, i8* %connector21,
    i32 %length23, i1 true)
2824 %string_term30 = getelementptr i8, i8* %new_string28, i32 %new_length24
2825 store i8 0, i8* %string_term30, align 1
2826 store i8* %new_string28, i8** %res, align 8
2827 br label %while
2828 }
2829
2830 define %list_item** @split(i8* %text, i8 %separator) {
2831 entry:
2832   %right = alloca i32, align 4
2833   %left = alloca i32, align 4
2834   %text_length = alloca i32, align 4
2835   %result = alloca %list_item**, align 8
2836   %text1 = alloca i8*, align 8
2837   store i8* %text, i8** %text1, align 8
2838   %separator2 = alloca i8, align 1
2839   store i8 %separator, i8* %separator2, align 1
2840   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
2841   %list = bitcast i8* %malloccall to %list_item**
2842   store %list_item* null, %list_item** %list, align 8
2843   store %list_item** %list, %list_item*** %result, align 8
2844   %text3 = load i8*, i8** %text1, align 8
2845   %length = call i32 @string_length(i8* %text3, i32 0)
2846   store i32 %length, i32* %text_length, align 4
2847   store i32 0, i32* %left, align 4
2848   store i32 0, i32* %right, align 4
2849   br label %while
2850
2851 while:                                     ; preds = %merge7, %entry
2852   %right21 = load i32, i32* %right, align 4
2853   %text_length22 = load i32, i32* %text_length, align 4
2854   %tmp23 = icmp slt i32 %right21, %text_length22
2855   br i1 %tmp23, label %while_body, label %merge

```



```

2856
2857 merge:                                ; preds = %while
2858   %result24 = load %list_item**, %list_item*** %result, align 8
2859   %list_ptr25 = load %list_item*, %list_item** %result24, align 8
2860   %text26 = load i8*, i8** %text1, align 8
2861   %left27 = load i32, i32* %left, align 4
2862   %get_char_ptr28 = getelementptr i8, i8* %text26, i32 %left27
2863   %left29 = load i32, i32* %left, align 4
2864   %right30 = load i32, i32* %right, align 4
2865   %subb31 = sub i32 %right30, %left29
2866   %length_w_nul32 = add i32 %subb31, 1
2867   %mallocsize33 = mul i32 %length_w_nul32, ptrtoint (i8* getelementptr (i8,
      i8* null, i32 1) to i32)
2868   %new_string35 = tail call i8* @malloc(i32 %mallocsize33)
2869   %string_term36 = getelementptr i8, i8* %new_string35, i32 %subb31
2870   store i8 0, i8* %string_term36, align 1
2871   call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string35, i8* %
      get_char_ptr28, i32 %subb31, i1 true)
2872   %length37 = call i32 @list_length(%list_item* %list_ptr25, i32 0)
2873   %list_ptr_ptr38 = call %list_item** @insert_string(%list_item** %result24,
      i8* %new_string35, i32 %length37)
2874   store %list_item** %list_ptr_ptr38, %list_item*** %result, align 8
2875   %result39 = load %list_item**, %list_item*** %result, align 8
2876   ret %list_item** %result39
2877
2878 while_body:                            ; preds = %while
2879   %text4 = load i8*, i8** %text1, align 8
2880   %right5 = load i32, i32* %right, align 4
2881   %get_char_ptr = getelementptr i8, i8* %text4, i32 %right5
2882   %get_char = load i8, i8* %get_char_ptr, align 1
2883   %separator6 = load i8, i8* %separator2, align 1
2884   %tmp = icmp eq i8 %get_char, %separator6
2885   br i1 %tmp, label %then, label %else
2886
2887 merge7:                                ; preds = %else, %then
2888   br label %while
2889
2890 then:                                  ; preds = %while_body
2891   %result8 = load %list_item**, %list_item*** %result, align 8
2892   %list_ptr = load %list_item*, %list_item** %result8, align 8
2893   %text9 = load i8*, i8** %text1, align 8
2894   %left10 = load i32, i32* %left, align 4
2895   %get_char_ptr11 = getelementptr i8, i8* %text9, i32 %left10
2896   %left12 = load i32, i32* %left, align 4
2897   %right13 = load i32, i32* %right, align 4
2898   %subb = sub i32 %right13, %left12
2899   %length_w_nul = add i32 %subb, 1
2900   %mallocsize = mul i32 %length_w_nul, ptrtoint (i8* getelementptr (i8, i8*
      null, i32 1) to i32)
2901   %new_string = tail call i8* @malloc(i32 %mallocsize)
2902   %string_term = getelementptr i8, i8* %new_string, i32 %subb
2903   store i8 0, i8* %string_term, align 1
2904   call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %get_char_ptr11,
      i32 %subb, i1 true)
2905   %length15 = call i32 @list_length(%list_item* %list_ptr, i32 0)
2906   %list_ptr_ptr = call %list_item** @insert_string(%list_item** %result8, i8
      * %new_string, i32 %length15)
2907   store %list_item** %list_ptr_ptr, %list_item*** %result, align 8
2908   %right16 = load i32, i32* %right, align 4

```

```

2909 %tmp17 = add i32 %right16, 1
2910 store i32 %tmp17, i32* %right, align 4
2911 %right18 = load i32, i32* %right, align 4
2912 store i32 %right18, i32* %left, align 4
2913 br label %merge7
2914
2915 else:                                     ; preds = %while_body
2916 %right19 = load i32, i32* %right, align 4
2917 %tmp20 = add i32 %right19, 1
2918 store i32 %tmp20, i32* %right, align 4
2919 br label %merge7
2920 }
2921
2922 define i8* @string_reverse(i8* %text) {
2923 entry:
2924 %index = alloca i32, align 4
2925 %string_length = alloca i32, align 4
2926 %res = alloca i8*, align 8
2927 %text1 = alloca i8*, align 8
2928 store i8* %text, i8** %text1, align 8
2929 store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.ll, i32 0,
    i32 0), i8** %res, align 8
2930 %text2 = load i8*, i8** %text1, align 8
2931 %length = call i32 @string_length(i8* %text2, i32 0)
2932 store i32 %length, i32* %string_length, align 4
2933 %string_length3 = load i32, i32* %string_length, align 4
2934 %tmp = sub i32 %string_length3, 1
2935 store i32 %tmp, i32* %index, align 4
2936 br label %while
2937
2938 while:                                     ; preds = %while_body, %
    entry
2939 %index10 = load i32, i32* %index, align 4
2940 %tmp11 = icmp sge i32 %index10, 0
2941 br i1 %tmp11, label %while_body, label %merge
2942
2943 merge:                                     ; preds = %while
2944 %res12 = load i8*, i8** %res, align 8
2945 ret i8* %res12
2946
2947 while_body:                               ; preds = %while
2948 %res4 = load i8*, i8** %res, align 8
2949 %text5 = load i8*, i8** %text1, align 8
2950 %index6 = load i32, i32* %index, align 4
2951 %get_char_ptr = getelementptr i8, i8* %text5, i32 %index6
2952 %get_char = load i8, i8* %get_char_ptr, align 1
2953 %length7 = call i32 @string_length(i8* %res4, i32 0)
2954 %new_length = add i32 %length7, 1
2955 %new_length_nul = add i32 %new_length, 1
2956 %mallocsize = mul i32 %new_length_nul, ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32)
2957 %new_string = tail call i8* @malloc(i32 %mallocsize)
2958 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %res4, i32 %
    length7, i1 true)
2959 %new_spot = getelementptr i8, i8* %new_string, i32 %length7
2960 store i8 %get_char, i8* %new_spot, align 1
2961 %string_term = getelementptr i8, i8* %new_string, i32 %new_length
2962 store i8 0, i8* %string_term, align 1
2963 store i8* %new_string, i8** %res, align 8

```

```

2964 %index8 = load i32, i32* %index, align 4
2965 %tmp9 = sub i32 %index8, 1
2966 store i32 %tmp9, i32* %index, align 4
2967 br label %while
2968 }
2969
2970 define il @startswith(i8* %text, i8 %s) {
2971 entry:
2972   %text1 = alloca i8*, align 8
2973   store i8* %text, i8** %text1, align 8
2974   %s2 = alloca i8, align 1
2975   store i8 %s, i8* %s2, align 1
2976   %s3 = load i8, i8* %s2, align 1
2977   %text4 = load i8*, i8** %text1, align 8
2978   %get_char_ptr = getelementptr i8, i8* %text4, i32 0
2979   %get_char = load i8, i8* %get_char_ptr, align 1
2980   %tmp = icmp eq i8 %s3, %get_char
2981   ret il %tmp
2982 }
2983
2984 define il @endswith(i8* %text, i8 %e) {
2985 entry:
2986   %string_length = alloca i32, align 4
2987   %text1 = alloca i8*, align 8
2988   store i8* %text, i8** %text1, align 8
2989   %e2 = alloca i8, align 1
2990   store i8 %e, i8* %e2, align 1
2991   %text3 = load i8*, i8** %text1, align 8
2992   %length = call i32 @string_length(i8* %text3, i32 0)
2993   store i32 %length, i32* %string_length, align 4
2994   %e4 = load i8, i8* %e2, align 1
2995   %text5 = load i8*, i8** %text1, align 8
2996   %string_length6 = load i32, i32* %string_length, align 4
2997   %tmp = sub i32 %string_length6, 1
2998   %get_char_ptr = getelementptr i8, i8* %text5, i32 %tmp
2999   %get_char = load i8, i8* %get_char_ptr, align 1
3000   %tmp7 = icmp eq i8 %e4, %get_char
3001   ret il %tmp7
3002 }
3003
3004 define %list_item** @string_to_list(i8* %text) {
3005 entry:
3006   %c = alloca i8, align 1
3007   %for_index = alloca i32, align 4
3008   %result = alloca %list_item**, align 8
3009   %text1 = alloca i8*, align 8
3010   store i8* %text, i8** %text1, align 8
3011   %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
3012     i1** null, i32 1) to i32))
3012   %list = bitcast i8* %mallocall to %list_item**
3013   store %list_item* null, %list_item** %list, align 8
3014   store %list_item** %list, %list_item*** %result, align 8
3015   store i32 0, i32* %for_index, align 4
3016   store i8 0, i8* %c, align 1
3017   br label %while
3018
3019 while:                                     ; preds = %while_body, %
3020   entry
3021   %for_index7 = load i32, i32* %for_index, align 4

```

```

3021 %text8 = load i8*, i8** %text1, align 8
3022 %length9 = call i32 @string_length(i8* %text8, i32 0)
3023 %tmp10 = icmp slt i32 %for_index7, %length9
3024 br i1 %tmp10, label %while_body, label %merge
3025
3026 merge:                                     ; preds = %while
3027 %result11 = load %list_item**, %list_item*** %result, align 8
3028 ret %list_item** %result11
3029
3030 while_body:                               ; preds = %while
3031 %text2 = load i8*, i8** %text1, align 8
3032 %for_index3 = load i32, i32* %for_index, align 4
3033 %get_char_ptr = getelementptr i8, i8* %text2, i32 %for_index3
3034 %get_char = load i8, i8* %get_char_ptr, align 1
3035 store i8 %get_char, i8* %c, align 1
3036 %for_index4 = load i32, i32* %for_index, align 4
3037 %tmp = add i32 %for_index4, 1
3038 store i32 %tmp, i32* %for_index, align 4
3039 %result5 = load %list_item**, %list_item*** %result, align 8
3040 %list_ptr = load %list_item*, %list_item** %result5, align 8
3041 %c6 = load i8, i8* %c, align 1
3042 %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
3043 %list_ptr_ptr = call %list_item** @insert_char(%list_item** %result5, i8 %
    c6, i32 %length)
3044 store %list_item** %list_ptr_ptr, %list_item*** %result, align 8
3045 br label %while
3046 }
3047
3048 define i8 @lower(i8 %c) {
3049 entry:
3050 %c_ref = alloca i8, align 1
3051 %for_index = alloca i32, align 4
3052 %index = alloca i32, align 4
3053 %c1 = alloca i8, align 1
3054 store i8 %c, i8* %c1, align 1
3055 store i32 -1, i32* %index, align 4
3056 store i32 0, i32* %for_index, align 4
3057 store i8 0, i8* %c_ref, align 1
3058 br label %while
3059
3060 while:                                     ; preds = %merge9, %entry
3061 %for_index24 = load i32, i32* %for_index, align 4
3062 %ASCII25 = load %list_item**, %list_item*** @ASCII, align 8
3063 %i1ist26 = load %list_item*, %list_item** %ASCII25, align 8
3064 %length = call i32 @list_length(%list_item* %i1ist26, i32 0)
3065 %tmp27 = icmp slt i32 %for_index24, %length
3066 br i1 %tmp27, label %while_body, label %merge
3067
3068 merge:                                     ; preds = %while
3069 %c28 = load i8, i8* %c1, align 1
3070 ret i8 %c28
3071
3072 while_body:                               ; preds = %while
3073 %ASCII = load %list_item**, %list_item*** @ASCII, align 8
3074 %i1ist = load %list_item*, %list_item** %ASCII, align 8
3075 %for_index2 = load i32, i32* %for_index, align 4
3076 %_result = call %list_item* @list_access(%list_item* %i1ist, i32 %
    for_index2)
3077 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,

```

```

    i32 0, i32 0
3078 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
3079 %data = load i8, i8* %data_ptr, align 1
3080 store i8 %data, i8* %c_ref, align 1
3081 %for_index3 = load i32, i32* %for_index, align 4
3082 %tmp = add i32 %for_index3, 1
3083 store i32 %tmp, i32* %for_index, align 4
3084 %index4 = load i32, i32* %index, align 4
3085 %tmp5 = add i32 %index4, 1
3086 store i32 %tmp5, i32* %index, align 4
3087 %c6 = load i8, i8* %c1, align 1
3088 %c_ref7 = load i8, i8* %c_ref, align 1
3089 %tmp8 = icmp eq i8 %c6, %c_ref7
3090 br i1 %tmp8, label %then, label %else23
3091
3092 merge9:                                ; preds = %else23, %
    merge12
3093 br label %while
3094
3095 then:                                    ; preds = %while_body
    %index10 = load i32, i32* %index, align 4
3096 %tmp11 = icmp slt i32 %index10, 26
3097 br i1 %tmp11, label %then13, label %else
3098
3099
3100 merge12:                                ; No predecessors!
    br label %merge9
3101
3102
3103 then13:                                  ; preds = %then
    %c14 = load i8, i8* %c1, align 1
3104 ret i8 %c14
3105
3106
3107 else:                                    ; preds = %then
    %ASCI115 = load %list_item**, %list_item** @ASCII, align 8
3108 %i16 = load %list_item*, %list_item* %ASCI115, align 8
3109 %index17 = load i32, i32* %index, align 4
3110 %tmp18 = sub i32 %index17, 26
3111 %_result19 = call %list_item* @list_access(%list_item* %i16, i32 %
    tmp18)
3112 %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
3113 %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
3114 %data22 = load i8, i8* %data_ptr21, align 1
3115 ret i8 %data22
3116
3117
3118 else23:                                  ; preds = %while_body
    br label %merge9
3119
3120 }
3121
3122 define i8 @upper(i8 %c) {
3123 entry:
3124 %c_ref = alloca i8, align 1
3125 %for_index = alloca i32, align 4
3126 %index = alloca i32, align 4
3127 %c1 = alloca i8, align 1
3128 store i8 %c, i8* %c1, align 1
3129 store i32 -1, i32* %index, align 4
3130 store i32 0, i32* %for_index, align 4
3131 store i8 0, i8* %c_ref, align 1
3132 br label %while

```

```

3133
3134 while:                                     ; preds = %merge9, %entry
3135     %for_index24 = load i32, i32* %for_index, align 4
3136     %ASCII25 = load %list_item**, %list_item*** @ASCII, align 8
3137     %ilist26 = load %list_item*, %list_item** %ASCII25, align 8
3138     %length = call i32 @list_length(%list_item* %ilist26, i32 0)
3139     %tmp27 = icmp slt i32 %for_index24, %length
3140     br i1 %tmp27, label %while_body, label %merge
3141
3142 merge:                                     ; preds = %while
3143     %c28 = load i8, i8* %c1, align 1
3144     ret i8 %c28
3145
3146 while_body:                               ; preds = %while
3147     %ASCII = load %list_item**, %list_item*** @ASCII, align 8
3148     %ilist = load %list_item*, %list_item** %ASCII, align 8
3149     %for_index2 = load i32, i32* %for_index, align 4
3150     %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
        for_index2)
3151     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
        i32 0, i32 0
3152     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
3153     %data = load i8, i8* %data_ptr, align 1
3154     store i8 %data, i8* %c_ref, align 1
3155     %for_index3 = load i32, i32* %for_index, align 4
3156     %tmp = add i32 %for_index3, 1
3157     store i32 %tmp, i32* %for_index, align 4
3158     %index4 = load i32, i32* %index, align 4
3159     %tmp5 = add i32 %index4, 1
3160     store i32 %tmp5, i32* %index, align 4
3161     %c6 = load i8, i8* %c1, align 1
3162     %c_ref7 = load i8, i8* %c_ref, align 1
3163     %tmp8 = icmp eq i8 %c6, %c_ref7
3164     br i1 %tmp8, label %then, label %else23
3165
3166 merge9:                                   ; preds = %else23, %
        merge12
3167     br label %while
3168
3169 then:                                     ; preds = %while_body
3170     %index10 = load i32, i32* %index, align 4
3171     %tmp11 = icmp sgt i32 %index10, 25
3172     br i1 %tmp11, label %then13, label %else
3173
3174 merge12:                                  ; No predecessors!
3175     br label %merge9
3176
3177 then13:                                   ; preds = %then
3178     %c14 = load i8, i8* %c1, align 1
3179     ret i8 %c14
3180
3181 else:                                     ; preds = %then
3182     %ASCII15 = load %list_item**, %list_item*** @ASCII, align 8
3183     %ilist16 = load %list_item*, %list_item** %ASCII15, align 8
3184     %index17 = load i32, i32* %index, align 4
3185     %tmp18 = add i32 %index17, 26
3186     %_result19 = call %list_item* @list_access(%list_item* %ilist16, i32 %
        tmp18)
3187     %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %

```

```

    _result19, i32 0, i32 0
3188 %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
3189 %data22 = load i8, i8* %data_ptr21, align 1
3190 ret i8 %data22
3191
3192 else23:                                     ; preds = %while_body
3193     br label %merge9
3194 }
3195
3196 define i1 @strcmp(i8* %str1, i8* %str2) {
3197 entry:
3198     %c2 = alloca i8, align 1
3199     %c1 = alloca i8, align 1
3200     %i = alloca i32, align 4
3201     %str11 = alloca i8*, align 8
3202     store i8* %str1, i8** %str11, align 8
3203     %str22 = alloca i8*, align 8
3204     store i8* %str2, i8** %str22, align 8
3205     %str13 = load i8*, i8** %str11, align 8
3206     %length = call i32 @string_length(i8* %str13, i32 0)
3207     %str24 = load i8*, i8** %str22, align 8
3208     %length5 = call i32 @string_length(i8* %str24, i32 0)
3209     %tmp = icmp ne i32 %length, %length5
3210     br i1 %tmp, label %then, label %else
3211
3212 merge:                                     ; preds = %else
3213     store i32 0, i32* %i, align 4
3214     br label %while
3215
3216 then:                                     ; preds = %entry
3217     ret i1 false
3218
3219 else:                                     ; preds = %entry
3220     br label %merge
3221
3222 while:                                     ; preds = %merge16, %merge
3223     %i21 = load i32, i32* %i, align 4
3224     %str122 = load i8*, i8** %str11, align 8
3225     %length23 = call i32 @string_length(i8* %str122, i32 0)
3226     %tmp24 = icmp slt i32 %i21, %length23
3227     br i1 %tmp24, label %while_body, label %merge6
3228
3229 merge6:                                     ; preds = %while
3230     ret i1 true
3231
3232 while_body:                               ; preds = %while
3233     %str17 = load i8*, i8** %str11, align 8
3234     %i8 = load i32, i32* %i, align 4
3235     %get_char_ptr = getelementptr i8, i8* %str17, i32 %i8
3236     %get_char = load i8, i8* %get_char_ptr, align 1
3237     store i8 %get_char, i8* %c1, align 1
3238     %str29 = load i8*, i8** %str22, align 8
3239     %i10 = load i32, i32* %i, align 4
3240     %get_char_ptr11 = getelementptr i8, i8* %str29, i32 %i10
3241     %get_char12 = load i8, i8* %get_char_ptr11, align 1
3242     store i8 %get_char12, i8* %c2, align 1
3243     %c113 = load i8, i8* %c1, align 1
3244     %c214 = load i8, i8* %c2, align 1
3245     %tmp15 = icmp ne i8 %c113, %c214

```

```

3246     br il %tmp15, label %then17, label %else18
3247
3248 merge16:                                     ; preds = %else18
3249     %i19 = load i32, i32* %i, align 4
3250     %tmp20 = add i32 %i19, 1
3251     store i32 %tmp20, i32* %i, align 4
3252     br label %while
3253
3254 then17:                                       ; preds = %while_body
3255     ret il false
3256
3257 else18:                                       ; preds = %while_body
3258     br label %merge16
3259 }
3260
3261 declare noalias i8* @malloc(i32)
3262
3263 define %list_item* @list_access(%list_item* %0, i32 %1) {
3264 entry:
3265     %is_zero = icmp eq i32 %1, 0
3266     br il %is_zero, label %then, label %else
3267
3268 then:   ; preds = %entry
3269     ret %list_item* %0
3270
3271 else:   ; preds = %entry
3272     %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
3273     1
3274     %next = load %list_item*, %list_item** %next_ptr, align 8
3275     %sub = sub i32 %1, 1
3276     %result = call %list_item* @list_access(%list_item* %next, i32 %sub)
3277     ret %list_item* %result
3278 }
3279
3280 define i32 @list_length(%list_item* %0, i32 %1) {
3281 entry:
3282     %ptr_is_null = icmp eq %list_item* %0, null
3283     br il %ptr_is_null, label %then, label %else
3284
3285 then:   ; preds = %entry
3286     ret i32 %1
3287
3288 else:   ; preds = %entry
3289     %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
3290     1
3291     %next = load %list_item*, %list_item** %next_ptr, align 8
3292     %add = add i32 %1, 1
3293     %result = call i32 @list_length(%list_item* %next, i32 %add)
3294     ret i32 %result
3295 }
3296
3297 define %list_item** @range_function(i32 %0, i32 %1, %list_item** %2, i32 %3)
3298 {
3299 entry:
3300     %is_last = icmp eq i32 %0, %1
3301     br il %is_last, label %then, label %else
3302
3303 then:   ; preds = %entry
3304     ret %list_item** %2

```



```

3302
3303 else:                                     ; preds = %entry
3304   %head_ptr_ptr = call @insert_int(%list_item** %2, i32 %0, i32
3305     %3)
3306   %next_s = add i32 1, %0
3307   %next_length = add i32 1, %3
3308   %4 = call %list_item** @range_function(i32 %next_s, i32 %1, %list_item** %
3309     head_ptr_ptr, i32 %next_length)
3310   ret %list_item** %4
3311 }
3312
3313 define %list_item** @insert_int(%list_item** %0, i32 %1, i32 %2) {
3314 entry:
3315   %list_ptr = load %list_item*, %list_item** %0, align 8
3316   %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
3317     i1** null, i32 1) to i32))
3318   %new_list_ptr_ptr = bitcast i8* %mallocall to %list_item**
3319   store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3320   %last_node_ptr_ptr = call %list_item** @list_copy_int(%list_item* %
3321     list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3322   %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3323   %temp = alloca %list_item, align 8
3324   %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3325   store %list_item* %new_list_ptr, %list_item** %next, align 8
3326   %mallocall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
3327     getelementptr (%list_item, %list_item* null, i32 1) to i32))
3328   %data_node = bitcast i8* %mallocall1 to %list_item*
3329   %mallocall2 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
3330     , i32* null, i32 1) to i32))
3331   %data = bitcast i8* %mallocall2 to i32*
3332   store i32 %1, i32* %data, align 4
3333   %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3334     i32 0
3335   %cast = bitcast i32* %data to i8*
3336   store i8* %cast, i8** %dat, align 8
3337   %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3338   %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
3339     1
3340   %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3341     i32 1
3342   %temp4 = load %list_item*, %list_item** %test, align 8
3343   store %list_item* %temp4, %list_item** %dat3, align 8
3344   store %list_item* %data_node, %list_item** %test, align 8
3345   %temp5 = load %list_item*, %list_item** %next, align 8
3346   store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3347   ret %list_item** %new_list_ptr_ptr
3348 }
3349
3350 define %list_item** @list_copy_int(%list_item* %0, i32 %1, %list_item** %2)
3351 {
3352 entry:
3353   %is_zero = icmp eq i32 %1, 0
3354   %ptr_is_null = icmp eq %list_item* %0, null
3355   %or_conds = or i1 %is_zero, %ptr_is_null
3356   br i1 %or_conds, label %then, label %else
3357
3358 then:                                     ; preds = %entry
3359   ret %list_item** %2
3360
3361 else:
3362   %list_ptr = load %list_item*, %list_item* %0, align 8
3363   %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
3364     i1** null, i32 1) to i32))
3365   %new_list_ptr_ptr = bitcast i8* %mallocall to %list_item**
3366   store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3367   %last_node_ptr_ptr = call %list_item** @list_copy_int(%list_item* %
3368     list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3369   %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3370   %temp = alloca %list_item, align 8
3371   %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3372   store %list_item* %new_list_ptr, %list_item** %next, align 8
3373   %mallocall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
3374     getelementptr (%list_item, %list_item* null, i32 1) to i32))
3375   %data_node = bitcast i8* %mallocall1 to %list_item*
3376   %mallocall2 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
3377     , i32* null, i32 1) to i32))
3378   %data = bitcast i8* %mallocall2 to i32*
3379   store i32 %1, i32* %data, align 4
3380   %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3381     i32 0
3382   %cast = bitcast i32* %data to i8*
3383   store i8* %cast, i8** %dat, align 8
3384   %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3385   %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
3386     1
3387   %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3388     i32 1
3389   %temp4 = load %list_item*, %list_item** %test, align 8
3390   store %list_item* %temp4, %list_item** %dat3, align 8
3391   store %list_item* %data_node, %list_item** %test, align 8
3392   %temp5 = load %list_item*, %list_item** %next, align 8
3393   store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3394   ret %list_item** %new_list_ptr_ptr
3395 }

```

```

3351 else:                                     ; preds = %entry
3352   %alloca1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
3353   %new_struct_ptr = bitcast i8* %alloca1 to %list_item*
3354   store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3355   %alloca11 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
      , i32* null, i32 1) to i32))
3356   %ltyp = bitcast i8* %alloca11 to i32*
3357   %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
      0, i32 0
3358   %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3359   %cast_old_data_ptr = bitcast i8* %old_data_ptr to i32*
3360   %old_data = load i32, i32* %cast_old_data_ptr, align 4
3361   store i32 %old_data, i32* %ltyp, align 4
3362   %data_ptr_cast = bitcast i32* %ltyp to i8*
3363   %store_new_data = getelementptr inbounds %list_item, %list_item* %
      new_struct_ptr, i32 0, i32 0
3364   store i8* %data_ptr_cast, i8** %store_new_data, align 8
3365   store %list_item* %new_struct_ptr, %list_item** %2, align 8
3366   %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
      i32 0, i32 1
3367   %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
      1
3368   %next2 = load %list_item*, %list_item** %next_ptr, align 8
3369   %sub = sub i32 %1, 1
3370   %3 = call %list_item** @list_copy_int(%list_item* %next2, i32 %sub, %
      list_item** %next)
3371   ret %list_item** %3
3372 }
3373
3374 define %list_item** @insert_string(%list_item** %0, i8* %1, i32 %2) {
3375 entry:
3376   %list_ptr = load %list_item*, %list_item** %0, align 8
3377   %alloca1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
      i1** null, i32 1) to i32))
3378   %new_list_ptr_ptr = bitcast i8* %alloca1 to %list_item**
3379   store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3380   %last_node_ptr_ptr = call %list_item** @list_copy_string(%list_item* %
      list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3381   %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3382   %temp = alloca %list_item, align 8
3383   %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3384   store %list_item* %new_list_ptr, %list_item** %next, align 8
3385   %alloca11 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
3386   %data_node = bitcast i8* %alloca11 to %list_item*
3387   %alloca12 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
      *, i1** null, i32 1) to i32))
3388   %data = bitcast i8* %alloca12 to i8**
3389   store i8* %1, i8** %data, align 8
3390   %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
      i32 0
3391   %cast = bitcast i8** %data to i8*
3392   store i8* %cast, i8** %dat, align 8
3393   %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3394   %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
      1
3395   %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
      i32 1

```

```

3396 %temp4 = load %list_item*, %list_item** %test, align 8
3397 store %list_item* %temp4, %list_item** %dat3, align 8
3398 store %list_item* %data_node, %list_item** %test, align 8
3399 %temp5 = load %list_item*, %list_item** %next, align 8
3400 store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3401 ret %list_item** %new_list_ptr_ptr
3402 }
3403
3404 define %list_item** @list_copy_string(%list_item* %0, i32 %1, %list_item**
    %2) {
3405 entry:
3406     %is_zero = icmp eq i32 %1, 0
3407     %ptr_is_null = icmp eq %list_item* %0, null
3408     %or_conds = or i1 %is_zero, %ptr_is_null
3409     br i1 %or_conds, label %then, label %else
3410
3411 then:                                     ; preds = %entry
3412     ret %list_item** %2
3413
3414 else:                                     ; preds = %entry
3415     %allocacll = tail call i8* @malloc(i32 ptrtoint (%list_item*
        %getelementptr (%list_item, %list_item* null, i32 1) to i32))
3416     %new_struct_ptr = bitcast i8* %allocacll to %list_item*
3417     store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3418     %allocacll1 = tail call i8* @malloc(i32 ptrtoint (i1** %getelementptr (i1
        *, i1** null, i32 1) to i32))
3419     %ltyp = bitcast i8* %allocacll1 to i8**
3420     %old_data_ptr_ptr = %getelementptr inbounds %list_item, %list_item* %0, i32
        0, i32 0
3421     %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3422     %cast_old_data_ptr = bitcast i8* %old_data_ptr to i8**
3423     %old_data = load i8*, i8** %cast_old_data_ptr, align 8
3424     store i8* %old_data, i8** %ltyp, align 8
3425     %data_ptr_cast = bitcast i8** %ltyp to i8*
3426     %store_new_data = %getelementptr inbounds %list_item, %list_item* %
        new_struct_ptr, i32 0, i32 0
3427     store i8* %data_ptr_cast, i8** %store_new_data, align 8
3428     store %list_item* %new_struct_ptr, %list_item** %2, align 8
3429     %next = %getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
        i32 0, i32 1
3430     %next_ptr = %getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
        1
3431     %next2 = load %list_item*, %list_item** %next_ptr, align 8
3432     %sub = sub i32 %1, 1
3433     %3 = call %list_item** @list_copy_string(%list_item* %next2, i32 %sub, %
        list_item** %next)
3434     ret %list_item** %3
3435 }
3436
3437 define %list_item** @insert_char(%list_item** %0, i8 %1, i32 %2) {
3438 entry:
3439     %list_ptr = load %list_item*, %list_item** %0, align 8
3440     %allocacll = tail call i8* @malloc(i32 ptrtoint (i1** %getelementptr (i1*,
        i1** null, i32 1) to i32))
3441     %new_list_ptr_ptr = bitcast i8* %allocacll to %list_item**
3442     store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3443     %last_node_ptr_ptr = call %list_item** @list_copy_char(%list_item* %
        list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3444     %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8

```

```

3445 %temp = alloca %list_item, align 8
3446 %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3447 store %list_item* %new_list_ptr, %list_item** %next, align 8
3448 %mallocall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
3449 %data_node = bitcast i8* %mallocall1 to %list_item*
3450 %data = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
3451 store i8 %1, i8* %data, align 1
3452 %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 0
3453 store i8* %data, i8** %dat, align 8
3454 %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3455 %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
    1
3456 %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 1
3457 %temp4 = load %list_item*, %list_item** %test, align 8
3458 store %list_item* %temp4, %list_item** %dat3, align 8
3459 store %list_item* %data_node, %list_item** %test, align 8
3460 %temp5 = load %list_item*, %list_item** %next, align 8
3461 store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3462 ret %list_item** %new_list_ptr_ptr
3463 }
3464
3465 define %list_item** @list_copy_char(%list_item* %0, i32 %1, %list_item** %2)
    {
3466 entry:
3467     %is_zero = icmp eq i32 %1, 0
3468     %ptr_is_null = icmp eq %list_item* %0, null
3469     %or_conds = or i1 %is_zero, %ptr_is_null
3470     br i1 %or_conds, label %then, label %else
3471
3472 then:                                     ; preds = %entry
3473     ret %list_item** %2
3474
3475 else:                                     ; preds = %entry
3476     %mallocall = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
3477     %new_struct_ptr = bitcast i8* %mallocall to %list_item*
3478     store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3479     %ltyp = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
        null, i32 1) to i32))
3480     %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
        0, i32 0
3481     %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3482     %old_data = load i8, i8* %old_data_ptr, align 1
3483     store i8 %old_data, i8* %ltyp, align 1
3484     %store_new_data = getelementptr inbounds %list_item, %list_item* %
        new_struct_ptr, i32 0, i32 0
3485     store i8* %ltyp, i8** %store_new_data, align 8
3486     store %list_item* %new_struct_ptr, %list_item** %2, align 8
3487     %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
        i32 0, i32 1
3488     %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
        1
3489     %next2 = load %list_item*, %list_item** %next_ptr, align 8
3490     %sub = sub i32 %1, 1
3491     %3 = call %list_item** @list_copy_char(%list_item* %next2, i32 %sub, %

```

```

    list_item** %next)
3492   ret %list_item** %3
3493 }
3494
3495 define %list_item** @insert_bool(%list_item** %0, i1 %1, i32 %2) {
3496 entry:
3497   %list_ptr = load %list_item*, %list_item** %0, align 8
3498   %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
3499                                     i1** null, i32 1) to i32))
3500   %new_list_ptr_ptr = bitcast i8* %mallocall to %list_item**
3501   store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3502   %last_node_ptr_ptr = call %list_item** @list_copy_bool(%list_item* %
3503     list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3504   %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3505   %temp = alloca %list_item, align 8
3506   %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3507   store %list_item* %new_list_ptr, %list_item** %next, align 8
3508   %mallocall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
3509     getelementptr (%list_item, %list_item* null, i32 1) to i32))
3510   %data_node = bitcast i8* %mallocall1 to %list_item*
3511   %mallocall2 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
3512     i1* null, i32 1) to i32))
3513   %data = bitcast i8* %mallocall2 to i1*
3514   store i1 %1, i1* %data, align 1
3515   %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3516     i32 0
3517   %cast = bitcast i1* %data to i8*
3518   store i8* %cast, i8** %dat, align 8
3519   %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3520   %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
3521     1
3522   %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3523     i32 1
3524   %temp4 = load %list_item*, %list_item** %test, align 8
3525   store %list_item* %temp4, %list_item** %dat3, align 8
3526   store %list_item* %data_node, %list_item** %test, align 8
3527   %temp5 = load %list_item*, %list_item** %next, align 8
3528   store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3529   ret %list_item** %new_list_ptr_ptr
3530 }
3531
3532 define %list_item** @list_copy_bool(%list_item* %0, i32 %1, %list_item** %2)
3533 {
3534 entry:
3535   %is_zero = icmp eq i32 %1, 0
3536   %ptr_is_null = icmp eq %list_item* %0, null
3537   %or_conds = or i1 %is_zero, %ptr_is_null
3538   br i1 %or_conds, label %then, label %else
3539
3540 then:
3541   ; preds = %entry
3542   ret %list_item** %2
3543
3544 else:
3545   ; preds = %entry
3546   %mallocall = tail call i8* @malloc(i32 ptrtoint (%list_item*
3547     getelementptr (%list_item, %list_item* null, i32 1) to i32))
3548   %new_struct_ptr = bitcast i8* %mallocall to %list_item*
3549   store %list_item* zeroinitializer, %list_item* %new_struct_ptr, align 1
3550   %mallocall1 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
3551     i1* null, i32 1) to i32))

```

```

3540 %ltyp = bitcast i8* %allocacall1 to i1*
3541 %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
    0, i32 0
3542 %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3543 %cast_old_data_ptr = bitcast i8* %old_data_ptr to i1*
3544 %old_data = load i1, i1* %cast_old_data_ptr, align 1
3545 store i1 %old_data, i1* %ltyp, align 1
3546 %data_ptr_cast = bitcast i1* %ltyp to i8*
3547 %store_new_data = getelementptr inbounds %list_item, %list_item* %
    new_struct_ptr, i32 0, i32 0
3548 store i8* %data_ptr_cast, i8** %store_new_data, align 8
3549 store %list_item* %new_struct_ptr, %list_item** %2, align 8
3550 %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
    i32 0, i32 1
3551 %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
3552 %next2 = load %list_item*, %list_item** %next_ptr, align 8
3553 %sub = sub i32 %1, 1
3554 %3 = call %list_item** @list_copy_bool(%list_item* %next2, i32 %sub, %
    list_item** %next)
3555 ret %list_item** %3
3556 }
3557
3558 define %list_item** @insert_float(%list_item** %0, double %1, i32 %2) {
3559 entry:
3560 %list_ptr = load %list_item*, %list_item** %0, align 8
3561 %allocacall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
3562 %new_list_ptr_ptr = bitcast i8* %allocacall to %list_item**
3563 store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3564 %last_node_ptr_ptr = call %list_item** @list_copy_float(%list_item* %
    list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3565 %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3566 %temp = alloca %list_item, align 8
3567 %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3568 store %list_item* %new_list_ptr, %list_item** %next, align 8
3569 %allocacall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
3570 %data_node = bitcast i8* %allocacall1 to %list_item*
3571 %allocacall2 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (
    double, double* null, i32 1) to i32))
3572 %data = bitcast i8* %allocacall2 to double*
3573 store double %1, double* %data, align 8
3574 %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 0
3575 %cast = bitcast double* %data to i8*
3576 store i8* %cast, i8** %dat, align 8
3577 %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3578 %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
    1
3579 %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 1
3580 %temp4 = load %list_item*, %list_item** %test, align 8
3581 store %list_item* %temp4, %list_item** %dat3, align 8
3582 store %list_item* %data_node, %list_item** %test, align 8
3583 %temp5 = load %list_item*, %list_item** %next, align 8
3584 store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3585 ret %list_item** %new_list_ptr_ptr
3586 }

```

```

3587
3588 define %list_item** @list_copy_float(%list_item* %0, i32 %1, %list_item**
    %2) {
3589 entry:
3590     %is_zero = icmp eq i32 %1, 0
3591     %ptr_is_null = icmp eq %list_item* %0, null
3592     %or_conds = or i1 %is_zero, %ptr_is_null
3593     br i1 %or_conds, label %then, label %else
3594
3595 then:                                     ; preds = %entry
3596     ret %list_item** %2
3597
3598 else:                                     ; preds = %entry
3599     %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
        %getelementptr (%list_item, %list_item* null, i32 1) to i32))
3600     %new_struct_ptr = bitcast i8* %malloccall to %list_item*
3601     store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3602     %malloccall1 = tail call i8* @malloc(i32 ptrtoint (double* %getelementptr (
        double, double* null, i32 1) to i32))
3603     %ltyp = bitcast i8* %malloccall1 to double*
3604     %old_data_ptr_ptr = %getelementptr inbounds %list_item, %list_item* %0, i32
        0, i32 0
3605     %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3606     %cast_old_data_ptr = bitcast i8* %old_data_ptr to double*
3607     %old_data = load double, double* %cast_old_data_ptr, align 8
3608     store double %old_data, double* %ltyp, align 8
3609     %data_ptr_cast = bitcast double* %ltyp to i8*
3610     %store_new_data = %getelementptr inbounds %list_item, %list_item* %
        new_struct_ptr, i32 0, i32 0
3611     store i8* %data_ptr_cast, i8** %store_new_data, align 8
3612     store %list_item* %new_struct_ptr, %list_item** %2, align 8
3613     %next = %getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
        i32 0, i32 1
3614     %next_ptr = %getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
        1
3615     %next2 = load %list_item*, %list_item** %next_ptr, align 8
3616     %sub = sub i32 %1, 1
3617     %3 = call %list_item** @list_copy_float(%list_item* %next2, i32 %sub, %
        list_item** %next)
3618     ret %list_item** %3
3619 }
3620
3621 define i1 @strcmp_function(i8* %0, i8* %1) {
3622 entry:
3623     %length = call i32 @string_length(i8* %0, i32 0)
3624     %length1 = call i32 @string_length(i8* %1, i32 0)
3625     %same_length = icmp ne i32 %length, %length1
3626     br i1 %same_length, label %then, label %else
3627
3628 then:                                     ; preds = %entry
3629     ret i1 false
3630
3631 else:                                     ; preds = %entry
3632     %last_index = sub i32 %length, 1
3633     %res = call i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %last_index,
        i32 0)
3634     ret i1 %res
3635 }
3636

```

```

3637 define i32 @string_length(i8* %0, i32 %1) {
3638 entry:
3639   %char = load i8, i8* %0, align 1
3640   %ptr_is_null = icmp eq i8 %char, 0
3641   br i1 %ptr_is_null, label %then, label %else
3642
3643 then:                                     ; preds = %entry
3644   ret i32 %1
3645
3646 else:                                     ; preds = %entry
3647   %next_ptr = getelementptr i8, i8* %0, i32 1
3648   %add = add i32 %1, 1
3649   %result = call i32 @string_length(i8* %next_ptr, i32 %add)
3650   ret i32 %result
3651 }
3652
3653 define i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %2, i32 %3) {
3654 entry:
3655   %charA_ptr = getelementptr i8, i8* %0, i32 %3
3656   %charA = load i8, i8* %charA_ptr, align 1
3657   %charB_ptr = getelementptr i8, i8* %1, i32 %3
3658   %charB = load i8, i8* %charB_ptr, align 1
3659   %not_same = icmp ne i8 %charA, %charB
3660   br i1 %not_same, label %then_not_same, label %else_same
3661
3662 then_not_same:                           ; preds = %entry
3663   ret i1 false
3664
3665 else_same:                               ; preds = %entry
3666   %last_char = icmp eq i32 %3, %2
3667   br i1 %last_char, label %then_end, label %else_not_end
3668
3669 then_end:                                ; preds = %else_same
3670   ret i1 true
3671
3672 else_not_end:                             ; preds = %else_same
3673   %next_index = add i32 1, %3
3674   %res = call i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %2, i32 %
     next_index)
3675   ret i1 %res
3676 }
3677
3678 ; Function Attrs: argmemonly nounwind willreturn
3679 declare void @llvm.memcpy.p0i8.p0i8.i32(i8* noalias nocapture writeonly, i8*
     noalias nocapture readonly, i32, i1 immarg) #0
3680
3681 attributes #0 = { argmemonly nounwind willreturn }

```

### 9.3.2 maximum\_subarray.tm

Below is the TEAM source code:

```

1  /* Given an integer array nums, find the contiguous subarray (containing at
2     least one number) which has the largest sum and return its sum. */
3  int max(int x, int y):
4     if x > y:
5         return x;
6     end
7     return y;

```



```

7 end
8
9 int maxSubArray(list nums):
10
11     int total_sum = nums[0];
12     int max_sum = nums[0];
13
14
15     for num in nums[1:]:
16         total_sum = max(total_sum + num, num);
17         max_sum = max(max_sum, total_sum);
18     end
19
20     return max_sum;
21 end
22
23 print("%d\n", maxSubArray([-2,1,-3,4,-1,2,1,-5,4]));
24 // Output: 6
25 // Explanation: [4,-1,2,1] has the largest sum = 6.

```

The program is expected to have the following output:

```

1 6

```

Below is the LLVM generated by the compiler:

```

1 ; ModuleID = 'TEAM'
2 source_filename = "TEAM"
3
4 %list_item = type <{ i8*, %list_item* }>
5
6 @string = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
7
8 declare i32 @printf(i8*, ...)
9
10 declare double @pow(double, double)
11
12 declare i8* @fopen(i8*, i8*)
13
14 declare i32 @close(i8*)
15
16 declare i8* @readline(i8*)
17
18 declare i8* @write(i8*, i8*)
19
20 declare i1 @match(i8*, i8*)
21
22 declare i8* @find(i8*, i8*)
23
24 declare i8* @replace(i8*, i8*, i8*, i32)
25
26 declare i8* @replace_all(i8*, i8*, i8*)
27
28 declare %list_item** @find_all(i8*, i8*)
29
30 define i32 @main() {
31 entry:
32     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
33         i1** null, i32 1) to i32))
34     %list = bitcast i8* %mallocall to %list_item**

```

```

34 %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
35 %list_item = bitcast i8* %malloccall1 to %list_item*
36 store %list_item zeroinitializer, %list_item* %list_item, align 1
37 %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
    , i32* null, i32 1) to i32))
38 %copied = bitcast i8* %malloccall2 to i32*
39 store i32 4, i32* %copied, align 4
40 %cast_ptr = bitcast i32* %copied to i8*
41 %data_ptr_container = getelementptr inbounds %list_item, %list_item* %
    list_item, i32 0, i32 0
42 store i8* %cast_ptr, i8** %data_ptr_container, align 8
43 %next = getelementptr inbounds %list_item, %list_item* %list_item, i32 0,
    i32 1
44 store %list_item* null, %list_item** %next, align 8
45 %malloccall3 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
46 %list_item4 = bitcast i8* %malloccall3 to %list_item*
47 store %list_item zeroinitializer, %list_item* %list_item4, align 1
48 %malloccall5 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
    , i32* null, i32 1) to i32))
49 %copied6 = bitcast i8* %malloccall5 to i32*
50 store i32 -5, i32* %copied6, align 4
51 %cast_ptr7 = bitcast i32* %copied6 to i8*
52 %data_ptr_container8 = getelementptr inbounds %list_item, %list_item* %
    list_item4, i32 0, i32 0
53 store i8* %cast_ptr7, i8** %data_ptr_container8, align 8
54 %next9 = getelementptr inbounds %list_item, %list_item* %list_item4, i32
    0, i32 1
55 store %list_item* %list_item, %list_item** %next9, align 8
56 %malloccall10 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
57 %list_item11 = bitcast i8* %malloccall10 to %list_item*
58 store %list_item zeroinitializer, %list_item* %list_item11, align 1
59 %malloccall12 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
60 %copied13 = bitcast i8* %malloccall12 to i32*
61 store i32 1, i32* %copied13, align 4
62 %cast_ptr14 = bitcast i32* %copied13 to i8*
63 %data_ptr_container15 = getelementptr inbounds %list_item, %list_item* %
    list_item11, i32 0, i32 0
64 store i8* %cast_ptr14, i8** %data_ptr_container15, align 8
65 %next16 = getelementptr inbounds %list_item, %list_item* %list_item11, i32
    0, i32 1
66 store %list_item* %list_item4, %list_item** %next16, align 8
67 %malloccall17 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
68 %list_item18 = bitcast i8* %malloccall17 to %list_item*
69 store %list_item zeroinitializer, %list_item* %list_item18, align 1
70 %malloccall19 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
71 %copied20 = bitcast i8* %malloccall19 to i32*
72 store i32 2, i32* %copied20, align 4
73 %cast_ptr21 = bitcast i32* %copied20 to i8*
74 %data_ptr_container22 = getelementptr inbounds %list_item, %list_item* %
    list_item18, i32 0, i32 0
75 store i8* %cast_ptr21, i8** %data_ptr_container22, align 8
76 %next23 = getelementptr inbounds %list_item, %list_item* %list_item18, i32
    0, i32 1

```

```

77 store %list_item* %list_item11, %list_item** %next23, align 8
78 %malloccall124 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
79 %list_item25 = bitcast i8* %malloccall124 to %list_item*
80 store %list_item zeroinitializer, %list_item* %list_item25, align 1
81 %malloccall126 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
82 %copied27 = bitcast i8* %malloccall126 to i32*
83 store i32 -1, i32* %copied27, align 4
84 %cast_ptr28 = bitcast i32* %copied27 to i8*
85 %data_ptr_container29 = getelementptr inbounds %list_item, %list_item* %
    list_item25, i32 0, i32 0
86 store i8* %cast_ptr28, i8** %data_ptr_container29, align 8
87 %next30 = getelementptr inbounds %list_item, %list_item* %list_item25, i32
    0, i32 1
88 store %list_item* %list_item18, %list_item** %next30, align 8
89 %malloccall131 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
90 %list_item32 = bitcast i8* %malloccall131 to %list_item*
91 store %list_item zeroinitializer, %list_item* %list_item32, align 1
92 %malloccall133 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
93 %copied34 = bitcast i8* %malloccall133 to i32*
94 store i32 4, i32* %copied34, align 4
95 %cast_ptr35 = bitcast i32* %copied34 to i8*
96 %data_ptr_container36 = getelementptr inbounds %list_item, %list_item* %
    list_item32, i32 0, i32 0
97 store i8* %cast_ptr35, i8** %data_ptr_container36, align 8
98 %next37 = getelementptr inbounds %list_item, %list_item* %list_item32, i32
    0, i32 1
99 store %list_item* %list_item25, %list_item** %next37, align 8
100 %malloccall138 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
101 %list_item39 = bitcast i8* %malloccall138 to %list_item*
102 store %list_item zeroinitializer, %list_item* %list_item39, align 1
103 %malloccall140 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
104 %copied41 = bitcast i8* %malloccall140 to i32*
105 store i32 -3, i32* %copied41, align 4
106 %cast_ptr42 = bitcast i32* %copied41 to i8*
107 %data_ptr_container43 = getelementptr inbounds %list_item, %list_item* %
    list_item39, i32 0, i32 0
108 store i8* %cast_ptr42, i8** %data_ptr_container43, align 8
109 %next44 = getelementptr inbounds %list_item, %list_item* %list_item39, i32
    0, i32 1
110 store %list_item* %list_item32, %list_item** %next44, align 8
111 %malloccall145 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
112 %list_item46 = bitcast i8* %malloccall145 to %list_item*
113 store %list_item zeroinitializer, %list_item* %list_item46, align 1
114 %malloccall147 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
115 %copied48 = bitcast i8* %malloccall147 to i32*
116 store i32 1, i32* %copied48, align 4
117 %cast_ptr49 = bitcast i32* %copied48 to i8*
118 %data_ptr_container50 = getelementptr inbounds %list_item, %list_item* %
    list_item46, i32 0, i32 0
119 store i8* %cast_ptr49, i8** %data_ptr_container50, align 8
120 %next51 = getelementptr inbounds %list_item, %list_item* %list_item46, i32

```

```

    0, i32 1
121 store %list_item* %list_item39, %list_item** %next51, align 8
122 %alloca152 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
123 %list_item53 = bitcast i8* %alloca152 to %list_item*
124 store %list_item zeroinitializer, %list_item* %list_item53, align 1
125 %alloca154 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
126 %copied55 = bitcast i8* %alloca154 to i32*
127 store i32 -2, i32* %copied55, align 4
128 %cast_ptr56 = bitcast i32* %copied55 to i8*
129 %data_ptr_container57 = getelementptr inbounds %list_item, %list_item* %
    list_item53, i32 0, i32 0
130 store i8* %cast_ptr56, i8** %data_ptr_container57, align 8
131 %next58 = getelementptr inbounds %list_item, %list_item* %list_item53, i32
    0, i32 1
132 store %list_item* %list_item46, %list_item** %next58, align 8
133 store %list_item* %list_item53, %list_item** %list, align 8
134 %_result = call i32 @maxSubArray_int(%list_item** %list)
135 %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8
    ], [4 x i8]* @string, i32 0, i32 0), i32 %_result)
136 ret i32 0
137 }
138
139 define i32 @maxSubArray_int(%list_item** %nums) {
140 entry:
141   %num = alloca i32, align 4
142   %for_index = alloca i32, align 4
143   %max_sum = alloca i32, align 4
144   %total_sum = alloca i32, align 4
145   %nums1 = alloca %list_item**, align 8
146   store %list_item** %nums, %list_item*** %nums1, align 8
147   %nums2 = load %list_item**, %list_item*** %nums1, align 8
148   %ilist = load %list_item*, %list_item** %nums2, align 8
149   %_result = call %list_item* @list_access(%list_item* %ilist, i32 0)
150   %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
151   %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
152   %cast_data_ptr = bitcast i8* %data_ptr to i32*
153   %data = load i32, i32* %cast_data_ptr, align 4
154   store i32 %data, i32* %total_sum, align 4
155   %nums3 = load %list_item**, %list_item*** %nums1, align 8
156   %ilist4 = load %list_item*, %list_item** %nums3, align 8
157   %_result5 = call %list_item* @list_access(%list_item* %ilist4, i32 0)
158   %data_ptr_ptr6 = getelementptr inbounds %list_item, %list_item* %_result5,
    i32 0, i32 0
159   %data_ptr7 = load i8*, i8** %data_ptr_ptr6, align 8
160   %cast_data_ptr8 = bitcast i8* %data_ptr7 to i32*
161   %data9 = load i32, i32* %cast_data_ptr8, align 4
162   store i32 %data9, i32* %max_sum, align 4
163   store i32 0, i32* %for_index, align 4
164   store i32 0, i32* %num, align 4
165   br label %while
166
167 while:
    entry
    %for_index28 = load i32, i32* %for_index, align 4
168   %nums29 = load %list_item**, %list_item*** %nums1, align 8
169   %ilist30 = load %list_item*, %list_item** %nums29, align 8
170

```

```

171 %start31 = call %list_item* @list_access(%list_item* %ilist30, i32 1)
172 %mallocall32 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
173 %new_list_ptr_ptr33 = bitcast i8* %mallocall32 to %list_item**
174 %0 = call %list_item** @list_copy_int(%list_item* %start31, i32 -1, %
    list_item** %new_list_ptr_ptr33)
175 %ilist34 = load %list_item*, %list_item** %new_list_ptr_ptr33, align 8
176 %length = call i32 @list_length(%list_item* %ilist34, i32 0)
177 %tmp35 = icmp slt i32 %for_index28, %length
178 br i1 %tmp35, label %while_body, label %merge
179
180 merge:                                ; preds = %while
181 %max_sum36 = load i32, i32* %max_sum, align 4
182 ret i32 %max_sum36
183
184 while_body:                            ; preds = %while
185 %nums10 = load %list_item**, %list_item** %nums1, align 8
186 %ilist11 = load %list_item*, %list_item** %nums10, align 8
187 %start = call %list_item* @list_access(%list_item* %ilist11, i32 1)
188 %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
189 %new_list_ptr_ptr = bitcast i8* %mallocall to %list_item**
190 %1 = call %list_item** @list_copy_int(%list_item* %start, i32 -1, %
    list_item** %new_list_ptr_ptr)
191 %ilist12 = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
192 %for_index13 = load i32, i32* %for_index, align 4
193 %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
194 %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
195 %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
196 %cast_data_ptr17 = bitcast i8* %data_ptr16 to i32*
197 %data18 = load i32, i32* %cast_data_ptr17, align 4
198 store i32 %data18, i32* %num, align 4
199 %for_index19 = load i32, i32* %for_index, align 4
200 %tmp = add i32 %for_index19, 1
201 store i32 %tmp, i32* %for_index, align 4
202 %num20 = load i32, i32* %num, align 4
203 %total_sum21 = load i32, i32* %total_sum, align 4
204 %num22 = load i32, i32* %num, align 4
205 %tmp23 = add i32 %total_sum21, %num22
206 %_result24 = call i32 @max(i32 %tmp23, i32 %num20)
207 store i32 %_result24, i32* %total_sum, align 4
208 %total_sum25 = load i32, i32* %total_sum, align 4
209 %max_sum26 = load i32, i32* %max_sum, align 4
210 %_result27 = call i32 @max(i32 %max_sum26, i32 %total_sum25)
211 store i32 %_result27, i32* %max_sum, align 4
212 br label %while
213 }
214
215 define i32 @max(i32 %x, i32 %y) {
216 entry:
217 %x1 = alloca i32, align 4
218 store i32 %x, i32* %x1, align 4
219 %y2 = alloca i32, align 4
220 store i32 %y, i32* %y2, align 4
221 %x3 = load i32, i32* %x1, align 4
222 %y4 = load i32, i32* %y2, align 4
223 %tmp = icmp sgt i32 %x3, %y4

```

```

224     br i1 %tmp, label %then, label %else
225
226 merge:                                     ; preds = %else
227     %y6 = load i32, i32* %y2, align 4
228     ret i32 %y6
229
230 then:                                       ; preds = %entry
231     %x5 = load i32, i32* %x1, align 4
232     ret i32 %x5
233
234 else:                                       ; preds = %entry
235     br label %merge
236 }
237
238 declare noalias i8* @malloc(i32)
239
240 define %list_item* @list_access(%list_item* %0, i32 %1) {
241 entry:
242     %is_zero = icmp eq i32 %1, 0
243     br i1 %is_zero, label %then, label %else
244
245 then:                                       ; preds = %entry
246     ret %list_item* %0
247
248 else:                                       ; preds = %entry
249     %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
        1
250     %next = load %list_item*, %list_item** %next_ptr, align 8
251     %sub = sub i32 %1, 1
252     %result = call %list_item* @list_access(%list_item* %next, i32 %sub)
253     ret %list_item* %result
254 }
255
256 define %list_item** @list_copy_int(%list_item* %0, i32 %1, %list_item** %2)
    {
257 entry:
258     %is_zero = icmp eq i32 %1, 0
259     %ptr_is_null = icmp eq %list_item* %0, null
260     %or_conds = or i1 %is_zero, %ptr_is_null
261     br i1 %or_conds, label %then, label %else
262
263 then:                                       ; preds = %entry
264     ret %list_item** %2
265
266 else:                                       ; preds = %entry
267     %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
268     %new_struct_ptr = bitcast i8* %malloccall to %list_item*
269     store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
270     %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
        , i32* null, i32 1) to i32))
271     %ltyp = bitcast i8* %malloccall1 to i32*
272     %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
        0, i32 0
273     %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
274     %cast_old_data_ptr = bitcast i8* %old_data_ptr to i32*
275     %old_data = load i32, i32* %cast_old_data_ptr, align 4
276     store i32 %old_data, i32* %ltyp, align 4
277     %data_ptr_cast = bitcast i32* %ltyp to i8*

```

```

278 %store_new_data = getelementptr inbounds %list_item, %list_item* %
    new_struct_ptr, i32 0, i32 0
279 store i8* %data_ptr_cast, i8** %store_new_data, align 8
280 store %list_item* %new_struct_ptr, %list_item** %2, align 8
281 %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
    i32 0, i32 1
282 %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
283 %next2 = load %list_item*, %list_item** %next_ptr, align 8
284 %sub = sub i32 %1, 1
285 %3 = call %list_item** @list_copy_int(%list_item* %next2, i32 %sub, %
    list_item** %next)
286 ret %list_item** %3
287 }
288
289 define i32 @list_length(%list_item* %0, i32 %1) {
290 entry:
291 %ptr_is_null = icmp eq %list_item* %0, null
292 br i1 %ptr_is_null, label %then, label %else
293
294 then:                                     ; preds = %entry
295 ret i32 %1
296
297 else:                                     ; preds = %entry
298 %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
299 %next = load %list_item*, %list_item** %next_ptr, align 8
300 %add = add i32 %1, 1
301 %result = call i32 @list_length(%list_item* %next, i32 %add)
302 ret i32 %result
303 }

```

## 9.4 Breakdown of Responsibilities

It is difficult at the time when this report is written to determine exactly who wrote which test programs. We all wrote tests and often offered suggestions to others in terms of what should be tested. But in particular, Naoki tested for the file IO, Lulu tested list polymorphism, Saurav tested list slicing and other list related functionality, and Yingjie tested list built-in and string standard library functions.

## 10 Lessons Learned

### 10.1 Lulu

I gained an immense appreciation for the amount of work and thoughts that others have put into building modern day compilers, especially after knowing that even a basic compiler requires extensive research and careful design decisions. Additionally, I never actively thought about all of the decisions that goes into designing a programming language and how difficult they were to make. Secondly, I learned the sheer necessity of testing. Many times during the project did I have to remind myself to test thoroughly and punctually after the completion of a new feature, as there were several bugs that indicated bigger problems that sometimes reflected the need of changing a major component of our infrastructure. I learned that testing early and often will avoid having to discover bugs several weeks down the line. Perhaps my biggest takeaway is the value in maintaining a largely scoped project. While I believed in a fast and frequent production, I realized that diving deep into a new feature without being aware the complexity and intricacies of implementing such a feature, while exciting and fun, may end up as a discouraging experience. This made me take a step back, reevaluate the most optimal path to take in order to achieve my goals, and reconsider which features should be within the scope of a project.

### 10.2 Yingjie

This class removed a big layer of abstraction and revealed how the compiler works behind the scenes. It made me see how interesting and yet how complex this topic is. By working extensively with OCaml, I also see the big advantage of functional programming; everything can be so elegantly written, and it can catch so many errors even before the program is run. Working on a large project with multiple people also made me realize the importance of planning and communication. There were so many instances where I figured out easier and more straightforward implementations for specific features only after spending hours chasing pointers in LLVM. For future groups, I would recommend making very specific plans, especially at the code generation phase. Clear documentation is also crucial; we implemented the list feature in our language as a linked list under the hood. There were times when I misread a pointer to a pointer as a pointer to data, and that was simply disastrous.

### 10.3 Naoki

This project allowed me to gain both a high-level understanding of how a modern-day compiler works and low-level details of how each part is implemented. Working on each of the compiler sequentially allowed me to gain that understanding. I also quite enjoyed and learned a lot during the design phase of the project since I got exposed to a lot of things you have to consider when designing a language: thinking about what kind of purposes the language should be useful for, how to make the syntax intuitive and considering trade-offs between static typing and dynamic typing. Furthermore, working on such a large-scale project made me realize how important it is to plan ahead of time, break up the project into smaller parts and test things as we add them. Overall, I feel that with the knowledge I have gained from this project and class, I have a better understanding of why languages are designed a certain way and how to write code that is more compiler friendly. For future teams, I would stress the importance of testing early and having a detailed timeline from the start.

### 10.4 Saurav

Being a part of a semester long project with a team, I was reminded of importance of planning, designing and team coordination. Fortunately, our team coordinated pretty well among each other. However, there were few ideas that we did not plan well during the starting of the project that required big architectural fix in the later part of the semester. In the technical sides of compilers, I learned how a feature that looks simple on the high level takes a lot of design choices



and complex implementation in the compiler. This made me feel fortunate for being able to work on high level languages and appreciate modern compilers more. I was also reminded of how sticking to the functional language paradigm helps us avoid endless debugging for run-time errors. Likewise, I learned the importance of a Documentation and Language Reference Manual while working with pointers and memory that required instructions I was not familiar with. I also got a clearer picture of where optimizations fit in the compiler. Since performance was not the biggest priority, I learned to use this leeway to make some parts of the code more readable and neat-looking.

# 11 Appendix

## 11.1 Git Log

| Author        | Date       | Message                                                         |
|---------------|------------|-----------------------------------------------------------------|
| Lulu          | 2021-02-27 | Initial commit                                                  |
| Lulu Zheng    | 2021-02-27 | first draft of scanner                                          |
| Lulu Zheng    | 2021-02-28 | fixed bug for slcomment                                         |
| Lulu Zheng    | 2021-03-06 | parser and ast draft                                            |
| gyawalisaurav | 2021-03-06 | Added stuff for TEAM to microc AST                              |
| Yingjie Ling  | 2021-03-07 | Merge pull request #1 from luluzheng1/ast                       |
| Lulu Zheng    | 2021-03-07 | revised ast and parser                                          |
| Lulu Zheng    | 2021-03-08 | still need to fix conflicts                                     |
| Lulu Zheng    | 2021-03-08 | added arrow and colon                                           |
| Lulu Zheng    | 2021-03-08 | added arrow and colon                                           |
| Yingjie Ling  | 2021-03-09 | did slight modification and generated output file               |
| Yingjie Ling  | 2021-03-09 | cleaned directory and added fst and snd for three tuples        |
| Lulu Zheng    | 2021-03-09 | updated gitignore to include .output and .ml files              |
| Lulu Zheng    | 2021-03-09 | removed SEMI                                                    |
| Lulu Zheng    | 2021-03-09 | removed conflicts                                               |
| Lulu Zheng    | 2021-03-09 | removed SEMI                                                    |
| Lulu          | 2021-03-09 | Merge pull request #2 from luluzheng1/parser_0308               |
| Lulu Zheng    | 2021-03-09 | Merge branch 'parser' of github.com:luluzheng1/TEAM into parser |
| Lulu Zheng    | 2021-03-09 | added range precedence and associativity                        |
| Lulu Zheng    | 2021-03-09 | revised ast                                                     |
| Lulu Zheng    | 2021-03-09 | adding toplevel                                                 |
| Lulu Zheng    | 2021-03-09 | modify .gitignore                                               |
| Lulu Zheng    | 2021-03-09 | added newline as EOL token                                      |
| Lulu Zheng    | 2021-03-09 | adding Id as an expr                                            |
| Lulu Zheng    | 2021-03-09 | fixed pretty printing                                           |
| nokada11      | 2021-03-09 | Add tests                                                       |
| nokada11      | 2021-03-09 | Add script to run tests                                         |
| Yingjie Ling  | 2021-03-09 | added list_assign                                               |
| Yingjie Ling  | 2021-03-10 | Merge pull request #3 from luluzheng1/test_tim                  |
| Lulu Zheng    | 2021-03-10 | merging test into parser                                        |
| Lulu Zheng    | 2021-03-10 | disallow float to be of 1. format                               |
| Lulu Zheng    | 2021-03-10 | fixed shift/reduce conflict                                     |
| Lulu Zheng    | 2021-03-10 | disallow float to be of 1. format                               |
| Yingjie Ling  | 2021-03-10 | Merge pull request #4 from luluzheng1/parser                    |
| Yingjie Ling  | 2021-03-10 | Create README.md                                                |
| Yingjie Ling  | 2021-03-10 | Update README.md                                                |
| Yingjie Ling  | 2021-03-10 | Update README.md                                                |
| nokada11      | 2021-03-11 | Add ^ to scanner                                                |
| nokada11      | 2021-03-11 | Remove vdecls from fbody and program, and move it to stmt       |
| nokada11      | 2021-03-11 | Add empty case for decls                                        |
| nokada11      | 2021-03-11 | Remove outdated code and comments                               |

|                |            |                                                        |
|----------------|------------|--------------------------------------------------------|
| Yingjie Ling   | 2021-03-11 | Merge pull request #6 from luluzheng1/restructure      |
| nokada11       | 2021-03-11 | Remove fst and snd definitions since they are built-in |
| nokada11       | 2021-03-11 | Add negative tests                                     |
| nokada11       | 2021-03-11 | Add error message to test output                       |
| Naoki Okada    | 2021-03-11 | Merge pull request #5 from luluzheng1/README           |
| gyawalisaurav  | 2021-03-11 | Added more test files                                  |
| Yingjie Ling   | 2021-03-11 | Merge pull request #7 from luluzheng1/restructure      |
| Yingjie Ling   | 2021-03-11 | Merge pull request #8 from luluzheng1/more_tests       |
| gyawalisaurav  | 2021-03-11 | Changed test function                                  |
| gyawalisaurav  | 2021-03-11 | Made runtests executable                               |
| Lulu Zheng     | 2021-03-11 | modify tests to new syntax                             |
| Lulu Zheng     | 2021-03-11 | changed syntax to using SEMI delimiters                |
| Lulu Zheng     | 2021-03-11 | fixed pretty printing                                  |
| Lulu Zheng     | 2021-03-11 | comments to tests                                      |
| Lulu Zheng     | 2021-03-11 | mentioned any incomplete work                          |
| Lulu Zheng     | 2021-03-11 | fixed formatting                                       |
| Lulu Zheng     | 2021-03-11 | added formatting and colored printing                  |
| Lulu Zheng     | 2021-03-11 | fixed indentation                                      |
| Lulu Zheng     | 2021-03-11 | fixed indentation                                      |
| Yingjie Ling   | 2021-03-11 | deleted unnecessary import                             |
| Yingjie Ling   | 2021-03-11 | Merge pull request #9 from luluzheng1/new_test         |
| Lulu Zheng     | 2021-03-11 | merge commit fixed changes                             |
| Lulu Zheng     | 2021-03-11 | added file type                                        |
| Lulu Zheng     | 2021-03-11 | removed file todo with README                          |
| Lulu Zheng     | 2021-03-11 | removed import work as per Tim's request               |
| Lulu Zheng     | 2021-03-11 | removed IMPORT                                         |
| Lulu Zheng     | 2021-03-11 | added coloring                                         |
| Yingjie Ling   | 2021-03-11 | Merge pull request #10 from luluzheng1/parser_test     |
| Lulu Zheng     | 2021-03-11 | added coloring                                         |
| Lulu           | 2021-03-11 | Merge pull request #11 from luluzheng1/parser_test     |
| Lulu Zheng     | 2021-03-11 | removed unnecessary code                               |
| Lulu           | 2021-03-11 | Merge pull request #12 from luluzheng1/parser_test     |
| gyawalisaurav  | 2021-03-20 | Added sexprs and sstmts                                |
| gyawalisaurav  | 2021-03-20 | Added string_of_sexpr                                  |
| gyawalisaurav  | 2021-03-20 | Added remaining pretty printing functions              |
| gyawalisaurav  | 2021-03-20 | Fixed pretty printing                                  |
| gyawalisaurav  | 2021-03-20 | Added symantic checking for few exps and stmts         |
| gyawalisaurav  | 2021-03-20 | Added SAST printing argument                           |
| Saurav Gyawali | 2021-03-20 | Merge pull request #13 from luluzheng1/sast            |
| Lulu Zheng     | 2021-03-23 | added unknown type for empty lists                     |
| Lulu Zheng     | 2021-03-23 | added listlit in expr                                  |
| Lulu Zheng     | 2021-03-23 | added exceptions.ml                                    |
| Lulu Zheng     | 2021-03-23 | added MismatchTypes exception                          |

|              |            |                                                                                          |
|--------------|------------|------------------------------------------------------------------------------------------|
| Lulu Zheng   | 2021-03-23 | added scope                                                                              |
| Lulu Zheng   | 2021-03-23 | added InvalidBinaryOperation exception                                                   |
| Lulu Zheng   | 2021-03-23 | added Binop expr                                                                         |
| Lulu Zheng   | 2021-03-23 | fixed usage from microc to team                                                          |
| Lulu Zheng   | 2021-03-23 | added unary operations and Invalid UopException                                          |
| Lulu Zheng   | 2021-03-24 | added ListAssign and NonListAccess, InvalidIndex exceptions                              |
| Lulu Zheng   | 2021-03-24 | added AssignOp                                                                           |
| Yingjie Ling | 2021-03-25 | Added duplicate error and void type error and their corresponding functions              |
| nokada11     | 2021-03-26 | Add skeleton for checking function semantics                                             |
| Yingjie Ling | 2021-03-27 | Merge pull request #15 from luluzheng1/semant_global                                     |
| Yingjie Ling | 2021-03-27 | added more tests                                                                         |
| Yingjie Ling | 2021-03-27 | fixed minor error                                                                        |
| Yingjie Ling | 2021-03-27 | cleaned up dir                                                                           |
| Yingjie Ling | 2021-03-28 | changed the AST print and fixed minor bug in runtests.py                                 |
| Yingjie Ling | 2021-03-28 | added SEMI for expr's and fixed indentation                                              |
| Yingjie Ling | 2021-03-28 | modified runtests.py for easier results checking                                         |
| Lulu Zheng   | 2021-03-28 | commit before switching branch                                                           |
| Lulu Zheng   | 2021-03-28 | changed indent function and fixed bugs in pretty printing                                |
| Lulu Zheng   | 2021-03-28 | fixed reverse evaluation bug                                                             |
| Lulu Zheng   | 2021-03-28 | update logs to most recent build                                                         |
| Lulu Zheng   | 2021-03-28 | fixed merge conflict                                                                     |
| Lulu Zheng   | 2021-03-28 | formatted document with ocamlformat                                                      |
| Lulu Zheng   | 2021-03-28 | added exception signatures for voidtype and duplicate exceptions                         |
| Lulu Zheng   | 2021-03-28 | formatted with ocamlformat                                                               |
| Lulu Zheng   | 2021-03-28 | added WrongNumberArgs and IllegalArgument exceptions and Call                            |
| Yingjie Ling | 2021-03-28 | added one hof test                                                                       |
| Lulu Zheng   | 2021-03-28 | added SliceExpr, End and NoExpr and created some exception used within these expressions |
| Lulu Zheng   | 2021-03-28 | ocaml formatter                                                                          |
| Yingjie Ling | 2021-03-29 | fixed minor bug in pretty print. COLON after function def is mandatory now               |
| Yingjie Ling | 2021-03-29 | added formal arguments type to pretty print                                              |
| Lulu Zheng   | 2021-03-29 | reversed evaluation order                                                                |
| Lulu Zheng   | 2021-03-29 | added comments                                                                           |
| Lulu Zheng   | 2021-03-29 | fixed listlit type to be recursive                                                       |
| Yingjie Ling | 2021-03-29 | Merge pull request #17 from luluzheng1/parser                                            |
| Lulu Zheng   | 2021-03-29 | fixed merge conflicts                                                                    |
| Lulu Zheng   | 2021-03-29 | fixed typo in binop exp                                                                  |
| Lulu Zheng   | 2021-03-29 | created test folder for semant.ml                                                        |
| Lulu Zheng   | 2021-03-29 | removed old test folder                                                                  |
| Lulu         | 2021-03-29 | Merge pull request #16 from luluzheng1/semant_expr                                       |
| Yingjie Ling | 2021-03-29 | implemented checkResults in runtests.py                                                  |
| Yingjie Ling | 2021-03-29 | fixed typo                                                                               |

|                |            |                                                                                                                                     |
|----------------|------------|-------------------------------------------------------------------------------------------------------------------------------------|
| gyawalisaurav  | 2021-03-30 | Added SDeclaration semant check                                                                                                     |
| gyawalisaurav  | 2021-03-30 | Merge branch 'codegen_expr' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> into semant_expr |
| Lulu Zheng     | 2021-03-30 | exception changed to open                                                                                                           |
| gyawalisaurav  | 2021-03-30 | Merge branch 'codegen_expr' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> into semant_expr |
| nokada11       | 2021-03-30 | Add built_in_decls, add_func and function_decls                                                                                     |
| gyawalisaurav  | 2021-03-30 | Fixed Sdeclaration and added an exception                                                                                           |
| Lulu Zheng     | 2021-03-30 | added tags                                                                                                                          |
| Lulu Zheng     | 2021-03-30 | added statements for return and blocks and codegen                                                                                  |
| Yingjie Ling   | 2021-03-30 | implemented senamntic check for functions                                                                                           |
| Lulu Zheng     | 2021-03-30 | finished statements                                                                                                                 |
| Lulu           | 2021-03-30 | Merge pull request #18 from luluzheng1/codegen_expr                                                                                 |
| Yingjie Ling   | 2021-03-30 | Merge branch 'semant_func' into semant_function_main                                                                                |
| Yingjie Ling   | 2021-03-30 | Merge pull request #19 from luluzheng1/semant_function_main                                                                         |
| Yingjie Ling   | 2021-03-30 | fixed multiple errors related to functions and changed the sfunc_decl                                                               |
| gyawalisaurav  | 2021-04-01 | Added basic codegen.ml                                                                                                              |
| gyawalisaurav  | 2021-04-01 | Updated top level for compiling                                                                                                     |
| gyawalisaurav  | 2021-04-01 | Added hello world team program                                                                                                      |
| gyawalisaurav  | 2021-04-01 | Updated hello world                                                                                                                 |
| Yingjie Ling   | 2021-04-01 | noted that length and append need to be treated as special cases                                                                    |
| gyawalisaurav  | 2021-04-01 | Added scoping and variable stuff                                                                                                    |
| gyawalisaurav  | 2021-04-01 | Added more scoping stuff                                                                                                            |
| Saurav Gyawali | 2021-04-01 | Merge pull request #20 from luluzheng1/semant_func                                                                                  |
| gyawalisaurav  | 2021-04-01 | Fixed Global Variables                                                                                                              |
| gyawalisaurav  | 2021-04-01 | Added block scope                                                                                                                   |
| gyawalisaurav  | 2021-04-01 | Added Boolean                                                                                                                       |
| gyawalisaurav  | 2021-04-01 | Added While statement                                                                                                               |
| gyawalisaurav  | 2021-04-01 | Added basic makefile                                                                                                                |
| Lulu Zheng     | 2021-04-02 | fixed evaluation order in semant                                                                                                    |
| Lulu Zheng     | 2021-04-02 | added declaration test                                                                                                              |
| Lulu           | 2021-04-02 | Merge pull request #21 from luluzheng1/semant_debug                                                                                 |
| Lulu Zheng     | 2021-04-02 | fixed reverse evaluation bug                                                                                                        |
| Lulu           | 2021-04-02 | Merge pull request #22 from luluzheng1/semant_debug                                                                                 |
| Saurav Gyawali | 2021-04-02 | Merge pull request #23 from luluzheng1/codegen                                                                                      |
| gyawalisaurav  | 2021-04-01 | Changed default to compiling                                                                                                        |
| Yingjie Ling   | 2021-04-01 | testing more hof's                                                                                                                  |
| Yingjie Ling   | 2021-04-01 | added a second hof test and now checking all built-in tests against gold standard                                                   |
| Yingjie Ling   | 2021-04-01 | fixed directory name typo                                                                                                           |

|               |            |                                                                             |
|---------------|------------|-----------------------------------------------------------------------------|
| Yingjie Ling  | 2021-04-02 | added more patterns for types and resolved ambiguous type                   |
| Yingjie Ling  | 2021-04-02 | Merge branch 'main' into test_hof                                           |
| Yingjie Ling  | 2021-04-02 | Merge pull request #24 from luluzheng1/test_hof                             |
| gyawalisaurav | 2021-04-02 | Added a dummy case to supress warning                                       |
| gyawalisaurav | 2021-04-02 | removed log file                                                            |
| gyawalisaurav | 2021-04-02 | Fixed waiting for more input bug                                            |
| Yingjie Ling  | 2021-04-02 | runtests.py now supports ast, sast, and codegen                             |
| Lulu Zheng    | 2021-04-02 | readded Makefile                                                            |
| Yingjie Ling  | 2021-04-02 | now running diff for codegen testmode if output differs from the standard   |
| Lulu Zheng    | 2021-04-02 | fixed annoying return not last bug                                          |
| Lulu Zheng    | 2021-04-02 | added codegen compile toplevel                                              |
| Lulu Zheng    | 2021-04-02 | Merge branch 'test_codegen' of github.com:luluzheng1/TEAM into test_codegen |
| Yingjie Ling  | 2021-04-02 | started working on list implementation in codegen                           |
| nokada11      | 2021-04-03 | Check if there is a return statement in non-void functions                  |
| nokada11      | 2021-04-03 | Add .vscode to gitignore                                                    |
| Lulu          | 2021-04-03 | Merge branch 'main' into test_codegen                                       |
| Lulu          | 2021-04-03 | Merge pull request #25 from luluzheng1/test_codegen                         |
| Lulu Zheng    | 2021-04-03 | getting rid of statements                                                   |
| Lulu          | 2021-04-03 | Merge pull request #26 from luluzheng1/bug                                  |
| Lulu Zheng    | 2021-04-03 | updated DRAFT of README                                                     |
| Lulu Zheng    | 2021-04-03 | fixed single file execution command                                         |
| Lulu Zheng    | 2021-04-03 | prettified README                                                           |
| Lulu          | 2021-04-03 | Merge pull request #27 from luluzheng1/README                               |
| Lulu Zheng    | 2021-04-03 | added float + int binop operations, changed add variables                   |
| Lulu Zheng    | 2021-04-03 | added test for binop on float and int                                       |
| Lulu Zheng    | 2021-04-03 | made minor changes                                                          |
| Lulu Zheng    | 2021-04-04 | added more binop operations                                                 |
| Lulu Zheng    | 2021-04-04 | added invalidfloatbinop and invalidintbinop                                 |
| Lulu Zheng    | 2021-04-04 | adding more binop tests                                                     |
| Lulu Zheng    | 2021-04-04 | fixed exception name error                                                  |
| Lulu Zheng    | 2021-04-04 | added unop in expr                                                          |
| Lulu Zheng    | 2021-04-04 | added tests for boolean binops                                              |
| Lulu Zheng    | 2021-04-04 | added SAssign                                                               |
| Lulu Zheng    | 2021-04-04 | changed file now that assign works                                          |
| Lulu Zheng    | 2021-04-04 | added NotFound exception                                                    |
| Lulu Zheng    | 2021-04-05 | cleans .o files                                                             |
| Lulu Zheng    | 2021-04-05 | fixed command for single file execution                                     |
| Lulu Zheng    | 2021-04-05 | added string slicing                                                        |
| Lulu Zheng    | 2021-04-05 | adding more tests                                                           |
| Lulu Zheng    | 2021-04-05 | added script to compile file using llc                                      |
| Lulu Zheng    | 2021-04-05 | added string library that contains string slicing function                  |

|               |            |                                                          |
|---------------|------------|----------------------------------------------------------|
| Lulu Zheng    | 2021-04-05 | changed substring to string                              |
| Lulu Zheng    | 2021-04-06 | removing junk files                                      |
| Lulu Zheng    | 2021-04-06 | removed shutil                                           |
| gyawalisaurav | 2021-04-07 | Added initial unboxed list implementation                |
| gyawalisaurav | 2021-04-07 | Changed to boxed list and added access function          |
| gyawalisaurav | 2021-04-07 | Started double slicing                                   |
| gyawalisaurav | 2021-04-07 | Fixed variable names in ocaml and for llvm code          |
| gyawalisaurav | 2021-04-08 | Changed iterative access to recursive                    |
| gyawalisaurav | 2021-04-08 | Added list slicing                                       |
| gyawalisaurav | 2021-04-08 | Added more slicing functionality for lists               |
| gyawalisaurav | 2021-04-08 | Added list assign                                        |
| gyawalisaurav | 2021-04-08 | Removed warnings                                         |
| gyawalisaurav | 2021-04-08 | Fixed [a:] slices                                        |
| gyawalisaurav | 2021-04-08 | Added string slicing                                     |
| gyawalisaurav | 2021-04-08 | Added null character to string                           |
| gyawalisaurav | 2021-04-09 | Fixed global variables                                   |
| gyawalisaurav | 2021-04-10 | Got rid of unnecessary initialization                    |
| Lulu Zheng    | 2021-04-10 | changed for to include string                            |
| Lulu Zheng    | 2021-04-10 | changed syntax in tests                                  |
| Lulu Zheng    | 2021-04-10 | changed how for is checked                               |
| Lulu Zheng    | 2021-04-10 | added new codegen tests                                  |
| Lulu Zheng    | 2021-04-10 | finished if statement                                    |
| Lulu Zheng    | 2021-04-10 | changed type types to less functional                    |
| Lulu Zheng    | 2021-04-10 | added debugging code                                     |
| Lulu Zheng    | 2021-04-10 | added new exceptions for codegen                         |
| Lulu Zheng    | 2021-04-10 | added For in codegen                                     |
| Lulu Zheng    | 2021-04-10 | added check for continue and break if they are in loop   |
| Lulu Zheng    | 2021-04-10 | finished continue and break                              |
| Lulu Zheng    | 2021-04-10 | adding tests for break and continue                      |
| Lulu Zheng    | 2021-04-10 | compile script                                           |
| Lulu Zheng    | 2021-04-10 | new makefile with debugging                              |
| Lulu Zheng    | 2021-04-10 | added dummy for SEnd                                     |
| gyawalisaurav | 2021-04-11 | list functions are only defined once                     |
| gyawalisaurav | 2021-04-11 | Changed variable names and format                        |
| Lulu Zheng    | 2021-04-15 | changed comments                                         |
| Lulu Zheng    | 2021-04-15 | added polymorphic print function                         |
| Lulu Zheng    | 2021-04-15 | removed string.c, doing slicing builtin                  |
| Lulu Zheng    | 2021-04-15 | changing print functions to all use print                |
| Lulu Zheng    | 2021-04-15 | changed to polymorphic print                             |
| Lulu Zheng    | 2021-04-15 | added NotInLoop exception                                |
| Lulu          | 2021-04-15 | Merge pull request #28 from luluzheng1/codegen_expr_stmt |
| Lulu Zheng    | 2021-04-15 | changed call format in semant                            |
| Lulu Zheng    | 2021-04-15 | added type checking for append                           |
| gyawalisaurav | 2021-04-15 | Got rid of unnecessary pointer                           |
| Lulu Zheng    | 2021-04-15 | adding test for append                                   |
| gyawalisaurav | 2021-04-15 | Fixed heap allocation instructions                       |
| Lulu Zheng    | 2021-04-15 | x                                                        |

|                |            |                                                                                                                                    |
|----------------|------------|------------------------------------------------------------------------------------------------------------------------------------|
| Lulu           | 2021-04-15 | Merge pull request #29 from luluzheng1/semant_append_length                                                                        |
| Lulu Zheng     | 2021-04-15 | added semant checking for append and length                                                                                        |
| Lulu Zheng     | 2021-04-15 | adding tests for append and length                                                                                                 |
| Lulu           | 2021-04-15 | Merge pull request #30 from luluzheng1/semant_append_length                                                                        |
| nokada11       | 2021-04-16 | Separate list indexing and slicing                                                                                                 |
| nokada11       | 2021-04-16 | Replace ID with expr when indexing lists                                                                                           |
| Lulu Zheng     | 2021-04-17 | adding more reference logs                                                                                                         |
| Lulu           | 2021-04-17 | Merge pull request #31 from luluzheng1/semant_append_length                                                                        |
| Lulu Zheng     | 2021-04-17 | added indentation for sast                                                                                                         |
| Lulu           | 2021-04-17 | Merge pull request #32 from luluzheng1/semant_append_length                                                                        |
| gyawalisaurav  | 2021-04-17 | Changed ID in slice to expr                                                                                                        |
| gyawalisaurav  | 2021-04-17 | Changed ID in slice to expr in ast                                                                                                 |
| gyawalisaurav  | 2021-04-17 | Changed ID in slice to expr in sast and semant                                                                                     |
| gyawalisaurav  | 2021-04-17 | fixed list and string slicing                                                                                                      |
| gyawalisaurav  | 2021-04-17 | removed unnecessary comments from the end                                                                                          |
| gyawalisaurav  | 2021-04-17 | Got rid of outer test                                                                                                              |
| gyawalisaurav  | 2021-04-17 | Merge branch 'main' into codegen_list                                                                                              |
| gyawalisaurav  | 2021-04-17 | Fixed grammar in parser                                                                                                            |
| nokada11       | 2021-04-17 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> into main               |
| nokada11       | 2021-04-18 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> into extended_testsuite |
| nokada11       | 2021-04-18 | Add extended testsuite                                                                                                             |
| Lulu Zheng     | 2021-04-18 | changed casting from const to build so that binop on variables work                                                                |
| gyawalisaurav  | 2021-04-18 | Added list assign                                                                                                                  |
| gyawalisaurav  | 2021-04-18 | Removed assignop                                                                                                                   |
| Saurav Gyawali | 2021-04-18 | Merge pull request #33 from luluzheng1/codegen_list                                                                                |
| gyawalisaurav  | 2021-04-18 | Removed elif                                                                                                                       |
| Saurav Gyawali | 2021-04-18 | Merge pull request #34 from luluzheng1/codegen_list                                                                                |
| gyawalisaurav  | 2021-04-18 | removed fdecl from build_stmt                                                                                                      |
| gyawalisaurav  | 2021-04-18 | Changed max width to 80 cols                                                                                                       |
| nokada11       | 2021-04-18 | Add more test cases                                                                                                                |
| gyawalisaurav  | 2021-04-18 | Added list length                                                                                                                  |
| gyawalisaurav  | 2021-04-18 | Added string length                                                                                                                |
| Saurav Gyawali | 2021-04-18 | Merge pull request #35 from luluzheng1/codegen_list                                                                                |
| gyawalisaurav  | 2021-04-18 | Fixed list assign with [1:] format                                                                                                 |
| Lulu Zheng     | 2021-04-18 | added hofs checking in semant                                                                                                      |
| Lulu Zheng     | 2021-04-18 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">github.com:luluzheng1/TEAM</a> into semant_hofs                |
| gyawalisaurav  | 2021-04-18 | Fixed list assign                                                                                                                  |
| gyawalisaurav  | 2021-04-18 | Added additional case for indexing                                                                                                 |
| Lulu Zheng     | 2021-04-18 | updating standard ref files                                                                                                        |



|                |            |                                                                                                                                    |
|----------------|------------|------------------------------------------------------------------------------------------------------------------------------------|
| Lulu Zheng     | 2021-04-18 | removed commented code                                                                                                             |
| Lulu           | 2021-04-18 | Merge pull request #36 from luluzheng1/semant_hofs                                                                                 |
| Saurav Gyawali | 2021-04-18 | Merge pull request #37 from luluzheng1/codegen_list                                                                                |
| nokada11       | 2021-04-19 | Resolve merge conflicts                                                                                                            |
| nokada11       | 2021-04-19 | Resolve merge conflicts                                                                                                            |
| Lulu Zheng     | 2021-04-19 | adding regex library and pcre2 for regex                                                                                           |
| Lulu Zheng     | 2021-04-19 | adding pcre libs                                                                                                                   |
| Lulu Zheng     | 2021-04-19 | adding pcre libs                                                                                                                   |
| Lulu Zheng     | 2021-04-19 | adding regex functions                                                                                                             |
| Lulu Zheng     | 2021-04-19 | added more regex functions                                                                                                         |
| Lulu Zheng     | 2021-04-19 | adding tests for regex                                                                                                             |
| Lulu Zheng     | 2021-04-19 | optimized replaceall                                                                                                               |
| gyawalisaurav  | 2021-04-20 | Changed functions to be first class                                                                                                |
| Saurav Gyawali | 2021-04-20 | Merge pull request #38 from luluzheng1/codegen_list                                                                                |
| gyawalisaurav  | 2021-04-20 | Merge branch 'main' into codegen_hof                                                                                               |
| Saurav Gyawali | 2021-04-20 | Merge pull request #39 from luluzheng1/codegen_hof                                                                                 |
| Lulu Zheng     | 2021-04-20 | fix merge conflict                                                                                                                 |
| Lulu Zheng     | 2021-04-20 | buggy :(                                                                                                                           |
| Lulu Zheng     | 2021-04-20 | adding getlist                                                                                                                     |
| Lulu Zheng     | 2021-04-20 | adding getlist                                                                                                                     |
| nokada11       | 2021-04-20 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> into extended_testsuite |
| Yingjie Ling   | 2021-04-20 | append                                                                                                                             |
| gyawalisaurav  | 2021-04-20 | Fixed list slice                                                                                                                   |
| nokada11       | 2021-04-20 | Edit tests                                                                                                                         |
| nokada11       | 2021-04-20 | Fix merge conflicts                                                                                                                |
| nokada11       | 2021-04-20 | Add built-in open to semant                                                                                                        |
| Yingjie Ling   | 2021-04-21 | append function first implementation                                                                                               |
| Yingjie Ling   | 2021-04-21 | updated append implementation seg faulting still                                                                                   |
| Yingjie Ling   | 2021-04-21 | added more stuff                                                                                                                   |
| Yingjie Ling   | 2021-04-21 | fixed problems in append                                                                                                           |
| Yingjie Ling   | 2021-04-21 | cleaned up directory                                                                                                               |
| Yingjie Ling   | 2021-04-21 | cleaned up directory                                                                                                               |
| Yingjie Ling   | 2021-04-21 | Merge branch 'main' into codegen_append                                                                                            |
| Yingjie Ling   | 2021-04-21 | Merge pull request #40 from luluzheng1/codegen_append                                                                              |
| Lulu Zheng     | 2021-04-21 | adding getlist.c                                                                                                                   |
| Yingjie Ling   | 2021-04-22 | print bool                                                                                                                         |
| Yingjie Ling   | 2021-04-22 | print bool                                                                                                                         |
| Yingjie Ling   | 2021-04-22 | Merge pull request #41 from luluzheng1/print_bool                                                                                  |
| Yingjie Ling   | 2021-04-23 | append is need to be able to append before the index                                                                               |
| gyawalisaurav  | 2021-04-23 | Added insert                                                                                                                       |
| Yingjie Ling   | 2021-04-23 | enabled insert                                                                                                                     |
| Yingjie Ling   | 2021-04-23 | Merge pull request #42 from luluzheng1/insert                                                                                      |
| Yingjie Ling   | 2021-04-23 | fixed typo                                                                                                                         |

|                |            |                                                                                                                                    |
|----------------|------------|------------------------------------------------------------------------------------------------------------------------------------|
| Yingjie Ling   | 2021-04-23 | Merge pull request #43 from luluzheng1/insert                                                                                      |
| Yingjie Ling   | 2021-04-23 | adopted Saurav's solution                                                                                                          |
| Yingjie Ling   | 2021-04-23 | Merge branch 'main' into codegen_append_reverse                                                                                    |
| Yingjie Ling   | 2021-04-23 | Merge pull request #44 from luluzheng1/codegen_append_reverse                                                                      |
| nokada11       | 2021-04-24 | Add readline to semant                                                                                                             |
| nokada11       | 2021-04-24 | Add readline to codegen                                                                                                            |
| nokada11       | 2021-04-24 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> into extended_testsuite |
| nokada11       | 2021-04-24 | Update tests                                                                                                                       |
| Lulu Zheng     | 2021-04-24 | fixed minor bug for comments                                                                                                       |
| nokada11       | 2021-04-24 | Update expected output for bad tests                                                                                               |
| nokada11       | 2021-04-24 | Don't pipe llvm code to lli for bad tests                                                                                          |
| nokada11       | 2021-04-24 | Add fileio.c                                                                                                                       |
| Yingjie Ling   | 2021-04-24 | added insert and changed top level                                                                                                 |
| Yingjie Ling   | 2021-04-24 | ensured that the codegen gets the actual type of the list                                                                          |
| Yingjie Ling   | 2021-04-24 | implemented range                                                                                                                  |
| Yingjie Ling   | 2021-04-24 | insert append are mutating list and range is fully functional                                                                      |
| Lulu Zheng     | 2021-04-24 | added code for findall                                                                                                             |
| Lulu Zheng     | 2021-04-24 | finished findall                                                                                                                   |
| Lulu Zheng     | 2021-04-24 | removed getlist as a dependency                                                                                                    |
| Lulu Zheng     | 2021-04-24 | removed getlist                                                                                                                    |
| Lulu Zheng     | 2021-04-24 | removed getlist                                                                                                                    |
| Lulu Zheng     | 2021-04-24 | adding tests for findall                                                                                                           |
| Lulu Zheng     | 2021-04-24 | accidental delete :/ poop                                                                                                          |
| Lulu Zheng     | 2021-04-24 | fixed merge conflicts                                                                                                              |
| Lulu Zheng     | 2021-04-24 | updated tests                                                                                                                      |
| Lulu           | 2021-04-24 | Merge pull request #45 from luluzheng1/semant_hofs                                                                                 |
| nokada11       | 2021-04-26 | Add write function to fileio.c                                                                                                     |
| nokada11       | 2021-04-26 | Add write and close                                                                                                                |
| Yingjie Ling   | 2021-04-26 | enabled for loop and modified append, insert                                                                                       |
| Yingjie Ling   | 2021-04-26 | enabled char printing                                                                                                              |
| nokada11       | 2021-04-26 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> into main               |
| Yingjie Ling   | 2021-04-26 | noexpr fixed                                                                                                                       |
| Yingjie Ling   | 2021-04-26 | for supports string now                                                                                                            |
| Yingjie Ling   | 2021-04-26 | fixed string for                                                                                                                   |
| Yingjie Ling   | 2021-04-26 | cleaned up directory                                                                                                               |
| gyawalisaurav  | 2021-04-26 | Added declaration without initialization feature                                                                                   |
| Saurav Gyawali | 2021-04-26 | Merge pull request #47 from luluzheng1/declaration                                                                                 |
| nokada11       | 2021-04-26 | Add semant check for c-style print                                                                                                 |
| nokada11       | 2021-04-26 | Add exceptions we need to handle in print part of semant                                                                           |
| nokada11       | 2021-04-26 | Add code to call c-style print                                                                                                     |
| gyawalisaurav  | 2021-04-26 | added special case for declaring list and string without initialization                                                            |

|                |            |                                                                                                                                    |
|----------------|------------|------------------------------------------------------------------------------------------------------------------------------------|
| Saurav Gyawali | 2021-04-26 | Merge pull request #49 from luluzheng1/declaration                                                                                 |
| nokada11       | 2021-04-26 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> into main               |
| Naoki Okada    | 2021-04-26 | Add Extended Testsuite section to README                                                                                           |
| Yingjie Ling   | 2021-04-26 | added split standard library functino                                                                                              |
| nokada11       | 2021-04-27 | Resolve merge conflicts                                                                                                            |
| nokada11       | 2021-04-27 | Update list test                                                                                                                   |
| nokada11       | 2021-04-27 | Update string test                                                                                                                 |
| nokada11       | 2021-04-27 | Update variables test                                                                                                              |
| Naoki Okada    | 2021-04-27 | Merge pull request #48 from luluzheng1/codegen_print                                                                               |
| nokada11       | 2021-04-27 | Modify list test                                                                                                                   |
| Yingjie Ling   | 2021-04-27 | Merge branch 'insert_append_for_library' into main+Tim                                                                             |
| nokada11       | 2021-04-27 | Modify semant check for print                                                                                                      |
| Lulu Zheng     | 2021-04-27 | removed unknown from codegen                                                                                                       |
| Lulu Zheng     | 2021-04-27 | adding instructions for installing PCRE2                                                                                           |
| Naoki Okada    | 2021-04-27 | Merge branch 'main+Tim' into fix-print                                                                                             |
| Naoki Okada    | 2021-04-27 | Merge pull request #52 from luluzheng1/fix-print                                                                                   |
| Yingjie Ling   | 2021-04-27 | fixed some stuff                                                                                                                   |
| Yingjie Ling   | 2021-04-27 | Merge branch 'main+Tim' of <a href="https://github.com/luluzheng1/TEAM">github.com:luluzheng1/TEAM</a> into main+Tim               |
| nokada11       | 2021-04-27 | Fix print in codegen                                                                                                               |
| gyawalisaaurav | 2021-04-27 | Fixed noexpr                                                                                                                       |
| Yingjie Ling   | 2021-04-27 | added string_to_list                                                                                                               |
| Yingjie Ling   | 2021-04-27 | Merge pull request #53 from luluzheng1/main+Tim                                                                                    |
| gyawalisaaurav | 2021-04-27 | Fixed for loop for string                                                                                                          |
| Saurav Gyawali | 2021-04-27 | Merge pull request #54 from luluzheng1/declaration_fix                                                                             |
| gyawalisaaurav | 2021-04-27 | changed declaration positions to beginning of func                                                                                 |
| Lulu Zheng     | 2021-04-27 | making changes to list                                                                                                             |
| Saurav Gyawali | 2021-04-27 | Merge pull request #55 from luluzheng1/declaration_fix                                                                             |
| Saurav Gyawali | 2021-04-27 | Merge pull request #56 from luluzheng1/declaration_fix                                                                             |
| gyawalisaaurav | 2021-04-27 | Fixed memcpy name                                                                                                                  |
| nokada11       | 2021-04-27 | Resolver merge conflicts                                                                                                           |
| nokada11       | 2021-04-27 | Modify arith test                                                                                                                  |
| nokada11       | 2021-04-27 | Modify function test                                                                                                               |
| nokada11       | 2021-04-27 | Modify while and variables test                                                                                                    |
| nokada11       | 2021-04-27 | Modify formattedPrint test                                                                                                         |
| nokada11       | 2021-04-27 | Add append to list test                                                                                                            |
| Saurav Gyawali | 2021-04-27 | Merge pull request #58 from luluzheng1/declaration_fix                                                                             |
| nokada11       | 2021-04-27 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> into extended_testsuite |
| nokada11       | 2021-04-27 | Disable regex in Makefile for now                                                                                                  |

|              |            |                                                               |
|--------------|------------|---------------------------------------------------------------|
| nokada11     | 2021-04-27 | Modify string test                                            |
| nokada11     | 2021-04-27 | Update README                                                 |
| Naoki Okada  | 2021-04-27 | Merge pull request #57 from<br>luluzheng1/extended_testsuite  |
| Lulu Zheng   | 2021-04-27 | more changes to accomodate list                               |
| Lulu Zheng   | 2021-04-27 | merge conflicts                                               |
| Yingjie Ling | 2021-04-28 | added some COOL standard library functions                    |
| Yingjie Ling | 2021-04-29 | cleaned up list.tm                                            |
| Yingjie Ling | 2021-04-29 | reverse implementation                                        |
| Lulu Zheng   | 2021-04-29 | making progress..                                             |
| Lulu Zheng   | 2021-04-29 | adding resolve                                                |
| Lulu Zheng   | 2021-04-29 | minor fix                                                     |
| Lulu Zheng   | 2021-04-29 | changing syntax of files                                      |
| Yingjie Ling | 2021-04-29 | fixed reverse for list                                        |
| Yingjie Ling | 2021-04-29 | cleaned up directory                                          |
| Yingjie Ling | 2021-04-29 | Merge pull request #59 from<br>luluzheng1/reverse             |
| Yingjie Ling | 2021-04-29 | cleaned up code                                               |
| Yingjie Ling | 2021-04-29 | .DS_Store banished                                            |
| Yingjie Ling | 2021-04-29 | DS_store banished                                             |
| Yingjie Ling | 2021-04-29 | deleted test.py                                               |
| Yingjie Ling | 2021-04-29 | resolved conflict                                             |
| Yingjie Ling | 2021-04-29 | Merge pull request #62 from<br>luluzheng1/standardLibrary     |
| Lulu Zheng   | 2021-04-30 | syntax change                                                 |
| Lulu Zheng   | 2021-04-30 | adding new tests for list type                                |
| Lulu Zheng   | 2021-04-30 | more list resolve                                             |
| Lulu Zheng   | 2021-04-30 | change syntax                                                 |
| Lulu Zheng   | 2021-04-30 | fix list resolve                                              |
| Lulu Zheng   | 2021-04-30 | fixed stntax in tests                                         |
| Lulu Zheng   | 2021-04-30 | changing syntax for lists                                     |
| Lulu Zheng   | 2021-04-30 | fixing tests and ref                                          |
| Lulu Zheng   | 2021-04-30 | removed TODO statements                                       |
| Lulu Zheng   | 2021-04-30 | more tests                                                    |
| Lulu Zheng   | 2021-05-01 | writing list stdlib funcs                                     |
| Yingjie Ling | 2021-05-01 | fixed length bug                                              |
| Lulu Zheng   | 2021-05-01 | adding better error messages, prepending<br>stdlib to code    |
| Lulu Zheng   | 2021-05-01 | adding .PHONY                                                 |
| Lulu Zheng   | 2021-05-01 | added contains                                                |
| Lulu Zheng   | 2021-05-01 | added contains functions                                      |
| Lulu Zheng   | 2021-05-01 | prepending stdlib files                                       |
| Lulu Zheng   | 2021-05-01 | fixed merge conflicts                                         |
| Lulu Zheng   | 2021-05-01 | fixed list syntax errors                                      |
| Lulu Zheng   | 2021-05-01 | changed stdlib file name                                      |
| Lulu Zheng   | 2021-05-03 | adding contains                                               |
| Lulu Zheng   | 2021-05-04 | added more list resolve stuff                                 |
| Lulu Zheng   | 2021-05-04 | remove function                                               |
| Lulu Zheng   | 2021-05-04 | fixed bug with incorrect order of statements in<br>else block |
| Lulu Zheng   | 2021-05-04 | added remove functions                                        |
| Lulu Zheng   | 2021-05-04 | prepend list library functions                                |

|               |            |                                                                                                            |
|---------------|------------|------------------------------------------------------------------------------------------------------------|
| Lulu Zheng    | 2021-05-04 | adding test for contains                                                                                   |
| Lulu Zheng    | 2021-05-04 | more list tests                                                                                            |
| Lulu Zheng    | 2021-05-04 | removes test                                                                                               |
| Lulu Zheng    | 2021-05-04 | removed unnecessary resolving                                                                              |
| Lulu Zheng    | 2021-05-04 | changed to new list syntax                                                                                 |
| Lulu Zheng    | 2021-05-04 | removed -r option                                                                                          |
| Lulu          | 2021-05-04 | Merge pull request #63 from luluzheng1/unknown_type                                                        |
| Lulu Zheng    | 2021-05-04 | fixed function in for loop                                                                                 |
| Lulu          | 2021-05-04 | Merge pull request #64 from luluzheng1/unknown_type                                                        |
| Yingjie Ling  | 2021-05-04 | changed lli to llc                                                                                         |
| Yingjie Ling  | 2021-05-04 | cleaned up directory and banished log files                                                                |
| nokada11      | 2021-05-04 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> |
| nokada11      | 2021-05-05 | Resolve merge conflicts                                                                                    |
| nokada11      | 2021-05-05 | Remove string from Makefile                                                                                |
| Naoki Okada   | 2021-05-05 | Merge pull request #46 from luluzheng1/fileio                                                              |
| nokada11      | 2021-05-05 | Merge branch 'main' of <a href="https://github.com/luluzheng1/TEAM">https://github.com/luluzheng1/TEAM</a> |
| nokada11      | 2021-05-05 | Add string concat with +                                                                                   |
| Naoki Okada   | 2021-05-05 | Merge pull request #65 from luluzheng1/concat-string                                                       |
| Yingjie Ling  | 2021-05-05 | Merge remote-tracking branch 'origin' into pythonScript                                                    |
| Yingjie Ling  | 2021-05-05 | modified runtest.py and added contains                                                                     |
| Lulu Zheng    | 2021-05-06 | added string join function                                                                                 |
| Lulu Zheng    | 2021-05-06 | removed contains and added comments                                                                        |
| Lulu Zheng    | 2021-05-06 | added join instantiation                                                                                   |
| Lulu          | 2021-05-06 | Merge pull request #66 from luluzheng1/unknown_type                                                        |
| Lulu Zheng    | 2021-05-06 | added new exceptions and comments                                                                          |
| Lulu          | 2021-05-06 | Merge pull request #67 from luluzheng1/unknown_type                                                        |
| nokada11      | 2021-05-07 | Add semant for string + char                                                                               |
| nokada11      | 2021-05-07 | Add codegen for string + char                                                                              |
| nokada11      | 2021-05-07 | Add semant for char + string                                                                               |
| nokada11      | 2021-05-07 | Implement string + char and char + string in codegen                                                       |
| Naoki Okada   | 2021-05-07 | Merge pull request #68 from luluzheng1/string+char                                                         |
| nokada11      | 2021-05-07 | Fix compilation step in compile.sh                                                                         |
| Naoki Okada   | 2021-05-07 | Merge pull request #69 from luluzheng1/fix-compilescrip                                                    |
| nokada11      | 2021-05-07 | Remove fclose from write                                                                                   |
| gyawalisaurav | 2021-05-08 | Added list concat                                                                                          |
| gyawalisaurav | 2021-05-08 | Fixed warning                                                                                              |
| Lulu Zheng    | 2021-05-08 | uncommented string_reverse                                                                                 |
| Lulu          | 2021-05-08 | Merge pull request #70 from luluzheng1/unknown_type                                                        |
| Naoki Okada   | 2021-05-08 | Merge pull request #71 from luluzheng1/fixes                                                               |
| Lulu Zheng    | 2021-05-08 | updating tests                                                                                             |

|              |            |                                                                             |
|--------------|------------|-----------------------------------------------------------------------------|
| Lulu         | 2021-05-08 | Merge pull request #72 from luluzheng1/unknown_type                         |
| Lulu Zheng   | 2021-05-08 | Merge branch 'main' into pythonScript                                       |
| Yingjie Ling | 2021-05-08 | python script + contains + == for string                                    |
| Lulu Zheng   | 2021-05-08 | merge conflict                                                              |
| Lulu Zheng   | 2021-05-08 | fixed comment bug                                                           |
| Yingjie Ling | 2021-05-08 | used subprocess and compile.sh                                              |
| Yingjie Ling | 2021-05-08 | Merge branch 'pythonScript' of github.com:luluzheng1/TEAM into pythonScript |
| Lulu Zheng   | 2021-05-08 | removing print_list                                                         |
| Lulu Zheng   | 2021-05-08 | Merge branch 'pythonScript' of github.com:luluzheng1/TEAM into pythonScript |
| Yingjie Ling | 2021-05-08 | cleaned up python script                                                    |
| Yingjie Ling | 2021-05-08 | Merge branch 'pythonScript' of github.com:luluzheng1/TEAM into pythonScript |
| Yingjie Ling | 2021-05-08 | Merge pull request #73 from luluzheng1/pythonScript                         |
| nokada11     | 2021-05-08 | Update sast test gold standards                                             |
| nokada11     | 2021-05-08 | Update ast test gold standards                                              |
| nokada11     | 2021-05-08 | Change flag for semantic check to -s                                        |
| nokada11     | 2021-05-08 | Remove fileio stuff produced at compile time                                |
| nokada11     | 2021-05-08 | Add built-in library stuff produced at compile time to gitignore            |
| nokada11     | 2021-05-08 | Add instruction to remove any dSYM folders                                  |
| nokada11     | 2021-05-08 | Add tests                                                                   |
| nokada11     | 2021-05-08 | Add txt files for file i/o tests                                            |
| nokada11     | 2021-05-08 | Correct contains test                                                       |
| nokada11     | 2021-05-08 | Update runtests                                                             |
| Naoki Okada  | 2021-05-08 | Merge pull request #74 from luluzheng1/test                                 |
| Lulu Zheng   | 2021-05-08 | removing -r                                                                 |
| Yingjie Ling | 2021-05-08 | added ASCII art                                                             |
| Lulu Zheng   | 2021-05-08 | fixed slice assign type error                                               |
| Lulu         | 2021-05-08 | Merge pull request #75 from luluzheng1/list_add                             |
| Lulu Zheng   | 2021-05-08 | organizing directories to be cleaner                                        |
| Lulu         | 2021-05-08 | Merge pull request #76 from luluzheng1/reorganization                       |
| Lulu Zheng   | 2021-05-08 | moving test_text_files inside of tests directory                            |
| Lulu         | 2021-05-08 | Merge pull request #77 from luluzheng1/reorganization                       |
| Lulu Zheng   | 2021-05-09 | changed commands in readme                                                  |
| Lulu         | 2021-05-09 | Merge pull request #78 from luluzheng1/reorganization                       |
| Lulu Zheng   | 2021-05-09 | adding make clean in pcre2                                                  |
| Lulu Zheng   | 2021-05-09 | fixed resolve back to generic error                                         |

## 11.2 Complete Code Listing

### 11.2.1 standard\_library/list.tm

```
1  /* Author: Yingjie L., Lulu Z. */
2  bool contains(list l, list items):
3      list res = [];
4      for item in items:
5          res = append(res, false);
6          for ref in l:
7              if ref == item:
8                  res[length(res) - 1] = true;
9              end
10         end
11     end
12
13     for r in res:
14         if not r:
15             return false;
16         end
17     end
18     return true;
19 end
20
21 contains([1, 2], [1]);
22 contains(["hello", "world"], ["hello"]);
23 contains(['h', 'e'], ['h']);
24 contains([true, false], [true]);
25 contains([1.1, 1.2], [1.1]);
26
27 list remove_int(list l, int elem, bool all):
28     list retlist = [];
29     int i = 0;
30     int len = length(l);
31     int remove_index = 0;
32
33     if all:
34         while i < len:
35             if l[i] == elem:
36                 i += 1;
37                 continue;
38             end
39             retlist = append(retlist, l[i]);
40             i += 1;
41         end
42     else:
43         while i < len:
44             if l[i] == elem:
45                 remove_index = i;
46                 break;
47             end
48
49             retlist = append(retlist, l[i]);
50
51             i += 1;
52         end
53         /* can be optimized with list concat */
54         if i != len:
55             for index in remove_index+1..len:
56                 retlist = append(retlist, l[index]);
```

```

57         end
58     end
59 end
60
61     return retlist;
62 end
63
64
65 list remove_float(list l, float elem, bool all):
66     list retlist = [];
67     int i = 0;
68     int len = length(l);
69     int remove_index = 0;
70
71     if all:
72         while i < len:
73             if l[i] == elem:
74                 i += 1;
75                 continue;
76             end
77             retlist = append(retlist, l[i]);
78             i += 1;
79         end
80     else:
81         while i < len:
82             if l[i] == elem:
83                 remove_index = i;
84                 break;
85             end
86
87             retlist = append(retlist, l[i]);
88
89             i += 1;
90         end
91         /* can be optimized with list concat */
92         if i != len:
93             for index in remove_index+1..len:
94                 retlist = append(retlist, l[index]);
95             end
96         end
97     end
98
99     return retlist;
100 end
101
102 list remove_bool(list l, bool elem, bool all):
103     list retlist = [];
104     int i = 0;
105     int len = length(l);
106     int remove_index = 0;
107
108     if all:
109         while i < len:
110             if l[i] == elem:
111                 i += 1;
112                 continue;
113             end
114             retlist = append(retlist, l[i]);
115             i += 1;

```



```

116         end
117     else:
118         while i < len:
119             if l[i] == elem:
120                 remove_index = i;
121                 break;
122             end
123
124             retlist = append(retlist, l[i]);
125
126             i += 1;
127         end
128         /* can be optimized with list concat */
129         if i != len:
130             for index in remove_index+1..len:
131                 retlist = append(retlist, l[index]);
132             end
133         end
134     end
135
136     return retlist;
137 end
138
139
140 list remove_char(list l, char elem, bool all):
141     list retlist = [];
142     int i = 0;
143     int len = length(l);
144     int remove_index = 0;
145
146     if all:
147         while i < len:
148             if l[i] == elem:
149                 i += 1;
150                 continue;
151             end
152             retlist = append(retlist, l[i]);
153             i += 1;
154         end
155     else:
156         while i < len:
157             if l[i] == elem:
158                 remove_index = i;
159                 break;
160             end
161
162             retlist = append(retlist, l[i]);
163
164             i += 1;
165         end
166         /* can be optimized with list concat */
167         if i != len:
168             for index in remove_index+1..len:
169                 retlist = append(retlist, l[index]);
170             end
171         end
172     end
173
174     return retlist;

```

```

175 end
176
177 list remove_string(list l, string elem, bool all):
178     list retlist = [];
179     int i = 0;
180     int len = length(l);
181     int remove_index = 0;
182
183     if all:
184         while i < len:
185             if strcmp(l[i], elem):
186                 i += 1;
187                 continue;
188             end
189             retlist = append(retlist, l[i]);
190             i += 1;
191         end
192     else:
193         while i < len:
194             if strcmp(l[i], elem):
195                 remove_index = i;
196                 break;
197             end
198
199             retlist = append(retlist, l[i]);
200
201             i += 1;
202         end
203         /* can be optimized with list concat */
204         if i != len:
205             for index in remove_index+1..len:
206                 retlist = append(retlist, l[index]);
207             end
208         end
209     end
210
211     return retlist;
212 end
213
214 remove_int([1], 1, true);
215 remove_float([1.5], 1.5, false);
216 remove_bool([true], true, true);
217 remove_char(['a'], 'a', true);
218 remove_string(["hello"], "what", true);

```

### 11.2.2 standard\_library/string.tm

```

1 /* Author: Yingjie L. */
2 list ASCII = [
3     'a', 'b', 'c', 'd', 'e',
4     'f', 'g', 'h', 'i', 'j',
5     'k', 'l', 'm', 'n', 'o',
6     'p', 'q', 'r', 's', 't',
7     'u', 'v', 'w', 'x', 'y',
8     'z',
9     'A', 'B', 'C', 'D', 'E',
10    'F', 'G', 'H', 'I', 'J',
11    'K', 'L', 'M', 'N', 'O',
12    'P', 'Q', 'R', 'S', 'T',

```

```

13         'U', 'V', 'W', 'X', 'Y',
14         'Z'
15     ];
16
17 list split(string text, char separator):
18     list result = [];
19     int text_length = length(text);
20     int left = 0;
21     int right = 0;
22     while right < text_length:
23         if text[right] == separator:
24             result = append(result, text[left:right]);
25             right = right + 1;
26             left = right;
27         else:
28             right = right + 1;
29         end
30     end
31     result = append(result, text[left:right]);
32     return result;
33 end
34
35
36 string join(list text_list, string connector):
37     string res = "";
38     int list_length = length(text_list);
39     for index in 0..(list_length - 1):
40         res = (res + text_list[index] + connector);
41     end
42     res = res + text_list[list_length - 1];
43     return res;
44 end
45
46 join(["hello"], "");
47
48 string string_reverse(string text):
49     string res = "";
50     int string_length = length(text);
51     int index = string_length - 1;
52     while index >= 0:
53         res = res + text[index];
54         index -= 1;
55     end
56     return res;
57 end
58
59 bool startswith(string text, char s):
60     return s == text[0];
61 end
62
63 bool endswith(string text, char e):
64     int string_length = length(text);
65     return e == text[string_length - 1];
66 end
67
68 list string_to_list(string text):
69     list result = [];
70     for c in text:
71         result = append(result, c);

```

```

72     end
73     return result;
74 end
75
76 char lower(char c):
77     int index = -1;
78     for c_ref in ASCII:
79         index += 1;
80         if c == c_ref:
81             if index < 26:
82                 return c;
83             else:
84                 return ASCII[index - 26];
85             end
86         end
87     end
88     return c;
89 end
90
91 char upper(char c):
92     int index = -1;
93     for c_ref in ASCII:
94         index += 1;
95         if c == c_ref:
96             if index > 25:
97                 return c;
98             else:
99                 return ASCII[index + 26];
100            end
101        end
102    end
103    return c;
104 end
105
106 bool strcmp(string str1, string str2):
107     if length(str1) != length(str2):
108         return false;
109     end
110
111     int i = 0;
112     while i < length(str1):
113         char c1 = str1[i];
114         char c2 = str2[i];
115         if c1 != c2:
116             return false;
117         end
118         i += 1;
119     end
120     return true;
121 end

```

### 11.2.3 src/scanner.ml1

```

1 (* Ocamllex scanner for TEAM *)
2 (* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
3
4 {
5     open Parser
6     exception Scan_error of string

```

```

7 let fail ch = raise (Scan_error (Char.escaped ch))
8 let unescape s =
9   Scanf.sscanf ("\" ^ s ^ "\"") "%S%!" (fun x -> x)
10 }
11
12 let digit = ['0' - '9']
13 let digits = digit+
14 let float = digits '.' digits
15 let ascii = ([ ' '-'!' '#'-'[ ' ']' '- '~' ])
16 let char = "'" (ascii | digit) "'"
17 let escaped_char = "\\\" ['\\' ' ' ' ' 'n' 'r' 't']
18 let string = "'" ( (ascii | escaped_char)* as s) "'"
19 rule token = parse
20   [ ' ' '\t' '\r' '\n' ] { token lexbuf } (* Whitespace *)
21   | ';' { SEMI }
22   | "/*" { comment lexbuf } (* Blocky Comments *)
23   | "//" { slcomment lexbuf } (* Single line Comments *)
24   | '(' { LPAREN }
25   | ')' { RPAREN }
26   | '[' { LSQUARE }
27   | ']' { RSQUARE }
28   | ":" { COLON }
29   | ',' { COMMA }
30   | '+' { PLUS }
31   | '-' { MINUS }
32   | '*' { TIMES }
33   | '/' { DIVIDE }
34   | '%' { MOD }
35   | '^' { EXP }
36   | "+=" { ADDASN }
37   | "-=" { SUBASN }
38   | "*=" { MULASN }
39   | "/=" { DIVASN }
40   | "%=" { MODASN }
41   | '=' { ASSIGN }
42   | ".." { RANGE }
43   | "==" { EQ }
44   | "!=" { NEQ }
45   | '<' { LT }
46   | "<=" { LEQ }
47   | '>' { GT }
48   | ">=" { GEQ }
49   | "and" { AND }
50   | "or" { OR }
51   | "not" { NOT }
52   | "if" { IF }
53   | "elif" { ELSEIF }
54   | "else" { ELSE }
55   | "for" { FOR }
56   | "in" { IN }
57   | "while" { WHILE }
58   | "break" { BREAK }
59   | "continue" { CONTINUE }
60   | "return" { RETURN }
61   | "end" { END }
62   | "int" { INT }
63   | "float" { FLOAT }
64   | "bool" { BOOL }
65   | "string" { STRING }

```

```

66 | "char"    { CHAR }
67 | "void"    { VOID }
68 | "file"    { FILE }
69 | "true"    { BLIT(true) }
70 | "false"   { BLIT(false) }
71 | "list"    { LIST }
72 | "->"      { ARROW }
73 | digits as lxm { LITERAL(int_of_string lxm) }
74 | float as lxm { FLIT(float_of_string lxm) }
75 | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
76 | char as lxm { CLIT( String.get lxm 1 ) }
77 | string    { SLIT(unescape s) }
78 | eof { EOF }
79 | _ as char { fail char }
80
81
82 and comment = parse
83     "*/"    { token lexbuf }
84 | _        { comment lexbuf }
85
86 and slcomment = parse
87     '\n'    { token lexbuf }
88 | eof      { token lexbuf }
89 | _        { slcomment lexbuf }

```

#### 11.2.4 src/parser.mly

```

1 /* Ocaml yacc parser for TEAM */
2 /* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. */
3 %{
4 open Ast
5 %}
6
7 %token LPAREN RPAREN LSQUARE RSQUARE COMMA ARROW COLON SEMI
8 %token PLUS MINUS TIMES DIVIDE MOD EXP
9 %token ADDASN SUBASN MULASN DIVASN MODASN ASSIGN NOT
10 %token EQ NEQ LT LEQ GT GEQ RANGE AND OR
11 %token IF ELSEIF ELSE FOR IN WHILE BREAK CONTINUE RETURN END
12 %token INT FLOAT BOOL STRING CHAR VOID
13 %token LIST FILE
14 %token <bool> BLIT
15 %token <int> LITERAL
16 %token <float> FLIT
17 %token <string> ID
18 %token <char> CLIT
19 %token <string> SLIT
20 %token EOF
21
22 %start program
23 %type <Ast.program> program
24
25 %left ARROW
26 %right ASSIGN ADDASN SUBASN MULASN DIVASN MODASN
27 %left OR
28 %left AND
29 %left EQ NEQ
30 %left LT GT LEQ GEQ
31 %nonassoc RANGE
32 %left PLUS MINUS

```

```

33 %left TIMES DIVIDE MOD
34 %left EXP
35 %right NOT
36
37 %%
38
39 program:
40   decls EOF { ( List.rev (fst $1), List.rev (snd $1)) }
41
42 decls:
43   /* nothing */ { ([], []) }
44   | decls fdecl { (($2 :: fst $1), snd $1) }
45   | decls stmt { (fst $1, ($2 :: snd $1)) }
46
47 fdecl:
48   typ ID LPAREN formals_opt RPAREN COLON stmt_list END
49   { {
50     typ = $1;
51     fname = $2;
52     formals = $4;
53     body = List.rev $7;
54   } }
55 formals_opt:
56   /* nothing */ { [] }
57   | formals_list { List.rev $1 }
58
59 formals_list:
60   typ ID { [($1, $2)] }
61   | formals_list COMMA typ ID { ($3, $4) :: $1 }
62
63 typ:
64   INT { Int }
65   | FLOAT { Float }
66   | BOOL { Bool }
67   | CHAR { Char }
68   | STRING { String }
69   | VOID { Void }
70   | FILE { File }
71   | LIST { List Unknown }
72   | typ_list ARROW typ { Func($1, $3) }
73
74 typ_list_helper:
75   typ { [$1] }
76   | typ_list_helper COMMA typ { $3 :: $1 }
77
78 typ_list:
79   LPAREN RPAREN { [] }
80   | LPAREN typ_list_helper RPAREN { List.rev $2 }
81
82 vdecl:
83   typ ID ASSIGN expr { Declaration($1, $2, $4) }
84   | typ ID { Declaration($1, $2, Noexpr) }
85
86 stmt_list:
87   /* nothing */ { [] }
88   | stmt_list stmt { $2 :: $1 }
89
90 stmt:
91   | vdecl SEMI { $1 }

```

```

92 | expr SEMI { Expr $1 }
93 | RETURN expr_opt SEMI { Return $2 }
94 | IF internal_if { $2 }
95 | FOR ID IN expr COLON stmt_list END { For($2, $4, Block(List.rev $6)) }
96 | WHILE expr COLON stmt_list END { While($2, Block(List.rev $4)) }
97 | BREAK SEMI { Break }
98 | CONTINUE SEMI { Continue }
99
100 internal_if:
101   expr COLON stmt_list else_list END { If($1, Block(List.rev $3), $4) }
102
103 else_list:
104   /* nothing */ { Block([]) }
105   | ELSEIF expr COLON stmt_list else_list { If($2, Block(List.rev $4), $5) }
106   | ELSE COLON stmt_list { Block(List.rev $3) }
107
108 expr_opt:
109   /* nothing */ { Noexpr }
110   | expr { $1 }
111
112 primary:
113   LITERAL { IntLit($1) }
114   | BLIT   { BoolLit($1) }
115   | FLIT   { FloatLit($1) }
116   | CLIT   { CharLit($1) }
117   | SLIT   { StringLit($1) }
118   | ID     { Id($1) }
119   | LSQUARE list_literal RSQUARE { ListLit(List.rev $2) }
120   | LPAREN expr RPAREN { $2 }
121
122 bracket_expr:
123   primary {$1}
124   | bracket_expr LSQUARE index RSQUARE {SliceExpr($1, $3)}
125   | bracket_expr LPAREN args_opt RPAREN { Call($1, $3) }
126
127 unary_expr:
128   bracket_expr {$1}
129   | MINUS unary_expr {Unop(Neg, $2)}
130   | NOT unary_expr {Unop(Not, $2)}
131
132 expr:
133   unary_expr {$1}
134   | expr PLUS expr { Binop($1, Add, $3) }
135   | expr MINUS expr { Binop($1, Sub, $3) }
136   | expr TIMES expr { Binop($1, Mult, $3) }
137   | expr DIVIDE expr { Binop($1, Div, $3) }
138   | expr EXP expr { Binop($1, Exp, $3) }
139   | expr EQ expr { Binop($1, Equal, $3) }
140   | expr NEQ expr { Binop($1, Neq, $3) }
141   | expr LT expr { Binop($1, Less, $3) }
142   | expr LEQ expr { Binop($1, Leq, $3) }
143   | expr GT expr { Binop($1, Greater, $3) }
144   | expr GEQ expr { Binop($1, Geq, $3) }
145   | expr AND expr { Binop($1, And, $3) }
146   | expr OR expr { Binop($1, Or, $3) }
147   | expr MOD expr { Binop($1, Mod, $3) }
148   | expr RANGE expr { Binop($1, Range, $3) }
149   | expr ASSIGN expr { Assign($1, $3) }
150   | expr ADDASN expr { Assign($1, Binop($1, Add, $3)) }

```



```

151 | expr SUBASN expr { Assign($1, Binop($1, Sub, $3)) }
152 | expr MULASN expr { Assign($1, Binop($1, Mult, $3)) }
153 | expr DIVASN expr { Assign($1, Binop($1, Div, $3)) }
154 | expr MODASN expr { Assign($1, Binop($1, Mod, $3)) }
155
156 index:
157     expr { Index $1 }
158 | expr COLON expr { Slice($1, $3) }
159 | COLON expr { Slice(IntLit 0, $2) }
160 | expr COLON { Slice($1, End) }
161 | COLON { Slice(IntLit 0, End) }
162
163 list_literal:
164     /* nothing */ { [] }
165 | expr { [$1] }
166 | list_literal COMMA expr { $3 :: $1 }
167
168 args_opt:
169     /* nothing */ { [] }
170 | args_list { List.rev $1 }
171
172 args_list:
173     expr { [$1] }
174 | args_list COMMA expr { $3 :: $1 }

```

### 11.2.5 src/ast.ml

```

1 (* Abstract Syntax Tree and functions for printing it *)
2 (* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
3 type op =
4   | Add
5   | Sub
6   | Mult
7   | Div
8   | Mod
9   | Equal
10  | Neq
11  | Less
12  | Leq
13  | Greater
14  | Geq
15  | And
16  | Or
17  | Exp
18  | Range
19
20 type uop = Neg | Not
21
22 type expr =
23   | IntLit of int
24   | FloatLit of float
25   | BoolLit of bool
26   | CharLit of char
27   | StringLit of string
28   | ListLit of expr list
29   | Id of string
30   | Binop of expr * op * expr
31   | Unop of uop * expr
32   | Assign of expr * expr

```

```

33 | Call of expr * expr list
34 | SliceExpr of expr * slce
35 | End
36 | Noexpr
37
38 and slce = Index of expr | Slice of expr * expr
39
40 type typ =
41 | Int
42 | Bool
43 | Float
44 | Void
45 | Char
46 | String
47 | List of typ
48 | Func of typ list * typ
49 | File
50 | Unknown
51 (* An empty list has an list unknown type *)
52
53 type bind = typ * string
54
55 type stmt =
56 | Block of stmt list
57 | Expr of expr
58 | Return of expr
59 | If of expr * stmt * stmt
60 | For of string * expr * stmt
61 | While of expr * stmt
62 | Declaration of typ * string * expr
63 | Break
64 | Continue
65
66 type func_decl = {typ: typ; fname: string; formals: bind list; body: stmt
    list}
67
68 type program = func_decl list * stmt list
69
70 (* Pretty-printing functions *)
71
72 let string_of_op = function
73 | Add -> "+"
74 | Sub -> "-"
75 | Mult -> "*"
76 | Div -> "/"
77 | Mod -> "%"
78 | Equal -> "=="
79 | Neq -> "!="
80 | Less -> "<"
81 | Leq -> "<="
82 | Greater -> ">"
83 | Geq -> ">="
84 | And -> "and"
85 | Or -> "or"
86 | Exp -> "^"
87 | Range -> ".."
88
89 let string_of_uop = function Neg -> "-" | Not -> "not "
90

```

```

91 let rec string_of_expr = function
92   | IntLit l -> string_of_int l
93   | FloatLit l -> string_of_float l
94   | BoolLit true -> "true"
95   | BoolLit false -> "false"
96   | CharLit c -> String.make 1 c
97   | StringLit s -> "\"" ^ s ^ "\""
98   | ListLit l -> "[" ^ String.concat "," (List.map string_of_expr l) ^ "]"
99   | SliceExpr (e, s) -> (
100     match s with
101     | Index i -> string_of_expr e ^ "[" ^ string_of_expr i ^ "]"
102     | Slice (i, j) ->
103       string_of_expr e ^ "[" ^ string_of_expr i ^ ":" ^ string_of_expr j ^
104         "]"
105     )
106   | Id s -> s
107   | Binop (e1, o, e2) -> (
108     match o with
109     | Range -> string_of_expr e1 ^ string_of_op o ^ string_of_expr e2
110     | _ -> string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr
111       e2
112   )
113   | Unop (o, e) -> string_of_uop o ^ string_of_expr e
114   | Assign (v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
115   | Call (f, el) ->
116     string_of_expr f ^ "("
117     ^ String.concat ", " (List.map string_of_expr el)
118     ^ ")"
119   | End -> ""
120   | Noexpr -> ""
121
122 let indent stmts_as_string =
123   let rec take k xs =
124     match k with
125     | 0 -> []
126     | k -> (
127       match xs with [] -> failwith "take" | y :: ys -> y :: take (k - 1) ys
128     )
129   in
130   let l = String.split_on_char '\n' stmts_as_string in
131   let indent_stmt stmts_as_list =
132     List.map (fun stmt_as_string -> "\t" ^ stmt_as_string) stmts_as_list
133   in
134   String.concat "\n" (indent_stmt (take (List.length l - 1) l)) ^ "\n"
135
136 let rec string_of_stmt = function
137   | Block stmts -> String.concat "" (List.map string_of_stmt stmts)
138   | Expr expr -> string_of_expr expr ^ ";\n"
139   | Return expr -> "return " ^ string_of_expr expr ^ ";\n"
140   | If (e, s1, s2) -> (
141     match s2 with
142     | Block [] ->
143       "if " ^ string_of_expr e ^ ":\n" ^ indent (string_of_stmt s1) ^ "end\n"
144     | _ ->
145       "if " ^ string_of_expr e ^ ":\n"
146       ^ indent (string_of_stmt s1)
147       ^ "else:\n"
148       ^ indent (string_of_stmt s2)
149       ^ "end\n"
150   )

```

```

146 | For (s, e2, st) ->
147   "for " ^ s ^ " in " ^ string_of_expr e2 ^ ":\n"
148   ^ indent (string_of_stmt st)
149   ^ "end\n"
150 | While (e, s) ->
151   "while " ^ string_of_expr e ^ ":\n" ^ indent (string_of_stmt s) ^ "end\n"
152 | Declaration (t, id, e) -> (
153   match e with
154   | Noexpr -> string_of_typ t ^ " " ^ id ^ ";\n"
155   | _ -> string_of_typ t ^ " " ^ id ^ " = " ^ string_of_expr e ^ ";\n" )
156 | Break -> "break;\n"
157 | Continue -> "continue;\n"
158
159 and string_of_typ = function
160 | Int -> "int"
161 | Bool -> "bool"
162 | Float -> "float"
163 | Void -> "void"
164 | Char -> "char"
165 | String -> "string"
166 | List t -> "list<" ^ string_of_typ t ^ ">"
167 | Func (a, r) ->
168   "("
169   ^ String.concat "," (List.map string_of_typ a)
170   ^ ")" ^ "->" ^ string_of_typ r
171 | File -> "file"
172 | Unknown -> "?"
173
174 let string_of_formals (formals : bind list) : string =
175   String.concat ", "
176     (List.map
177       (fun (formal : bind) -> string_of_typ (fst formal) ^ " " ^ snd formal
178       )
179       formals )
180
181 let string_of_fdecl fdecl =
182   string_of_typ fdecl.typ ^ " " ^ fdecl.fname ^ "("
183   ^ string_of_formals fdecl.formals
184   ^ "):\n"
185   ^ indent (String.concat "" (List.map string_of_stmt fdecl.body))
186   ^ "end\n"
187
188 let string_of_program (funcs, stmts) =
189   String.concat "\n" (List.map string_of_fdecl funcs)
190   ^ String.concat "" (List.map string_of_stmt stmts)

```

### 11.2.6 src/sast.ml

```

1 (* Semantically-checked Abstract Syntax Tree and functions for printing it
2    *)
3 (* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
4 open Ast
5
6 module StringMap = Map.Make (String)
7
8 type symbol_table =
9   { variables: typ StringMap.t
10     ; (* Variables bound in current block *)
11     functions: func_decl list

```

```

10      (* Functions that may need to be re-instantiated with concrete types
11      *)
12      ; list_variables: symbol_table ref StringMap.t
13      ; parent: symbol_table option (* Enclosing scope *) }
14 type sexpr = typ * sx
15
16 and sx =
17   | SIntLit of int
18   | SFloatLit of float
19   | SBoolLit of bool
20   | SCharLit of char
21   | SStringLit of string
22   | SListLit of sexpr list
23   | SId of string
24   | SBinop of sexpr * op * sexpr
25   | SUnop of uop * sexpr
26   | SAssign of sexpr * sexpr
27   | SCall of sexpr * sexpr list
28   | SSliceExpr of sexpr * sslce
29   | SNoexpr
30   | SEnd
31
32 and sslce = SIndex of sexpr | SSlice of sexpr * sexpr
33
34 type sstmt =
35   | SBlock of sstmt list
36   | SExpr of sexpr
37   | SReturn of sexpr
38   | SIf of sexpr * sstmt * sstmt
39   | SFor of string * sexpr * sstmt
40   | SWhile of sexpr * sstmt
41   | SDeclaration of typ * string * sexpr
42   | SBreak
43   | SContinue
44
45 type sfunc_decl =
46   {styp: typ; sfname: string; sformals: bind list; sbody: sstmt list}
47
48 (* Same as symbol table, but all functions are semantically checked *)
49 type resolved_table =
50   { rvariables: typ StringMap.t
51   ; (* Variables bound in current block *)
52     rfunctions: sfunc_decl list
53   ; (* Functions that may need to be re-instantiated with concrete types
54     *)
55     ; rlist_variables: resolved_table ref StringMap.t
56     ; rparent: resolved_table option (* Enclosing scope *) }
57
58 type program = sfunc_decl list * sstmt list
59
60 (* Pretty-printing functions *)
61 let rec string_of_sexpr (t, e) =
62   "(" ^ string_of_typ t ^ " : "
63   ^ ( match e with
64     | SIntLit i -> string_of_int i
65     | SFloatLit f -> string_of_float f
66     | SBoolLit true -> "true"

```

```

67 | SBoolLit false -> "false"
68 | SCharLit c -> String.make 1 c
69 | SStringLit s -> "\"" ^ s ^ "\""
70 | SListLit l -> "[" ^ String.concat "," (List.map string_of_sexpr l) ^ "]"
71 | SSliceExpr (e, s) -> (
72   match s with
73   | SIndex i -> string_of_sexpr e ^ "[" ^ string_of_sexpr i ^ "]"
74   | SSlice (i, j) ->
75     string_of_sexpr e ^ "[" ^ string_of_sexpr i ^ ":" ^
string_of_sexpr j
76     ^ "]" )
77 | SId s -> s
78 | SBinop (e1, o, e2) -> (
79   match o with
80   | Range -> string_of_sexpr e1 ^ string_of_op o ^ string_of_sexpr e2
81   | _ ->
82     string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr
e2 )
83 | SUnop (o, e) -> string_of_uop o ^ string_of_sexpr e
84 | SAssign (v, e) -> string_of_sexpr v ^ " = " ^ string_of_sexpr e
85 | SCall (f, el) ->
86   string_of_sexpr f ^ "("
87   ^ String.concat ", " (List.map string_of_sexpr el)
88   ^ ")"
89 | SEnd -> ""
90 | SNoexpr -> "" )
91 ^ ")"
92
93 let indent stmts_as_string =
94   let rec take k xs =
95     match k with
96     | 0 -> []
97     | k -> (
98       match xs with [] -> failwith "take" | y :: ys -> y :: take (k - 1) ys
99       )
100   in
101   let l = String.split_on_char '\n' stmts_as_string in
102   let indent_stmt stmts_as_list =
103     List.map (fun stmt_as_string -> "\t" ^ stmt_as_string) stmts_as_list
104   in
105   String.concat "\n" (indent_stmt (take (List.length l - 1) l)) ^ "\n"
106
107 let rec string_of_sstmt = function
108 | SBlock stmts -> String.concat "" (List.map string_of_sstmt stmts)
109 | SExpr expr -> string_of_sexpr expr ^ "\n"
110 | SReturn expr -> "return " ^ string_of_sexpr expr ^ "\n"
111 | SIf (e, s1, s2) -> (
112   match s2 with
113   | SBlock [] ->
114     "if " ^ string_of_sexpr e ^ ":\n"
115     ^ indent (string_of_sstmt s1)
116     ^ "end\n"
117   | _ ->
118     "if " ^ string_of_sexpr e ^ ":\n"
119     ^ indent (string_of_sstmt s1)
120     ^ "else:\n"
121     ^ indent (string_of_sstmt s2)
122     ^ "end\n" )

```

```

122 | SFor (s, e2, st) ->
123   "for " ^ s ^ " in " ^ string_of_sexpr e2 ^ ":\n "
124   ^ indent (string_of_sstmt st)
125   ^ "end\n"
126 | SWhile (e, s) ->
127   "while " ^ string_of_sexpr e ^ ":\n"
128   ^ indent (string_of_sstmt s)
129   ^ "end\n"
130 | SDeclaration (t, id, (tp, e)) -> (
131   match e with
132   | SNoexpr -> string_of_typ t ^ " " ^ id ^ "\n"
133   | _ -> string_of_typ t ^ " " ^ id ^ " = " ^ string_of_sexpr (tp, e) ^ "\n" )
134 | SBreak -> "break\n"
135 | SContinue -> "continue\n"
136
137 let string_of_sfdecl fdecl =
138   string_of_typ fdecl.styp ^ " " ^ fdecl.sfname ^ "("
139   ^ String.concat ", " (List.map snd fdecl.sformals)
140   ^ ")\n"
141   ^ indent (String.concat "" (List.map string_of_sstmt fdecl.sbody))
142   ^ "end\n"
143
144 let string_of_sprogram (funcs, stmts) =
145   String.concat "" (List.map string_of_sfdecl funcs)
146   ^ "\n"
147   ^ String.concat "" (List.map string_of_sstmt stmts)

```

### 11.2.7 src/exceptions.ml

```

1 (* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
2 open Ast
3
4 exception NonUniformTypeContainer of typ * typ
5
6 exception UndefinedId of string
7
8 exception MismatchedTypes of typ * typ * expr
9
10 exception InvalidBinaryOperation of typ * op * typ * expr
11
12 exception InvalidUnaryOperation of typ * uop * expr
13
14 exception IllegalAssignment of typ * op option * typ * expr
15
16 exception IllegalDeclaration of typ * typ * stmt
17
18 exception NonListAccess of typ * typ * expr
19
20 exception InvalidIndex of typ * expr
21
22 exception TypeError of string
23
24 exception CannotRedefineBuiltIn of string
25
26 exception AlreadyDefined of string
27
28 exception VoidType of string
29

```

```

30 exception Duplicate of string
31
32 exception UndefinedFunction
33
34 exception WrongNumberOfArgs of int * int * expr
35
36 exception PrintMissingArgs of expr
37
38 exception PrintBadArgs of char
39
40 exception PrintWrongType of expr
41
42 exception PrintWrongNumArgs of int * int
43
44 exception PrintTypeError of typ * typ
45
46 exception IllegalArgument of typ * typ * expr
47
48 exception IllegalSlice of expr * typ
49
50 exception WrongIndex of typ * expr
51
52 exception WrongSliceIndex of typ * typ * expr * expr
53
54 exception ReturnNotLast
55
56 exception ReturnOutsideFunction
57
58 exception NoReturnInNonVoidFunction
59
60 exception ReturnMismatchedTypes of typ * typ * stmt
61
62 exception NotInLoop of string
63
64 exception AppendNonList of typ
65
66 exception LengthWrongArgument of typ
67
68 exception UnsupportedPrint of typ
69
70 exception AssignNonVar of expr
71
72 exception IllegalFname
73
74 exception IllegalFor
75
76 (* Resolver Exceptions *)
77 exception IllegalSSlice
78
79 (* Codegen Exceptions *)
80 exception InvalidFloatBinop
81
82 exception InvalidIntBinop
83
84 exception InvalidStringBinop
85
86 exception NotFound of string
87
88 exception ImpossibleElif

```



```

89
90 let handle_error (e : exn) =
91   match e with
92   | NonUniformTypeContainer (t1, t2) ->
93     let s1 = string_of_typ t1 and s2 = string_of_typ t2 in
94     raise
95       (TypeError
96        (Printf.sprintf
97         "Type error: Lists can only contain one type. Expected '%s',
98         but \
99           got '%s'"
100          s1 s2 ) )
101   | UndefinedId n ->
102     raise
103       (TypeError
104        (Printf.sprintf "Error: variable '%s' was used before it was
105         defined"
106          n ) )
107   | MismatchedTypes (t1, t2, e) ->
108     let s1 = string_of_typ t1
109     and s2 = string_of_typ t2
110     and s3 = string_of_expr e in
111     raise
112       (TypeError
113        (Printf.sprintf
114         "Type error: Expected value of type '%s', but got a value of \
115         type '%s' in '%s'"
116         s1 s2 s3 ) )
117   | InvalidBinaryOperation (t1, op, t2, e) ->
118     let s1 = string_of_typ t1
119     and s2 = string_of_op op
120     and s3 = string_of_typ t2
121     and s4 = string_of_expr e in
122     raise
123       (TypeError
124        (Printf.sprintf
125         "Type error: Illegal binary operator '%s' '%s' '%s' in '%s'"
126         s1 s2
127         s3 s4 ) )
128   | InvalidUnaryOperation (t, op, e) ->
129     let s1 = string_of_typ t
130     and s2 = string_of_uop op
131     and s3 = string_of_expr e in
132     raise
133       (TypeError
134        (Printf.sprintf "Error: Illegal unary operator '%s' '%s' in '%s'"
135         s2
136         s1 s3 ) )
137   | IllegalAssignment (t1, op, t2, e) ->
138     let s1 = string_of_typ t1
139     and s3 = string_of_typ t2
140     and s4 = string_of_expr e in
141     let s2 = match op with Some x -> string_of_op x | None -> "" in
142     raise
143       (TypeError
144        (Printf.sprintf "Error: Illegal assignment '%s' '%s'='%s' in '%s'
145         '"
146         s1 s2 s3 s4 ) )
147   | IllegalDeclaration (t1, t2, s) ->

```

```

143     let s1 = string_of_typ t1
144     and s2 = string_of_typ t2
145     and s3 = string_of_stmt s in
146     raise
147       (TypeError
148        (Printf.sprintf "Error: Illegal declaration '%s' '%s' in '%s'" s1
149          s2
150            s3 ) )
151 | NonListAccess (t1, t2, e) ->
152   let s1 = string_of_typ t1
153   and s2 = string_of_typ t2
154   and s3 = string_of_expr e in
155   raise
156     (TypeError
157      (Printf.sprintf
158       "Type Error: Expected a lvalue of type List('%s'), but got \
159       lvalue of type '%s' in '%s'"
160       s1 s2 s3 ) )
161 | InvalidIndex (t1, e) ->
162   let s1 = string_of_typ t1 and s2 = string_of_expr e in
163   raise
164     (TypeError
165      (Printf.sprintf
166       "Type Error: Expected index of type Int, but got type '%s' in \
167       '%s'"
168       s1 s2 ) )
169 | CannotRedefineBuiltIn s ->
170   raise
171     (TypeError
172      (Printf.sprintf
173       "Error: Function '%s' may not be defined as it exists as a \
174       built \
175       in function"
176       s ) )
177 | AlreadyDefined s ->
178   raise
179     (TypeError
180      (Printf.sprintf
181       "Error: '%s' cannot be redefined in the current scope" s ) )
182 | VoidType n ->
183   raise
184     (TypeError
185      (Printf.sprintf "Type Error: variable '%s' cannot have type Void"
186       n)
187      )
188 | Duplicate n ->
189   raise
190     (TypeError
191      (Printf.sprintf "Error: variable name '%s' has already defined" n
192       ) )
193 | UndefinedFunction ->
194   raise
195     (TypeError
196      (Printf.sprintf "Error: function was called, but it is undefined"
197       ) )
198 | WrongNumberOfArgs (exp, act, e) ->
199   let s1 = string_of_int exp
200   and s2 = string_of_int act

```

```

196     and s3 = string_of_expr e in
197     raise
198     (TypeError
199      (Printf.sprintf "Error: expected '%s' arguments but got '%s' in
200      '%s'"
201      s1 s2 s3 ) )
202 | PrintMissingArgs e ->
203     let s1 = string_of_expr e in
204     raise
205     (TypeError
206      (Printf.sprintf
207       "Error: expected more than 0 arguments but got 0 in '%s'" s1 )
208     )
209 | PrintBadArgs e ->
210     raise
211     (TypeError
212      (Printf.sprintf "Error: expected either c, f or i but got '%c'" e
213      ) )
214 | PrintWrongType e ->
215     let s1 = string_of_expr e in
216     raise (TypeError (Printf.sprintf "Error: expected string but got '%s'"
217     s1))
218 | PrintWrongNumArgs (i1, i2) ->
219     let s1 = string_of_int i1 and s2 = string_of_int i2 in
220     raise
221     (TypeError
222      (Printf.sprintf "Error: expected %s addition arguments, but got %
223      s"
224      s1 s2 ) )
225 | PrintTypeError (t1, t2) ->
226     let s1 = string_of_typ t1 and s2 = string_of_typ t2 in
227     raise
228     (TypeError
229      (Printf.sprintf
230       "Type Error: expected argument of type '%s' but got '%s'
231       instead"
232       s1 s2 ) )
233 | IllegalArgument (t1, t2, e) ->
234     let s1 = string_of_typ t1
235     and s2 = string_of_typ t2
236     and s3 = string_of_expr e in
237     raise
238     (TypeError
239      (Printf.sprintf
240       "Type Error: Illegal argument found in '%s'. Expected argument
241       \
242       of type '%s' but got '%s'"
243       s3 s1 s2 ) )
244 | IllegalSlice (e, t) ->
245     let s1 = string_of_expr e and s2 = string_of_typ t in
246     raise
247     (TypeError
248      (Printf.sprintf
249       "Type Error: Illegal Slice expression, expected '%s' to be \
250       either a list or array, but got a '%s'"
251       s1 s2 ) )
252 | WrongIndex (t, e) ->
253     let s1 = string_of_typ t and s2 = string_of_expr e in
254     raise

```

```

248         (TypeError
249             (Printf.sprintf
250                 "Type Error: Expected index to be an int, but got '%s' in '%s'
251             " s1
252                 s2 ) )
253 | WrongSliceIndex (t1, t2, e1, e2) ->
254     let s1 = string_of_type t1
255     and s2 = string_of_type t2
256     and s3 = string_of_expr e1
257     and s4 = string_of_expr e2 in
258     raise
259     (TypeError
260         (Printf.sprintf
261             "Type Error: Expected left and right values of slice
262             expression \
263             to be of type int, but got type '%s' and '%s' in '%s' and '%s'
264             "
265             s1 s2 s3 s4 ) )
266 | ReturnNotLast ->
267     raise
268     (TypeError
269         (Printf.sprintf
270             "Error: There are unreachable statements after return" ) )
271 | ReturnOutsideFunction ->
272     raise
273     (TypeError
274         (Printf.sprintf "Error: Return statement is outside of a function
275         ")
276     )
277 | NoReturnInNonVoidFunction ->
278     raise
279     (TypeError
280         (Printf.sprintf
281             "Error: No return statement in function returning non-void" )
282     )
283 | ReturnMismatchedTypes (t1, t2, s) ->
284     let s1 = string_of_type t1
285     and s2 = string_of_type t2
286     and s3 = string_of_stmt s in
287     raise
288     (TypeError
289         (Printf.sprintf
290             "Type Error: Expected value of type '%s', but got a value of \
291             type '%s' in '%s'
292             "
293             s1 s2 s3 ) )
294 | InvalidFloatBinop ->
295     raise
296     (Failure
297         "Internal Error: Invalid operation on float. Semant should have \
298         rejected this" )
299 | InvalidIntBinop ->
300     raise
301     (Failure
302         "Internal Error: Invalid operation on int. Semant should have \
303         rejected this" )
304 | InvalidStringBinop ->
305     raise
306     (Failure
307         "Internal Error: Invalid operation on string. Semant should have

```

```

302         \
           rejected this" )
303 | NotFound s ->
304     raise
305     (Failure (Printf.sprintf "Internal Error: Variable '%s' not in scope
           " s))
306 | ImpossibleElif ->
307     raise
308     (Failure
309     (Printf.sprintf
310     "Internal Error: Corrupted Tree. Semant should have rejected
           this" )
311     )
312 | NotInLoop s ->
313     raise
314     (Failure
315     (Printf.sprintf
316     "Error: Expected '%s' to be in a loop, but it was not" s ) )
317 | AppendNonList t ->
318     let s = string_of_type t in
319     raise
320     (Failure
321     (Printf.sprintf
322     "Type Error: Expected first argument to append to be of type \
           list, but got type '%s' instead"
323     s ) )
324 | LengthWrongArgument t ->
325     let s = string_of_type t in
326     raise
327     (Failure
328     (Printf.sprintf
329     "Type Error: Expected argument to length to be of type list or
           \
330     string, but got type '%s' instead"
331     s ) )
332 | UnsupportedPrint t ->
333     let s = string_of_type t in
334     raise
335     (Failure
336     (Printf.sprintf
337     "Type Error: Print does not support printing for type '%s'" s
338     ) )
339 | AssignNonVar e ->
340     let s = string_of_expr e in
341     raise
342     (Failure
343     (Printf.sprintf
344     "Error: Cannot assign to a non variable or non slice
           expression \
345     in '%s'"
346     s ) )
347 | IllegalFname ->
348     raise (Failure (Printf.sprintf "Error: Function name is not an SId"))
349 | IllegalFor ->
350     raise
351     (Failure (Printf.sprintf "For loop can only operate on string or
           list"))
352 | IllegalSSlice ->
353     raise

```

```

354         (Failure
355           (Printf.sprintf
356             "Internal Error: Illegal Slice, should have been rejected in \
357             Semant" ) )
358 | e -> raise (TypeError (Printexc.to_string e))

```

### 11.2.8 src/semant.ml

```

1 (* Semantic checking for the TEAM compiler *)
2 (* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
3
4 open Ast
5 open Sast
6 module StringMap = Map.Make (String)
7 open List
8 module E = Exceptions
9
10 (* Semantic checking of the AST. Returns an SAST if successful, throws an
11    exception if something is wrong. *)
12
13 let check (functions, statements) =
14   let func_ty fd =
15     let param_types = List.map (fun (a, _) -> a) fd.formals in
16     Func (param_types, fd.typ)
17   in
18   let built_in_decls =
19     let add_bind map (name, formalTypes, returnType) =
20       StringMap.add name
21         (func_ty {typ= returnType; fname= name; formals= formalTypes; body=
22           []})
23       map
24     in
25     List.fold_left add_bind StringMap.empty
26     [ ("print", [(Unknown, "x")], Void)
27     ; ("open", [(String, "file_name"); (String, "mode")], File)
28     ; ("readline", [(File, "file_handle")], String)
29     ; ("write", [(File, "file_handle"); (String, "content")], Void)
30     ; ("close", [(File, "file_handle")], Void)
31     ; ("match", [(String, "target"); (String, "regex")], Bool)
32     ; ("find", [(String, "target"); (String, "regex")], String)
33     ; ( "replace"
34       , [ (String, "target")
35         ; (String, "regex")
36         ; (String, "replace")
37         ; (Int, "count") ]
38       , String )
39     ; ( "replaceall"
40       , [(String, "target"); (String, "regex"); (String, "replace")]
41       , String )
42     ; ("findall", [(String, "target"); (String, "regex")], List String)
43     ; ( "append"
44       , [(List Unknown, "input_list"); (Unknown, "element")]
45       , List Unknown )
46     ; ( "insert"
47       , [(List Unknown, "input_list"); (Unknown, "element"); (Int, "index"
48         , List Unknown )
49       ; ("length", [(Unknown, "input_list")], Int)
50       ; ("reverse", [(List Unknown, "input_list")], List Unknown) ]

```

```

50 in
51 (* fd.typ *)
52 let add_func map fd =
53   let n = fd.fname in
54   (* Name of the function *)
55   match fd with
56   | _ when StringMap.mem n built_in_decls ->
57     raise (E.CannotRedefineBuiltIn fd.fname)
58   | _ when StringMap.mem n map -> raise (E.AlreadyDefined fd.fname)
59   | _ -> StringMap.add n (func_ty fd) map
60 in
61 let function_decls = List.fold_left add_func built_in_decls functions in
62 let variable_table =
63   { variables= function_decls
64   ; functions
65   ; list_variables= StringMap.empty
66   ; parent= None }
67 in
68 (* Create a reference to the global table. The scope will be passed
69    through
70    recursive calls and be mutated when we need to add a new variable *)
71 let global_scope = ref variable_table in
72 let check_binds (to_check : bind list) =
73   let name_compare (_, n1) (_, n2) = compare n1 n2 in
74   let check_it checked binding =
75     match binding with
76     | Void, name -> raise (E.VoidType name)
77     | _, n1 -> (
78       match checked with
79       (* No duplicate bindings *)
80       | (_, n2) :: _ when n1 = n2 -> raise (E.Duplicate n2)
81       | _ -> binding :: checked )
82   in
83   let _ = List.fold_left check_it [] (List.sort name_compare to_check) in
84   to_check
85 in
86 (* Finding a variable, beginning in a given scope and searching upwards *)
87 let rec type_of_identifier (scope : symbol_table ref) name =
88   try StringMap.find name !scope.variables
89   with Not_found -> (
90     match !scope.parent with
91     | Some parent -> type_of_identifier (ref parent) name
92     | _ -> raise (E.UndefinedId name) )
93 in
94 (* Add a variable to the given scope *)
95 let add_var_to_scope (scope : symbol_table ref) id ty =
96   try
97     let _ = StringMap.find id !scope.variables in
98     raise (E.Duplicate id)
99   with Not_found ->
100     scope :=
101       { variables= StringMap.add id ty !scope.variables
102       ; functions= !scope.functions
103       ; list_variables= !scope.list_variables
104       ; parent= !scope.parent }
105 in
106 let update_var (scope : symbol_table ref) id ty =
107   scope :=

```

```

108     { variables= StringMap.add id ty !scope.variables
109       ; functions= !scope.functions
110       ; list_variables= !scope.list_variables
111       ; parent= !scope.parent }
112 in
113 (* Add the scope of list variable with name id *)
114 let add_list_scope id (scope : symbol_table ref) =
115   try
116     let _ = StringMap.find id !scope.variables in
117     raise (E.Duplicate id)
118   with Not_found ->
119     scope :=
120       { variables= !scope.variables
121         ; functions= !scope.functions
122         ; list_variables= StringMap.add id scope !scope.list_variables
123         ; parent= !scope.parent }
124 in
125 (* For finding a list outside of current scope that hasn't been type
126   inferred *)
127 let get_list_scope id scope = StringMap.find_opt id !scope.list_variables
128 in
129 let check_assign lvaluet rvaluet err =
130   if lvaluet = rvaluet then lvaluet
131   else
132     let ret =
133       match (lvaluet, rvaluet) with
134       | List Unknown, List ty -> List ty
135       | List ty, List Unknown -> List ty
136       | Void, List Unknown -> List Unknown
137       | ty, Unknown -> ty
138       | Unknown, ty -> ty
139       | _ -> raise err
140     in
141     ret
142 in
143 let rec innermost_ty ty =
144   match ty with List t -> innermost_ty t | nonlist_ty -> nonlist_ty
145 in
146 (* Return a semantically-checked expression with a type *)
147 let rec expr scope exp =
148   match exp with
149   | IntLit l -> (Int, SIntLit l)
150   | FloatLit l -> (Float, SFloatLit l)
151   | BoolLit l -> (Bool, SBoolLit l)
152   | CharLit l -> (Char, SCharLit l)
153   | StringLit l -> (String, SStringLit l)
154   | ListLit es -> (
155     let ts = List.map (fun x -> fst (expr scope x)) es in
156     match ts with
157     | [] -> (List Unknown, SListLit [])
158     | x :: _ ->
159       let ty = List.hd ts in
160       let check_type e =
161         let ty', e' = expr scope e in
162         if ty' = ty then (ty', e')
163         else raise (E.NonUniformTypeContainer (ty, ty'))
164       in
165       (List x, SListLit (List.map check_type es)) )
166   | Id s -> (type_of_identifier scope s, SId s)

```



```

165 | Binop (e1, op, e2) as e ->
166   let t1, e1' = expr scope e1 and t2, e2' = expr scope e2 in
167   let same = t1 = t2 in
168   let ty =
169     match op with
170     | (Add | Sub | Mult | Div | Mod) when same && t1 = Int -> Int
171     | (Add | Sub | Mult | Div) when same && t1 = Float -> Float
172     | (Add | Sub | Mult | Div) when t1 = Int && t2 = Float -> Float
173     | (Add | Sub | Mult | Div) when t1 = Float && t2 = Int -> Float
174     | Add when t1 = String && t2 = String -> String
175     | Add when t1 = String && t2 = Char -> String
176     | Add when t1 = Char && t2 = String -> String
177     | Exp when same && t1 = Int -> Int
178     | Exp when same && t1 = Float -> Float
179     | Exp when t1 = Int && t2 = Float -> Float
180     | Exp when t1 = Float && t2 = Int -> Float
181     | (Equal | Neq) when t1 = Int && t2 = Float -> Bool
182     | (Equal | Neq) when t1 = Float && t2 = Int -> Bool
183     | (Equal | Neq) when same -> Bool
184     | (Equal | Neq) when t1 = Unknown || t2 = Unknown -> Bool
185     | (Less | Leq | Greater | Geq) when t1 = Int && t2 = Float -> Bool
186     | (Less | Leq | Greater | Geq) when t1 = Float && t2 = Int -> Bool
187     | (Less | Leq | Greater | Geq) when same && t1 = Int -> Bool
188     | (Less | Leq | Greater | Geq) when same && t1 = Float -> Bool
189     | (And | Or) when same && t1 = Bool -> Bool
190     | Range when same && t1 = Int -> List Int
191     | Add when same && match t1 with List _ -> true | _ -> false -> t1
192     | _ when t1 = Unknown && t2 <> Unknown -> t2
193     | _ when t1 <> Unknown && t2 = Unknown -> t1
194     | _ when same && t1 = Unknown -> t1
195     | _ -> raise (E.InvalidBinaryOperation (t1, op, t2, e))
196   in
197   (ty, SBinop ((t1, e1'), op, (t2, e2')))
198 | Unop (op, e) as ex ->
199   let t, e' = expr scope e in
200   let ty =
201     match op with
202     | Neg when t = Int || t = Float -> t
203     | Not when t = Bool -> Bool
204     | _ -> raise (E.InvalidUnaryOperation (t, op, ex))
205   in
206   (ty, SUnop (op, (t, e')))
207 | Assign (s, e) as ex ->
208   let lt, s' = expr scope s in
209   let rt, e' = expr scope e in
210   let lrt = check_assign lt rt (E.IllegalAssignment (lt, None, rt, ex))
211   in
212   let s_name =
213     match s with
214     | Id n -> n
215     | SliceExpr (Id n, _) -> n
216     | _ -> raise (E.AssignNonVar ex)
217   in
218   let is_slice = match s with SliceExpr _ -> true | _ -> false in
219   let is_list = match lt with List _ -> true | _ -> false in
220   let non_slice =
221     match (lt, rt) with
222     | List _, List _ | Void, List _ ->

```

```

223         let _ =
224             (* update the list variable in the scope it is defined in *)
225             match get_list_scope s_name scope with
226             | Some sc -> update_var sc s_name lrt
227             | None -> ()
228         in
229             (lrt, SAssign ((lrt, s'), (rt, e'))))
230     | _ -> (lrt, SAssign ((lt, s'), (rt, e'))))
231 in
232 if is_slice && is_list then
233     let _ = update_var scope s_name lt in
234     let _ =
235         (* update the list variable in the scope it is defined in *)
236         match get_list_scope s_name scope with
237         | Some sc -> update_var sc s_name lrt
238         | None -> ()
239     in
240         (lt, SAssign ((lt, s'), (rt, e'))))
241 else non_slice
242 | Call (fname, args) as call -> (
243     let check_length frmls =
244         if List.length args != List.length frmls then
245             raise
246             (E.WrongNumberOfArgs (List.length frmls, List.length args,
247 call))
248         else ()
249     in
250     let fty, fname' = expr scope fname in
251     let formals, ret_type =
252         match fty with
253         | Func (f, r) -> (f, r)
254         | _ -> raise E.UndefinedFunction
255     in
256     let _ = if fname <> Id "print" then check_length formals else () in
257     match fname with
258     | Id "print" ->
259         let et, _ = expr scope (hd args) in
260         let _ =
261             match et with String -> () | _ -> raise (E.UnsupportedPrint et
262 )
263         in
264         ( Void
265         , SCall
266           ((Func ([et], Void), SId "print"), List.map (expr scope)
267 args)
268         )
269     | Id "append" ->
270         let args' = List.map (expr scope) args in
271         let et1, args1' = hd args' in
272         let et2, args2' = hd (tl args') in
273         let inner_ty =
274             match et1 with List ty -> ty | _ -> raise (E.AppendNonList et1
275 )
276         in
277         let is_list et = match et with List _ -> true | _ -> false in
278         let _ =
279             if
280                 is_list et2
281                 && innermost_ty et2 = Unknown

```

```

278         && innermost_ty et1 <> Unknown
279     then ()
280     else if innermost_ty et1 <> et2 && innermost_ty et1 != Unknown
281     then raise (E.MismatchedTypes (inner_ty, et2, call))
282     else ()
283 in
284 if innermost_ty et1 = Unknown then
285     ( List et2
286     , SCall
287       ( (Func ([List et2; et2], List et2), SId "append")
288       , [(List et2, args1'); (et2, args2')] ) )
289 else if is_list et2 && innermost_ty et2 = Unknown then
290     ( et1
291     , SCall
292       ( (Func ([et1; et1], et1), SId "append")
293       , [(et1, args1'); (et1, args2')] ) )
294 else
295     ( List et2
296     , SCall ((Func ([List et2; et2], List et2), SId "append"),
args')
297     )
298 | Id "reverse" ->
299     let args' = List.map (expr scope) args in
300     let et1, _ = hd args' in
301     let _ =
302         match et1 with
303         | List _ -> ()
304         (* | String -> () *)
305         | _ -> raise (E.LengthWrongArgument et1)
306     in
307     (et1, SCall ((Func ([et1], et1), SId "reverse"), args'))
308
309 | Id "insert" ->
310     let args' = List.map (expr scope) args in
311     let et1, args1' = hd args' in
312     let et2, args2' = hd (tl args') in
313     let et3, args3' = hd (List.rev args') in
314     let inner_ty =
315         match et1 with List ty -> ty | _ -> raise (E.AppendNonList et1
)
316
317     in
318     let is_list et = match et with List _ -> true | _ -> false in
319     let _ =
320         if
321             is_list et2
322             && innermost_ty et2 = Unknown
323             && innermost_ty et1 <> Unknown
324         then ()
325         else if innermost_ty et1 <> et2 && innermost_ty et1 != Unknown
326         then raise (E.MismatchedTypes (inner_ty, et2, call))
327         else ()
328     in
329     if innermost_ty et1 = Unknown then
330         ( List et2
331         , SCall
332           ( (Func ([List et2; et2], List et2), SId "insert")
333           , [(List et2, args1'); (et2, args2'); (et3, args3')] ) )
334     else if is_list et2 && innermost_ty et2 = Unknown then
335         ( et1

```

```

335         , SCall
336         ( (Func ([et1; et1], et1), SId "insert")
337         , [(et1, args1'); (et1, args2'); (et3, args3')] ) )
338     else
339         ( List et2
340         , SCall ((Func ([List et2; et2], List et2), SId "insert"),
args')
341         )
342
343     | Id "length" ->
344         let args' = List.map (expr scope) args in
345         let et1, _ = hd args' in
346         let _ =
347             match et1 with
348             | List _ -> ()
349             | String -> ()
350             | _ -> raise (E.LengthWrongArgument et1)
351         in
352         (Int, SCall ((Func ([List Int], Int), SId "length"), args'))
353
354     | Id "split" ->
355         let args' = List.map (expr scope) args in
356         let et1, _ = hd args' in
357         let _ =
358             match et1 with
359             | String -> ()
360             | _ -> raise (Failure "Mismatched types")
361         in
362         ( List String
363         , SCall ((Func ([String; Char], List String), SId "split"), args
')
364         )
365
366     | Id "string_to_list" ->
367         let args' = List.map (expr scope) args in
368         let et1, _ = hd args' in
369         let _ =
370             match et1 with
371             | String -> ()
372             | _ -> raise (Failure "Mismatched types")
373         in
374         ( List Char
375         , SCall ((Func ([String], List Char), SId "string_to_list"),
args')
376         )
377
378     | _ ->
379         let check_call ft e =
380             let et, e' = expr scope e in
381             (check_assign ft et (E.IllegalArgument (ft, et, e)), e')
382         in
383         let args' = List.map2 check_call formals args in
384         let arg_types = List.map (fun e -> fst (expr scope e)) args in
385         (* get all list type arguments *)
386         let list_args =
387             List.filter
388             (fun x -> match x with List _ -> true | _ -> false)
389             arg_types
390         in
391         if length list_args > 0 then
392             let inner_types =
393                 List.map

```

```

391         (fun t -> match t with List inner_ty -> inner_ty | ty ->
ty)
392         arg_types
393     in
394     (* convert list types to strings *)
395     let type_specific_name =
396         String.concat "_"
397         (List.map (fun t -> string_of_ty t) inner_types)
398     in
399     let func_name =
400         match fname with Id s -> s | _ -> raise E.IllegalFname
401     in
402     (* look up the old function *)
403     let look_up_func =
404         List.find_opt (fun f -> f.fname = func_name) functions
405     in
406     let generic_func =
407         match look_up_func with
408         | Some f -> f
409         | None -> raise E.UndefinedFunction
410     in
411     let new_fname =
412         String.concat "_" [generic_func.fname; type_specific_name]
413     in
414     let new_func_created =
415         List.find_opt
416             (fun f -> f.fname = new_fname)
417             !global_scope.functions
418     in
419     if Option.is_none new_func_created then
420         (* make a copy of the new function where type is resolved to
a
421             specific kind of list *)
422         let modified_func =
423             { typ= ret_type
424             ; fname= new_fname
425             ; formals=
426                 List.combine arg_types (List.map snd generic_func.
formals)
427             ; body= generic_func.body }
428         in
429         (* add the new type specified function, remove the generic
430             function *)
431         let _ =
432             global_scope :=
433             { variables= !global_scope.variables
434             ; functions=
435                 modified_func
436             ::
437                 List.filter
438                     (fun f -> f.fname != generic_func.fname)
439                     !global_scope.functions
440             ; list_variables= !scope.list_variables
441             ; parent= !global_scope.parent }
442         in
443         ( ret_type
444         , SCall ((Func (arg_types, ret_type), SId new_fname), args')
445     )
else

```

```

446         ( ret_type
447         , SCall ((Func (arg_types, ret_type), SId new_fname), args')
448         )
449     else (ret_type, SCall ((fty, fname'), args')) )
450 | SliceExpr (lexpr, slce) as slice ->
451     let lt, lexpr' = expr scope lexpr in
452     let check_slice_expr =
453         match slice with
454         | Index e ->
455             let t, e' = expr scope e in
456             let id_type =
457                 match lt with
458                 | List ty -> ty
459                 | String -> Char
460                 | _ -> raise (E.IllegalSlice (slice, lt))
461             in
462             if t = Int then
463                 (id_type, SSliceExpr ((lt, lexpr'), SIndex (t, e')))
464             else raise (E.WrongIndex (t, e))
465         | Slice (e1, e2) ->
466             let t1, e1' = expr scope e1 and t2, e2' = expr scope e2 in
467             let id_type =
468                 match lt with
469                 | List _ -> lt
470                 | String -> lt
471                 | _ -> raise (E.IllegalSlice (slice, lt))
472             in
473             if t1 = Int && t1 = t2 then
474                 ( id_type
475                 , SSliceExpr ((lt, lexpr'), SSlice ((t1, e1'), (t2, e2'))) )
476             else raise (E.WrongSliceIndex (t1, t2, e1, e2))
477     in
478     check_slice_expr
479 | End -> (Int, SEnd)
480 | Noexpr -> (Void, SNoexpr)
481 in
482 let check_bool_expr scope e =
483     let t', e' = expr scope e in
484     if t' != Bool then raise (E.MismatchedTypes (t', Bool, e)) else (t', e')
485 in
486 (* check void type variable *)
487 let check_void_type ty name =
488     match ty with Void -> raise (E.VoidType name) | _ -> ty
489 in
490 let dummy = {typ= Int; fname= "toplevel"; formals= []; body= []} in
491 (* Checks if there are any statements after return *)
492 let rec check_return sl typ =
493     match sl with
494     | [] -> if typ != Void then raise E.NoReturnInNonVoidFunction else ()
495     | [Return _] -> ()
496     | Return _ :: _ -> raise E.ReturnNotLast
497     | _ :: ss -> check_return ss typ
498 in
499 (* Return a semantically-checked statement containing exprs *)
500 let rec check_stmt scope stmt loop fdecl =
501     match stmt with
502     | Expr e -> SExpr (expr scope e)
503     | Block sl ->
504         let new_scope =

```

```

504     { variables= StringMap.empty
505       ; functions= !scope.functions
506       ; list_variables= !scope.list_variables
507       ; parent= Some !scope }
508   in
509   let new_scope_ref = ref new_scope in
510   let rec check_stmt_list = function
511     | Block sl :: ss -> check_stmt_list (sl @ ss)
512     | s :: ss ->
513       check_stmt new_scope_ref s loop fdecl :: check_stmt_list ss
514     | [] -> []
515   in
516   SBlock (List.rev (check_stmt_list (List.rev sl)))
517 | Return e as return -> (
518   match fdecl.fname with
519   | "oplevel" -> raise E.ReturnOutsideFunction
520   | _ ->
521     let t, e' = expr scope e in
522     let is_return_list =
523       match (t, fdecl.typ) with
524       | List _, List Unknown -> true
525       | _ -> false
526     in
527     if is_return_list then
528       let look_up_func =
529         (* Look up functions in the symbol table *)
530         List.find_opt (fun f -> f.fname = fdecl.fname) !scope.
functions
531       in
532       let func =
533         match look_up_func with
534         | Some f -> f
535         | None -> raise E.UndefinedFunction
536       in
537       let modified_func =
538         {typ= t; fname= func.fname; formals= func.formals; body= func.
body}
539       in
540       (* update function return type to be type specific *)
541       let _ =
542         global_scope :=
543           { variables= !global_scope.variables
544             ; functions=
545               modified_func
546             ::
547               List.filter
548                 (fun f -> f.fname != func.fname)
549                 !global_scope.functions
550             ; list_variables= !scope.list_variables
551             ; parent= !global_scope.parent }
552       in
553       SReturn (t, e')
554     else if t = fdecl.typ then SReturn (t, e')
555     else raise (E.ReturnMismatchedTypes (fdecl.typ, t, return)) )
556 | If (p, b1, b2) ->
557   SIf
558     ( check_bool_expr scope p
559       , check_stmt scope b1 loop fdecl
560       , check_stmt scope b2 loop fdecl )

```

```

561 | For (s, e, st) ->
562   let t, e' = expr scope e in
563   let s_ty =
564     match t with
565     | List ty -> ty
566     | String -> Char
567     | _ -> raise E.IllegalFor
568   in
569   let _ = add_var_to_scope scope s s_ty in
570   let sexpr = SFor (s, (t, e'), check_stmt scope st (loop + 1) fdecl)
in
571   (* remove the indexing variable in for loop from current scope *)
572   let _ =
573     scope :=
574       { variables= StringMap.remove s !scope.variables
575       ; functions= !scope.functions
576       ; list_variables= !scope.list_variables
577       ; parent= !scope.parent }
578   in
579   sexpr
580 | While (p, s) ->
581   SWhile (check_bool_expr scope p, check_stmt scope s (loop + 1) fdecl)
)
582 | Declaration (ty, s, e) as decl ->
583   let expr_ty, e' = expr scope e in
584   let _ = check_void_type ty s in
585   let same = expr_ty = ty in
586   let is_generic_list =
587     match expr_ty with
588     | List Unknown -> true
589     | List _ -> false
590     | _ -> false
591   in
592   if (same && not is_generic_list) || e' = SNoexpr then
593     let _ = add_var_to_scope scope s ty in
594     SDeclaration (ty, s, (expr_ty, e'))
595   else if expr_ty = Unknown then
596     let _ = add_var_to_scope scope s ty in
597     SDeclaration (ty, s, (expr_ty, e'))
598   (* update type of list on LHS of assignment to the type of the LHS
*)
599   else if expr_ty = Unknown then
600     let _ = add_var_to_scope scope s ty in
601     SDeclaration (ty, s, (expr_ty, e'))
602   else if ty = List Unknown && not is_generic_list then
603     let _ = add_var_to_scope scope s expr_ty in
604     SDeclaration (expr_ty, s, (expr_ty, e'))
605   else
606     let _ =
607       (* if type of LHS is already specific, do not revert back to
generic *)
608       match (expr_ty, e') with
609       | List Unknown, _ ->
610         let _ = add_list_scope s scope in
611         add_var_to_scope scope s ty
612       | _ -> raise (E.IllegalDeclaration (ty, expr_ty, decl))
613     in
614     SDeclaration (ty, s, (expr_ty, e'))
615 | Break -> if loop > 0 then SBreak else raise (E.NotInLoop "Break")

```



```

616 | Continue -> if loop > 0 then SContinue else raise (E.NotInLoop "
    Continue")
617 in
618 let check_functions func =
619   let formals' = check_binds func.formals in
620   let add_formal map (ty, name) = StringMap.add name ty map in
621   let func_variable_table =
622     { variables= List.fold_left add_formal StringMap.empty formals'
623     ; functions= !global_scope.functions
624     ; list_variables= !global_scope.list_variables
625     ; parent= Some !global_scope }
626   in
627   let func_scope = ref func_variable_table in
628   let _ = check_return func.body func.typ in
629   let body' = check_stmt func_scope (Block func.body) 0 func in
630   if func.typ = List Unknown then
631     let updated_func =
632       List.filter (fun f -> f.fname = func.fname) !global_scope.functions
633     in
634     let _ =
635       if length updated_func = 0 then raise E.UndefinedFunction else ()
636     in
637     (* update function type to be type specific *)
638     { styp= (hd updated_func).typ
639     ; sfname= func.fname
640     ; sformals= formals'
641     ; sbody= [body'] }
642   else {styp= func.typ; sfname= func.fname; sformals= formals'; sbody= [
        body']}
643 in
644 let check_stmts stmt = check_stmt global_scope stmt 0 dummy in
645 let statements' =
646   try List.map check_stmts statements with e -> E.handle_error e
647 in
648 let functions' =
649   try List.map check_functions !global_scope.functions
650   with e -> E.handle_error e
651 in
652 (functions', statements')

```

### 11.2.9 src/resolve.ml

```

1 (* Authors: Lulu Z. *)
2 open Ast
3 open Sast
4 module StringMap = Map.Make (String)
5 open List
6 module E = Exceptions
7
8 let resolve (functions, statements) =
9   let variable_table : resolved_table =
10     { rvariables= StringMap.empty
11     ; rfunctions= functions
12     ; rlist_variables= StringMap.empty
13     ; rparent= None }
14   in
15   (* Create a reference to the global table. The scope will be passed
        through
16     recursive calls and be mutated when we need to add a new variable *)

```

```

17 let global_scope : resolved_table ref = ref variable_table in
18 (* Add a variable to the given scope *)
19 let add_var (scope : resolved_table ref) id ty =
20   scope :=
21     { rvariables= StringMap.add id ty !scope.rvariables
22       ; rfunctions= !scope.rfunctions
23       ; rlist_variables= !scope.rlist_variables
24       ; rparent= !scope.rparent }
25 in
26 (* Finding a variable, beginning in a given scope and searching upwards *)
27 let rec type_of_identifier (scope : resolved_table ref) id =
28   try StringMap.find id !scope.rvariables
29   with Not_found -> (
30     match !scope.rparent with
31     | Some parent -> type_of_identifier (ref parent) id
32     | _ -> raise (E.UndefinedId id) )
33 in
34 (* Add the scope of list variable with name id *)
35 let add_list_scope id (scope : resolved_table ref) =
36   scope :=
37     { rvariables= !scope.rvariables
38       ; rfunctions= !scope.rfunctions
39       ; rlist_variables= StringMap.add id scope !scope.rlist_variables
40       ; rparent= !scope.rparent }
41 in
42 (* For finding a list outside of current scope that hasn't been type
43    inferred *)
44 let get_list_scope id scope = StringMap.find_opt id !scope.rlist_variables
45 in
46 let look_up_func (funcs : sfunc_decl list) fname =
47   List.find_opt (fun f -> f.sfname = fname) funcs
48 in
49 let func_ty fd =
50   let param_types = List.map (fun (a, _) -> a) fd.sformals in
51   Func (param_types, fd.styp)
52 in
53 let rec expr scope ((t, e) : sexpr) =
54   match e with
55   | SIntLit i -> (t, SIntLit i)
56   | SFloatLit f -> (t, SFloatLit f)
57   | SBoolLit b -> (t, SBoolLit b)
58   | SCharLit c -> (t, SCharLit c)
59   | SStringLit s -> (t, SStringLit s)
60   | SId n -> (
61     try
62       if (type_of_identifier scope n, SId n) = (List Unknown, SId n) then
63         (t, SId n)
64       else (type_of_identifier scope n, SId n)
65     with _ -> (t, SId n) )
66   | SListLit l -> (t, SListLit l)
67   | SSliceExpr (lexpr, slce) ->
68     let lt, lexpr' = expr scope lexpr in
69     let check_slice_expr =
70       match slce with
71       | SIndex e ->
72         let t, e' = expr scope e in
73         let id_type =
74           match lt with
75           | List ty -> ty

```

```

74         | String -> Char
75         | _ -> raise E.IllegalSSlice
76     in
77         (id_ty, SSliceExpr ((lt, lepr'), SIndex (t, e')))
78 | SSlice _ -> (
79     (* look up new inferred type from the symbol table *)
80     let id_ty, id =
81         match lepr with
82         | List Unknown, SId s -> (
83             try (type_of_identifier scope s, SId s)
84             with _ -> (List Unknown, SId s) )
85         | ty, expr -> (ty, expr)
86     in
87     match id_ty with
88     | List Unknown -> (t, SSliceExpr (lepr, slce))
89     | List _ -> (id_ty, SSliceExpr ((id_ty, id), slce))
90     | _ -> (t, SSliceExpr (lepr, slce)) )
91 in
92     check_slice_expr
93 | SBinop (e1, op, e2) -> (t, SBinop (expr scope e1, op, expr scope e2))
94 | SUnop (op, e) -> (t, SUnop (op, e))
95 | SAssign (le, re) ->
96     let lt, le' = expr scope le in
97     let rt, re' = expr scope re in
98     let s_name =
99         match le' with
100         | SId n -> n
101         | SSliceExpr (_, SId n), _ -> n
102         | _ -> raise (Failure "Can't assign to a non variable")
103     in
104     let ret =
105         match (lt, le') with
106         | List _, SId s ->
107             let _ = add_var scope s rt in
108             let _ =
109                 (* update the list variable in the scope it is defined in *)
110                 match get_list_scope s_name scope with
111                 | Some sc -> add_var sc s_name rt
112                 | None -> ()
113             in
114             (rt, SAssign ((rt, le'), (rt, re')))
115         | _ -> (t, SAssign (le, re))
116     in
117     ret
118 | SCall (f, args) -> (
119     match f with
120     | _, SId "print" ->
121         (* resolve type of list if arguments are list indexing exprs *)
122         let resolved_args = List.map (expr scope) args in
123         (t, SCall (f, resolved_args))
124     | _, SId "append" -> (t, SCall (f, args))
125     | _, SId "insert" -> (t, SCall (f, args))
126     | _, SId "length" -> (t, SCall (f, args))
127     | _, SId "find" -> (t, SCall (f, args))
128     | _, SId "findall" -> (t, SCall (f, args))
129     | _, SId "match" -> (t, SCall (f, args))
130     | _, SId "replace" -> (t, SCall (f, args))
131     | _, SId "replaceall" -> (t, SCall (f, args))
132     | fty, SId fname ->

```

```

133     let option_func = look_up_func functions fname in
134     if Option.is_some option_func then
135         let func = Option.get option_func in
136         let args' = List.map (expr scope) args in
137         let resolved_arg_tys = List.map fst args' in
138         if resolved_arg_tys <> List.map fst args then
139             let inner_tys =
140                 List.map
141                     (fun t -> match t with List inner_ty -> inner_ty | ty ->
ty)
142                         resolved_arg_tys
143             in
144             let type_specific_name =
145                 String.concat "_"
146                 (List.map (fun t -> string_of_typ t) inner_tys)
147             in
148             (* get new type resolved function name *)
149             let new_fname = String.concat "_" [fname; type_specific_name]
in
150
151             (* if new function hasn't been created yet *)
152             let new_func_created =
153                 List.find_opt (fun f -> f.sfname = new_fname) functions
154             in
155             let _, ret_type =
156                 match fty with
157                 | Func (f, r) -> (f, r)
158                 | _ -> raise E.UndefinedFunction
159             in
160             (* create new func with new types *)
161             if Option.is_none new_func_created then
162                 let modified_func =
163                     { styp= ret_type
164                     ; sfname= new_fname
165                     ; sformals=
166                         List.combine resolved_arg_tys (List.map snd func.
sformals)
167                     ; sbody= func.sbody }
168                 in
169                 (* remove old function from the function list *)
170                 let _ =
171                     global_scope :=
172                     { rvariables= !global_scope.rvariables
173                     ; rfunctions=
174                         modified_func
175                         ::
176                         List.filter
177                             (fun f -> f.sfname != fname)
178                             !global_scope.rfunctions
179                     ; rlist_variables= !scope.rlist_variables
180                     ; rparent= !global_scope.rparent }
181                 in
182                 ( ret_type
183                 , SCall
184                     ((Func (resolved_arg_tys, ret_type), SId new_fname),
args')
185                 )
186             else
187                 ( ret_type
188                 , SCall

```

```

188         ((Func (resolved_arg_tys, ret_type), SId new_fname),
args')
189     )
190     else
191         let ret =
192             match func.styp with
193             | List _ -> (func.styp, SCall ((func_ty func, SId fname),
args'))
194             | _ -> (t, SCall (f, args'))
195         in
196         ret
197     else (t, SCall (f, args))
198 | _, _ -> raise E.IllegalFname )
199 | SEnd -> (t, SEnd)
200 | SNoexpr -> (Void, SNoexpr)
201 in
202 let rec stmt scope st =
203     match st with
204     | SExpr e -> SExpr (expr scope e)
205     | SBlock sl ->
206         let new_scope : resolved_table =
207             { rvariables= StringMap.empty
208             ; rfunctions= !scope.rfunctions
209             ; rlist_variables= !scope.rlist_variables
210             ; rparent= Some !scope }
211         in
212         let new_scope_ref = ref new_scope in
213         let rec stmt_list = function
214             | SBlock sl :: ss -> stmt_list (sl @ ss)
215             | s :: ss -> stmt new_scope_ref s :: stmt_list ss
216             | [] -> []
217         in
218         SBlock (stmt_list sl)
219     | SReturn e -> SReturn e
220     | SIf (p, then_stmt, else_stmt) -> SIf (expr scope p, then_stmt,
else_stmt)
221     | SFor (s, e, sl) ->
222         let t, e' = expr scope e in
223         let list_name = match e' with SId s -> Some s | _ -> None in
224         (* get resolved type if expr sl is a list *)
225         let inner_ty =
226             match t with
227             | List ty -> ty
228             | String -> Char
229             | _ -> raise E.IllegalFor
230         in
231         let resolved_ty =
232             if t = List Unknown && Option.is_some list_name then
233                 type_of_identifier scope (Option.get list_name)
234             else t
235         in
236         let _ = add_var scope s inner_ty in
237         let sexpr = SFor (s, (resolved_ty, e'), stmt scope sl) in
238         let _ =
239             scope :=
240                 { rvariables= StringMap.remove s !scope.rvariables
241                 ; rfunctions= !scope.rfunctions
242                 ; rlist_variables= !scope.rlist_variables
243                 ; rparent= !scope.rparent }

```

```

244     in
245     sexpr
246 | SWhile (p, b) -> SWhile (p, stmt scope b)
247 | SDeclaration (ty, s, e) ->
248     let resolved_ty, e' = expr scope e in
249     let is_generic_list =
250         match resolved_ty with
251         | List Unknown -> true
252         | List _ -> false
253         | _ -> false
254     in
255     if is_generic_list || e' = SNoexpr then
256         let _ = add_list_scope s scope in
257         let _ = add_var scope s ty in
258         SDeclaration (ty, s, (resolved_ty, e'))
259         (* update type of list on LHS of assignment to the type of the LHS
260 *)
261     else if resolved_ty = Unknown then
262         let _ = add_var scope s ty in
263         SDeclaration (ty, s, (resolved_ty, e'))
264     else if ty = List Unknown && not is_generic_list then
265         let _ = add_var scope s resolved_ty in
266         SDeclaration (resolved_ty, s, (resolved_ty, e'))
267     else if resolved_ty = List Unknown then
268         let _ = add_list_scope s scope in
269         let _ = add_var scope s ty in
270         SDeclaration (ty, s, (resolved_ty, e'))
271     else SDeclaration (ty, s, (resolved_ty, e'))
272 | SBreak -> SBreak
273 | SContinue -> SContinue
274 in
275 let check_functions func =
276     let add_formal map (ty, name) = StringMap.add name ty map in
277     let func_variable_table =
278         { rvariables= List.fold_left add_formal StringMap.empty func.sformals
279         ; rfunctions= !global_scope.rfunctions
280         ; rlist_variables= !global_scope.rlist_variables
281         ; rparent= Some !global_scope }
282     in
283     let func_scope = ref func_variable_table in
284     let body' = stmt func_scope (SBlock func.sbody) in
285     if func.styp = List Unknown then
286         let updated_func =
287             List.filter (fun f -> f.sfname = func.sfname) !global_scope.
288             rfunctions
289         in
290         let _ =
291             if length updated_func = 0 then raise E.UndefinedFunction else ()
292         in
293         (* func.type should be updated to type specific *)
294         { styp= (hd updated_func).styp
295         ; sfname= func.sfname
296         ; sformals= func.sformals
297         ; sbody= [body'] }
298     else
299         { styp= func.styp
300         ; sfname= func.sfname
301         ; sformals= func.sformals
302         ; sbody= [body'] }

```

```

301 in
302 let statements' =
303   try List.map (stmt global_scope) statements with e -> E.handle_error e
304 in
305 let functions' =
306   try List.map check_functions !global_scope.rfunctions
307   with e -> E.handle_error e
308 in
309 (functions', statements')

```

### 11.2.10 src/codegen.ml

```

1 (* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
2 module L = Llvml
3 module A = Ast
4 open Sast
5 module E = Exceptions
6 module StringMap = Map.Make (String)
7 (* module Array *)
8
9 type var_table =
10   {lvariables: L.llvalue StringMap.t; parent: var_table ref option}
11
12 let translate (functions, statements) =
13   (* defining main function as the list of statements on the top level*)
14   let main_func =
15     {styp= A.Int; sfname= "main"; sformals= []; sbody= statements}
16   in
17   let functions = [main_func] @ functions in
18   let context = L.global_context () in
19   let i32_t = L.i32_type context
20   and char_t = L.i8_type context
21   and i8_t = L.i8_type context
22   and void_t = L.void_type context
23   and float_t = L.double_type context
24   and il_t = L.il_type context
25   and string_t = L.pointer_type (L.i8_type context)
26   and file_t = L.pointer_type (L.i8_type context)
27   and the_module = L.create_module context "TEAM" in
28   let list_struct_type = L.named_struct_type context "list_item" in
29   let list_struct_ptr = L.pointer_type list_struct_type in
30   (* list is a linked list with a ptr to the data and the next ptr *)
31   let _ =
32     L.struct_set_body list_struct_type
33     [|L.pointer_type i8_t; list_struct_ptr|]
34     true
35   in
36   (* Convert TEAM types to LLVM types *)
37   let rec ltype_of_typ = function
38     | A.Int -> i32_t
39     | A.String -> string_t
40     | A.Bool -> il_t
41     | A.Float -> float_t
42     | A.Void -> void_t
43     | A.Char -> char_t
44     | A.Unknown -> i32_t
45     | A.Func (args_t, ret_t) -> func_ty args_t ret_t
46     | A.List _ -> L.pointer_type list_struct_ptr
47     | A.File -> file_t

```

```

48 (* get the ptr to the function *)
49 and func_ty args_t ret_t =
50   let llret_type = ltype_of_typ ret_t in
51   let llargs = Array.map (fun t -> ltype_of_typ t) (Array.of_list args_t)
   in
52   L.pointer_type (L.function_type llret_type llargs)
53 in
54 (* print function *)
55 let printf_t : L.lltype =
56   L.var_arg_function_type i32_t [|L.pointer_type i8_t|]
57 in
58 let printf_func : L.llvalue =
59   L.declare_function "printf" printf_t the_module
60 in
61 (* power function *)
62 let pow_t : L.lltype = L.function_type float_t [|float_t; float_t|] in
63 let pow_func : L.llvalue = L.declare_function "pow" pow_t the_module in
64 let open_t : L.lltype = L.function_type string_t [|string_t; string_t|] in
65 let open_func : L.llvalue = L.declare_function "fopen" open_t the_module
   in
66 let close_t : L.lltype = L.function_type i32_t [|file_t|] in
67 let close_func : L.llvalue = L.declare_function "close" close_t the_module
   in
68 let readline_t : L.lltype = L.function_type string_t [|file_t|] in
69 let readline_func : L.llvalue =
70   L.declare_function "readline" readline_t the_module
71 in
72 let write_t : L.lltype = L.function_type string_t [|file_t; string_t|] in
73 let write_func : L.llvalue = L.declare_function "write" write_t the_module
   in
74 let match_t : L.lltype = L.function_type i1_t [|string_t; string_t|] in
75 let match_func : L.llvalue = L.declare_function "match" match_t the_module
   in
76 let find_t : L.lltype = L.function_type string_t [|string_t; string_t|] in
77 let find_func : L.llvalue = L.declare_function "find" find_t the_module in
78 let replace_t : L.lltype =
79   L.function_type string_t [|string_t; string_t; string_t; i32_t|]
80 in
81 let replace_func : L.llvalue =
82   L.declare_function "replace" replace_t the_module
83 in
84 let replaceall_t : L.lltype =
85   L.function_type string_t [|string_t; string_t; string_t|]
86 in
87 let replaceall_func : L.llvalue =
88   L.declare_function "replace_all" replaceall_t the_module
89 in
90 let findall_t : L.lltype =
91   L.function_type (L.pointer_type list_struct_ptr) [|string_t; string_t|]
92 in
93 let findall_func : L.llvalue =
94   L.declare_function "find_all" findall_t the_module
95 in
96 let var_table = {lvariables= StringMap.empty; parent= None} in
97 let globals = ref var_table in
98 let function_decls : L.llvalue StringMap.t =
99   let function_decl m fdecl =
100     let name = fdecl.sfname
101     and formal_types =

```



```

102   Array.of_list (List.map (fun (t, _) -> ltype_of_typ t) fdecl.
      sformals)
103   in
104     let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types in
105     StringMap.add name (L.define_function name ftype the_module) m
106   in
107   List.fold_left function_decl StringMap.empty functions
108 in
109 (* Generate code for a function *)
110 let build_function_body scope fdecl =
111   let the_function = StringMap.find fdecl.sfname function_decls in
112   let builder = L.builder_at_end context (L.entry_block the_function) in
113   let rec find_variable sc n =
114     try Some (StringMap.find n !sc.lvariables)
115     with Not_found -> (
116       match !sc.parent with None -> None | Some t -> find_variable t n )
117   in
118   (* Allocating space for function formal variables *)
119   let formals =
120     let add_formal m (t, n) p =
121       let () = L.set_value_name n p in
122       let local = L.build_alloca (ltype_of_typ t) n builder in
123       let _ = L.build_store p local builder in
124       StringMap.add n local m
125     in
126     List.fold_left2 add_formal StringMap.empty fdecl.sformals
127       (Array.to_list (L.params the_function))
128   in
129   let scope =
130     match fdecl.sfname with
131     | "main" -> scope
132     | _ -> ref {lvariables= formals; parent= Some scope}
133   in
134   let get_list_inner_typ = function
135     | A.List t -> t
136     | _ -> raise (Failure "Internal error: Not a list type")
137   in
138   let add_terminal builder instr =
139     match L.block_terminator (L.insertion_block builder) with
140     | Some _ -> ()
141     | None -> ignore (instr builder)
142   in
143   (* Function to generate code for a semantically checked expression *)
144   let rec expr sc builder ((t, e) : sexpr) =
145     match e with
146     | SIntLit i -> L.const_int i32_t i
147     | SFloatLit f -> L.const_float float_t f
148     | SBoolLit b -> L.const_int i1_t (if b then 1 else 0)
149     | SCharLit c -> L.const_int char_t (Char.code c)
150     | SStringLit s -> L.build_global_stringptr s "string" builder
151     | SId n -> (
152       let var = find_variable sc n in
153       match var with
154       | Some vv -> L.build_load vv n builder
155       | None -> (
156         (* Check if the variable is a function *)
157         match t with
158         | A.Func _ -> StringMap.find n function_decls
159         | _ -> raise (E.NotFound n) ) )

```

```

160 | SListLit l ->
161     (* Build a Linked list *)
162     let lst = L.build_malloc list_struct_ptr "list" builder in
163     let _ = L.build_store (build_list t l sc builder) lst builder in
164     lst
165 | SSliceExpr (lexpr, slice) -> (
166     let lt, _ = lexpr in
167     let l = expr sc builder lexpr in
168     match lt with
169     | A.String -> (
170         match slice with
171         | SIndex i ->
172             (* Getting a character indexed at i from a string *)
173             let ptr =
174                 L.build_gep l [|expr sc builder i|] "get_char_ptr" builder
175             in
176                 L.build_load ptr "get_char" builder
177     | SSlice (i, j) ->
178         (* Getting a slice of a string *)
179         let ptr =
180             L.build_gep l [|expr sc builder i|] "get_char_ptr" builder
181         in
182             let length =
183                 L.build_sub (expr sc builder j) (expr sc builder i) "subb"
184                 builder
185             in
186                 let length_w_nul =
187                     L.build_add length
188                     ((L.const_int i32_t) 1)
189                     "length_w_nul" builder
190                 in
191                     let new_str =
192                         L.build_array_malloc i8_t length_w_nul "new_string"
193                     in
194                         let nul =
195                             L.build_gep new_str [|length|] "string_term" builder
196                         in
197                             (* Adding null character to end of string *)
198                             let _ = L.build_store ((L.const_int i8_t) 0) nul builder in
199                             let mmcpy_t =
200                                 L.function_type void_t
201                                 [|L.pointer_type i8_t; L.pointer_type i8_t; i32_t; i1_t
202 |]
203                             in
204                                 let mmcpy =
205                                     L.declare_function "llvm.memcpy.p0i8.p0i8.i32" mmcpy_t
206                                     the_module
207                                 in
208                                     (* Copying the slice to newly allocated space *)
209                                     let _ =
210   L.build_call mmcpy
211   [|new_str; ptr; length; (L.const_int i1_t) 1|]
212   "" builder
213                                     in
214   new_str )
215     | A.List _ -> (
216         match slice with
217         | SIndex i ->

```

```

217         (* Getting an element at index i *)
218         let la_func = build_access_function () in
219         let l = L.build_load l "ilist" builder in
220         let item_ptr =
221             L.build_call la_func
222             [|l; expr sc builder i|]
223             "_result" builder
224         in
225         let data_ptr_ptr =
226             L.build_struct_gep item_ptr 0 "data_ptr_ptr" builder
227         in
228         let dat_ptr = L.build_load data_ptr_ptr "data_ptr" builder
in
229
230         (* Casting "void" pointer back to data pointer *)
231         let type_casted =
232             L.build_bitcast dat_ptr
233             (L.pointer_type (ltype_of_type t))
234             "cast_data_ptr" builder
235         in
236         L.build_load type_casted "data" builder
| SSlice (i, j) ->
237         (* Getting a list slice *)
238         let la_func = build_access_function () in
239         let l = L.build_load l "ilist" builder in
240         let i = expr sc builder i in
241         let item_ptr = L.build_call la_func [|l; i|] "start" builder
in
242
243         (* Checking for [i:] type of slice*)
244         let j =
245             match j with
246             | _, SEnd -> L.const_int i32_t (-1)
247             | _ -> L.build_sub (expr sc builder j) i "difference"
builder
248
249         in
250         let lc_func = build_copy_function t in
251         let new_list_ptr_ptr =
252             L.build_malloc list_struct_ptr "new_list_ptr_ptr" builder
253         in
254         (* Copy the list slice into a new list *)
255         let _ =
256             L.build_call lc_func
257             [|item_ptr; j; new_list_ptr_ptr|]
258             "" builder
259         in
260         new_list_ptr_ptr )
261     | _ -> raise (Failure "Internal error: invalid slice") )
262 | SBinop (e1, op, e2) ->
263     let t1, _ = e1
264     and t2, _ = e2
265     and e1' = expr sc builder e1
266     and e2' = expr sc builder e2 in
267     (* Check for operand types *)
268     if t1 = A.Float && t2 = A.Int then
269         let cast_e2' = L.build_sitofp e2' float_t "cast" builder in
270         match op with
271         | A.Add -> L.build_fadd e1' cast_e2' "tmp" builder
272         | A.Sub -> L.build_fsub e1' cast_e2' "tmp" builder
273         | A.Mult -> L.build_fmuls e1' cast_e2' "tmp" builder
274         | A.Div -> L.build_fdiv e1' cast_e2' "tmp" builder

```

```

273 | A.Exp -> L.build_call pow_func [|e1'; cast_e2'|] "exp" builder
274 | A.Equal -> L.build_fcmp L.Fcmp.Oeq e1' cast_e2' "tmp" builder
275 | A.Neq -> L.build_fcmp L.Fcmp.One e1' cast_e2' "tmp" builder
276 | A.Less -> L.build_fcmp L.Fcmp.Olt e1' cast_e2' "tmp" builder
277 | A.Leq -> L.build_fcmp L.Fcmp.Ole e1' cast_e2' "tmp" builder
278 | A.Greater -> L.build_fcmp L.Fcmp.Ogt e1' cast_e2' "tmp"
builder
279 | A.Geq -> L.build_fcmp L.Fcmp.Oge e1' cast_e2' "tmp" builder
280 | _ -> raise E.InvalidFloatBinop
281 else if t1 = A.Int && t2 = A.Float then
282   let cast_e1' = L.build_sitofp e1' float_t "cast" builder in
283   match op with
284   | A.Add -> L.build_fadd cast_e1' e2' "tmp" builder
285   | A.Sub -> L.build_fsub cast_e1' e2' "tmp" builder
286   | A.Mult -> L.build_fmul cast_e1' e2' "tmp" builder
287   | A.Div -> L.build_fdiv cast_e1' e2' "tmp" builder
288   | A.Exp -> L.build_call pow_func [|cast_e1'; e2'|] "exp" builder
289   | A.Equal -> L.build_fcmp L.Fcmp.Oeq cast_e1' e2' "tmp" builder
290   | A.Neq -> L.build_fcmp L.Fcmp.One cast_e1' e2' "tmp" builder
291   | A.Less -> L.build_fcmp L.Fcmp.Olt cast_e1' e2' "tmp" builder
292   | A.Leq -> L.build_fcmp L.Fcmp.Ole cast_e1' e2' "tmp" builder
293   | A.Greater -> L.build_fcmp L.Fcmp.Ogt cast_e1' e2' "tmp"
builder
294 | A.Geq -> L.build_fcmp L.Fcmp.Oge cast_e1' e2' "tmp" builder
295 | _ -> raise E.InvalidFloatBinop
296 else if t1 = A.Float && t2 = A.Float then
297   match op with
298   | A.Add -> L.build_fadd e1' e2' "tmp" builder
299   | A.Sub -> L.build_fsub e1' e2' "tmp" builder
300   | A.Mult -> L.build_fmul e1' e2' "tmp" builder
301   | A.Div -> L.build_fdiv e1' e2' "tmp" builder
302   | A.Exp -> L.build_call pow_func [|e1'; e2'|] "exp" builder
303   | A.Equal -> L.build_fcmp L.Fcmp.Oeq e1' e2' "tmp" builder
304   | A.Neq -> L.build_fcmp L.Fcmp.One e1' e2' "tmp" builder
305   | A.Less -> L.build_fcmp L.Fcmp.Olt e1' e2' "tmp" builder
306   | A.Leq -> L.build_fcmp L.Fcmp.Ole e1' e2' "tmp" builder
307   | A.Greater -> L.build_fcmp L.Fcmp.Ogt e1' e2' "tmp" builder
308   | A.Geq -> L.build_fcmp L.Fcmp.Oge e1' e2' "tmp" builder
309   | _ -> raise E.InvalidFloatBinop
310 else if t1 = A.Int && t2 = A.Int then
311   match op with
312   | A.Add -> L.build_add e1' e2' "tmp" builder
313   | A.Sub -> L.build_sub e1' e2' "tmp" builder
314   | A.Mult -> L.build_mul e1' e2' "tmp" builder
315   | A.Div -> L.build_sdiv e1' e2' "tmp" builder
316   | A.Mod -> L.build_srem e1' e2' "tmp" builder
317   | A.Exp ->
318     (* Casting integer to float *)
319     let cast_e1' = L.build_sitofp e1' float_t "cast" builder
320     and cast_e2' = L.build_sitofp e2' float_t "cast" builder in
321     let result =
322       L.build_call pow_func [|cast_e1'; cast_e2'|] "exp" builder
323     in
324     L.build_fptosi result i32_t "result" builder
325 | A.Equal -> L.build_icmp L.Icmp.Eq e1' e2' "tmp" builder
326 | A.Neq -> L.build_icmp L.Icmp.Ne e1' e2' "tmp" builder
327 | A.Less -> L.build_icmp L.Icmp.Slt e1' e2' "tmp" builder
328 | A.Leq -> L.build_icmp L.Icmp.Sle e1' e2' "tmp" builder
329 | A.Greater -> L.build_icmp L.Icmp.Sgt e1' e2' "tmp" builder

```

```

330 | A.Geq -> L.build_icmp L.Icmp.Sge e1' e2' "tmp" builder
331 | A.Range ->
332   (* Generating list for range operator *)
333   let range_function = build_range_function () in
334   let head_ptr_ptr =
335     L.build_malloc list_struct_ptr "head_ptr_ptr" builder
336   in
337   let _ =
338     L.build_store
339     (L.const_null list_struct_ptr)
340     head_ptr_ptr builder
341   in
342   L.build_call range_function
343   [|e1'; e2'; head_ptr_ptr; L.const_int i32_t 0|]
344   "range_list" builder
345 | _ -> raise E.InvalidIntBinop
346 else if t1 = A.Bool && t2 = A.Bool then
347   match op with
348   | A.And -> L.build_and e1' e2' "tmp" builder
349   | A.Or -> L.build_or e1' e2' "tmp" builder
350   | A.Equal -> L.build_icmp L.Icmp.Eq e1' e2' "tmp" builder
351   | A.Neq -> L.build_icmp L.Icmp.Ne e1' e2' "tmp" builder
352   | _ -> raise E.InvalidFloatBinop
353 else if t1 = A.Char && t2 = A.Char then
354   match op with
355   | A.Equal -> L.build_icmp L.Icmp.Eq e1' e2' "tmp" builder
356   | A.Neq -> L.build_icmp L.Icmp.Ne e1' e2' "tmp" builder
357   | _ -> raise E.InvalidFloatBinop
358 else if t1 = A.String && t2 = A.String then
359   match op with
360   | A.Add ->
361     (* Adding two strings together *)
362     let sl_func = build_string_length_function () in
363     let length1 =
364       L.build_call sl_func
365       [|e1'; L.const_int i32_t 0|]
366       "length" builder
367     in
368     let length2 =
369       L.build_call sl_func
370       [|e2'; L.const_int i32_t 0|]
371       "length" builder
372     in
373     let new_length =
374       L.build_add length1 length2 "new_length" builder
375     in
376     (* Adding the sum of lengths of strings with 1 to add null
377     *)
377     let new_length_w_null =
378       L.build_add new_length (L.const_int i32_t 1) "
new_length_nul"
379     builder
380     in
381     let new_str =
382       L.build_array_malloc i8_t new_length_w_null "new_string"
383       builder
384     in
385     (* Copying both strings to newly allocated space *)
386     let mmcpy_t =

```

```

387         L.function_type void_t
388         [|L.pointer_type i8_t; L.pointer_type i8_t; i32_t; i1_t
[]
389
390         in
391         let mmcpy =
392             L.declare_function "llvm.memcpy.p0i8.p0i8.i32" mmcpy_t
393             the_module
394
395         in
396         let _ =
397             L.build_call mmcpy
398             [|new_str; e1'; length1; (L.const_int i1_t) 1|]
399             " builder
400
401         in
402         let new_spot =
403             L.build_gep new_str [|length1|] "new_spot" builder
404
405         in
406         let _ =
407             L.build_call mmcpy
408             [|new_spot; e2'; length2; (L.const_int i1_t) 1|]
409             " builder
410
411         in
412         let nul =
413             L.build_gep new_str [|new_length|] "string_term" builder
414
415         in
416         let _ = L.build_store ((L.const_int i8_t) 0) nul builder in
417         new_str
418     | A.Equal ->
419         (* Checking if strings are equal *)
420         let strcmp_func = build_strcmp_function () in
421         L.build_call strcmp_func [|e1'; e2'|] "strcmp_eq" builder
422     | A.Neq ->
423         let strcmp_func = build_strcmp_function () in
424         let bool_val = L.build_call strcmp_func [|e1'; e2'|] "
strcmp_eq" builder in
425         L.build_select bool_val (L.const_int i1_t 0) (L.const_int
i1_t 1) "strcmp_neq" builder
426     | _ -> raise E.InvalidStringBinop
427 else if t1 = A.String && t2 = A.Char then
428     match op with
429     | A.Add ->
430         (* Adding character to a string *)
431         let sl_func = build_string_length_function () in
432         let length1 =
433             L.build_call sl_func
434             [|e1'; L.const_int i32_t 0|]
435             "length" builder
436
437         in
438         let length2 =
439             (L.const_int i32_t 1)
440
441         in
442         let new_length =
443             L.build_add length1 length2 "new_length" builder
444
445         in
446         let new_length_w_null =
447             L.build_add new_length (L.const_int i32_t 1) "new_length_nul
"
448
449         in
450         builder
451
452         in
453         let new_str =

```

```

442         L.build_array_malloc i8_t new_length_w_null "new_string"
443         builder
444     in
445     let mmcpy_t =
446         L.function_type void_t
447         [|L.pointer_type i8_t; L.pointer_type i8_t; i32_t; i1_t|]
448     in
449     let mmcpy =
450         L.declare_function "llvm.memcpy.p0i8.p0i8.i32" mmcpy_t
451         the_module
452     in
453     let _ =
454         L.build_call mmcpy
455         [|new_str; e1'; length1; (L.const_int i1_t) 1|]
456         "" builder
457     in
458     let new_spot =
459         L.build_gep new_str [|length1|] "new_spot" builder
460     in
461     let _ = L.build_store e2' new_spot builder in
462     let nul =
463         L.build_gep new_str [|new_length|] "string_term" builder
464     in
465     let _ = L.build_store ((L.const_int i8_t) 0) nul builder in
466     new_str
467 | _ -> raise E.InvalidStringBinop
468 else if t1 = A.Char && t2 = A.String then
469     (* Adding String to a character *)
470     match op with
471     | A.Add ->
472         let sl_func = build_string_length_function () in
473         let length1 = (L.const_int i32_t 1)
474         in
475         let length2 =
476             L.build_call sl_func
477             [|e2'; L.const_int i32_t 0|]
478             "length" builder
479         in
480         let new_length =
481             L.build_add length1 length2 "new_length" builder
482         in
483         let new_length_w_null =
484             L.build_add new_length (L.const_int i32_t 1) "new_length_nul
485
486         builder
487     in
488     let new_str =
489         L.build_array_malloc i8_t new_length_w_null "new_string"
490         builder
491     in
492     let mmcpy_t =
493         L.function_type void_t
494         [|L.pointer_type i8_t; L.pointer_type i8_t; i32_t; i1_t|]
495     in
496     let mmcpy =
497         L.declare_function "llvm.memcpy.p0i8.p0i8.i32" mmcpy_t
498         the_module
499     in
500     let _ = L.build_store e1' new_str builder in

```

```

500         let new_spot =
501             L.build_gep new_str [|length1|] "new_spot" builder
502         in
503         let _ =
504             L.build_call mmcpy
505                 [|new_spot; e2'; length2; (L.const_int i1_t) 1|]
506                 "" builder
507         in
508         let nul =
509             L.build_gep new_str [|new_length|] "string_term" builder
510         in
511         let _ = L.build_store ((L.const_int i8_t) 0) nul builder in
512         new_str
513     | _ -> raise E.InvalidStringBinop
514 else if ((match t1 with A.List _ -> true | _ -> false)) then
515     (* Concatenating two lists *)
516     (* Allocating new list *)
517     let new_list = L.build_malloc list_struct_ptr "new_list" builder
518 in
519     let _ = L.build_store (L.const_null list_struct_ptr) new_list
520 builder in
521     let slice = SSlice((A.Int, SIntLit 0), (A.Int, SIntLit 0)) in
522     let ilst = get_list_inner_typ t1 in
523     (* Copying the first list *)
524     let _ = build_asn_list sc builder ilst new_list slice (expr sc
525 builder e2) in
526     (* Copying the second list *)
527     let _ = build_asn_list sc builder ilst new_list slice (expr sc
528 builder e1) in
529     new_list
530 else (
531     print_endline (A.string_of_ttyp t1) ;
532     print_endline (A.string_of_ttyp t2) ;
533     raise (Failure "Not Yet Implemented") )
534 | SUnop (op, e) ->
535     (* Building unary operation code *)
536     let t, _ = e in
537     let e' = expr sc builder e in
538     ( match op with
539     | A.Neg when t = A.Float -> L.build_fneg
540     | A.Neg -> L.build_neg
541     | A.Not -> L.build_not )
542     e' "tmp" builder
543 | SAssign (le, re) ->
544     let re' = expr sc builder re in
545     let _ =
546         match le with
547         (* Updating regular variable assign *)
548         | _, SId s -> update_variable sc s re' builder
549         | _, SSliceExpr ((lst, ls), slc) -> (
550             match lst with
551             | A.List ilst ->
552                 (* Assigning list/element to a list slice/index *)
553                 let lis = expr sc builder (lst, ls) in
554                 build_asn_list sc builder ilst lis slc re'
555             | A.String -> raise (Failure "Cannot assign to string slice")
556             | _ -> raise (Failure "Internal Error") )
557         | _ -> raise (Failure "Internal Error")
558 in

```



```

555     re'
556 | SCall ((_, SId "length"), [(A.List lt, lst)]) ->
557     (* Building length function for list *)
558     let ll_func = build_list_length_function () in
559     let lst = expr sc builder (A.List lt, lst) in
560     let lst = L.build_load lst "ilist" builder in
561     L.build_call ll_func [|lst; L.const_int i32_t 0|] "length" builder
562 | SCall ((_, SId "length"), [(A.String, st)]) ->
563     (* Building length function for string *)
564     let sl_func = build_string_length_function () in
565     L.build_call sl_func
566     [|expr sc builder (A.String, st); L.const_int i32_t 0|]
567     "length" builder
568 (* Declaring builtin functions which are written in C *)
569 | SCall ((_, SId "open"), [(A.String, st); (A.String, st2)]) ->
570     L.build_call open_func
571     [|expr sc builder (A.String, st); expr sc builder (A.String, st2
572 )|]
573     "open" builder
574 | SCall ((_, SId "close"), [(A.File, file)]) ->
575     L.build_call close_func
576     [|expr sc builder (A.File, file)|]
577     "close" builder
578 | SCall ((_, SId "readline"), [(A.File, file)]) ->
579     L.build_call readline_func
580     [|expr sc builder (A.File, file)|]
581     "readline" builder
582 | SCall ((_, SId "write"), [(A.File, file); (A.String, st)]) ->
583     L.build_call write_func
584     [|expr sc builder (A.File, file); expr sc builder (A.String, st)
585 |]
586     "write" builder
587 (* Defining reverse function for lists *)
588 | SCall ((_, SId "reverse"), [(A.List lt, lst)]) ->
589     let reverse_func = build_list_reverse_function () in
590     let list_ptr_ptr = expr sc builder (A.List lt, lst) in
591     let list_ptr_ptr = L.build_load list_ptr_ptr "list_ptr" builder in
592     let new_list_ptr_ptr =
593     L.build_malloc list_struct_ptr "new_list_ptr_ptr" builder
594     in
595     let _ =
596     L.build_store
597     (L.const_null list_struct_ptr)
598     new_list_ptr_ptr builder
599     in
600     let lc_func = build_copy_function (A.List lt) in
601     let _ =
602     L.build_call lc_func
603     [|list_ptr_ptr; L.const_int i32_t (-1); new_list_ptr_ptr|]
604     "last_node_ptr_ptr" builder
605     in
606     L.build_call reverse_func [|new_list_ptr_ptr|] "reversed_list"
607 builder
608 | SCall ((_, SId "print"), args) ->
609     let eval_arg e =
610     let t, _ = e in
611     match t with
612     (* Adding the special case for boolean *)
613     | A.Bool ->

```

```

611         let bool_val = expr sc builder e in
612         let true_str =
613             L.build_global_stringptr "true" "string" builder
614         in
615         let false_str =
616             L.build_global_stringptr "false" "string" builder
617         in
618         let to_print =
619             L.build_select bool_val true_str false_str "bool_to_str"
620             builder
621         in
622         to_print
623     | _ -> expr sc builder e
624 in
625     let arg_list = List.map eval_arg args in
626     L.build_call printf_func (Array.of_list arg_list) "printf" builder
627     (* Declaring regex functions which are defined in C *)
628     | SCall (_, SId "match"), [(A.String, st); (A.String, st2)] ->
629         L.build_call match_func
630         [|expr sc builder (A.String, st); expr sc builder (A.String, st2
631 )|]
632         "match" builder
633     | SCall (_, SId "find"), [(A.String, st); (A.String, st2)] ->
634         L.build_call find_func
635         [|expr sc builder (A.String, st); expr sc builder (A.String, st2
636 )|]
637         "find" builder
638     | SCall
639         ( (_, SId "replace")
640         , [(A.String, st); (A.String, st2); (A.String, st3); (A.Int, i)] )
641     ->
642         L.build_call replace_func
643         [| expr sc builder (A.String, st)
644             ; expr sc builder (A.String, st2)
645             ; expr sc builder (A.String, st3)
646             ; expr sc builder (A.Int, i) |]
647         "replace" builder
648     | SCall
649         ( (_, SId "replaceall")
650         , [(A.String, st); (A.String, st2); (A.String, st3)] ) ->
651         L.build_call replaceall_func
652         [| expr sc builder (A.String, st)
653             ; expr sc builder (A.String, st2)
654             ; expr sc builder (A.String, st3) |]
655         "replaceall" builder
656     | SCall (_, SId "findall"), [(A.String, st); (A.String, st2)] ->
657         L.build_call findall_func
658         [|expr sc builder (A.String, st); expr sc builder (A.String, st2
659 )|]
660         "findall" builder
661     (* Appending to lists *)
662     | SCall (_, SId "append"), [(lt, lst); e] ->
663         let list_ptr_ptr = expr sc builder (lt, lst) in
664         let list_ptr = L.build_load list_ptr_ptr "list_ptr" builder in
665         let e' = expr sc builder e in
666         let ll_func = build_list_length_function () in
667         let length =
668             L.build_call ll_func
669             [|list_ptr; L.const_int i32_t 0|]

```

```

666         "length" builder
667     in
668         (* Using insert function to insert at the end *)
669         let insert_func = build_insert_function lt in
670         L.build_call insert_func
671             [|list_ptr_ptr; e'; length|]
672         "list_ptr_ptr" builder
673     (* Inserting to list at a certain index *)
674     | SCall ((_, SId "insert"), [(lt, lst); e; i]) ->
675         let list_ptr_ptr = expr sc builder (lt, lst) in
676         let e' = expr sc builder e in
677         let i' = expr sc builder i in
678         let insert_func = build_insert_function lt in
679         L.build_call insert_func [|list_ptr_ptr; e'; i'|] "list_ptr_ptr"
680         builder
681     (* Function call *)
682     | SCall (f, args) ->
683         (* Finding the function *)
684         let fdef = expr sc builder f in
685         (* Evaluating the function arguments *)
686         let llarg = List.rev (List.map (expr sc builder) (List.rev args))
in
687         let ret_type =
688             match f with
689             | A.Func (_, rett), _ -> rett
690             | _ -> raise (Failure "Internal Error")
691         in
692         let result = match ret_type with A.Void -> "" | _ -> "_result" in
693         L.build_call fdef (Array.of_list llarg) result builder
694     | SEnd -> raise (Failure "Not Yet Implemented")
695     | SNoexpr -> L.const_null i32_t
696     (* Function to generate code for assigning to list *)
697     and build_asn_list sc builder ilst lis slc re' =
698         match slc with
699         (* In case of assigning to index *)
700         | SIndex i ->
701             let la_func = build_access_function () in
702             let lis = L.build_load lis "ilist" builder in
703             (* Accessing the struct at the given index *)
704             let item_ptr =
705                 L.build_call la_func [|lis; expr sc builder i|] "result" builder
706             in
707             (* Accessing the data field in the struct *)
708             let data_ptr_ptr =
709                 L.build_struct_gep item_ptr 0 "data_ptpt" builder
710             in
711             (* Allocating new space for copying the data *)
712             let copy_data_ptr =
713                 L.build_malloc (ltype_of_typ ilst) "copy_ptr" builder
714             in
715             let _ = L.build_store re' copy_data_ptr builder in
716             (* Casting the data pointer to "void" pointer *)
717             let type_casted_copy =
718                 L.build_bitcast copy_data_ptr (L.pointer_type i8_t) "ccopy"
builder
719         in
720         (* Storing the pointer to data *)
721         let _ = L.build_store type_casted_copy data_ptr_ptr builder in
722         ()

```

```

723 (* In case of assigning a list to a slice *)
724 | SSlice (i, j) ->
725     let lsti = L.build_load lis "ilist" builder in
726     let rei = L.build_load re' "rei" builder in
727     let la_func = build_access_function () in
728     let lc_func = build_copy_function (A.List ilst) in
729     (* Getting the struct at index j *)
730     let end_ptr =
731         match j with
732         | _, SEnd -> L.const_null list_struct_ptr
733         | _ ->
734             L.build_call la_func
735                 [|lsti; expr sc builder j|]
736                 "result" builder
737     in
738     (* Allocating a dummy struct to change the length of list *)
739     let temp = L.build_alloca list_struct_type "temp" builder in
740     let next = L.build_struct_gep temp 1 "next" builder in
741     let _ = L.build_store lsti next builder in
742     (* Getting the struct at index i-1 *)
743     let item_ptr =
744         L.build_call la_func [|temp; expr sc builder i|] "result"
builder
745     in
746     let item_next = L.build_struct_gep item_ptr 1 "item_next" builder
in
747     (* Copying the right hand list to the next pointer at struct i-1
*)
748     let copy_end =
749         L.build_call lc_func
750             [|rei; L.const_int i32_t (-1); item_next|]
751             "copied" builder
752     in
753     (* Copying the list after index j to the new list *)
754     let _ = L.build_store end_ptr copy_end builder in
755     let _ =
756         L.build_store (L.build_load next "next" builder) lis builder
757     in
758     ()
759 (* Function to add a variable to scope *)
760 and add_variable_to_scope sc n v =
761     sc := {lvariables= StringMap.add n v !sc.lvariables; parent= !sc.
parent}
762 and update_variable sc (n : string) (e' : L.llvalue) builder =
763     let l_var =
764         match find_variable sc n with
765         | None -> raise (E.NotFound n)
766         | Some t -> t
767     in
768     let _ = L.build_store e' l_var builder in
769     sc :=
770         {lvariables= StringMap.add n l_var !sc.lvariables; parent= !sc.
parent}
771
772 (* Function to compare strings *)
773 and build_strcmp_function () =
774     match L.lookup_function "strcmp_function" the_module with
775     | Some func -> func
776     | None ->

```

```

777     (* returns 1 if the two strings are the same, 0 if otherwise *)
778     let strcmp_func_t =
779         L.function_type il_t [|string_t; string_t|]
780     in
781     let strcmp_func = L.define_function "strcmp_function"
strcmp_func_t the_module in
782     let strcmp_builder =
783         L.builder_at_end context (L.entry_block strcmp_func)
784     in
785     (* get the arguments *)
786     let stringA = L.param strcmp_func 0 in
787     let stringB = L.param strcmp_func 1 in
788
789     let sl_func = build_string_length_function () in
790     let length1 = L.build_call sl_func [|stringA; L.const_int i32_t
0|] "length" strcmp_builder in
791     let length2 = L.build_call sl_func [|stringB; L.const_int i32_t
0|] "length" strcmp_builder in
792
793     (* if lengths of the two strings are different, return false *)
794     let bool_val = L.build_icmp L.Icmp.Ne length1 length2 "same_length
" strcmp_builder in
795     let then_bb = L.append_block context "then" strcmp_func in
796     let _ = L.build_ret (L.const_int il_t 0) (L.builder_at_end context
then_bb) in
797     let else_bb = L.append_block context "else" strcmp_func in
798     let else_builder = L.builder_at_end context else_bb in
799     let strcmp_helper_func = build_strcmp_helper_function () in
800     let last_index =
801         L.build_sub length1 (L.const_int i32_t 1) "last_index"
else_builder
802     in
803     (* call strcmp_helper_func and return its result *)
804     let ret =
805         L.build_call strcmp_helper_func [|stringA; stringB; last_index;
(L.const_int i32_t 0)|]
806         "res" else_builder
807     in
808     let _ = L.build_ret ret else_builder in
809     let _ = L.build_cond_br bool_val then_bb else_bb strcmp_builder in
810     strcmp_func
811
812     (* helper function used by build_strcmp_function *)
813     and build_strcmp_helper_function () =
814         match L.lookup_function "strcmp_helper_function" the_module with
815         | Some func -> func
816         | None ->
817             let strcmp_helper_func_t =
818                 L.function_type il_t [|string_t; string_t; i32_t; i32_t|]
819             in
820             let strcmp_helper_func = L.define_function "strcmp_helper_function
" strcmp_helper_func_t the_module in
821             let strcmp_helper_builder =
822                 L.builder_at_end context (L.entry_block strcmp_helper_func)
823             in
824
825             (* get the arguments *)
826             let stringA = L.param strcmp_helper_func 0 in
827             let stringB = L.param strcmp_helper_func 1 in

```

```

828     let last_index = L.param strcmp_helper_func 2 in
829     let index = L.param strcmp_helper_func 3 in
830
831     (* load the character at index *)
832     let charA_ptr = L.build_gep stringA [|index|] "charA_ptr"
strcmp_helper_builder in
833     let charA = L.build_load charA_ptr "charA" strcmp_helper_builder
in
834     let charB_ptr = L.build_gep stringB [|index|] "charB_ptr"
strcmp_helper_builder in
835     let charB = L.build_load charB_ptr "charB" strcmp_helper_builder
in
836
837     (* if the character at index is not the same, return false *)
838     let bool_not_same_val = L.build_icmp L.Icmp.Ne charA charB "
not_same" strcmp_helper_builder in
839     let then_not_same_bb = L.append_block context "then_not_same"
strcmp_helper_func in
840     let _ = L.build_ret (L.const_int i1_t 0) (L.builder_at_end context
then_not_same_bb) in
841     let else_same_bb = L.append_block context "else_same"
strcmp_helper_func in
842     let else_same_builder = L.builder_at_end context else_same_bb in
843
844     (* if already at the last character, return true *)
845     let bool_at_end_val = L.build_icmp L.Icmp.Eq index last_index "
last_char" else_same_builder in
846     let then_end_bb = L.append_block context "then_end"
strcmp_helper_func in
847     let _ = L.build_ret (L.const_int i1_t 1) (L.builder_at_end context
then_end_bb) in
848     let else_not_end_bb = L.append_block context "else_not_end"
strcmp_helper_func in
849     let else_not_end_builder = L.builder_at_end context
else_not_end_bb in
850     let next_index =
851         L.build_add (L.const_int i32_t 1) index "next_index"
else_not_end_builder
852     in
853
854     (* recursively call strcmp_helper_func to compare the rest of the
string *)
855     let ret =
856         L.build_call strcmp_helper_func [|stringA; stringB; last_index;
next_index|]
857         "res" else_not_end_builder
858     in
859     let _ = L.build_ret ret else_not_end_builder in
860     let _ = L.build_cond_br bool_at_end_val then_end_bb
else_not_end_bb else_same_builder in
861     let _ = L.build_cond_br bool_not_same_val then_not_same_bb
else_same_bb strcmp_helper_builder in
862     strcmp_helper_func
863
864     (* Function to generate list from a range *)
865     and build_range_function () =
866         match L.lookup_function "range_function" the_module with
867         | Some func -> func
868         | None ->

```

```

869     let range_func_t =
870         L.function_type
871         (L.pointer_type list_struct_ptr)
872         [|i32_t; i32_t; L.pointer_type list_struct_ptr; i32_t|]
873     in
874     let range_func = L.define_function "range_function" range_func_t
the_module in
875     let range_builder =
876         L.builder_at_end context (L.entry_block range_func)
877     in
878
879     (* get the arguments *)
880     let s = L.param range_func 0 in
881     let e = L.param range_func 1 in
882     let head_ptr_ptr = L.param range_func 2 in
883     let curr_length = L.param range_func 3 in
884
885     (* return the list generated if the last element is generated *)
886     let bool_val = L.build_icmp L.Icmp.Eq s e "is_last" range_builder
in
887     let then_bb = L.append_block context "then" range_func in
888     let _ = L.build_ret head_ptr_ptr (L.builder_at_end context then_bb
) in
889     let else_bb = L.append_block context "else" range_func in
890     let else_builder = L.builder_at_end context else_bb in
891
892     (* call insert to append the next element *)
893     let insert_func = build_insert_function (A.List A.Int) in
894     let head_ptr_ptr =
895         L.build_call insert_func
896         [|head_ptr_ptr; s; curr_length|]
897         "head_ptr_ptr" else_builder
898     in
899     let next_s =
900         L.build_add (L.const_int i32_t 1) s "next_s" else_builder
901     in
902     let next_length =
903         L.build_add (L.const_int i32_t 1) curr_length "next_length"
904         else_builder
905     in
906
907     (* recursively call range_func to generate the rest of the
elements *)
908     let ret =
909         L.build_call range_func
910         [|next_s; e; head_ptr_ptr; next_length|]
911         "" else_builder
912     in
913     let _ = L.build_ret ret else_builder in
914     let _ = L.build_cond_br bool_val then_bb else_bb range_builder in
915     range_func
916
917     (* Function to deep copy a list *)
918     and build_copy_function typ =
919         let t = get_list_inner_typ typ in
920         let func_name = "list_copy_" ^ A.string_of_typ t in
921         (* Checking if the function is already defined *)
922         match L.lookup_function func_name the_module with
923         | Some func -> func

```

```

924 | None ->
925     (* Function defined takes list, length to be copied and space
926        where we want the list to be copied *)
927     let lc_func_t =
928         L.function_type
929             (L.pointer_type list_struct_ptr)
930             [|list_struct_ptr; i32_t; L.pointer_type list_struct_ptr|]
931     in
932     let lc_func = L.define_function func_name lc_func_t the_module in
933     let lc_builder = L.builder_at_end context (L.entry_block lc_func)
934 in
935     (* Checking if the index is 0 which means list is done copying*)
936     let i_cond =
937         L.build_icmp L.Icmp.Eq (L.param lc_func 1) (L.const_int i32_t 0)
938         "is_zero" lc_builder
939     in
940     (* Checking if the list being copied is a null pointer *)
941     let n_cond =
942         L.build_is_null (L.param lc_func 0) "ptr_is_null" lc_builder
943     in
944     let bool_val = L.build_or i_cond n_cond "or_conds" lc_builder in
945     let then_bb = L.append_block context "then" lc_func in
946     (* Base case of recursion: Return the pointer to the next pointer
947        of the struct at the end *)
948     let _ =
949         L.build_ret (L.param lc_func 2) (L.builder_at_end context
950         then_bb)
951     in
952     let else_bb = L.append_block context "else" lc_func in
953     let else_builder = L.builder_at_end context else_bb in
954     (* Allocating space for a new struct *)
955     let new_struct_ptr =
956         L.build_malloc list_struct_type "new_struct_ptr" else_builder
957     in
958     let _ =
959         L.build_store
960             (L.const_null list_struct_type)
961             new_struct_ptr else_builder
962     in
963     (* Allocating space for the data *)
964     let data_ptr = L.build_malloc (ltype_of_typ t) "ltyp" else_builder
965 in
966     (* Getting the old data to be copied *)
967     let old_data_ptr_ptr =
968         L.build_struct_gep (L.param lc_func 0) 0 "old_data_ptr_ptr"
969         else_builder
970     in
971     let old_data_ptr =
972         L.build_load old_data_ptr_ptr "old_data_ptr" else_builder
973     in
974     (* Casting the "void" pointer back to data pointer *)
975     let old_data_ptr =
976         L.build_bitcast old_data_ptr
977             (L.pointer_type (ltype_of_typ t))
978             "cast_old_data_ptr" else_builder
979     in
980     let old_data = L.build_load old_data_ptr "old_data" else_builder
981 in
982     (* Copy the old data into newly allocated space *)

```



```

979     let _ = L.build_store old_data data_ptr else_builder in
980     (* Casting the data pointer to "void" pointer *)
981     let data_ptr_cast =
982         L.build_bitcast data_ptr (L.pointer_type i8_t) "data_ptr_cast"
983         else_builder
984     in
985     (* Storing pointer to copied data in the new struct *)
986     let _ =
987         L.build_store data_ptr_cast
988         (L.build_struct_gep new_struct_ptr 0 "store_new_data"
else_builder)
989         else_builder
990     in
991     (* Storing the pointer to new struct in the previous struct's
992       next struct pointer field *)
993     let _ =
994         L.build_store new_struct_ptr (L.param lc_func 2) else_builder
995     in
996     let ptr_ptr =
997         L.build_struct_gep new_struct_ptr 1 "next" else_builder
998     in
999     (* Getting the pointer to next struct pointer field
1000       in current struct *)
1001     let next_ptr =
1002         L.build_struct_gep (L.param lc_func 0) 1 "next_ptr" else_builder
1003     in
1004     let next = L.build_load next_ptr "next" else_builder in
1005     (* Subtracting 1 from the index *)
1006     let sub =
1007         L.build_sub (L.param lc_func 1) (L.const_int i32_t 1) "sub"
1008         else_builder
1009     in
1010     (* Recursively calling the function for rest of the list *)
1011     let ret =
1012         L.build_call lc_func [|next; sub; ptr_ptr|] "" else_builder
1013     in
1014     let _ = L.build_ret ret else_builder in
1015     let _ = L.build_cond_br bool_val then_bb else_bb lc_builder in
1016     lc_func
1017
1018 and build_string_length_function () =
1019     match L.lookup_function "string_length" the_module with
1020     | Some func -> func
1021     | None ->
1022         (* Function takes in the string and current length *)
1023         let sl_func_t = L.function_type i32_t [|string_t; i32_t|] in
1024         let sl_func =
1025             L.define_function "string_length" sl_func_t the_module
1026         in
1027         let sl_builder = L.builder_at_end context (L.entry_block sl_func)
in
1028         (* Checking if the string is just a null character *)
1029         let bool_val =
1030             L.build_is_null
1031             (L.build_load (L.param sl_func 0) "char" sl_builder)
1032             "ptr_is_null" sl_builder
1033         in
1034         let then_bb = L.append_block context "then" sl_func in
1035         (* Returning the current length after finding a null character *)

```

```

1036     let _ =
1037         L.build_ret (L.param sl_func 1) (L.builder_at_end context
then_bb)
1038     in
1039     let else_bb = L.append_block context "else" sl_func in
1040     let else_builder = L.builder_at_end context else_bb in
1041     (* Getting pointer to next character in the string *)
1042     let next =
1043         L.build_gep (L.param sl_func 0)
1044             [|L.const_int i32_t 1|]
1045         "next_ptr" else_builder
1046     in
1047     (* Adding 1 to the current length *)
1048     let add =
1049         L.build_add (L.param sl_func 1) (L.const_int i32_t 1) "add"
1050         else_builder
1051     in
1052     (* Recursively calling length function with rest of the string*)
1053     let ret = L.build_call sl_func [|next; add|] "result" else_builder
in
1054     let _ = L.build_ret ret else_builder in
1055     let _ = L.build_cond_br bool_val then_bb else_bb sl_builder in
1056     sl_func
1057
1058     (* list_reverse_helper is used by list_reverse *)
1059     and build_list_reverse_helper_function () =
1060         match L.lookup_function "list_reverse_helper" the_module with
1061         | Some func -> func
1062         | None ->
1063             let reverse_helper_func_t =
1064                 L.function_type
1065                     (L.pointer_type list_struct_ptr)
1066                     [|L.pointer_type list_struct_ptr; L.pointer_type
list_struct_ptr|]
1067             in
1068             let reverse_helper_func =
1069                 L.define_function "list_reverse_helper" reverse_helper_func_t
1070                 the_module
1071             in
1072             let reverse_helper_builder =
1073                 L.builder_at_end context (L.entry_block reverse_helper_func)
1074             in
1075
1076             (* get arguments *)
1077             let prev_node_ptr_ptr = L.param reverse_helper_func 0 in
1078             let curr_node_ptr_ptr = L.param reverse_helper_func 1 in
1079
1080             (* get pointer to the previous node *)
1081             let prev_node_ptr =
1082                 L.build_load prev_node_ptr_ptr "prev_node_ptr"
1083                 reverse_helper_builder
1084             in
1085
1086             (* get pointer to the current node *)
1087             let curr_node_ptr =
1088                 L.build_load curr_node_ptr_ptr "curr_node_ptr"
1089                 reverse_helper_builder
1090             in
1091

```

```

1092     (* get next node *)
1093     let next_node_ptr_ptr =
1094         L.build_struct_gep curr_node_ptr 1 "next_node_ptr_ptr"
1095         reverse_helper_builder
1096     in
1097     let next_node_ptr =
1098         L.build_load next_node_ptr_ptr "next_node_ptr"
1099         reverse_helper_builder
1100     in
1101
1102     (* reverse direction *)
1103     let temp_ptr_ptr =
1104         L.build_malloc list_struct_ptr "temp_ptr_ptr"
reverse_helper_builder
1105     in
1106     let _ =
1107         L.build_store next_node_ptr temp_ptr_ptr reverse_helper_builder
1108     in
1109
1110     (* return pointer to the new list if the last element is reached
1111     *)
1112     let bool_val =
1113         L.build_is_null next_node_ptr "ptr_is_null"
reverse_helper_builder
1114     in
1115     let _ =
1116         L.build_store prev_node_ptr next_node_ptr_ptr
reverse_helper_builder
1117     in
1118     let then_bb = L.append_block context "then" reverse_helper_func in
1119     let _ =
1120         L.build_ret curr_node_ptr_ptr (L.builder_at_end context then_bb)
1121     in
1122     let else_bb = L.append_block context "else" reverse_helper_func in
1123     let else_builder = L.builder_at_end context else_bb in
1124
1125     (* recursively call reverse_helper_func to reverse the rest of the
1126     list *)
1127     let ret =
1128         L.build_call reverse_helper_func
1129         [|curr_node_ptr_ptr; temp_ptr_ptr|]
1130         "result" else_builder
1131     in
1132     let _ = L.build_ret ret else_builder in
1133     let _ =
1134         L.build_cond_br bool_val then_bb else_bb reverse_helper_builder
1135     in
1136     reverse_helper_func
1137
1138     (* building reverse function *)
1139     and build_list_reverse_function () =
1140         match L.lookup_function "list_reverse" the_module with
1141         | Some func -> func
1142         | None ->
1143             let reverse_func_t =
1144                 L.function_type
1145                 (L.pointer_type list_struct_ptr)
1146                 [|L.pointer_type list_struct_ptr|]
1147             in

```

```

1146     let reverse_func =
1147         L.define_function "list_reverse" reverse_func_t the_module
1148     in
1149     let reverse_builder =
1150         L.builder_at_end context (L.entry_block reverse_func)
1151     in
1152
1153     (* get arguments *)
1154     let list_ptr_ptr = L.param reverse_func 0 in
1155     let list_ptr = L.build_load list_ptr_ptr "list_ptr"
reverse_builder in
1156
1157     (* return the list if the head is null *)
1158     let bool_val_is_head_null =
1159         L.build_is_null list_ptr "ptr_is_null" reverse_builder
1160     in
1161     let then_head_null_bb = L.append_block context "then" reverse_func
in
1162     let _ =
1163         L.build_ret list_ptr_ptr
1164         (L.builder_at_end context then_head_null_bb)
1165     in
1166     let else_head_not_null_bb =
1167         L.append_block context "else" reverse_func
1168     in
1169     let else_head_not_null_builder =
1170         L.builder_at_end context else_head_not_null_bb
1171     in
1172
1173     (* get pointer to the list starting from the next node *)
1174     let next_ptr_ptr =
1175         L.build_struct_gep list_ptr 1 "next_ptr_ptr"
1176         else_head_not_null_builder
1177     in
1178     let next_ptr =
1179         L.build_load next_ptr_ptr "next_ptr" else_head_not_null_builder
1180     in
1181
1182     (* return the list if the list is a singleton *)
1183     let bool_val_is_next_null =
1184         L.build_is_null next_ptr "next_ptr_is_null"
1185         else_head_not_null_builder
1186     in
1187     let then_next_null_bb = L.append_block context "then_"
reverse_func in
1188     let _ =
1189         L.build_ret list_ptr_ptr
1190         (L.builder_at_end context then_next_null_bb)
1191     in
1192     let else_next_not_null_bb =
1193         L.append_block context "else_" reverse_func
1194     in
1195     let else_next_not_null_builder =
1196         L.builder_at_end context else_next_not_null_bb
1197     in
1198
1199     (* call reverse_helper_function to reverse the list *)
1200     let list_reverse_helper_function =
1201         build_list_reverse_helper_function ()

```

```

1202     in
1203     let head_ptr_ptr =
1204         L.build_call list_reverse_helper_function
1205             [|list_ptr_ptr; next_ptr_ptr|]
1206         "result" else_next_not_null_builder
1207     in
1208
1209     (* the next pointer of the head now points to nothing *)
1210     let _ =
1211         L.build_store
1212             (L.const_null list_struct_ptr)
1213             next_ptr_ptr else_next_not_null_builder
1214     in
1215     let _ = L.build_ret head_ptr_ptr else_next_not_null_builder in
1216     let _ =
1217         L.build_cond_br bool_val_is_head_null then_head_null_bb
1218             else_head_not_null_bb reverse_builder
1219     in
1220     let _ =
1221         L.build_cond_br bool_val_is_next_null then_next_null_bb
1222             else_next_not_null_bb else_head_not_null_builder
1223     in
1224     reverse_func
1225
1226     (* Building insert function *)
1227     and build_insert_function typ =
1228         let t = get_list_inner_type typ in
1229         (* insert function has to be defined for different types *)
1230         let func_name = "insert_" ^ A.string_of_type t in
1231         match L.lookup_function func_name the_module with
1232         | Some func -> func
1233         | None ->
1234             let ltype = ltype_of_type t in
1235             let insert_func_t =
1236                 L.function_type
1237                     (L.pointer_type list_struct_ptr)
1238                     [|L.pointer_type list_struct_ptr; ltype; i32_t|]
1239             in
1240             let insert_func =
1241                 L.define_function func_name insert_func_t the_module
1242             in
1243             let insert_builder =
1244                 L.builder_at_end context (L.entry_block insert_func)
1245             in
1246
1247             (* get arguments *)
1248             let list_ptr_ptr = L.param insert_func 0 in
1249             let e' = L.param insert_func 1 in
1250             let i' = L.param insert_func 2 in
1251             let list_ptr = L.build_load list_ptr_ptr "list_ptr" insert_builder
1252         in
1253
1254         (* malloc space for a new list *)
1255         let new_list_ptr_ptr =
1256             L.build_malloc list_struct_ptr "new_list_ptr_ptr" insert_builder
1257         in
1258         let _ =
1259             L.build_store
1260                 (L.const_null list_struct_ptr)

```

```

1260         new_list_ptr_ptr insert_builder
1261     in
1262
1263     (* get the length of the function *)
1264     let lc_func = build_copy_function typ in
1265
1266     (* copy the list to a new one so that the old list is not mutated
1267 *)
1268     let _ =
1269         L.build_call lc_func
1270         [|list_ptr; L.const_int i32_t (-1); new_list_ptr_ptr|]
1271         "last_node_ptr_ptr" insert_builder
1272     in
1273     let new_list_ptr =
1274         L.build_load new_list_ptr_ptr "new_list_ptr" insert_builder
1275     in
1276     let la_func = build_access_function () in
1277     let temp = L.build_alloca list_struct_type "temp" insert_builder
1278 in
1279     let next = L.build_struct_gep temp 1 "next" insert_builder in
1280     let _ = L.build_store new_list_ptr next insert_builder in
1281
1282     (* malloc space for the actual data *)
1283     let dat_struct =
1284         L.build_malloc list_struct_type "data_node" insert_builder
1285     in
1286
1287     (* malloc space for the pointer to the data *)
1288     let dat_ptr = L.build_malloc ltype "data" insert_builder in
1289     let _ = L.build_store e' dat_ptr insert_builder in
1290
1291     (* store the data pointer to the node struct *)
1292     let dat_ptr_ptr =
1293         L.build_struct_gep dat_struct 0 "dat" insert_builder
1294     in
1295     let type_casted =
1296         L.build_bitcast dat_ptr (L.pointer_type i8_t) "cast"
1297     in
1298     insert_builder
1299     in
1300     let _ = L.build_store type_casted dat_ptr_ptr insert_builder in
1301
1302     (* get the node specified by the index *)
1303     let item_ptr =
1304         L.build_call la_func [|temp; i'|] "result" insert_builder
1305     in
1306     let cur_next = L.build_struct_gep item_ptr 1 "test" insert_builder
1307 in
1308
1309     (* keep a record of what next currently pointer to *)
1310     let _ =
1311         L.build_store
1312         (L.build_load cur_next "temp" insert_builder)
1313         (L.build_struct_gep dat_struct 1 "dat" insert_builder)
1314     in
1315     insert_builder
1316 in
1317
1318     (* insertion happens here *)
1319     let _ = L.build_store dat_struct cur_next insert_builder in
1320

```

```

1315     (* connect old list to the new one *)
1316     let _ =
1317         L.build_store
1318         (L.build_load next "temp" insert_builder)
1319         new_list_ptr_ptr insert_builder
1320     in
1321     let _ = L.build_ret new_list_ptr_ptr insert_builder in
1322     insert_func
1323 (* Building function to get length of list *)
1324 and build_list_length_function () =
1325     match L.lookup_function "list_length" the_module with
1326     | Some func -> func
1327     | None ->
1328         (* Functiont takes a list and current length *)
1329         let ll_func_t = L.function_type i32_t [|list_struct_ptr; i32_t|]
1330     in
1331         let ll_func = L.define_function "list_length" ll_func_t the_module
1332         in
1333         let ll_builder = L.builder_at_end context (L.entry_block ll_func)
1334         in
1335             (* Checking if a list is null *)
1336             let bool_val =
1337                 L.build_is_null (L.param ll_func 0) "ptr_is_null" ll_builder
1338             in
1339             let then_bb = L.append_block context "then" ll_func in
1340             (* Return the current length if list is null *)
1341             let _ =
1342                 L.build_ret (L.param ll_func 1) (L.builder_at_end context
1343                 then_bb)
1344             in
1345             let else_bb = L.append_block context "else" ll_func in
1346             let else_builder = L.builder_at_end context else_bb in
1347             (* Get the next struct in the list *)
1348             let next_ptr =
1349                 L.build_struct_gep (L.param ll_func 0) 1 "next_ptr" else_builder
1350             in
1351             let next = L.build_load next_ptr "next" else_builder in
1352             (* Add 1 to the current length *)
1353             let add =
1354                 L.build_add (L.param ll_func 1) (L.const_int i32_t 1) "add"
1355                 else_builder
1356             in
1357             (* Recursively call the function with the rest of the list *)
1358             let ret = L.build_call ll_func [|next; add|] "result" else_builder
1359         in
1360             let _ = L.build_ret ret else_builder in
1361             let _ = L.build_cond_br bool_val then_bb else_bb ll_builder in
1362             ll_func
1363 (* Function to access a node at certain index *)
1364 and build_access_function () =
1365     match L.lookup_function "list_access" the_module with
1366     | Some func -> func
1367     | None ->
1368         (* Function takes in a list and index *)
1369         let la_func_t =
1370             L.function_type list_struct_ptr [|list_struct_ptr; i32_t|]
1371         in
1372         let la_func = L.define_function "list_access" la_func_t the_module
1373         in

```

```

1368     let la_builder = L.builder_at_end context (L.entry_block la_func)
1369   in
1370     (* Check if we are at the end of the list by comparing with null
1371      *)
1372     let bool_val =
1373       L.build_icmp L.Icmp.Eq (L.param la_func 1) (L.const_int i32_t 0)
1374       "is_zero" la_builder
1375     in
1376     let then_bb = L.append_block context "then" la_func in
1377     (* Return the current struct if index is 0 *)
1378     let _ =
1379       L.build_ret (L.param la_func 0) (L.builder_at_end context
1380 then_bb)
1381     in
1382     let else_bb = L.append_block context "else" la_func in
1383     let else_builder = L.builder_at_end context else_bb in
1384     (* Get the next node in the list *)
1385     let next_ptr =
1386       L.build_struct_gep (L.param la_func 0) 1 "next_ptr" else_builder
1387     in
1388     let next = L.build_load next_ptr "next" else_builder in
1389     (* Subtract 1 from the index *)
1390     let sub =
1391       L.build_sub (L.param la_func 1) (L.const_int i32_t 1) "sub"
1392       else_builder
1393     in
1394     (* Recursively call with the rest of the list *)
1395     let ret = L.build_call la_func [|next; sub|] "result" else_builder
1396   in
1397     let _ = L.build_ret ret else_builder in
1398     let _ = L.build_cond_br bool_val then_bb else_bb la_builder in
1399     la_func
1400
1401   (* Function to build a list *)
1402   and build_list list_typ lis (scope : var_table ref) builder =
1403     let typ = get_list_inner_typ list_typ in
1404     let ltyp = ltype_of_type typ in
1405     (* Function to build an individual struct *)
1406     let build_link prev data =
1407       (* Allocate a single struct *)
1408       let entry_ptr = L.build_malloc list_struct_type "list_item" builder
1409     in
1410       (* Initialize the allocated struct with null *)
1411       let _ =
1412         L.build_store (L.const_null list_struct_type) entry_ptr builder
1413       in
1414       (* Allocate space for the data *)
1415       let data_ptr = L.build_malloc ltyp "copied" builder in
1416       (* Store data in the allocated space *)
1417       let _ = L.build_store (expr scope builder data) data_ptr builder in
1418       (* Cast data pointer to "void" pointer *)
1419       let typcast_ptr =
1420         L.build_bitcast data_ptr (L.pointer_type i8_t) "cast_ptr" builder
1421       in
1422       (* Get the pointer to data pointer in the struct *)
1423       let data_ptr_container =
1424         L.build_struct_gep entry_ptr 0 "data_ptr_container" builder
1425       in
1426       (* Store the typecasted data pointer in the struct *)

```



```

1422     let _ = L.build_store typcast_ptr data_ptr_container builder in
1423     (* Get the pointer to next struct field *)
1424     let next = L.build_struct_gep entry_ptr 1 "next" builder in
1425     (* Store the previous struct ptr in current struct's next field *)
1426     let _ = L.build_store prev next builder in
1427     entry_ptr
1428   in
1429   let null_ptr = L.const_pointer_null list_struct_ptr in
1430   List.fold_left build_link null_ptr (List.rev lis)
1431
1432   (* Function to build code for Statements *)
1433   and build_stmt sc builder stmt loop =
1434     match stmt with
1435     | SBlock sl ->
1436       (* Create a new scope and build instruction for all statements *)
1437       let new_scope = ref {lvariables= StringMap.empty; parent= Some sc}
1438       in
1439       List.fold_left (fun b s -> build_stmt new_scope b s loop) builder
1440       sl
1441     | SExpr e ->
1442       (* Build instruction for expression *)
1443       let _ = expr sc builder e in
1444       builder
1445     | SReturn e ->
1446       (* Generate a return statement *)
1447       let _ =
1448         match fdecl.styp with
1449         | A.Void -> L.build_ret_void builder
1450         | _ -> L.build_ret (expr sc builder e) builder
1451       in
1452       builder
1453     | SIf (predicate, then_stmt, else_stmt) ->
1454       (* Same implementation as microc *)
1455       let bool_val = expr sc builder predicate in
1456       let merge_bb = L.append_block context "merge" the_function in
1457       let branch_instr = L.build_br merge_bb in
1458       let then_bb = L.append_block context "then" the_function in
1459       let then_builder =
1460         build_stmt sc (L.builder_at_end context then_bb) then_stmt loop
1461       in
1462       let () = add_terminal then_builder branch_instr in
1463       let else_bb = L.append_block context "else" the_function in
1464       let else_builder =
1465         build_stmt sc (L.builder_at_end context else_bb) else_stmt loop
1466       in
1467       let () = add_terminal else_builder branch_instr in
1468       let _ = L.build_cond_br bool_val then_bb else_bb builder in
1469       L.builder_at_end context merge_bb
1470     | SFor (s, (t, e), sl) ->
1471       (* An equivalent for loop code written using while loop and
1472        and other statemtns *)
1473       let equivalent =
1474         match t with
1475         | A.List s_ty ->
1476           (* Get length of the list *)
1477           let len_call =
1478             ( A.Int
1479             , SCall
1480             ((A.Func ([A.List A.Int], A.Int), SId "length"), [(t,

```

```

e))
1479         )
1480     in
1481     (* Create a new variable for index *)
1482     let index_expr = (A.Int, SId "for_index") in
1483     let while_cond =
1484         (A.Bool, SBinop (index_expr, A.Less, len_call))
1485     in
1486     (* Create a new variable to store the current list entry *)
1487     let element = (s_ty, SId s) in
1488     SBlock
1489         (* Declare the index variable *)
1490         [ SDeclaration (A.Int, "for_index", (A.Int, SIntLit 0))
1491         ; SDeclaration (s_ty, s, (A.Void, SNoexpr))
1492         ; SWhile
1493             ( while_cond
1494             , SBlock
1495                 [ SExpr
1496                     ( s_ty
1497                     , SAssign
1498                         ( element
1499                         , ( s_ty
1500                             , SSliceExpr ((t, e), SIndex index_expr)
1501                         ) )
1502                     ; SExpr
1503                         ( A.Int
1504                         , SAssign
1505                             ( index_expr
1506                             , ( A.Int
1507                                 , SBinop
1508                                     (index_expr, A.Add, (A.Int, SIntLit
1509   1))
1510                                 ) ) )
1511                         ; sl ] ) ]
1512     (* Using a similar strategy to for loop for lists*)
1513     | A.String ->
1514     let len_call =
1515         ( A.Int
1516         , SCall ((A.Func ([A.String], A.Int), SId "length"), [(t,
1517 e))
1518         )
1519     in
1520     let index_expr = (A.Int, SId "for_index") in
1521     let while_cond =
1522         (A.Bool, SBinop (index_expr, A.Less, len_call))
1523     in
1524     let element = (A.Char, SId s) in
1525     SBlock
1526         [ SDeclaration (A.Int, "for_index", (A.Int, SIntLit 0))
1527         ; SDeclaration (A.Char, s, (A.Void, SNoexpr))
1528         ; SWhile
1529             ( while_cond
1530             , SBlock
1531                 [ SExpr
1532                     ( A.Char
1533                     , SAssign
1534                         ( element
1535                         , ( A.Char

```

```

1534         , SSliceExpr ((t, e), SIndex index_expr)
1535     )
1536     ; SExpr
1537     ( A.Int
1538     , SAssign
1539     ( index_expr
1540     , ( A.Int
1541     , SBinop
1542     (index_expr, A.Add, (A.Int, SIntLit
1543     1))
1544     ) ) )
1545     ; sl ] ) ]
1546     | _ -> raise (Failure "internal error")
1547     in
1548     build_stmt sc builder equivalent loop
1549     | SDeclaration (t, n, e) ->
1550     let e =
1551     match e with
1552     (* Handling case for declaration without initial value *)
1553     | A.Void, SNoexpr -> (
1554     match t with
1555     | A.List _ ->
1556     let ptr_ptr =
1557     L.build_malloc list_struct_ptr "ptr_ptr" builder
1558     in
1559     let _ =
1560     L.build_store (L.const_null list_struct_ptr) ptr_ptr
1561     builder
1562     in
1563     ptr_ptr
1564     | A.String -> L.build_global_stringptr "" "string" builder
1565     | _ -> L.const_null (ltype_of_type t) )
1566     | asd -> expr sc builder asd
1567     in
1568     let _ =
1569     match fdecl.sfname with
1570     (* When Variable is declared on the top level *)
1571     | "main" ->
1572     (* Allocate as global for variables on top level *)
1573     let global =
1574     L.define_global n (L.const_null (ltype_of_type t))
1575     the_module
1576     in
1577     let _ = L.build_store e global builder in
1578     add_variable_to_scope sc n global
1579     (* When the variable is declared inside a function *)
1580     | _ ->
1581     (* Allocate space for variable on the top of function*)
1582     let init_pos = L.instr_begin (L.entry_block the_function) in
1583     let new_builder = L.builder_at context init_pos in
1584     let local = L.build_alloca (ltype_of_type t) n new_builder in
1585     let _ = L.build_store e local builder in
1586     add_variable_to_scope sc n local
1587     in
1588     builder
1589     | SWhile (predicate, body) ->
1590     (* Similar strategy as microc *)
1591     let pred_bb = L.append_block context "while" the_function in

```

```

1589     let _ = L.build_br pred_bb builder in
1590     let merge_bb = L.append_block context "merge" the_function in
1591     let body_bb = L.append_block context "while_body" the_function in
1592     (* Pass the builder for break and continue statements to refer *)
1593     let while_builder =
1594         build_stmt sc
1595             (L.builder_at_end context body_bb)
1596             body
1597             ((pred_bb, merge_bb) :: loop)
1598     in
1599     let () = add_terminal while_builder (L.build_br pred_bb) in
1600     let pred_builder = L.builder_at_end context pred_bb in
1601     let bool_val = expr sc pred_builder predicate in
1602     let _ = L.build_cond_br bool_val body_bb merge_bb pred_builder in
1603     L.builder_at_end context merge_bb
1604     (* Add instruction to builders passed from a loop *)
1605     | SBreak ->
1606         let () = add_terminal builder (L.build_br (snd (List.hd loop))) in
1607         builder
1608     | SContinue ->
1609         let () = add_terminal builder (L.build_br (fst (List.hd loop))) in
1610         builder
1611 in
1612 let builder =
1613     List.fold_left (fun b s -> build_stmt scope b s []) builder fdecl.
1614     sbody
1615 in
1616 (* Function to add terminals to a basic block *)
1617 add_terminal builder
1618     ( match fdecl.styp with
1619     | A.Void -> L.build_ret_void
1620     | A.Float -> L.build_ret (L.const_float float_t 0.0)
1621     | t -> L.build_ret (L.const_int (ltype_of_ttyp t) 0) )
1622 in
1623 (* Call build_function_body on all functions *)
1624 let functions' =
1625     try List.iter (build_function_body globals) functions
1626     with e -> E.handle_error e
1627 in
1628 functions' ; the_module

```

### 11.2.11 src/team.ml

```

1 (* team.ml: scan & parse & semantically analyze the input,
2  * check the resulting AST and generate SAST from it,
3  * generate LLVM IR, and dump the module
4  * Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
5 type action = Ast | Sast | Resolve | LLVM_IR
6
7 let make_err err = raise (Failure err)
8
9 (* Print the scanner error with the line and character number information *)
10 let scan_error lexbuf ch =
11     let start_p = Lexing.lexeme_start_p lexbuf in
12     let end_p = Lexing.lexeme_end_p lexbuf in
13     let line = string_of_int start_p.Lexing.pos_lnum in
14     let ch_start =
15         string_of_int (start_p.Lexing.pos_cnum - start_p.Lexing.pos_bol)
16     in

```

```

17 let ch_end = string_of_int (end_p.Lexing.pos_cnum - end_p.Lexing.pos_bol)
18   in
19 make_err
20   ( "Illegal character at line " ^ line ^ ", characters " ^ ch_start ^ "-"
21     ^ ch_end ^ ": " ^ "\"" ^ ch ^ "\"" )
22
23 (* Print the parser error with the line and character number information *)
24 let parse_error lexbuf =
25   let start_p = Lexing.lexeme_start_p lexbuf in
26   let end_p = Lexing.lexeme_end_p lexbuf in
27   let line = string_of_int start_p.Lexing.pos_lnum in
28   let ch_start =
29     string_of_int (start_p.Lexing.pos_cnum - start_p.Lexing.pos_bol)
30   in
31   let ch_end = string_of_int (end_p.Lexing.pos_cnum - end_p.Lexing.pos_bol)
32   in
33   let lexeme = Lexing.lexeme lexbuf in
34   make_err
35     ( "Syntax error at line " ^ line ^ ", characters " ^ ch_start ^ "-" ^
36       ch_end
37       ^ ": " ^ lexeme )
38
39 let parse lexbuf =
40   try Parser.program Scanner.token lexbuf with
41   | Scanner.Scan_error ch -> scan_error lexbuf ch
42   | Parsing.Parse_error -> parse_error lexbuf
43
44 let () =
45   let action = ref LLVM_IR in
46   let set_action a () = action := a in
47   let set_channel channel filename =
48     let file =
49       try open_in filename
50       with Sys_error s ->
51         let _ = print_endline s in
52         exit 1
53     in
54     channel := file
55   in
56   let speclist =
57     [ ("-a", Arg.Unit (set_action Ast), "Print the AST")
58       ; ("-s", Arg.Unit (set_action Resolve), "Print the Resolved SAST")
59       ; ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR") ]
60   in
61   let usage_msg = "usage: ./team.native [-a|-s|-l] [file.tm]" in
62   (* get buffers/channel for list and string standard library *)
63   let string_channel = ref stdin in
64   let list_channel = ref stdin in
65   let channel = ref stdin in
66   set_channel string_channel "standard_library/string.tm" ;
67   set_channel list_channel "standard_library/list.tm" ;
68   (* parse program and standard library *)
69   Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg
70   ;
71   let lexbuf = Lexing.from_channel !channel
72   and string_lexbuf = Lexing.from_channel !string_channel
73   and list_lexbuf = Lexing.from_channel !list_channel in
74   let ast = parse lexbuf in
75   let string_ast = parse string_lexbuf in

```

```

72 let list_ast = parse list_lexbuf in
73 (* prepend standard library to program *)
74 let ast =
75   ( fst string_ast @ fst list_ast @ fst ast
76   , snd string_ast @ snd list_ast @ snd ast )
77 in
78 let sast = Semant.check ast in
79 let resolved_sast = Resolve.resolve sast in
80 match !action with
81 | Ast -> print_string (Ast.string_of_program ast)
82 | Sast -> print_string (Sast.string_of_sprogram sast)
83 | Resolve -> print_string (Sast.string_of_sprogram resolved_sast)
84 | LLVM_IR ->
85   let m = Codegen.translate resolved_sast in
86   Llvm_analysis.assert_valid_module m ;
87   print_string (Llvm.string_of_llmodule m)

```

### 11.2.12 scripts/compile.sh

```

1 # Authors: Naoki O., Yingjie L., Lulu Z., Saurav G.
2 #!/bin/bash
3 export PATH=$PATH:/usr/local/opt/llvm/bin
4 # needed for compilation
5 CC="gcc"
6 LIBS="-g -Wall -lpcreposix -lpcre2-8"
7 # file is command line argument, filename is file without extension
8 file=$1
9 filename=$(echo "$file" | cut -f 1 -d ".")
10
11 func=$2
12
13 create() {
14   # generate code, need to pass in file since it could be modified
15   ./team.native -l "$1" > "$filename".ll
16   llc "$2".ll
17   eval "$CC $LIBS -o $filename.exe $filename.s regex.o fileio.o"
18   # echo "$filename.exe created"
19 }
20
21 # SCRIPT BEGINS HERE
22 if [[ $# -eq 0 ]]
23 then
24   echo "usage: ./compile.sh <file.tm> [func]"
25   exit 0
26 fi
27
28 if [ ! -e $file ]
29 then
30   echo "file $file not found"
31   exit 0
32 fi
33
34 if [[ "$func" == "clean" ]]
35 then
36   # echo "cleaning: $filename.s $filename.ll $filename.exe"
37   rm "$filename".s "$filename".ll "$filename".exe
38   rm -rf "$filename".exe.dSYM
39   exit 0
40 fi

```

```

41
42 # echo "${reset}compiling: $file"
43
44 create "$file" "$filename"
45
46 if [[ "$func" == "run" ]]
47 then
48     # echo "running: $filename.exe"
49     ./"$filename".exe
50 fi

```

### 11.2.13 c\_library/fileio.c

```

1 // Author: Naoki O.
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5 #include <assert.h>
6
7 FILE *open(char *filename, char *mode)
8 {
9     FILE *fp;
10    fp = fopen(filename, mode);
11    return fp;
12 }
13
14 int close(FILE *fp)
15 {
16    assert(fp != NULL);
17    return fclose(fp);
18 }
19
20 char *readline(FILE *fp)
21 {
22    assert(fp != NULL);
23    char *line_buf = NULL;
24    size_t line_buf_size = 0;
25    getline(&line_buf, &line_buf_size, fp);
26
27    return line_buf;
28 }
29
30 void write(FILE *fp, char *text)
31 {
32    assert(fp != NULL);
33    fputs(text, fp);
34 }
35
36 #ifdef BUILD_TEST
37 int main()
38 {
39    return 0;
40 }
41 #endif

```

### 11.2.14 c\_library/regex.c

```

1 // Author: Lulu Z.
2 #include <regex.h>

```

```

3 #include <stdio.h>
4 #include <assert.h>
5 #include <string.h>
6 #include <stdlib.h>
7 #define PCRE2_CODE_UNIT_WIDTH 8
8 #include <pcre2.h>
9
10 /* struct representation of a list node */
11 typedef struct list_item_
12 {
13     void *dat;
14     struct list_item_ *next;
15 } list_item;
16
17 list_item **append(char *s, list_item **head_ref)
18 {
19     list_item *node = malloc(sizeof(list_item));
20     char **s_ptr = malloc(sizeof(char *));
21     *s_ptr = s;
22     node->dat = s_ptr;
23     node->next = NULL;
24     list_item *last = *head_ref;
25
26     if (*head_ref == NULL)
27     {
28         *head_ref = node;
29         return head_ref;
30     }
31     while (last->next != NULL)
32     {
33         last = last->next;
34     }
35     last->next = node;
36
37     return head_ref;
38 }
39
40 /* prints the content in the list, for debugging purposes */
41 // void print_list(list_item **head_ref)
42 // {
43 //     list_item *temp = *head_ref;
44
45 //     if ((*head_ref)->dat == NULL && (*head_ref)->next == NULL)
46 //     {
47 //         return;
48 //     }
49
50 //     while (temp != NULL)
51 //     {
52 //         printf("%s\n", (*(char **)temp->dat));
53 //         if (temp->next == NULL)
54 //         {
55 //             break;
56 //         }
57
58 //         temp = temp->next;
59 //     }
60 // }
61

```



```

62 int length(list_item **head_ref)
63 {
64     list_item *temp = *head_ref;
65     int count = 0;
66     if ((*head_ref)->dat == NULL && (*head_ref)->next == NULL)
67     {
68         return count;
69     }
70
71     while (temp != NULL)
72     {
73         count += 1;
74         if (temp->next == NULL)
75         {
76             break;
77         }
78
79         temp = temp->next;
80     }
81     return count;
82 }
83 /*
84  * Match string against the extended regular expression in
85  * pattern, treating errors as no match.
86  *
87  * Return 1 for match, 0 for no match.
88  */
89
90 int match(const char *target, char *pattern)
91 {
92     int status;
93     regex_t re;
94
95     if (regcomp(&re, pattern, REG_EXTENDED | REG_NOSUB) != 0)
96     {
97         return (0); /* Report error. */
98     }
99     /* re = precompiled pattern
100      * string = pattern to be match in regex
101      * NULL = information regarding location of the matches
102      * 0 = to specify the change in matching behaviour */
103
104     status = regexec(&re, target, (size_t)0, NULL, 0);
105     regfree(&re);
106     if (status != 0)
107     {
108         return (0); /* Report error. */
109     }
110     return (1);
111 }
112
113 void substr(char *str, char *sub, int start, int len)
114 {
115     memcpy(sub, &str[start], len);
116     sub[len] = '\0';
117 }
118
119 /* Takes a target string and a regex and returns the
120  * first substring of the string that matches

```

```

121  * the regular expression.
122  *
123  * If no match is found, return the empty string.
124  */
125
126 char *find(char *target, char *regex)
127 {
128     pcre2_code *re;
129     PCRE2_SPTR subject = (const unsigned char *)target;
130     PCRE2_SPTR pattern = (const unsigned char *)regex;
131     int errornumber;
132     int rc;
133
134     PCRE2_SIZE erroroffset;
135     PCRE2_SIZE *ovector;
136     PCRE2_SIZE subject_length;
137
138     pcre2_match_data *match_data;
139
140     subject_length = (PCRE2_SIZE)strlen((char *)subject);
141
142     /* compile the regular expression pattern, and handle
143        any errors that are detected. */
144
145     re = pcre2_compile(
146         pattern,                /* the pattern */
147         PCRE2_ZERO_TERMINATED, /* indicates pattern is zero-terminated */
148         0,                      /* default options */
149         &errornumber,           /* for error number */
150         &erroroffset,           /* for error offset */
151         NULL);                  /* use default compile context */
152
153     /* Compilation failed: print the error message and exit. */
154
155     if (re == NULL)
156     {
157         PCRE2_UCHAR buffer[256];
158         pcre2_get_error_message(errornumber, buffer, sizeof(buffer));
159         printf("PCRE2 compilation failed at offset %d: %s\n", (int)
160             erroroffset,
161             buffer);
162         return "";
163     }
164
165     match_data = pcre2_match_data_create_from_pattern(re, NULL);
166
167     /* Now run the match. */
168
169     rc = pcre2_match(
170         re,                      /* the compiled pattern */
171         subject,                 /* the subject string */
172         subject_length,          /* the length of the subject */
173         0,                      /* start at offset 0 in the subject */
174         0,                      /* default options */
175         match_data,              /* block for storing the result */
176         NULL);                  /* use default match context */
177
178     /* Matching failed: handle error cases */

```

```

179     if (rc < 0)
180     {
181         switch (rc)
182         {
183             case PCRE2_ERROR_NOMATCH:
184                 /* No Match */
185                 break;
186                 /* Handle other special cases if you like */
187             default:
188                 /* Some other match error */
189                 break;
190         }
191         pcre2_match_data_free(match_data); /* Release memory used for the
match */
192         pcre2_code_free(re);                /* data and the compiled
pattern. */
193         return "";
194     }
195
196     /* Match succeeded. Get a pointer to the output vector,
197     * where string offsets are stored. */
198
199     ovector = pcre2_get_ovevector_pointer(match_data);
200     // printf("Match succeeded at offset %d\n", (int)ovevector[0]);
201
202     /* The output vector wasn't big enough.
203     * This should not happen, because we used
204     * pcre2_match_data_create_from_pattern() above. */
205
206     if (rc == 0)
207         printf("ovevector was not big enough for all the captured substrings\n
");
208
209     /* Show substrings stored in the output vector by number. */
210
211     PCRE2_SPTR substring_start = subject + ovector[0];
212     PCRE2_SIZE substring_length = ovector[1] - ovector[0];
213     char *sub = (char *)malloc(sizeof(char) * substring_length);
214     substr((char *)substring_start, sub, 0, (int)substring_length);
215     return sub;
216 }
217
218 list_item **find_all(char *target, char *regex)
219 {
220     pcre2_code *re;
221     PCRE2_SPTR name_table;
222     PCRE2_SPTR subject = (const unsigned char *)target;
223     PCRE2_SPTR pattern = (const unsigned char *)regex;
224     int crlf_is_newline;
225     int errornumber;
226     int i;
227     int rc;
228     int utf8;
229
230     uint32_t option_bits;
231     uint32_t namecount;
232     uint32_t name_entry_size;
233     uint32_t newline;
234

```

```

235 PCRE2_SIZE erroroffset;
236 PCRE2_SIZE *ovector;
237 PCRE2_SIZE subject_length;
238
239 pcre2_match_data *match_data;
240
241 subject_length = (PCRE2_SIZE)strlen((char *)subject);
242 list_item **ret = malloc(sizeof(list_item *));
243 *ret = NULL;
244 /* compile the regular expression pattern, and handle
245    any errors that are detected. */
246
247 re = pcre2_compile(
248     pattern,                /* the pattern */
249     PCRE2_ZERO_TERMINATED, /* indicates pattern is zero-terminated */
250     0,                      /* default options */
251     &errornumber,           /* for error number */
252     &erroroffset,          /* for error offset */
253     NULL);                  /* use default compile context */
254
255 /* Compilation failed: print the error message and exit. */
256
257 if (re == NULL)
258 {
259     PCRE2_UCHAR buffer[256];
260     pcre2_get_error_message(errornumber, buffer, sizeof(buffer));
261     printf("PCRE2 compilation failed at offset %d: %s\n", (int)
erroroffset,
262         buffer);
263     return ret;
264 }
265
266 match_data = pcre2_match_data_create_from_pattern(re, NULL);
267
268 /* Now run the match. */
269
270 rc = pcre2_match(
271     re,                /* the compiled pattern */
272     subject,           /* the subject string */
273     subject_length,    /* the length of the subject */
274     0,                /* start at offset 0 in the subject */
275     0,                /* default options */
276     match_data,        /* block for storing the result */
277     NULL);             /* use default match context */
278
279 /* Matching failed: handle error cases */
280
281 if (rc < 0)
282 {
283     list_item *list = malloc(sizeof(list_item));
284     switch (rc)
285     {
286     case PCRE2_ERROR_NOMATCH:
287         /* no match found, return empty list */
288         list->dat = NULL;
289         list->next = NULL;
290         *ret = list;
291         break;
292     /*

```

```

293     Handle other special cases if you like
294     */
295     default:
296         printf("Matching error %d\n", rc);
297         break;
298     }
299     pcre2_match_data_free(match_data); /* Release memory used for the
match */
300     pcre2_code_free(re);                /* data and the compiled
pattern. */
301     return ret;
302 }
303
304 /* Match succeeded. Get a pointer to the output vector, where string
offsets are
305 stored. */
306
307 ovector = pcre2_get_ovevector_pointer(match_data);
308 // printf("Match succeeded at offset %d\n", (int)ovevector[0]);
309
310 /* The output vector wasn't big enough. This should not happen, because
we used
311 pcre2_match_data_create_from_pattern() above. */
312
313 if (rc == 0)
314     printf("ovevector was not big enough for all the captured substrings\n
");
315
316 /* Show substrings stored in the output vector by number. Obviously, in
a real
317 application you might want to do things other than print them. */
318
319 PCRE2_SPTR substring_start = subject + ovector[0];
320 PCRE2_SIZE substring_length = ovector[1] - ovector[0];
321 char *sub = (char *)malloc(sizeof(char) * (substring_length + 1));
322 substr((char *)substring_start, sub, 0, (int)substring_length);
323 ret = append(sub, ret);
324 /* See if there are any named substrings, and if so, show them by name.
First
325 we have to extract the count of named parentheses from the pattern. */
326
327 (void)pcre2_pattern_info(
328     re, /* the compiled pattern */
329     PCRE2_INFO_NAMECOUNT, /* get the number of named substrings */
330     &namecount); /* where to put the answer */
331
332 if (namecount == 0)
333 {
334 }
335 else
336 {
337     PCRE2_SPTR tabptr;
338
339     /* Before we can access the substrings, we must extract the table
for
340 translating names to numbers, and the size of each entry in the table. */
341
342     (void)pcre2_pattern_info(
343         re, /* the compiled pattern */

```

```

344     PCRE2_INFO_NAMETABLE, /* address of the table */
345     &name_table);        /* where to put the answer */
346
347     (void)pcre2_pattern_info(
348         re,                /* the compiled pattern */
349         PCRE2_INFO_NAMEENTRYSIZE, /* size of each entry in the table */
350         &name_entry_size); /* where to put the answer */
351
352     /* Now we can scan the table and, for each entry, print the number,
353        the name,
354        and the substring itself. In the 8-bit library the number is held in two
355        bytes, most significant first. */
356
357     tabptr = name_table;
358     for (i = 0; i < namecount; i++)
359     {
360         int n = (tabptr[0] << 8) | tabptr[1];
361         printf("(%d) %s: %.*s\n", n, name_entry_size - 3, tabptr + 2,
362             (int)(ovector[2 * n + 1] - ovector[2 * n]), subject +
363             ovector[2 * n]);
364         tabptr += name_entry_size;
365     }
366
367     /* Before running the loop, check for UTF-8 and whether CRLF is a valid
368        sequence. First, find the options with which the regex was compiled and
369        extract
370        the UTF state. */
371
372     (void)pcre2_pattern_info(re, PCRE2_INFO_ALLOPTIONS, &option_bits);
373     utf8 = (option_bits & PCRE2_UTF) != 0;
374
375     /* Now find the newline convention and see whether CRLF is a valid
376        sequence. */
377
378     (void)pcre2_pattern_info(re, PCRE2_INFO_NEWLINE, &newline);
379     crlf_is_newline = newline == PCRE2_NEWLINE_ANY ||
380         newline == PCRE2_NEWLINE_CRLF ||
381         newline == PCRE2_NEWLINE_ANYCRLF;
382
383     /* Loop for second and subsequent matches */
384
385     for (;;)
386     {
387         uint32_t options = 0;                /* Normally no options */
388         PCRE2_SIZE start_offset = ovector[1]; /* Start at end of previous
389        match */
389
390         /* If the previous match was for an empty string, we are finished if
391            we are
392            at the end of the subject. Otherwise, arrange to run another match at the
393            same point to see if a non-empty match can be found. */
394
395         if (ovector[0] == ovector[1])
396         {
397             if (ovector[0] == subject_length)
398                 break;

```

```

396         options = PCRE2_NOTEMPTY_ATSTART | PCRE2_ANCHORED;
397     }
398     else
399     {
400         PCRE2_SIZE startchar = pcre2_get_startchar(match_data);
401         if (start_offset <= startchar)
402         {
403             if (startchar >= subject_length)
404                 break; /* Reached end of subject.
405             */
406             start_offset = startchar + 1; /* Advance by one character.
407             */
408             if (utf8) /* If UTF-8, it may be more
409             */
410             { /* than one code unit.
411             */
412                 for (; start_offset < subject_length; start_offset++)
413                     if ((subject[start_offset] & 0xc0) != 0x80)
414                         break;
415             }
416         }
417     }
418
419     /* Run the next matching operation */
420
421     rc = pcre2_match(
422         re, /* the compiled pattern */
423         subject, /* the subject string */
424         subject_length, /* the length of the subject */
425         start_offset, /* starting offset in the subject */
426         options, /* options */
427         match_data, /* block for storing the result */
428         NULL); /* use default match context */
429
430     if (rc == PCRE2_ERROR_NOMATCH)
431     {
432         if (options == 0)
433         {
434             break;
435         }
436         /* All matches found */
437         ovector[1] = start_offset + 1; /* Advance one code
438         unit */
439         if (crlf_is_newline && /* If CRLF is a newline
440         & */
441             start_offset < subject_length - 1 && /* we are at CRLF, */
442             subject[start_offset] == '\r' &&
443             subject[start_offset + 1] == '\n')
444             ovector[1] += 1; /* Advance by one more.
445         */
446         else if (utf8) /* Otherwise, ensure we
447         */
448         { /* advance a whole UTF-8
449         */
450             while (ovector[1] < subject_length) /* character. */
451             {
452                 if ((subject[ovector[1]] & 0xc0) != 0x80)
453                     break;
454                 ovector[1] += 1;

```

```

446         }
447     }
448     continue; /* Go round the loop again */
449 }
450
451 /* Other matching errors are not recoverable. */
452
453 if (rc < 0)
454 {
455     printf("Matching error %d\n", rc);
456     pcre2_match_data_free(match_data);
457     pcre2_code_free(re);
458     return ret;
459 }
460
461 /* Match succeeded */
462
463 // printf("\nMatch succeeded again at offset %d\n", (int)ovector[0])
464 ;
465
466 /* The match succeeded, but the output vector wasn't big enough.
467 This
468 should not happen. */
469
470 if (rc == 0)
471     printf("ovector was not big enough for all the captured
472 substrings\n");
473
474 /* We must guard against patterns such as /(?=\K)/ that use \K in
475 an
476 assertion to set the start of a match later than its end. In this
477 demonstration program, we just detect this case and give up. */
478
479 if (ovector[0] > ovector[1])
480 {
481     printf("\K was used in an assertion to set the match start
482 after its end.\n"
483           "From end to start the match was: %.*s\n",
484           (int)(ovector[0] - ovector[1]),
485           (char*)(subject + ovector[1]));
486     printf("Run abandoned\n");
487     pcre2_match_data_free(match_data);
488     pcre2_code_free(re);
489     return ret;
490 }
491
492 /* As before, show substrings stored in the output vector by number,
493 and then
494 also any named substrings. */
495
496 PCRE2_SPTR substring_start = subject + ovector[0];
497 size_t substring_length = ovector[1] - ovector[0];
498 char *sub = (char *)malloc(sizeof(char) * (substring_length + 1));
499 substr((char *)substring_start, sub, 0, (int)substring_length);
500 ret = append(sub, ret);
501
502 if (namecount == 0)
503 {
504

```



```

499     else
500     {
501         PCRE2_SPTR tabptr = name_table;
502         for (i = 0; i < namecount; i++)
503         {
504             int n = (tabptr[0] << 8) | tabptr[1];
505             printf("(%d) %*s: %.*s\n", n, name_entry_size - 3, tabptr +
2,
506                 (int)(ovector[2 * n + 1] - ovector[2 * n]), subject +
ovector[2 * n]);
507             tabptr += name_entry_size;
508         }
509     }
510 } /* End of loop to find second and subsequent matches */
511
512 pcre2_match_data_free(match_data);
513 pcre2_code_free(re);
514 return ret;
515 }
516
517 char *str_replace(char *orig, char *rep, char *with)
518 {
519     char *result; // the return string
520     char *ins;    // the next insert point
521     char *tmp;    // varies
522     int len_rep;  // length of rep (the string to remove)
523     int len_with; // length of with (the string to replace rep with)
524     int len_front; // distance between rep and end of last rep
525     int count;    // number of replacements
526
527     // sanity checks and initialization
528     if (!orig || !rep)
529         return NULL;
530     len_rep = strlen(rep);
531     if (len_rep == 0)
532         return NULL; // empty rep causes infinite loop during count
533     if (!with)
534         with = "";
535     len_with = strlen(with);
536
537     // count the number of replacements needed
538     ins = orig;
539     for (count = 0; (tmp = strstr(ins, rep)); ++count)
540     {
541         ins = tmp + len_rep;
542     }
543
544     tmp = result = malloc(strlen(orig) + (len_with - len_rep) * count + 1);
545
546     if (!result)
547         return orig;
548
549     // only replace the first one
550     ins = strstr(orig, rep);
551     len_front = ins - orig;
552     tmp = strncpy(tmp, orig, len_front) + len_front;
553     tmp = strcpy(tmp, with) + len_with;
554     orig += len_front + len_rep; // move to next "end of rep"
555

```

```

556     strcpy(tmp, orig);
557     return result;
558 }
559 // You must free the result if result is non-NULL.
560 char *str_replace_all(char *orig, char *rep, char *with)
561 {
562     char *result; // the return string
563     char *ins;    // the next insert point
564     char *tmp;    // varies
565     int len_rep;  // length of rep (the string to remove)
566     int len_with; // length of with (the string to replace rep with)
567     int len_front; // distance between rep and end of last rep
568     int count;    // number of replacements
569
570     // sanity checks and initialization
571     if (!orig || !rep)
572         return NULL;
573     len_rep = strlen(rep);
574     if (len_rep == 0)
575         return NULL; // empty rep causes infinite loop during count
576     if (!with)
577         with = "";
578     len_with = strlen(with);
579
580     // count the number of replacements needed
581     ins = orig;
582     for (count = 0; (tmp = strstr(ins, rep)); ++count)
583     {
584         ins = tmp + len_rep;
585     }
586
587     tmp = result = malloc(strlen(orig) + (len_with - len_rep) * count + 1);
588
589     if (!result)
590         return orig;
591
592     // first time through the loop, all the variable are set correctly
593     // from here on,
594     //     tmp points to the end of the result string
595     //     ins points to the next occurrence of rep in orig
596     //     orig points to the remainder of orig after "end of rep"
597     while (count--)
598     {
599         ins = strstr(orig, rep);
600         len_front = ins - orig;
601         tmp = strncpy(tmp, orig, len_front) + len_front;
602         tmp = strcpy(tmp, with) + len_with;
603         orig += len_front + len_rep; // move to next "end of rep"
604     }
605     strcpy(tmp, orig);
606     return result;
607 }
608
609 char *replace(char *target, char *regex, char *replc, int count)
610 {
611     char *result = malloc(sizeof(char) * strlen(target));
612     strcpy(result, target);
613     while (count--)
614     {

```

```

615     char *sub = find(result, regex);
616     if (strcmp(sub, "") == 0)
617     {
618         break;
619     }
620     result = str_replace(result, sub, replc);
621 }
622 return result;
623 }
624
625 char *replace_all(char *target, char *regex, char *replc)
626 {
627     char *result = malloc(sizeof(char) * strlen(target));
628     strcpy(result, target);
629     for (;;)
630     {
631         char *sub = find(result, regex);
632         if (strcmp(sub, "") == 0)
633         {
634             break;
635         }
636         result = str_replace(result, sub, replc);
637     }
638     return result;
639 }
640
641 #ifdef BUILD_TEST
642 void test_replace_all()
643 {
644     char *s = replace_all("google ggle goooogle", "go*gle", "replaced");
645     assert(strcmp(s, "replaced replaced replaced") == 0);
646     s = replace_all("This dog is grey and this cat is gray.", "gr(a|e)y", "
brown");
647     assert(strcmp(s, "This dog is brown and this cat is brown.") == 0);
648     s = replace_all("Hello hello hello", "hello", "bye");
649     assert(strcmp(s, "Hello bye bye") == 0);
650 }
651 void test_find_all()
652 {
653     list_item **list = find_all("hello hello hello", "hello");
654     assert(length(list) == 3);
655     list_item **list2 = find_all("google ggle goooogle", "go*gle");
656     assert(length(list2) == 3);
657     list_item **list3 = find_all("gray", "gr(a|e)y");
658     assert(length(list3) == 1);
659     list_item **list4 = find_all("2+2 3*3 4-4 5+5 6*6", "\\d+[\\++-x\\*]\\d+"
);
660     assert(length(list4) == 5);
661     list_item **list5 = find_all("This is a dog", "^dog");
662     assert(length(list5) == 0);
663 }
664 void test_find()
665 {
666     char *s = find("ggle google", "go*gle");
667     assert(strcmp(s, "ggle") == 0);
668     s = find("goooogle", "go*gle");
669     assert(strcmp(s, "goooogle") == 0);
670     s = find("grey", "gr(a|e)y");
671     assert(strcmp(s, "grey") == 0);

```

```

672     s = find("There goes the bat", "[b-chm-pP]at|ot");
673     assert(strcmp(s, "bat") == 0);
674     s = find("a", "\\w");
675     assert(strcmp(s, "a") == 0);
676     s = find("2+2", "\\d+[\\+-x\\*]\\d+");
677     assert(strcmp(s, "2+2") == 0);
678     s = find("x+y", "\\w+[\\+-x\\*]\\w+");
679     assert(strcmp(s, "x+y") == 0);
680     s = find("?+?", "\\W+[\\+-x\\*]\\W+");
681     assert(strcmp(s, "?+?") == 0);
682     // Begins with dog
683     s = find("dog collar", "^dog");
684     assert(strcmp(s, "dog") == 0);
685     // No match
686     s = find("This is a dog", "^dog");
687     assert(strcmp(s, "") == 0);
688     // Ends with dog
689     s = find("hot dog", "dog$");
690     assert(strcmp(s, "dog") == 0);
691     // Just dog
692     s = find("dog", "^dog$");
693     assert(strcmp(s, "dog") == 0);
694     // Time in 24 hr format
695     s = find("12:03", "^[01]?[0-9]|2[0-3]:[0-5][0-9]$");
696     assert(strcmp(s, "12:03") == 0);
697     // No match
698     s = find("z", "z{3,6}");
699     assert(strcmp(s, "") == 0);
700 }
701 void test_match()
702 {
703     int i = match("ggle google", "go*gle");
704     assert(i == 1);
705     i = match("gooogle", "go*gle");
706     assert(i == 1);
707     i = match("grey", "gr(a|e)y");
708     assert(i == 1);
709     i = match("There goes the bat", "[b-chm-pP]at|ot");
710     assert(i == 1);
711     i = match("a", "\\w");
712     assert(i == 1);
713     i = match("2+2", "\\d+[\\+-x\\*]\\d+");
714     assert(i == 1);
715     i = match("x+y", "\\w+[\\+-x\\*]\\w+");
716     assert(i == 1);
717     i = match("?+?", "\\W+[\\+-x\\*]\\W+");
718     assert(i == 1);
719     // Begins with dog
720     i = match("dog collar", "^dog");
721     assert(i == 1);
722     // No match
723     i = match("This is a dog", "^dog");
724     assert(i == 0);
725     // Ends with dog
726     i = match("hot dog", "dog$");
727     assert(i == 1);
728     // Just dog
729     i = match("dog", "^dog$");
730     assert(i == 1);

```

```

731 // Time in 24 hr format
732 i = match("12:03", "^[01]?[0-9]|2[0-3]):[0-5][0-9]$");
733 assert(i == 1);
734 // No match
735 i = match("z", "z{3,6}");
736 assert(i == 0);
737 }
738 void test_replace()
739 {
740     char *s = replace("ggle google", "go*gle", "replaced", 1);
741     assert(strcmp(s, "replaced google") == 0);
742     s = replace("ggle google", "go*gle", "replaced", 5);
743     assert(strcmp(s, "replaced replaced") == 0);
744     s = replace("ggle google", "go*gle", "replaced", 0);
745     assert(strcmp(s, "ggle google") == 0);
746     s = replace("google google google", "goo", "foo", 3);
747     assert(strcmp(s, "foogle foogle foogle") == 0);
748 }
749 int main()
750 {
751     test_replace_all();
752     test_find();
753     test_match();
754     test_replace();
755     test_find_all();
756     return 0;
757 }
758 #endif

```

### 11.2.15 README & Makefile

```

1 # TEAM
2
3 Note: This file is best viewed in a Markdown reader.
4
5 TEAM (Text Extraction And Manipulation) is a domain specific programming
   language designed for text processing, data extraction, and report
   generation. With its straightforward syntax and various built-in
   functions, TEAM offers a clean layer of abstraction for users to perform
   tasks that are often cumbersome to do in general purpose languages.
6
7 ## Compilation
8
9 To compile TEAM, do
10
11     make
12
13 ## Environment Setup
14
15 Team uses PCRE2 (Perl Compatible Regular Expressions) library to support
   regular expressions. To setup dependencies, do:
16
17     cd pcre2-10.36 && ./configure && make && make install && cd ..
18
19 Ocamlbuild sometimes does not like .o files in the directory when trying to
   compile TEAM. If you encounter an issue with Ocamlbuild requiring
   sanitization, please do:
20
21     cd pcre2-10.36 && make clean

```

```

22     cd .. && make clean && make
23
24 For more information on the PCRE2 API, please visit:
25
26 https://www.pcre.org/current/doc/html/pcre2api.html
27
28 ## Testing
29
30 To run all tests, do
31
32     python scripts/runtests.py -m all
33
34 To run tests in a specific directory, do
35
36     python scripts/runtests.py -m <mode>
37
38 | Mode      | Description
39 | ----- |
40 | ast       | run the tests in `ast_tests/` and validate the pretty printed
41 |           | ast against a gold standard. Tests that pass the validation are marked
42 |           | with OK! and those that don't are marked with FAILED!
43 |
44 | sast      | run the tests in `sast_tests/` and validates the pretty printed
45 |           | sast against a gold standard. Tests that pass the validation are marked
46 |           | with OK! and those that don't are marked with FAILED!
47 |
48 | codegen   | run the tests in `codegen_tests/`, compile the resulting LLVM
49 |           | code, execute the resulting file, and validate the output against a gold
50 |           | standard. If the validation fails, a diff of the two files will be
51 |           | printed to standard output. |
52
53 Files used for testing are located in `<mode>_tests/`.
54 The generated outputs are located in `<mode>_log/` and
55 the expected outputs (gold standard) are located in `<mode>_ref/`.
56
57 The default mode is ast if none was provided.
58
59 To execute a single TEAM file (file.tm), do
60
61     python scripts/runtests.py -t file.tm -r file.log -m <mode>
62
63 - -t specifies the file to be executed
64
65 - -r specifies the file that is the gold standard
66
67 - -m when running a single test, the mode can not be `all`.
68
69 ## Extended Testsuite
70
71 To run the extended testsuite, do
72
73     python scripts/runtests.py -m extended
74
75 7 positive tests included in this testsuite are as follows:

```

```

67
68 | Program          | Description
69 | ----- |
    |
70 | arith.tm          | Tests arithmetic operators (add, subtract, multiply,
    | divide) for int and float |
71 | string.tm         | Tests string slicing and indexing
    |
72 | list.tm           | Tests list slicing and indexing
    |
73 | function.tm       | Tests calling a user-defined function in the body of
    | another function |
74 | while.tm          | Tests while loop
    |
75 | scope.tm          | Tests local and global variables hold correct values
    |
76 | formattedPrint.tm | Tests print function with formatted strings
    |
77
78 3 negative tests included in this testsuite are as follows:
79
80 | Test              | Description
81 | ----- |
    |
82 | badDuplicate.tm   | Detects duplicate function definitions
    |
83 | badScope.tm      | Detects variables used out of scope
    |
84 | badReturn.tm     | Detects mismatch between a function's return type
    | specified by its signature and its actual return type in its body |
85
86 ## Group Members
87
88 - Wenlu (Lulu) Zheng: <lulu.zheng@tufts.edu>
89 - Yingjie Ling: <yingjie.ling@tufts.edu>
90 - Saurav Gyawali: <saurav.gyawali@tufts.edu>
91 - Naoki Okada: <naoki.okada@tufts.edu>

1 all: team.native fileio regex
2 team.native : ./src/parser.mly ./src/scanner.mll ./src/codegen.ml ./src/
    semant.ml ./src/resolve.ml ./src/team.ml
3 opam config exec -- \
4 ocamlbuild -use-ocamlfind ./src/team.native
5 # For built-in functions
6 .PHONY: fileio
7 fileio: ./c_library/fileio.c
8 gcc -c -Wall -g ./c_library/fileio.c
9 gcc -g -o fileio -DBUILD_TEST ./c_library/fileio.c
10
11 .PHONY: regex
12 regex : ./c_library/regex.c
13 gcc -c -Wall -g ./c_library/regex.c
14 gcc -g -lpcposix -lpcr2-8 -o regex -DBUILD_TEST ./c_library/regex.c
15
16 .PHONY : clean

```

```

17 clean :
18   ocamlbuild -clean
19   rm *.o
20   rm regex
21   rm fileio
22   rm -r *.dSYM/

```

### 11.2.16 Standard Library LLVM

The following LLVM is generated from the standard library functions shown in 11.2.1 and 11.2.2. Because the standard library functions are prepended to all tm files, the LLVM shown below appears in all target programs.

```

1 ; ModuleID = 'TEAM'
2 source_filename = "TEAM"
3
4 %list_item = type <{ i8*, %list_item* }>
5
6 @ASCII = global %list_item** null
7 @string = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
8 @string.1 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
9 @string.2 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
10 @string.3 = private unnamed_addr constant [6 x i8] c"wolrd\00", align 1
11 @string.4 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
12 @string.5 = private unnamed_addr constant [5 x i8] c"what\00", align 1
13 @string.6 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
14 @string.7 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
15 @string.8 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
16 @string.9 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
17 @string.10 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
18
19 declare i32 @printf(i8*, ...)
20
21 declare double @pow(double, double)
22
23 declare i8* @fopen(i8*, i8*)
24
25 declare i32 @close(i8*)
26
27 declare i8* @readline(i8*)
28
29 declare i8* @write(i8*, i8*)
30
31 declare i1 @match(i8*, i8*)
32
33 declare i8* @find(i8*, i8*)
34
35 declare i8* @replace(i8*, i8*, i8*, i32)
36
37 declare i8* @replace_all(i8*, i8*, i8*)
38
39 declare %list_item** @find_all(i8*, i8*)
40
41 define i32 @main() {
42 entry:
43   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
44     i1** null, i32 1) to i32))
45   %list = bitcast i8* %mallocall to %list_item**

```



```

45 %allocacll1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
46 %list_item = bitcast i8* %allocacll1 to %list_item*
47 store %list_item zeroinitializer, %list_item* %list_item, align 1
48 %copied = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
49 store i8 90, i8* %copied, align 1
50 %data_ptr_container = getelementptr inbounds %list_item, %list_item* %
    list_item, i32 0, i32 0
51 store i8* %copied, i8** %data_ptr_container, align 8
52 %next = getelementptr inbounds %list_item, %list_item* %list_item, i32 0,
    i32 1
53 store %list_item* null, %list_item** %next, align 8
54 %allocacll3 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
55 %list_item4 = bitcast i8* %allocacll3 to %list_item*
56 store %list_item zeroinitializer, %list_item* %list_item4, align 1
57 %copied6 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
58 store i8 89, i8* %copied6, align 1
59 %data_ptr_container7 = getelementptr inbounds %list_item, %list_item* %
    list_item4, i32 0, i32 0
60 store i8* %copied6, i8** %data_ptr_container7, align 8
61 %next8 = getelementptr inbounds %list_item, %list_item* %list_item4, i32
    0, i32 1
62 store %list_item* %list_item, %list_item** %next8, align 8
63 %allocacll9 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
64 %list_item10 = bitcast i8* %allocacll9 to %list_item*
65 store %list_item zeroinitializer, %list_item* %list_item10, align 1
66 %copied12 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
67 store i8 88, i8* %copied12, align 1
68 %data_ptr_container13 = getelementptr inbounds %list_item, %list_item* %
    list_item10, i32 0, i32 0
69 store i8* %copied12, i8** %data_ptr_container13, align 8
70 %next14 = getelementptr inbounds %list_item, %list_item* %list_item10, i32
    0, i32 1
71 store %list_item* %list_item4, %list_item** %next14, align 8
72 %allocacll15 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
73 %list_item16 = bitcast i8* %allocacll15 to %list_item*
74 store %list_item zeroinitializer, %list_item* %list_item16, align 1
75 %copied18 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
76 store i8 87, i8* %copied18, align 1
77 %data_ptr_container19 = getelementptr inbounds %list_item, %list_item* %
    list_item16, i32 0, i32 0
78 store i8* %copied18, i8** %data_ptr_container19, align 8
79 %next20 = getelementptr inbounds %list_item, %list_item* %list_item16, i32
    0, i32 1
80 store %list_item* %list_item10, %list_item** %next20, align 8
81 %allocacll21 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
82 %list_item22 = bitcast i8* %allocacll21 to %list_item*
83 store %list_item zeroinitializer, %list_item* %list_item22, align 1
84 %copied24 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
85 store i8 86, i8* %copied24, align 1

```

```

86 %data_ptr_container25 = getelementptr inbounds %list_item, %list_item* %
    list_item22, i32 0, i32 0
87 store i8* %copied24, i8** %data_ptr_container25, align 8
88 %next26 = getelementptr inbounds %list_item, %list_item* %list_item22, i32
    0, i32 1
89 store %list_item* %list_item16, %list_item** %next26, align 8
90 %allocaall127 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
91 %list_item28 = bitcast i8* %allocaall127 to %list_item*
92 store %list_item zeroinitializer, %list_item* %list_item28, align 1
93 %copied30 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
94 store i8 85, i8* %copied30, align 1
95 %data_ptr_container31 = getelementptr inbounds %list_item, %list_item* %
    list_item28, i32 0, i32 0
96 store i8* %copied30, i8** %data_ptr_container31, align 8
97 %next32 = getelementptr inbounds %list_item, %list_item* %list_item28, i32
    0, i32 1
98 store %list_item* %list_item22, %list_item** %next32, align 8
99 %allocaall133 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
100 %list_item34 = bitcast i8* %allocaall133 to %list_item*
101 store %list_item zeroinitializer, %list_item* %list_item34, align 1
102 %copied36 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
103 store i8 84, i8* %copied36, align 1
104 %data_ptr_container37 = getelementptr inbounds %list_item, %list_item* %
    list_item34, i32 0, i32 0
105 store i8* %copied36, i8** %data_ptr_container37, align 8
106 %next38 = getelementptr inbounds %list_item, %list_item* %list_item34, i32
    0, i32 1
107 store %list_item* %list_item28, %list_item** %next38, align 8
108 %allocaall139 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
109 %list_item40 = bitcast i8* %allocaall139 to %list_item*
110 store %list_item zeroinitializer, %list_item* %list_item40, align 1
111 %copied42 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
112 store i8 83, i8* %copied42, align 1
113 %data_ptr_container43 = getelementptr inbounds %list_item, %list_item* %
    list_item40, i32 0, i32 0
114 store i8* %copied42, i8** %data_ptr_container43, align 8
115 %next44 = getelementptr inbounds %list_item, %list_item* %list_item40, i32
    0, i32 1
116 store %list_item* %list_item34, %list_item** %next44, align 8
117 %allocaall145 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
118 %list_item46 = bitcast i8* %allocaall145 to %list_item*
119 store %list_item zeroinitializer, %list_item* %list_item46, align 1
120 %copied48 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
121 store i8 82, i8* %copied48, align 1
122 %data_ptr_container49 = getelementptr inbounds %list_item, %list_item* %
    list_item46, i32 0, i32 0
123 store i8* %copied48, i8** %data_ptr_container49, align 8
124 %next50 = getelementptr inbounds %list_item, %list_item* %list_item46, i32
    0, i32 1
125 store %list_item* %list_item40, %list_item** %next50, align 8

```

```

126 %mallocall151 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
127 %list_item52 = bitcast i8* %mallocall151 to %list_item*
128 store %list_item zeroinitializer, %list_item* %list_item52, align 1
129 %copied54 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
130 store i8 81, i8* %copied54, align 1
131 %data_ptr_container55 = getelementptr inbounds %list_item, %list_item* %
    list_item52, i32 0, i32 0
132 store i8* %copied54, i8** %data_ptr_container55, align 8
133 %next56 = getelementptr inbounds %list_item, %list_item* %list_item52, i32
    0, i32 1
134 store %list_item* %list_item46, %list_item** %next56, align 8
135 %mallocall157 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
136 %list_item58 = bitcast i8* %mallocall157 to %list_item*
137 store %list_item zeroinitializer, %list_item* %list_item58, align 1
138 %copied60 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
139 store i8 80, i8* %copied60, align 1
140 %data_ptr_container61 = getelementptr inbounds %list_item, %list_item* %
    list_item58, i32 0, i32 0
141 store i8* %copied60, i8** %data_ptr_container61, align 8
142 %next62 = getelementptr inbounds %list_item, %list_item* %list_item58, i32
    0, i32 1
143 store %list_item* %list_item52, %list_item** %next62, align 8
144 %mallocall163 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
145 %list_item64 = bitcast i8* %mallocall163 to %list_item*
146 store %list_item zeroinitializer, %list_item* %list_item64, align 1
147 %copied66 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
148 store i8 79, i8* %copied66, align 1
149 %data_ptr_container67 = getelementptr inbounds %list_item, %list_item* %
    list_item64, i32 0, i32 0
150 store i8* %copied66, i8** %data_ptr_container67, align 8
151 %next68 = getelementptr inbounds %list_item, %list_item* %list_item64, i32
    0, i32 1
152 store %list_item* %list_item58, %list_item** %next68, align 8
153 %mallocall169 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
154 %list_item70 = bitcast i8* %mallocall169 to %list_item*
155 store %list_item zeroinitializer, %list_item* %list_item70, align 1
156 %copied72 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
157 store i8 78, i8* %copied72, align 1
158 %data_ptr_container73 = getelementptr inbounds %list_item, %list_item* %
    list_item70, i32 0, i32 0
159 store i8* %copied72, i8** %data_ptr_container73, align 8
160 %next74 = getelementptr inbounds %list_item, %list_item* %list_item70, i32
    0, i32 1
161 store %list_item* %list_item64, %list_item** %next74, align 8
162 %mallocall175 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
163 %list_item76 = bitcast i8* %mallocall175 to %list_item*
164 store %list_item zeroinitializer, %list_item* %list_item76, align 1
165 %copied78 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
166 store i8 77, i8* %copied78, align 1

```

```

167 %data_ptr_container79 = getelementptr inbounds %list_item, %list_item* %
    list_item76, i32 0, i32 0
168 store i8* %copied78, i8** %data_ptr_container79, align 8
169 %next80 = getelementptr inbounds %list_item, %list_item* %list_item76, i32
    0, i32 1
170 store %list_item* %list_item70, %list_item** %next80, align 8
171 %alloca181 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
172 %list_item82 = bitcast i8* %alloca181 to %list_item*
173 store %list_item zeroinitializer, %list_item* %list_item82, align 1
174 %copied84 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
175 store i8 76, i8* %copied84, align 1
176 %data_ptr_container85 = getelementptr inbounds %list_item, %list_item* %
    list_item82, i32 0, i32 0
177 store i8* %copied84, i8** %data_ptr_container85, align 8
178 %next86 = getelementptr inbounds %list_item, %list_item* %list_item82, i32
    0, i32 1
179 store %list_item* %list_item76, %list_item** %next86, align 8
180 %alloca187 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
181 %list_item88 = bitcast i8* %alloca187 to %list_item*
182 store %list_item zeroinitializer, %list_item* %list_item88, align 1
183 %copied90 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
184 store i8 75, i8* %copied90, align 1
185 %data_ptr_container91 = getelementptr inbounds %list_item, %list_item* %
    list_item88, i32 0, i32 0
186 store i8* %copied90, i8** %data_ptr_container91, align 8
187 %next92 = getelementptr inbounds %list_item, %list_item* %list_item88, i32
    0, i32 1
188 store %list_item* %list_item82, %list_item** %next92, align 8
189 %alloca193 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
190 %list_item94 = bitcast i8* %alloca193 to %list_item*
191 store %list_item zeroinitializer, %list_item* %list_item94, align 1
192 %copied96 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
193 store i8 74, i8* %copied96, align 1
194 %data_ptr_container97 = getelementptr inbounds %list_item, %list_item* %
    list_item94, i32 0, i32 0
195 store i8* %copied96, i8** %data_ptr_container97, align 8
196 %next98 = getelementptr inbounds %list_item, %list_item* %list_item94, i32
    0, i32 1
197 store %list_item* %list_item88, %list_item** %next98, align 8
198 %alloca199 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
199 %list_item100 = bitcast i8* %alloca199 to %list_item*
200 store %list_item zeroinitializer, %list_item* %list_item100, align 1
201 %copied102 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
202 store i8 73, i8* %copied102, align 1
203 %data_ptr_container103 = getelementptr inbounds %list_item, %list_item* %
    list_item100, i32 0, i32 0
204 store i8* %copied102, i8** %data_ptr_container103, align 8
205 %next104 = getelementptr inbounds %list_item, %list_item* %list_item100,
    i32 0, i32 1
206 store %list_item* %list_item94, %list_item** %next104, align 8

```

```

207 %malloccall1105 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
208 %list_item106 = bitcast i8* %malloccall1105 to %list_item*
209 store %list_item zeroinitializer, %list_item* %list_item106, align 1
210 %copied108 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
211 store i8 72, i8* %copied108, align 1
212 %data_ptr_container109 = getelementptr inbounds %list_item, %list_item* %
    list_item106, i32 0, i32 0
213 store i8* %copied108, i8** %data_ptr_container109, align 8
214 %next110 = getelementptr inbounds %list_item, %list_item* %list_item106,
    i32 0, i32 1
215 store %list_item* %list_item100, %list_item** %next110, align 8
216 %malloccall111 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
217 %list_item112 = bitcast i8* %malloccall111 to %list_item*
218 store %list_item zeroinitializer, %list_item* %list_item112, align 1
219 %copied114 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
220 store i8 71, i8* %copied114, align 1
221 %data_ptr_container115 = getelementptr inbounds %list_item, %list_item* %
    list_item112, i32 0, i32 0
222 store i8* %copied114, i8** %data_ptr_container115, align 8
223 %next116 = getelementptr inbounds %list_item, %list_item* %list_item112,
    i32 0, i32 1
224 store %list_item* %list_item106, %list_item** %next116, align 8
225 %malloccall117 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
226 %list_item118 = bitcast i8* %malloccall117 to %list_item*
227 store %list_item zeroinitializer, %list_item* %list_item118, align 1
228 %copied120 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
229 store i8 70, i8* %copied120, align 1
230 %data_ptr_container121 = getelementptr inbounds %list_item, %list_item* %
    list_item118, i32 0, i32 0
231 store i8* %copied120, i8** %data_ptr_container121, align 8
232 %next122 = getelementptr inbounds %list_item, %list_item* %list_item118,
    i32 0, i32 1
233 store %list_item* %list_item112, %list_item** %next122, align 8
234 %malloccall123 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
235 %list_item124 = bitcast i8* %malloccall123 to %list_item*
236 store %list_item zeroinitializer, %list_item* %list_item124, align 1
237 %copied126 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
238 store i8 69, i8* %copied126, align 1
239 %data_ptr_container127 = getelementptr inbounds %list_item, %list_item* %
    list_item124, i32 0, i32 0
240 store i8* %copied126, i8** %data_ptr_container127, align 8
241 %next128 = getelementptr inbounds %list_item, %list_item* %list_item124,
    i32 0, i32 1
242 store %list_item* %list_item118, %list_item** %next128, align 8
243 %malloccall129 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
244 %list_item130 = bitcast i8* %malloccall129 to %list_item*
245 store %list_item zeroinitializer, %list_item* %list_item130, align 1
246 %copied132 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
247 store i8 68, i8* %copied132, align 1

```

```

248 %data_ptr_container133 = getelementptr @inbounds %list_item, %list_item* %
    list_item130, i32 0, i32 0
249 store i8* %copied132, i8** %data_ptr_container133, align 8
250 %next134 = getelementptr @inbounds %list_item, %list_item* %list_item130,
    i32 0, i32 1
251 store %list_item* %list_item124, %list_item** %next134, align 8
252 %alloca1135 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
253 %list_item136 = bitcast i8* %alloca1135 to %list_item*
254 store %list_item zeroinitializer, %list_item* %list_item136, align 1
255 %copied138 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
256 store i8 67, i8* %copied138, align 1
257 %data_ptr_container139 = getelementptr @inbounds %list_item, %list_item* %
    list_item136, i32 0, i32 0
258 store i8* %copied138, i8** %data_ptr_container139, align 8
259 %next140 = getelementptr @inbounds %list_item, %list_item* %list_item136,
    i32 0, i32 1
260 store %list_item* %list_item130, %list_item** %next140, align 8
261 %alloca1141 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
262 %list_item142 = bitcast i8* %alloca1141 to %list_item*
263 store %list_item zeroinitializer, %list_item* %list_item142, align 1
264 %copied144 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
265 store i8 66, i8* %copied144, align 1
266 %data_ptr_container145 = getelementptr @inbounds %list_item, %list_item* %
    list_item142, i32 0, i32 0
267 store i8* %copied144, i8** %data_ptr_container145, align 8
268 %next146 = getelementptr @inbounds %list_item, %list_item* %list_item142,
    i32 0, i32 1
269 store %list_item* %list_item136, %list_item** %next146, align 8
270 %alloca1147 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
271 %list_item148 = bitcast i8* %alloca1147 to %list_item*
272 store %list_item zeroinitializer, %list_item* %list_item148, align 1
273 %copied150 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
274 store i8 65, i8* %copied150, align 1
275 %data_ptr_container151 = getelementptr @inbounds %list_item, %list_item* %
    list_item148, i32 0, i32 0
276 store i8* %copied150, i8** %data_ptr_container151, align 8
277 %next152 = getelementptr @inbounds %list_item, %list_item* %list_item148,
    i32 0, i32 1
278 store %list_item* %list_item142, %list_item** %next152, align 8
279 %alloca1153 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
280 %list_item154 = bitcast i8* %alloca1153 to %list_item*
281 store %list_item zeroinitializer, %list_item* %list_item154, align 1
282 %copied156 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
283 store i8 122, i8* %copied156, align 1
284 %data_ptr_container157 = getelementptr @inbounds %list_item, %list_item* %
    list_item154, i32 0, i32 0
285 store i8* %copied156, i8** %data_ptr_container157, align 8
286 %next158 = getelementptr @inbounds %list_item, %list_item* %list_item154,
    i32 0, i32 1
287 store %list_item* %list_item148, %list_item** %next158, align 8

```

```

288 %malloccall1159 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
289 %list_item160 = bitcast i8* %malloccall1159 to %list_item*
290 store %list_item zeroinitializer, %list_item* %list_item160, align 1
291 %copied162 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
292 store i8 121, i8* %copied162, align 1
293 %data_ptr_container163 = getelementptr inbounds %list_item, %list_item* %
    list_item160, i32 0, i32 0
294 store i8* %copied162, i8** %data_ptr_container163, align 8
295 %next164 = getelementptr inbounds %list_item, %list_item* %list_item160,
    i32 0, i32 1
296 store %list_item* %list_item154, %list_item** %next164, align 8
297 %malloccall1165 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
298 %list_item166 = bitcast i8* %malloccall1165 to %list_item*
299 store %list_item zeroinitializer, %list_item* %list_item166, align 1
300 %copied168 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
301 store i8 120, i8* %copied168, align 1
302 %data_ptr_container169 = getelementptr inbounds %list_item, %list_item* %
    list_item166, i32 0, i32 0
303 store i8* %copied168, i8** %data_ptr_container169, align 8
304 %next170 = getelementptr inbounds %list_item, %list_item* %list_item166,
    i32 0, i32 1
305 store %list_item* %list_item160, %list_item** %next170, align 8
306 %malloccall1171 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
307 %list_item172 = bitcast i8* %malloccall1171 to %list_item*
308 store %list_item zeroinitializer, %list_item* %list_item172, align 1
309 %copied174 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
310 store i8 119, i8* %copied174, align 1
311 %data_ptr_container175 = getelementptr inbounds %list_item, %list_item* %
    list_item172, i32 0, i32 0
312 store i8* %copied174, i8** %data_ptr_container175, align 8
313 %next176 = getelementptr inbounds %list_item, %list_item* %list_item172,
    i32 0, i32 1
314 store %list_item* %list_item166, %list_item** %next176, align 8
315 %malloccall1177 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
316 %list_item178 = bitcast i8* %malloccall1177 to %list_item*
317 store %list_item zeroinitializer, %list_item* %list_item178, align 1
318 %copied180 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
319 store i8 118, i8* %copied180, align 1
320 %data_ptr_container181 = getelementptr inbounds %list_item, %list_item* %
    list_item178, i32 0, i32 0
321 store i8* %copied180, i8** %data_ptr_container181, align 8
322 %next182 = getelementptr inbounds %list_item, %list_item* %list_item178,
    i32 0, i32 1
323 store %list_item* %list_item172, %list_item** %next182, align 8
324 %malloccall1183 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
325 %list_item184 = bitcast i8* %malloccall1183 to %list_item*
326 store %list_item zeroinitializer, %list_item* %list_item184, align 1
327 %copied186 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
328 store i8 117, i8* %copied186, align 1

```



```

329 %data_ptr_container187 = getelementptr @inbounds %list_item, %list_item* %
    list_item184, i32 0, i32 0
330 store i8* %copied186, i8** %data_ptr_container187, align 8
331 %next188 = getelementptr @inbounds %list_item, %list_item* %list_item184,
    i32 0, i32 1
332 store %list_item* %list_item178, %list_item** %next188, align 8
333 %alloca1189 = tail call i8* @malloc(i32 @ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
334 %list_item190 = bitcast i8* %alloca1189 to %list_item*
335 store %list_item zeroinitializer, %list_item* %list_item190, align 1
336 %copied192 = tail call i8* @malloc(i32 @ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
337 store i8 116, i8* %copied192, align 1
338 %data_ptr_container193 = getelementptr @inbounds %list_item, %list_item* %
    list_item190, i32 0, i32 0
339 store i8* %copied192, i8** %data_ptr_container193, align 8
340 %next194 = getelementptr @inbounds %list_item, %list_item* %list_item190,
    i32 0, i32 1
341 store %list_item* %list_item184, %list_item** %next194, align 8
342 %alloca1195 = tail call i8* @malloc(i32 @ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
343 %list_item196 = bitcast i8* %alloca1195 to %list_item*
344 store %list_item zeroinitializer, %list_item* %list_item196, align 1
345 %copied198 = tail call i8* @malloc(i32 @ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
346 store i8 115, i8* %copied198, align 1
347 %data_ptr_container199 = getelementptr @inbounds %list_item, %list_item* %
    list_item196, i32 0, i32 0
348 store i8* %copied198, i8** %data_ptr_container199, align 8
349 %next200 = getelementptr @inbounds %list_item, %list_item* %list_item196,
    i32 0, i32 1
350 store %list_item* %list_item190, %list_item** %next200, align 8
351 %alloca1201 = tail call i8* @malloc(i32 @ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
352 %list_item202 = bitcast i8* %alloca1201 to %list_item*
353 store %list_item zeroinitializer, %list_item* %list_item202, align 1
354 %copied204 = tail call i8* @malloc(i32 @ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
355 store i8 114, i8* %copied204, align 1
356 %data_ptr_container205 = getelementptr @inbounds %list_item, %list_item* %
    list_item202, i32 0, i32 0
357 store i8* %copied204, i8** %data_ptr_container205, align 8
358 %next206 = getelementptr @inbounds %list_item, %list_item* %list_item202,
    i32 0, i32 1
359 store %list_item* %list_item196, %list_item** %next206, align 8
360 %alloca1207 = tail call i8* @malloc(i32 @ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
361 %list_item208 = bitcast i8* %alloca1207 to %list_item*
362 store %list_item zeroinitializer, %list_item* %list_item208, align 1
363 %copied210 = tail call i8* @malloc(i32 @ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
364 store i8 113, i8* %copied210, align 1
365 %data_ptr_container211 = getelementptr @inbounds %list_item, %list_item* %
    list_item208, i32 0, i32 0
366 store i8* %copied210, i8** %data_ptr_container211, align 8
367 %next212 = getelementptr @inbounds %list_item, %list_item* %list_item208,
    i32 0, i32 1
368 store %list_item* %list_item202, %list_item** %next212, align 8

```



```

369 %malloccall1213 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
370 %list_item214 = bitcast i8* %malloccall1213 to %list_item*
371 store %list_item zeroinitializer, %list_item* %list_item214, align 1
372 %copied216 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
373 store i8 112, i8* %copied216, align 1
374 %data_ptr_container217 = getelementptr inbounds %list_item, %list_item* %
    list_item214, i32 0, i32 0
375 store i8* %copied216, i8** %data_ptr_container217, align 8
376 %next218 = getelementptr inbounds %list_item, %list_item* %list_item214,
    i32 0, i32 1
377 store %list_item* %list_item208, %list_item** %next218, align 8
378 %malloccall1219 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
379 %list_item220 = bitcast i8* %malloccall1219 to %list_item*
380 store %list_item zeroinitializer, %list_item* %list_item220, align 1
381 %copied222 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
382 store i8 111, i8* %copied222, align 1
383 %data_ptr_container223 = getelementptr inbounds %list_item, %list_item* %
    list_item220, i32 0, i32 0
384 store i8* %copied222, i8** %data_ptr_container223, align 8
385 %next224 = getelementptr inbounds %list_item, %list_item* %list_item220,
    i32 0, i32 1
386 store %list_item* %list_item214, %list_item** %next224, align 8
387 %malloccall1225 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
388 %list_item226 = bitcast i8* %malloccall1225 to %list_item*
389 store %list_item zeroinitializer, %list_item* %list_item226, align 1
390 %copied228 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
391 store i8 110, i8* %copied228, align 1
392 %data_ptr_container229 = getelementptr inbounds %list_item, %list_item* %
    list_item226, i32 0, i32 0
393 store i8* %copied228, i8** %data_ptr_container229, align 8
394 %next230 = getelementptr inbounds %list_item, %list_item* %list_item226,
    i32 0, i32 1
395 store %list_item* %list_item220, %list_item** %next230, align 8
396 %malloccall1231 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
397 %list_item232 = bitcast i8* %malloccall1231 to %list_item*
398 store %list_item zeroinitializer, %list_item* %list_item232, align 1
399 %copied234 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
400 store i8 109, i8* %copied234, align 1
401 %data_ptr_container235 = getelementptr inbounds %list_item, %list_item* %
    list_item232, i32 0, i32 0
402 store i8* %copied234, i8** %data_ptr_container235, align 8
403 %next236 = getelementptr inbounds %list_item, %list_item* %list_item232,
    i32 0, i32 1
404 store %list_item* %list_item226, %list_item** %next236, align 8
405 %malloccall1237 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
406 %list_item238 = bitcast i8* %malloccall1237 to %list_item*
407 store %list_item zeroinitializer, %list_item* %list_item238, align 1
408 %copied240 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
409 store i8 108, i8* %copied240, align 1

```

```

410 %data_ptr_container241 = getelementptr inbounds %list_item*, %list_item* %
    list_item238, i32 0, i32 0
411 store i8* %copied240, i8** %data_ptr_container241, align 8
412 %next242 = getelementptr inbounds %list_item*, %list_item* %list_item238,
    i32 0, i32 1
413 store %list_item* %list_item232, %list_item** %next242, align 8
414 %allocaall243 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
415 %list_item244 = bitcast i8* %allocaall243 to %list_item*
416 store %list_item zeroinitializer, %list_item* %list_item244, align 1
417 %copied246 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
418 store i8 107, i8* %copied246, align 1
419 %data_ptr_container247 = getelementptr inbounds %list_item*, %list_item* %
    list_item244, i32 0, i32 0
420 store i8* %copied246, i8** %data_ptr_container247, align 8
421 %next248 = getelementptr inbounds %list_item*, %list_item* %list_item244,
    i32 0, i32 1
422 store %list_item* %list_item238, %list_item** %next248, align 8
423 %allocaall249 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
424 %list_item250 = bitcast i8* %allocaall249 to %list_item*
425 store %list_item zeroinitializer, %list_item* %list_item250, align 1
426 %copied252 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
427 store i8 106, i8* %copied252, align 1
428 %data_ptr_container253 = getelementptr inbounds %list_item*, %list_item* %
    list_item250, i32 0, i32 0
429 store i8* %copied252, i8** %data_ptr_container253, align 8
430 %next254 = getelementptr inbounds %list_item*, %list_item* %list_item250,
    i32 0, i32 1
431 store %list_item* %list_item244, %list_item** %next254, align 8
432 %allocaall255 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
433 %list_item256 = bitcast i8* %allocaall255 to %list_item*
434 store %list_item zeroinitializer, %list_item* %list_item256, align 1
435 %copied258 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
436 store i8 105, i8* %copied258, align 1
437 %data_ptr_container259 = getelementptr inbounds %list_item*, %list_item* %
    list_item256, i32 0, i32 0
438 store i8* %copied258, i8** %data_ptr_container259, align 8
439 %next260 = getelementptr inbounds %list_item*, %list_item* %list_item256,
    i32 0, i32 1
440 store %list_item* %list_item250, %list_item** %next260, align 8
441 %allocaall261 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
442 %list_item262 = bitcast i8* %allocaall261 to %list_item*
443 store %list_item zeroinitializer, %list_item* %list_item262, align 1
444 %copied264 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
445 store i8 104, i8* %copied264, align 1
446 %data_ptr_container265 = getelementptr inbounds %list_item*, %list_item* %
    list_item262, i32 0, i32 0
447 store i8* %copied264, i8** %data_ptr_container265, align 8
448 %next266 = getelementptr inbounds %list_item*, %list_item* %list_item262,
    i32 0, i32 1
449 store %list_item* %list_item256, %list_item** %next266, align 8

```

```

450 %malloccall1267 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
451 %list_item268 = bitcast i8* %malloccall1267 to %list_item*
452 store %list_item zeroinitializer, %list_item* %list_item268, align 1
453 %copied270 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
454 store i8 103, i8* %copied270, align 1
455 %data_ptr_container271 = getelementptr inbounds %list_item, %list_item* %
    list_item268, i32 0, i32 0
456 store i8* %copied270, i8** %data_ptr_container271, align 8
457 %next272 = getelementptr inbounds %list_item, %list_item* %list_item268,
    i32 0, i32 1
458 store %list_item* %list_item262, %list_item** %next272, align 8
459 %malloccall1273 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
460 %list_item274 = bitcast i8* %malloccall1273 to %list_item*
461 store %list_item zeroinitializer, %list_item* %list_item274, align 1
462 %copied276 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
463 store i8 102, i8* %copied276, align 1
464 %data_ptr_container277 = getelementptr inbounds %list_item, %list_item* %
    list_item274, i32 0, i32 0
465 store i8* %copied276, i8** %data_ptr_container277, align 8
466 %next278 = getelementptr inbounds %list_item, %list_item* %list_item274,
    i32 0, i32 1
467 store %list_item* %list_item268, %list_item** %next278, align 8
468 %malloccall1279 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
469 %list_item280 = bitcast i8* %malloccall1279 to %list_item*
470 store %list_item zeroinitializer, %list_item* %list_item280, align 1
471 %copied282 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
472 store i8 101, i8* %copied282, align 1
473 %data_ptr_container283 = getelementptr inbounds %list_item, %list_item* %
    list_item280, i32 0, i32 0
474 store i8* %copied282, i8** %data_ptr_container283, align 8
475 %next284 = getelementptr inbounds %list_item, %list_item* %list_item280,
    i32 0, i32 1
476 store %list_item* %list_item274, %list_item** %next284, align 8
477 %malloccall1285 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
478 %list_item286 = bitcast i8* %malloccall1285 to %list_item*
479 store %list_item zeroinitializer, %list_item* %list_item286, align 1
480 %copied288 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
481 store i8 100, i8* %copied288, align 1
482 %data_ptr_container289 = getelementptr inbounds %list_item, %list_item* %
    list_item286, i32 0, i32 0
483 store i8* %copied288, i8** %data_ptr_container289, align 8
484 %next290 = getelementptr inbounds %list_item, %list_item* %list_item286,
    i32 0, i32 1
485 store %list_item* %list_item280, %list_item** %next290, align 8
486 %malloccall1291 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
487 %list_item292 = bitcast i8* %malloccall1291 to %list_item*
488 store %list_item zeroinitializer, %list_item* %list_item292, align 1
489 %copied294 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
490 store i8 99, i8* %copied294, align 1

```

```

491 %data_ptr_container295 = getelementptr @inbounds %list_item, %list_item* %
    list_item292, i32 0, i32 0
492 store i8* %copied294, i8** %data_ptr_container295, align 8
493 %next296 = getelementptr @inbounds %list_item, %list_item* %list_item292,
    i32 0, i32 1
494 store %list_item* %list_item286, %list_item** %next296, align 8
495 %alloca1297 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
496 %list_item298 = bitcast i8* %alloca1297 to %list_item*
497 store %list_item zeroinitializer, %list_item* %list_item298, align 1
498 %copied300 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
499 store i8 98, i8* %copied300, align 1
500 %data_ptr_container301 = getelementptr @inbounds %list_item, %list_item* %
    list_item298, i32 0, i32 0
501 store i8* %copied300, i8** %data_ptr_container301, align 8
502 %next302 = getelementptr @inbounds %list_item, %list_item* %list_item298,
    i32 0, i32 1
503 store %list_item* %list_item292, %list_item** %next302, align 8
504 %alloca1303 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
505 %list_item304 = bitcast i8* %alloca1303 to %list_item*
506 store %list_item zeroinitializer, %list_item* %list_item304, align 1
507 %copied306 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
508 store i8 97, i8* %copied306, align 1
509 %data_ptr_container307 = getelementptr @inbounds %list_item, %list_item* %
    list_item304, i32 0, i32 0
510 store i8* %copied306, i8** %data_ptr_container307, align 8
511 %next308 = getelementptr @inbounds %list_item, %list_item* %list_item304,
    i32 0, i32 1
512 store %list_item* %list_item298, %list_item** %next308, align 8
513 store %list_item* %list_item304, %list_item** %list, align 8
514 store %list_item** %list, %list_item*** @ASCII, align 8
515 %alloca1309 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
516 %list310 = bitcast i8* %alloca1309 to %list_item**
517 %alloca1311 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
518 %list_item312 = bitcast i8* %alloca1311 to %list_item*
519 store %list_item zeroinitializer, %list_item* %list_item312, align 1
520 %alloca1313 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
521 %copied314 = bitcast i8* %alloca1313 to i8**
522 store i8* getelementptr @inbounds ([6 x i8], [6 x i8]* @string.1, i32 0,
    i32 0), i8** %copied314, align 8
523 %cast_ptr = bitcast i8** %copied314 to i8*
524 %data_ptr_container315 = getelementptr @inbounds %list_item, %list_item* %
    list_item312, i32 0, i32 0
525 store i8* %cast_ptr, i8** %data_ptr_container315, align 8
526 %next316 = getelementptr @inbounds %list_item, %list_item* %list_item312,
    i32 0, i32 1
527 store %list_item* null, %list_item** %next316, align 8
528 store %list_item* %list_item312, %list_item** %list310, align 8
529 %result = call i8* @join_string_string(%list_item** %list310, i8*
    getelementptr @inbounds ([1 x i8], [1 x i8]* @string, i32 0, i32 0))
530 %alloca1317 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
531 %list318 = bitcast i8* %alloca1317 to %list_item**

```

```

532 %malloccall1319 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
533 %list_item320 = bitcast i8* %malloccall1319 to %list_item*
534 store %list_item zeroinitializer, %list_item* %list_item320, align 1
535 %malloccall1321 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
536 %copied322 = bitcast i8* %malloccall1321 to i32*
537 store i32 1, i32* %copied322, align 4
538 %cast_ptr323 = bitcast i32* %copied322 to i8*
539 %data_ptr_container324 = getelementptr inbounds %list_item, %list_item* %
    list_item320, i32 0, i32 0
540 store i8* %cast_ptr323, i8** %data_ptr_container324, align 8
541 %next325 = getelementptr inbounds %list_item, %list_item* %list_item320,
    i32 0, i32 1
542 store %list_item* null, %list_item** %next325, align 8
543 store %list_item* %list_item320, %list_item** %list318, align 8
544 %malloccall1326 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
545 %list327 = bitcast i8* %malloccall1326 to %list_item**
546 %malloccall1328 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
547 %list_item329 = bitcast i8* %malloccall1328 to %list_item*
548 store %list_item zeroinitializer, %list_item* %list_item329, align 1
549 %malloccall1330 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
550 %copied331 = bitcast i8* %malloccall1330 to i32*
551 store i32 2, i32* %copied331, align 4
552 %cast_ptr332 = bitcast i32* %copied331 to i8*
553 %data_ptr_container333 = getelementptr inbounds %list_item, %list_item* %
    list_item329, i32 0, i32 0
554 store i8* %cast_ptr332, i8** %data_ptr_container333, align 8
555 %next334 = getelementptr inbounds %list_item, %list_item* %list_item329,
    i32 0, i32 1
556 store %list_item* null, %list_item** %next334, align 8
557 %malloccall1335 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
558 %list_item336 = bitcast i8* %malloccall1335 to %list_item*
559 store %list_item zeroinitializer, %list_item* %list_item336, align 1
560 %malloccall1337 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
561 %copied338 = bitcast i8* %malloccall1337 to i32*
562 store i32 1, i32* %copied338, align 4
563 %cast_ptr339 = bitcast i32* %copied338 to i8*
564 %data_ptr_container340 = getelementptr inbounds %list_item, %list_item* %
    list_item336, i32 0, i32 0
565 store i8* %cast_ptr339, i8** %data_ptr_container340, align 8
566 %next341 = getelementptr inbounds %list_item, %list_item* %list_item336,
    i32 0, i32 1
567 store %list_item* %list_item329, %list_item** %next341, align 8
568 store %list_item* %list_item336, %list_item** %list327, align 8
569 %_result342 = call i1 @contains_int_int(%list_item** %list327, %list_item
    ** %list318)
570 %malloccall1343 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
571 %list344 = bitcast i8* %malloccall1343 to %list_item**
572 %malloccall1345 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
573 %list_item346 = bitcast i8* %malloccall1345 to %list_item*
574 store %list_item zeroinitializer, %list_item* %list_item346, align 1

```

```

575 %malloccall1347 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
576 %copied348 = bitcast i8* %malloccall1347 to i8**
577 store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.2, i32 0,
    i32 0), i8** %copied348, align 8
578 %cast_ptr349 = bitcast i8** %copied348 to i8*
579 %data_ptr_container350 = getelementptr inbounds %list_item, %list_item* %
    list_item346, i32 0, i32 0
580 store i8* %cast_ptr349, i8** %data_ptr_container350, align 8
581 %next351 = getelementptr inbounds %list_item, %list_item* %list_item346,
    i32 0, i32 1
582 store %list_item* null, %list_item** %next351, align 8
583 store %list_item* %list_item346, %list_item** %list344, align 8
584 %malloccall1352 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
585 %list353 = bitcast i8* %malloccall1352 to %list_item**
586 %malloccall1354 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
587 %list_item355 = bitcast i8* %malloccall1354 to %list_item*
588 store %list_item zeroinitializer, %list_item* %list_item355, align 1
589 %malloccall1356 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
590 %copied357 = bitcast i8* %malloccall1356 to i8**
591 store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.3, i32 0,
    i32 0), i8** %copied357, align 8
592 %cast_ptr358 = bitcast i8** %copied357 to i8*
593 %data_ptr_container359 = getelementptr inbounds %list_item, %list_item* %
    list_item355, i32 0, i32 0
594 store i8* %cast_ptr358, i8** %data_ptr_container359, align 8
595 %next360 = getelementptr inbounds %list_item, %list_item* %list_item355,
    i32 0, i32 1
596 store %list_item* null, %list_item** %next360, align 8
597 %malloccall1361 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
598 %list_item362 = bitcast i8* %malloccall1361 to %list_item*
599 store %list_item zeroinitializer, %list_item* %list_item362, align 1
600 %malloccall1363 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
601 %copied364 = bitcast i8* %malloccall1363 to i8**
602 store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.4, i32 0,
    i32 0), i8** %copied364, align 8
603 %cast_ptr365 = bitcast i8** %copied364 to i8*
604 %data_ptr_container366 = getelementptr inbounds %list_item, %list_item* %
    list_item362, i32 0, i32 0
605 store i8* %cast_ptr365, i8** %data_ptr_container366, align 8
606 %next367 = getelementptr inbounds %list_item, %list_item* %list_item362,
    i32 0, i32 1
607 store %list_item* %list_item355, %list_item** %next367, align 8
608 store %list_item* %list_item362, %list_item** %list353, align 8
609 %_result368 = call i1 @contains_string_string(%list_item** %list353, %
    list_item** %list344)
610 %malloccall1369 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
611 %list370 = bitcast i8* %malloccall1369 to %list_item**
612 %malloccall1371 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
613 %list_item372 = bitcast i8* %malloccall1371 to %list_item*
614 store %list_item zeroinitializer, %list_item* %list_item372, align 1

```

```

615 %copied374 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
616 store i8 104, i8* %copied374, align 1
617 %data_ptr_container375 = getelementptr inbounds %list_item, %list_item* %
    list_item372, i32 0, i32 0
618 store i8* %copied374, i8** %data_ptr_container375, align 8
619 %next376 = getelementptr inbounds %list_item, %list_item* %list_item372,
    i32 0, i32 1
620 store %list_item* null, %list_item** %next376, align 8
621 store %list_item* %list_item372, %list_item** %list370, align 8
622 %malloccall377 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
623 %list378 = bitcast i8* %malloccall377 to %list_item**
624 %malloccall379 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
625 %list_item380 = bitcast i8* %malloccall379 to %list_item*
626 store %list_item zeroinitializer, %list_item* %list_item380, align 1
627 %copied382 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
628 store i8 101, i8* %copied382, align 1
629 %data_ptr_container383 = getelementptr inbounds %list_item, %list_item* %
    list_item380, i32 0, i32 0
630 store i8* %copied382, i8** %data_ptr_container383, align 8
631 %next384 = getelementptr inbounds %list_item, %list_item* %list_item380,
    i32 0, i32 1
632 store %list_item* null, %list_item** %next384, align 8
633 %malloccall385 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
634 %list_item386 = bitcast i8* %malloccall385 to %list_item*
635 store %list_item zeroinitializer, %list_item* %list_item386, align 1
636 %copied388 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
637 store i8 104, i8* %copied388, align 1
638 %data_ptr_container389 = getelementptr inbounds %list_item, %list_item* %
    list_item386, i32 0, i32 0
639 store i8* %copied388, i8** %data_ptr_container389, align 8
640 %next390 = getelementptr inbounds %list_item, %list_item* %list_item386,
    i32 0, i32 1
641 store %list_item* %list_item380, %list_item** %next390, align 8
642 store %list_item* %list_item386, %list_item** %list378, align 8
643 %_result391 = call i1 @contains_char_char(%list_item** %list378, %
    list_item** %list370)
644 %malloccall392 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
645 %list393 = bitcast i8* %malloccall392 to %list_item**
646 %malloccall394 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
647 %list_item395 = bitcast i8* %malloccall394 to %list_item*
648 store %list_item zeroinitializer, %list_item* %list_item395, align 1
649 %malloccall396 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
650 %copied397 = bitcast i8* %malloccall396 to i1*
651 store i1 true, i1* %copied397, align 1
652 %cast_ptr398 = bitcast i1* %copied397 to i8*
653 %data_ptr_container399 = getelementptr inbounds %list_item, %list_item* %
    list_item395, i32 0, i32 0
654 store i8* %cast_ptr398, i8** %data_ptr_container399, align 8
655 %next400 = getelementptr inbounds %list_item, %list_item* %list_item395,
    i32 0, i32 1

```



```

656 store %list_item* null, %list_item** %next400, align 8
657 store %list_item* %list_item395, %list_item** %list393, align 8
658 %malloccall401 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
659 %list402 = bitcast i8* %malloccall401 to %list_item**
660 %malloccall403 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
661 %list_item404 = bitcast i8* %malloccall403 to %list_item*
662 store %list_item zeroinitializer, %list_item* %list_item404, align 1
663 %malloccall405 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
664 %copied406 = bitcast i8* %malloccall405 to i1*
665 store i1 false, i1* %copied406, align 1
666 %cast_ptr407 = bitcast i1* %copied406 to i8*
667 %data_ptr_container408 = getelementptr inbounds %list_item, %list_item* %
    list_item404, i32 0, i32 0
668 store i8* %cast_ptr407, i8** %data_ptr_container408, align 8
669 %next409 = getelementptr inbounds %list_item, %list_item* %list_item404,
    i32 0, i32 1
670 store %list_item* null, %list_item** %next409, align 8
671 %malloccall410 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
672 %list_item411 = bitcast i8* %malloccall410 to %list_item*
673 store %list_item zeroinitializer, %list_item* %list_item411, align 1
674 %malloccall412 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
675 %copied413 = bitcast i8* %malloccall412 to i1*
676 store i1 true, i1* %copied413, align 1
677 %cast_ptr414 = bitcast i1* %copied413 to i8*
678 %data_ptr_container415 = getelementptr inbounds %list_item, %list_item* %
    list_item411, i32 0, i32 0
679 store i8* %cast_ptr414, i8** %data_ptr_container415, align 8
680 %next416 = getelementptr inbounds %list_item, %list_item* %list_item411,
    i32 0, i32 1
681 store %list_item* %list_item404, %list_item** %next416, align 8
682 store %list_item* %list_item411, %list_item** %list402, align 8
683 %_result417 = call i1 @contains_bool_bool(%list_item** %list402, %
    list_item** %list393)
684 %malloccall418 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
685 %list419 = bitcast i8* %malloccall418 to %list_item**
686 %malloccall420 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
687 %list_item421 = bitcast i8* %malloccall420 to %list_item*
688 store %list_item zeroinitializer, %list_item* %list_item421, align 1
689 %malloccall422 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
    (double, double* null, i32 1) to i32))
690 %copied423 = bitcast i8* %malloccall422 to double*
691 store double 1.100000e+00, double* %copied423, align 8
692 %cast_ptr424 = bitcast double* %copied423 to i8*
693 %data_ptr_container425 = getelementptr inbounds %list_item, %list_item* %
    list_item421, i32 0, i32 0
694 store i8* %cast_ptr424, i8** %data_ptr_container425, align 8
695 %next426 = getelementptr inbounds %list_item, %list_item* %list_item421,
    i32 0, i32 1
696 store %list_item* null, %list_item** %next426, align 8
697 store %list_item* %list_item421, %list_item** %list419, align 8
698 %malloccall427 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))

```



```

699 %list428 = bitcast i8* %malloccall427 to %list_item**
700 %malloccall429 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
701 %list_item430 = bitcast i8* %malloccall429 to %list_item*
702 store %list_item zeroinitializer, %list_item* %list_item430, align 1
703 %malloccall431 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
    (double, double* null, i32 1) to i32))
704 %copied432 = bitcast i8* %malloccall431 to double*
705 store double 1.200000e+00, double* %copied432, align 8
706 %cast_ptr433 = bitcast double* %copied432 to i8*
707 %data_ptr_container434 = getelementptr inbounds %list_item, %list_item* %
    list_item430, i32 0, i32 0
708 store i8* %cast_ptr433, i8** %data_ptr_container434, align 8
709 %next435 = getelementptr inbounds %list_item, %list_item* %list_item430,
    i32 0, i32 1
710 store %list_item* null, %list_item** %next435, align 8
711 %malloccall436 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
712 %list_item437 = bitcast i8* %malloccall436 to %list_item*
713 store %list_item zeroinitializer, %list_item* %list_item437, align 1
714 %malloccall438 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
    (double, double* null, i32 1) to i32))
715 %copied439 = bitcast i8* %malloccall438 to double*
716 store double 1.100000e+00, double* %copied439, align 8
717 %cast_ptr440 = bitcast double* %copied439 to i8*
718 %data_ptr_container441 = getelementptr inbounds %list_item, %list_item* %
    list_item437, i32 0, i32 0
719 store i8* %cast_ptr440, i8** %data_ptr_container441, align 8
720 %next442 = getelementptr inbounds %list_item, %list_item* %list_item437,
    i32 0, i32 1
721 store %list_item* %list_item430, %list_item** %next442, align 8
722 store %list_item* %list_item437, %list_item** %list428, align 8
723 %_result443 = call i1 @contains_float_float(%list_item** %list428, %
    list_item** %list419)
724 %malloccall444 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
725 %list445 = bitcast i8* %malloccall444 to %list_item**
726 %malloccall446 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
727 %list_item447 = bitcast i8* %malloccall446 to %list_item*
728 store %list_item zeroinitializer, %list_item* %list_item447, align 1
729 %malloccall448 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
730 %copied449 = bitcast i8* %malloccall448 to i32*
731 store i32 1, i32* %copied449, align 4
732 %cast_ptr450 = bitcast i32* %copied449 to i8*
733 %data_ptr_container451 = getelementptr inbounds %list_item, %list_item* %
    list_item447, i32 0, i32 0
734 store i8* %cast_ptr450, i8** %data_ptr_container451, align 8
735 %next452 = getelementptr inbounds %list_item, %list_item* %list_item447,
    i32 0, i32 1
736 store %list_item* null, %list_item** %next452, align 8
737 store %list_item* %list_item447, %list_item** %list445, align 8
738 %_result453 = call %list_item** @remove_int_int_int_bool(%list_item** %
    list445, i32 1, i1 true)
739 %malloccall454 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
740 %list455 = bitcast i8* %malloccall454 to %list_item**

```

```

741 %malloccall456 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
742 %list_item457 = bitcast i8* %malloccall456 to %list_item*
743 store %list_item zeroinitializer, %list_item* %list_item457, align 1
744 %malloccall458 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
    (double, double* null, i32 1) to i32))
745 %copied459 = bitcast i8* %malloccall458 to double*
746 store double 1.500000e+00, double* %copied459, align 8
747 %cast_ptr460 = bitcast double* %copied459 to i8*
748 %data_ptr_container461 = getelementptr inbounds %list_item, %list_item* %
    list_item457, i32 0, i32 0
749 store i8* %cast_ptr460, i8** %data_ptr_container461, align 8
750 %next462 = getelementptr inbounds %list_item, %list_item* %list_item457,
    i32 0, i32 1
751 store %list_item* null, %list_item** %next462, align 8
752 store %list_item* %list_item457, %list_item** %list455, align 8
753 %_result463 = call %list_item** @remove_float_float_float_bool(%list_item
    ** %list455, double 1.500000e+00, i1 false)
754 %malloccall464 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
755 %list465 = bitcast i8* %malloccall464 to %list_item**
756 %malloccall466 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
757 %list_item467 = bitcast i8* %malloccall466 to %list_item*
758 store %list_item zeroinitializer, %list_item* %list_item467, align 1
759 %malloccall468 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
760 %copied469 = bitcast i8* %malloccall468 to i1*
761 store i1 true, i1* %copied469, align 1
762 %cast_ptr470 = bitcast i1* %copied469 to i8*
763 %data_ptr_container471 = getelementptr inbounds %list_item, %list_item* %
    list_item467, i32 0, i32 0
764 store i8* %cast_ptr470, i8** %data_ptr_container471, align 8
765 %next472 = getelementptr inbounds %list_item, %list_item* %list_item467,
    i32 0, i32 1
766 store %list_item* null, %list_item** %next472, align 8
767 store %list_item* %list_item467, %list_item** %list465, align 8
768 %_result473 = call %list_item** @remove_bool_bool_bool_bool(%list_item** %
    list465, i1 true, i1 true)
769 %malloccall474 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
770 %list475 = bitcast i8* %malloccall474 to %list_item**
771 %malloccall476 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
772 %list_item477 = bitcast i8* %malloccall476 to %list_item*
773 store %list_item zeroinitializer, %list_item* %list_item477, align 1
774 %copied479 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
775 store i8 97, i8* %copied479, align 1
776 %data_ptr_container480 = getelementptr inbounds %list_item, %list_item* %
    list_item477, i32 0, i32 0
777 store i8* %copied479, i8** %data_ptr_container480, align 8
778 %next481 = getelementptr inbounds %list_item, %list_item* %list_item477,
    i32 0, i32 1
779 store %list_item* null, %list_item** %next481, align 8
780 store %list_item* %list_item477, %list_item** %list475, align 8
781 %_result482 = call %list_item** @remove_char_char_char_bool(%list_item** %
    list475, i8 97, i1 true)

```

```

782 %malloccall483 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
783 %list484 = bitcast i8* %malloccall483 to %list_item**
784 %malloccall485 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
785 %list_item486 = bitcast i8* %malloccall485 to %list_item*
786 store %list_item zeroinitializer, %list_item* %list_item486, align 1
787 %malloccall487 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
788 %copied488 = bitcast i8* %malloccall487 to i8**
789 store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.6, i32 0,
    i32 0), i8** %copied488, align 8
790 %cast_ptr489 = bitcast i8** %copied488 to i8*
791 %data_ptr_container490 = getelementptr inbounds %list_item, %list_item* %
    list_item486, i32 0, i32 0
792 store i8* %cast_ptr489, i8** %data_ptr_container490, align 8
793 %next491 = getelementptr inbounds %list_item, %list_item* %list_item486,
    i32 0, i32 1
794 store %list_item* null, %list_item** %next491, align 8
795 store %list_item* %list_item486, %list_item** %list484, align 8
796 %_result492 = call %list_item** @remove_string_string_string_bool(%
    list_item** %list484, i8* getelementptr inbounds ([5 x i8], [5 x i8]*
    @string.5, i32 0, i32 0), i1 true)
797 ret i32 0
798 }
799
800 define %list_item** @remove_string_string_string_bool(%list_item** %l, i8* %
    elem, i1 %all) {
801 entry:
802 %index = alloca i32, align 4
803 %for_index = alloca i32, align 4
804 %remove_index = alloca i32, align 4
805 %len = alloca i32, align 4
806 %i = alloca i32, align 4
807 %retlist = alloca %list_item**, align 8
808 %l1 = alloca %list_item**, align 8
809 store %list_item** %l, %list_item*** %l1, align 8
810 %elem2 = alloca i8*, align 8
811 store i8* %elem, i8** %elem2, align 8
812 %all3 = alloca i1, align 1
813 store i1 %all, i1* %all3, align 1
814 %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
815 %list = bitcast i8* %malloccall to %list_item**
816 store %list_item* null, %list_item** %list, align 8
817 store %list_item** %list, %list_item*** %retlist, align 8
818 store i32 0, i32* %i, align 4
819 %l4 = load %list_item**, %list_item*** %l1, align 8
820 %ilist = load %list_item*, %list_item** %l4, align 8
821 %length = call i32 @list_length(%list_item* %ilist, i32 0)
822 store i32 %length, i32* %len, align 4
823 store i32 0, i32* %remove_index, align 4
824 %all5 = load i1, i1* %all3, align 1
825 br i1 %all5, label %then, label %else30
826
827 merge:                                ; preds = %merge68, %
    merge6
828 %retlist109 = load %list_item**, %list_item*** %retlist, align 8
829 ret %list_item** %retlist109

```

```

830
831 then:                                     ; preds = %entry
832     br label %while
833
834 while:                                     ; preds = %merge12, %
835     then13, %then
836     %i27 = load i32, i32* %i, align 4
837     %len28 = load i32, i32* %len, align 4
838     %tmp29 = icmp slt i32 %i27, %len28
839     br i1 %tmp29, label %while_body, label %merge6
840
841 merge6:                                     ; preds = %while
842     br label %merge
843
844 while_body:                               ; preds = %while
845     %elem7 = load i8*, i8** %elem2, align 8
846     %l18 = load %list_item**, %list_item*** %l1, align 8
847     %ilist9 = load %list_item*, %list_item** %l18, align 8
848     %i10 = load i32, i32* %i, align 4
849     %_result = call %list_item* @list_access(%list_item* %ilist9, i32 %i10)
850     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
851         i32 0, i32 0
852     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
853     %cast_data_ptr = bitcast i8* %data_ptr to i8**
854     %data = load i8*, i8** %cast_data_ptr, align 8
855     %_resultt11 = call i1 @strcmp(i8* %data, i8* %elem7)
856     br i1 %_resultt11, label %then13, label %else
857
858 merge12:                                   ; preds = %else
859     %retlist15 = load %list_item**, %list_item*** %retlist, align 8
860     %list_ptr = load %list_item*, %list_item** %retlist15, align 8
861     %l16 = load %list_item**, %list_item*** %l1, align 8
862     %ilist17 = load %list_item*, %list_item** %l16, align 8
863     %i18 = load i32, i32* %i, align 4
864     %_resultt19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
865     %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %_resultt19,
866         i32 0, i32 0
867     %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
868     %cast_data_ptr22 = bitcast i8* %data_ptr21 to i8**
869     %data23 = load i8*, i8** %cast_data_ptr22, align 8
870     %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
871     %list_ptr_ptr = call %list_item** @insert_string(%list_item** %retlist15,
872         i8* %data23, i32 %length24)
873     store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
874     %i25 = load i32, i32* %i, align 4
875     %tmp26 = add i32 %i25, 1
876     store i32 %tmp26, i32* %i, align 4
877     br label %while
878
879 then13:                                   ; preds = %while_body
880     %i14 = load i32, i32* %i, align 4
881     %tmp = add i32 %i14, 1
882     store i32 %tmp, i32* %i, align 4
883     br label %while
884
885 else:                                     ; preds = %while_body
886     br label %merge12
887
888 else30:                                   ; preds = %entry

```

```

885     br label %while31
886
887 while31:                                     ; preds = %merge44, %
888     else30
889     %i62 = load i32, i32* %i, align 4
890     %len63 = load i32, i32* %len, align 4
891     %tmp64 = icmp slt i32 %i62, %len63
892     br i1 %tmp64, label %while_body33, label %merge32
893 merge32:                                     ; preds = %while31, %
894     then45
895     %i65 = load i32, i32* %i, align 4
896     %len66 = load i32, i32* %len, align 4
897     %tmp67 = icmp ne i32 %i65, %len66
898     br i1 %tmp67, label %then69, label %else108
899 while_body33:                               ; preds = %while31
900     %elem34 = load i8*, i8** %elem2, align 8
901     %l35 = load %list_item**, %list_item*** %l1, align 8
902     %ilist36 = load %list_item*, %list_item** %l35, align 8
903     %i37 = load i32, i32* %i, align 4
904     %_result38 = call %list_item* @list_access(%list_item* %ilist36, i32 %i37)
905     %data_ptr_ptr39 = getelementptr inbounds %list_item, %list_item* %
906     _result38, i32 0, i32 0
907     %data_ptr40 = load i8*, i8** %data_ptr_ptr39, align 8
908     %cast_data_ptr41 = bitcast i8* %data_ptr40 to i8**
909     %data42 = load i8*, i8** %cast_data_ptr41, align 8
910     %_result43 = call i1 @strcmp(i8* %data42, i8* %elem34)
911     br i1 %_result43, label %then45, label %else47
912 merge44:                                     ; preds = %else47
913     %retlist48 = load %list_item**, %list_item*** %retlist, align 8
914     %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
915     %l50 = load %list_item**, %list_item*** %l1, align 8
916     %ilist51 = load %list_item*, %list_item** %l50, align 8
917     %i52 = load i32, i32* %i, align 4
918     %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
919     %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
920     _result53, i32 0, i32 0
921     %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
922     %cast_data_ptr56 = bitcast i8* %data_ptr55 to i8**
923     %data57 = load i8*, i8** %cast_data_ptr56, align 8
924     %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
925     %list_ptr_ptr59 = call %list_item** @insert_string(%list_item** %retlist48
926     , i8* %data57, i32 %length58)
927     store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
928     %i60 = load i32, i32* %i, align 4
929     %tmp61 = add i32 %i60, 1
930     store i32 %tmp61, i32* %i, align 4
931     br label %while31
932 then45:                                     ; preds = %while_body33
933     %i46 = load i32, i32* %i, align 4
934     store i32 %i46, i32* %remove_index, align 4
935     br label %merge32
936 else47:                                     ; preds = %while_body33
937     br label %merge44
938

```

```

939 merge68:                                     ; preds = %else108, %
    merge71
940 br label %merge
941
942 then69:                                       ; preds = %merge32
943 store i32 0, i32* %for_index, align 4
944 store i32 0, i32* %index, align 4
945 br label %while70
946
947 while70:                                     ; preds = %while_body72, %
    then69
948 %for_index98 = load i32, i32* %for_index, align 4
949 %remove_index99 = load i32, i32* %remove_index, align 4
950 %tmp100 = add i32 %remove_index99, 1
951 %len101 = load i32, i32* %len, align 4
952 %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
953 %head_ptr_ptr103 = bitcast i8* %malloccall102 to %list_item**
954 store %list_item* null, %list_item** %head_ptr_ptr103, align 8
955 %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
956 %i105 = load %list_item*, %list_item** %range_list104, align 8
957 %length106 = call i32 @list_length(%list_item* %i105, i32 0)
958 %tmp107 = icmp slt i32 %for_index98, %length106
959 br i1 %tmp107, label %while_body72, label %merge71
960
961 merge71:                                     ; preds = %while70
962 br label %merge68
963
964 while_body72:                               ; preds = %while70
965 %remove_index73 = load i32, i32* %remove_index, align 4
966 %tmp74 = add i32 %remove_index73, 1
967 %len75 = load i32, i32* %len, align 4
968 %malloccall176 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
969 %head_ptr_ptr = bitcast i8* %malloccall176 to %list_item**
970 store %list_item* null, %list_item** %head_ptr_ptr, align 8
971 %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
972 %i177 = load %list_item*, %list_item** %range_list, align 8
973 %for_index78 = load i32, i32* %for_index, align 4
974 %_result79 = call %list_item* @list_access(%list_item* %i177, i32 %
    for_index78)
975 %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
976 %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
977 %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
978 %data83 = load i32, i32* %cast_data_ptr82, align 4
979 store i32 %data83, i32* %index, align 4
980 %for_index84 = load i32, i32* %for_index, align 4
981 %tmp85 = add i32 %for_index84, 1
982 store i32 %tmp85, i32* %for_index, align 4
983 %retlist86 = load %list_item**, %list_item** %retlist, align 8
984 %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
985 %l88 = load %list_item**, %list_item** %l1, align 8
986 %i189 = load %list_item*, %list_item** %l88, align 8
987 %index90 = load i32, i32* %index, align 4
988 %_result91 = call %list_item* @list_access(%list_item* %i189, i32 %
    index90)

```

```

989 %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
990 %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
991 %cast_data_ptr94 = bitcast i8* %data_ptr93 to i8**
992 %data95 = load i8*, i8** %cast_data_ptr94, align 8
993 %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
994 %list_ptr_ptr97 = call %list_item** @insert_string(%list_item** %retlist86
    , i8* %data95, i32 %length96)
995 store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
996 br label %while70
997
998 else108:                                ; preds = %merge32
999     br label %merge68
1000 }
1001
1002 define %list_item** @remove_char_char_char_bool(%list_item** %l, i8 %elem,
    i1 %all) {
1003 entry:
1004     %index = alloca i32, align 4
1005     %for_index = alloca i32, align 4
1006     %remove_index = alloca i32, align 4
1007     %len = alloca i32, align 4
1008     %i = alloca i32, align 4
1009     %retlist = alloca %list_item**, align 8
1010     %l1 = alloca %list_item**, align 8
1011     store %list_item** %l, %list_item*** %l1, align 8
1012     %elem2 = alloca i8, align 1
1013     store i8 %elem, i8* %elem2, align 1
1014     %all3 = alloca i1, align 1
1015     store i1 %all, i1* %all3, align 1
1016     %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))
1017     %list = bitcast i8* %malloccall to %list_item**
1018     store %list_item* null, %list_item** %list, align 8
1019     store %list_item** %list, %list_item*** %retlist, align 8
1020     store i32 0, i32* %i, align 4
1021     %l4 = load %list_item**, %list_item*** %l1, align 8
1022     %ilist = load %list_item*, %list_item** %l4, align 8
1023     %length = call i32 @list_length(%list_item* %ilist, i32 0)
1024     store i32 %length, i32* %len, align 4
1025     store i32 0, i32* %remove_index, align 4
1026     %all5 = load i1, i1* %all3, align 1
1027     br i1 %all5, label %then, label %else29
1028
1029 merge:                                ; preds = %merge65, %
    merge6
1030     %retlist104 = load %list_item**, %list_item*** %retlist, align 8
1031     ret %list_item** %retlist104
1032
1033 then:                                ; preds = %entry
1034     br label %while
1035
1036 while:                                ; preds = %merge11, %
    then12, %then
1037     %i26 = load i32, i32* %i, align 4
1038     %len27 = load i32, i32* %len, align 4
1039     %tmp28 = icmp slt i32 %i26, %len27
1040     br i1 %tmp28, label %while_body, label %merge6
1041

```

```

1042 merge6:                                     ; preds = %while
1043     br label %merge
1044
1045 while_body:                                   ; preds = %while
1046     %l7 = load %list_item**, %list_item*** %l1, align 8
1047     %ilist8 = load %list_item*, %list_item** %l7, align 8
1048     %i9 = load i32, i32* %i, align 4
1049     %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1050     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
1051         i32 0, i32 0
1052     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1053     %data = load i8, i8* %data_ptr, align 1
1054     %elem10 = load i8, i8* %elem2, align 1
1055     %tmp = icmp eq i8 %data, %elem10
1056     br i1 %tmp, label %then12, label %else
1057
1058 merge11:                                     ; preds = %else
1059     %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1060     %list_ptr = load %list_item*, %list_item** %retlist15, align 8
1061     %l16 = load %list_item**, %list_item*** %l1, align 8
1062     %ilist17 = load %list_item*, %list_item** %l16, align 8
1063     %i18 = load i32, i32* %i, align 4
1064     %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
1065     %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
1066         _result19, i32 0, i32 0
1067     %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
1068     %data22 = load i8, i8* %data_ptr21, align 1
1069     %length23 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1070     %list_ptr_ptr = call %list_item** @insert_char(%list_item** %retlist15, i8
1071         %data22, i32 %length23)
1072     store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1073     %i24 = load i32, i32* %i, align 4
1074     %tmp25 = add i32 %i24, 1
1075     store i32 %tmp25, i32* %i, align 4
1076     br label %while
1077
1078 then12:                                     ; preds = %while_body
1079     %i13 = load i32, i32* %i, align 4
1080     %tmp14 = add i32 %i13, 1
1081     store i32 %tmp14, i32* %i, align 4
1082     br label %while
1083
1084 else:                                       ; preds = %while_body
1085     br label %merge11
1086
1087 else29:                                   ; preds = %entry
1088     br label %while30
1089
1090 while30:                                   ; preds = %merge42, %
1091     else29
1092     %i59 = load i32, i32* %i, align 4
1093     %len60 = load i32, i32* %len, align 4
1094     %tmp61 = icmp slt i32 %i59, %len60
1095     br i1 %tmp61, label %while_body32, label %merge31
1096
1097 merge31:                                   ; preds = %while30, %
1098     then43
1099     %i62 = load i32, i32* %i, align 4
1100     %len63 = load i32, i32* %len, align 4

```



```

1096 %tmp64 = icmp ne i32 %i62, %len63
1097 br i1 %tmp64, label %then66, label %else103
1098
1099 while_body32:                                ; preds = %while30
1100 %l33 = load %list_item**, %list_item*** %l1, align 8
1101 %ilist34 = load %list_item*, %list_item** %l33, align 8
1102 %i35 = load i32, i32* %i, align 4
1103 %_result36 = call %list_item* @list_access(%list_item* %ilist34, i32 %i35)
1104 %data_ptr_ptr37 = getelementptr inbounds %list_item, %list_item* %
    _result36, i32 0, i32 0
1105 %data_ptr38 = load i8*, i8** %data_ptr_ptr37, align 8
1106 %data39 = load i8, i8* %data_ptr38, align 1
1107 %elem40 = load i8, i8* %elem2, align 1
1108 %tmp41 = icmp eq i8 %data39, %elem40
1109 br i1 %tmp41, label %then43, label %else45
1110
1111 merge42:                                    ; preds = %else45
1112 %retlist46 = load %list_item**, %list_item*** %retlist, align 8
1113 %list_ptr47 = load %list_item*, %list_item** %retlist46, align 8
1114 %l48 = load %list_item**, %list_item*** %l1, align 8
1115 %ilist49 = load %list_item*, %list_item** %l48, align 8
1116 %i50 = load i32, i32* %i, align 4
1117 %_result51 = call %list_item* @list_access(%list_item* %ilist49, i32 %i50)
1118 %data_ptr_ptr52 = getelementptr inbounds %list_item, %list_item* %
    _result51, i32 0, i32 0
1119 %data_ptr53 = load i8*, i8** %data_ptr_ptr52, align 8
1120 %data54 = load i8, i8* %data_ptr53, align 1
1121 %length55 = call i32 @list_length(%list_item* %list_ptr47, i32 0)
1122 %list_ptr_ptr56 = call %list_item** @insert_char(%list_item** %retlist46,
    i8 %data54, i32 %length55)
1123 store %list_item** %list_ptr_ptr56, %list_item*** %retlist, align 8
1124 %i57 = load i32, i32* %i, align 4
1125 %tmp58 = add i32 %i57, 1
1126 store i32 %tmp58, i32* %i, align 4
1127 br label %while30
1128
1129 then43:                                    ; preds = %while_body32
1130 %i44 = load i32, i32* %i, align 4
1131 store i32 %i44, i32* %remove_index, align 4
1132 br label %merge31
1133
1134 else45:                                    ; preds = %while_body32
1135 br label %merge42
1136
1137 merge65:                                    ; preds = %else103, %
    merge68
1138 br label %merge
1139
1140 then66:                                    ; preds = %merge31
1141 store i32 0, i32* %for_index, align 4
1142 store i32 0, i32* %index, align 4
1143 br label %while67
1144
1145 while67:                                    ; preds = %while_body69, %
    then66
1146 %for_index93 = load i32, i32* %for_index, align 4
1147 %remove_index94 = load i32, i32* %remove_index, align 4
1148 %tmp95 = add i32 %remove_index94, 1
1149 %len96 = load i32, i32* %len, align 4

```

```

1150 %malloccall97 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
1151 %head_ptr_ptr98 = bitcast i8* %malloccall97 to %list_item**
1152 store %list_item* null, %list_item** %head_ptr_ptr98, align 8
1153 %range_list99 = call %list_item** @range_function(i32 %tmp95, i32 %len96,
    %list_item** %head_ptr_ptr98, i32 0)
1154 %ilist100 = load %list_item*, %list_item** %range_list99, align 8
1155 %length101 = call i32 @list_length(%list_item* %ilist100, i32 0)
1156 %tmp102 = icmp slt i32 %for_index93, %length101
1157 br i1 %tmp102, label %while_body69, label %merge68
1158
1159 merge68:                                ; preds = %while67
1160 br label %merge65
1161
1162 while_body69:                            ; preds = %while67
1163 %remove_index70 = load i32, i32* %remove_index, align 4
1164 %tmp71 = add i32 %remove_index70, 1
1165 %len72 = load i32, i32* %len, align 4
1166 %malloccall73 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
1167 %head_ptr_ptr = bitcast i8* %malloccall73 to %list_item**
1168 store %list_item* null, %list_item** %head_ptr_ptr, align 8
1169 %range_list = call %list_item** @range_function(i32 %tmp71, i32 %len72, %
    list_item** %head_ptr_ptr, i32 0)
1170 %ilist74 = load %list_item*, %list_item** %range_list, align 8
1171 %for_index75 = load i32, i32* %for_index, align 4
1172 %_result76 = call %list_item* @list_access(%list_item* %ilist74, i32 %
    for_index75)
1173 %data_ptr_ptr77 = getelementptr inbounds %list_item, %list_item* %
    _result76, i32 0, i32 0
1174 %data_ptr78 = load i8*, i8** %data_ptr_ptr77, align 8
1175 %cast_data_ptr = bitcast i8* %data_ptr78 to i32*
1176 %data79 = load i32, i32* %cast_data_ptr, align 4
1177 store i32 %data79, i32* %index, align 4
1178 %for_index80 = load i32, i32* %for_index, align 4
1179 %tmp81 = add i32 %for_index80, 1
1180 store i32 %tmp81, i32* %for_index, align 4
1181 %retlist82 = load %list_item**, %list_item*** %retlist, align 8
1182 %list_ptr83 = load %list_item*, %list_item** %retlist82, align 8
1183 %l84 = load %list_item**, %list_item*** %l1, align 8
1184 %ilist85 = load %list_item*, %list_item** %l84, align 8
1185 %index86 = load i32, i32* %index, align 4
1186 %_result87 = call %list_item* @list_access(%list_item* %ilist85, i32 %
    index86)
1187 %data_ptr_ptr88 = getelementptr inbounds %list_item, %list_item* %
    _result87, i32 0, i32 0
1188 %data_ptr89 = load i8*, i8** %data_ptr_ptr88, align 8
1189 %data90 = load i8, i8* %data_ptr89, align 1
1190 %length91 = call i32 @list_length(%list_item* %list_ptr83, i32 0)
1191 %list_ptr_ptr92 = call %list_item** @insert_char(%list_item** %retlist82,
    i8 %data90, i32 %length91)
1192 store %list_item** %list_ptr_ptr92, %list_item*** %retlist, align 8
1193 br label %while67
1194
1195 else103:                                ; preds = %merge31
1196 br label %merge65
1197 }
1198

```

```

1199 define %list_item** @remove_bool_bool_bool_bool(%list_item** %l, i1 %elem,
1200         i1 %all) {
1201 entry:
1202     %index = alloca i32, align 4
1203     %for_index = alloca i32, align 4
1204     %remove_index = alloca i32, align 4
1205     %len = alloca i32, align 4
1206     %i = alloca i32, align 4
1207     %retlist = alloca %list_item**, align 8
1208     %l1 = alloca %list_item**, align 8
1209     store %list_item** %l, %list_item*** %l1, align 8
1210     %elem2 = alloca i1, align 1
1211     store i1 %elem, i1* %elem2, align 1
1212     %all3 = alloca i1, align 1
1213     store i1 %all, i1* %all3, align 1
1214     %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
1215         i1** null, i32 1) to i32))
1216     %list = bitcast i8* %malloccall to %list_item**
1217     store %list_item* null, %list_item** %list, align 8
1218     store %list_item** %list, %list_item*** %retlist, align 8
1219     store i32 0, i32* %i, align 4
1220     %l4 = load %list_item**, %list_item*** %l1, align 8
1221     %ilist = load %list_item*, %list_item** %l4, align 8
1222     %length = call i32 @list_length(%list_item* %ilist, i32 0)
1223     store i32 %length, i32* %len, align 4
1224     store i32 0, i32* %remove_index, align 4
1225     %all5 = load i1, i1* %all3, align 1
1226     br i1 %all5, label %then, label %else30
1227
1228 merge:
1229     ; preds = %merge68, %
1230     merge6
1231     %retlist109 = load %list_item**, %list_item*** %retlist, align 8
1232     ret %list_item** %retlist109
1233
1234 then:
1235     ; preds = %entry
1236     br label %while
1237
1238 while:
1239     ; preds = %merge11, %
1240     then12, %then
1241     %i27 = load i32, i32* %i, align 4
1242     %len28 = load i32, i32* %len, align 4
1243     %tmp29 = icmp slt i32 %i27, %len28
1244     br i1 %tmp29, label %while_body, label %merge6
1245
1246 merge6:
1247     ; preds = %while
1248     br label %merge
1249
1250 while_body:
1251     ; preds = %while
1252     %l7 = load %list_item**, %list_item*** %l1, align 8
1253     %ilist8 = load %list_item*, %list_item** %l7, align 8
1254     %i9 = load i32, i32* %i, align 4
1255     @_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1256     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* @_result,
1257         i32 0, i32 0
1258     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1259     %cast_data_ptr = bitcast i8* %data_ptr to i1*
1260     %data = load i1, i1* %cast_data_ptr, align 1
1261     %elem10 = load i1, i1* %elem2, align 1
1262     %tmp = icmp eq i1 %data, %elem10

```

```

1253     br i1 %tmp, label %then12, label %else
1254
1255 merge11:                                     ; preds = %else
1256     %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1257     %list_ptr = load %list_item*, %list_item** %retlist15, align 8
1258     %l16 = load %list_item**, %list_item*** %l1, align 8
1259     %ilist17 = load %list_item*, %list_item** %l16, align 8
1260     %i18 = load i32, i32* %i, align 4
1261     %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
1262     %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
        _result19, i32 0, i32 0
1263     %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
1264     %cast_data_ptr22 = bitcast i8* %data_ptr21 to i1*
1265     %data23 = load i1, i1* %cast_data_ptr22, align 1
1266     %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1267     %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %retlist15, i1
        %data23, i32 %length24)
1268     store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1269     %i25 = load i32, i32* %i, align 4
1270     %tmp26 = add i32 %i25, 1
1271     store i32 %tmp26, i32* %i, align 4
1272     br label %while
1273
1274 then12:                                     ; preds = %while_body
1275     %i13 = load i32, i32* %i, align 4
1276     %tmp14 = add i32 %i13, 1
1277     store i32 %tmp14, i32* %i, align 4
1278     br label %while
1279
1280 else:                                       ; preds = %while_body
1281     br label %merge11
1282
1283 else30:                                    ; preds = %entry
1284     br label %while31
1285
1286 while31:                                   ; preds = %merge44, %
        else30
1287     %i62 = load i32, i32* %i, align 4
1288     %len63 = load i32, i32* %len, align 4
1289     %tmp64 = icmp slt i32 %i62, %len63
1290     br i1 %tmp64, label %while_body33, label %merge32
1291
1292 merge32:                                   ; preds = %while31, %
        then45
1293     %i65 = load i32, i32* %i, align 4
1294     %len66 = load i32, i32* %len, align 4
1295     %tmp67 = icmp ne i32 %i65, %len66
1296     br i1 %tmp67, label %then69, label %else108
1297
1298 while_body33:                             ; preds = %while31
1299     %l34 = load %list_item**, %list_item*** %l1, align 8
1300     %ilist35 = load %list_item*, %list_item** %l34, align 8
1301     %i36 = load i32, i32* %i, align 4
1302     %_result37 = call %list_item* @list_access(%list_item* %ilist35, i32 %i36)
1303     %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
        _result37, i32 0, i32 0
1304     %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
1305     %cast_data_ptr40 = bitcast i8* %data_ptr39 to i1*
1306     %data41 = load i1, i1* %cast_data_ptr40, align 1

```

```

1307 %elem42 = load i1, i1* %elem2, align 1
1308 %tmp43 = icmp eq i1 %data41, %elem42
1309 br i1 %tmp43, label %then45, label %else47
1310
1311 merge44:                                     ; preds = %else47
1312 %retlist48 = load %list_item**, %list_item*** %retlist, align 8
1313 %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
1314 %l50 = load %list_item**, %list_item*** %l1, align 8
1315 %ilist51 = load %list_item*, %list_item** %l50, align 8
1316 %i52 = load i32, i32* %i, align 4
1317 %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
1318 %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
    _result53, i32 0, i32 0
1319 %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
1320 %cast_data_ptr56 = bitcast i8* %data_ptr55 to i1*
1321 %data57 = load i1, i1* %cast_data_ptr56, align 1
1322 %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
1323 %list_ptr_ptr59 = call %list_item** @insert_bool(%list_item** %retlist48,
    i1 %data57, i32 %length58)
1324 store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
1325 %i60 = load i32, i32* %i, align 4
1326 %tmp61 = add i32 %i60, 1
1327 store i32 %tmp61, i32* %i, align 4
1328 br label %while31
1329
1330 then45:                                     ; preds = %while_body33
1331 %i46 = load i32, i32* %i, align 4
1332 store i32 %i46, i32* %remove_index, align 4
1333 br label %merge32
1334
1335 else47:                                     ; preds = %while_body33
1336 br label %merge44
1337
1338 merge68:                                     ; preds = %else108, %
    merge71
1339 br label %merge
1340
1341 then69:                                     ; preds = %merge32
1342 store i32 0, i32* %for_index, align 4
1343 store i32 0, i32* %index, align 4
1344 br label %while70
1345
1346 while70:                                     ; preds = %while_body72, %
    then69
1347 %for_index98 = load i32, i32* %for_index, align 4
1348 %remove_index99 = load i32, i32* %remove_index, align 4
1349 %tmp100 = add i32 %remove_index99, 1
1350 %len101 = load i32, i32* %len, align 4
1351 %mallocall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
1352 %head_ptr_ptr103 = bitcast i8* %mallocall102 to %list_item**
1353 store %list_item* null, %list_item** %head_ptr_ptr103, align 8
1354 %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
1355 %ilist105 = load %list_item*, %list_item** %range_list104, align 8
1356 %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
1357 %tmp107 = icmp slt i32 %for_index98, %length106
1358 br i1 %tmp107, label %while_body72, label %merge71
1359

```

```

1360 merge71:                                     ; preds = %while70
1361     br label %merge68
1362
1363 while_body72:                                   ; preds = %while70
1364     %remove_index73 = load i32, i32* %remove_index, align 4
1365     %tmp74 = add i32 %remove_index73, 1
1366     %len75 = load i32, i32* %len, align 4
1367     %alloca176 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
1368         *, i1** null, i32 1) to i32))
1369     %head_ptr_ptr = bitcast i8* %alloca176 to %list_item**
1370     store %list_item* null, %list_item** %head_ptr_ptr, align 8
1371     %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
1372         list_item** %head_ptr_ptr, i32 0)
1373     %i177 = load %list_item*, %list_item** %range_list, align 8
1374     %for_index78 = load i32, i32* %for_index, align 4
1375     %_result79 = call %list_item* @list_access(%list_item* %i177, i32 %
1376         for_index78)
1377     %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
1378         _result79, i32 0, i32 0
1379     %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
1380     %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
1381     %data83 = load i32, i32* %cast_data_ptr82, align 4
1382     store i32 %data83, i32* %index, align 4
1383     %for_index84 = load i32, i32* %for_index, align 4
1384     %tmp85 = add i32 %for_index84, 1
1385     store i32 %tmp85, i32* %for_index, align 4
1386     %retlist86 = load %list_item**, %list_item*** %retlist, align 8
1387     %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
1388     %l188 = load %list_item**, %list_item*** %l1, align 8
1389     %i189 = load %list_item*, %list_item** %l188, align 8
1390     %index90 = load i32, i32* %index, align 4
1391     %_result91 = call %list_item* @list_access(%list_item* %i189, i32 %
1392         index90)
1393     %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
1394         _result91, i32 0, i32 0
1395     %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
1396     %cast_data_ptr94 = bitcast i8* %data_ptr93 to i1*
1397     %data95 = load i1, i1* %cast_data_ptr94, align 1
1398     %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
1399     %list_ptr_ptr97 = call %list_item** @insert_bool(%list_item** %retlist86,
1400         i1 %data95, i32 %length96)
1401     store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
1402     br label %while70
1403
1404 else108:                                       ; preds = %merge32
1405     br label %merge68
1406 }
1407
1408 define %list_item** @remove_float_float_float_bool(%list_item** %l, double %
1409     elem, i1 %all) {
1410 entry:
1411     %index = alloca i32, align 4
1412     %for_index = alloca i32, align 4
1413     %remove_index = alloca i32, align 4
1414     %len = alloca i32, align 4
1415     %i = alloca i32, align 4
1416     %retlist = alloca %list_item**, align 8
1417     %l1 = alloca %list_item**, align 8
1418     store %list_item** %l, %list_item*** %l1, align 8

```

```

1411 %elem2 = alloca double, align 8
1412 store double %elem, double* %elem2, align 8
1413 %all3 = alloca i1, align 1
1414 store i1 %all, i1* %all3, align 1
1415 %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
1416     i1** null, i32 1) to i32))
1416 %list = bitcast i8* %malloccall to %list_item**
1417 store %list_item* null, %list_item** %list, align 8
1418 store %list_item** %list, %list_item*** %retlist, align 8
1419 store i32 0, i32* %i, align 4
1420 %l4 = load %list_item**, %list_item*** %l1, align 8
1421 %ilist = load %list_item*, %list_item** %l4, align 8
1422 %length = call i32 @list_length(%list_item* %ilist, i32 0)
1423 store i32 %length, i32* %len, align 4
1424 store i32 0, i32* %remove_index, align 4
1425 %all5 = load i1, i1* %all3, align 1
1426 br i1 %all5, label %then, label %else30
1427
1428 merge:                                ; preds = %merge68, %
    merge6
1429 %retlist109 = load %list_item**, %list_item*** %retlist, align 8
1430 ret %list_item** %retlist109
1431
1432 then:                                ; preds = %entry
    br label %while
1433
1434
1435 while:                                ; preds = %merge11, %
    then12, %then
1436 %i27 = load i32, i32* %i, align 4
1437 %len28 = load i32, i32* %len, align 4
1438 %tmp29 = icmp slt i32 %i27, %len28
1439 br i1 %tmp29, label %while_body, label %merge6
1440
1441 merge6:                                ; preds = %while
    br label %merge
1442
1443
1444 while_body:                            ; preds = %while
    %l7 = load %list_item**, %list_item*** %l1, align 8
1445 %ilist8 = load %list_item*, %list_item** %l7, align 8
1446 %i9 = load i32, i32* %i, align 4
1447 %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1448 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
1449     i32 0, i32 0
1450 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1451 %cast_data_ptr = bitcast i8* %data_ptr to double*
1452 %data = load double, double* %cast_data_ptr, align 8
1453 %elem10 = load double, double* %elem2, align 8
1454 %tmp = fcmp oeq double %data, %elem10
1455 br i1 %tmp, label %then12, label %else
1456
1457 merge11:                              ; preds = %else
    %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1458 %list_ptr = load %list_item*, %list_item** %retlist15, align 8
1459 %l16 = load %list_item**, %list_item*** %l1, align 8
1460 %ilist17 = load %list_item*, %list_item** %l16, align 8
1461 %i18 = load i32, i32* %i, align 4
1462 %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
1463 %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
1464     _result19, i32 0, i32 0

```

```

1465 %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
1466 %cast_data_ptr22 = bitcast i8* %data_ptr21 to double*
1467 %data23 = load double, double* %cast_data_ptr22, align 8
1468 %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1469 %list_ptr_ptr = call %list_item** @insert_float(%list_item** %retlist15,
    double %data23, i32 %length24)
1470 store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1471 %i25 = load i32, i32* %i, align 4
1472 %tmp26 = add i32 %i25, 1
1473 store i32 %tmp26, i32* %i, align 4
1474 br label %while
1475
1476 then12:                                     ; preds = %while_body
1477 %i13 = load i32, i32* %i, align 4
1478 %tmp14 = add i32 %i13, 1
1479 store i32 %tmp14, i32* %i, align 4
1480 br label %while
1481
1482 else:                                       ; preds = %while_body
1483 br label %merge11
1484
1485 else30:                                    ; preds = %entry
1486 br label %while31
1487
1488 while31:                                   ; preds = %merge44, %
    else30
1489 %i62 = load i32, i32* %i, align 4
1490 %len63 = load i32, i32* %len, align 4
1491 %tmp64 = icmp slt i32 %i62, %len63
1492 br i1 %tmp64, label %while_body33, label %merge32
1493
1494 merge32:                                   ; preds = %while31, %
    then45
1495 %i65 = load i32, i32* %i, align 4
1496 %len66 = load i32, i32* %len, align 4
1497 %tmp67 = icmp ne i32 %i65, %len66
1498 br i1 %tmp67, label %then69, label %else108
1499
1500 while_body33:                             ; preds = %while31
1501 %l34 = load %list_item**, %list_item*** %l1, align 8
1502 %i135 = load %list_item*, %list_item** %l34, align 8
1503 %i36 = load i32, i32* %i, align 4
1504 %_result37 = call %list_item* @list_access(%list_item* %i135, i32 %i36)
1505 %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
    _result37, i32 0, i32 0
1506 %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
1507 %cast_data_ptr40 = bitcast i8* %data_ptr39 to double*
1508 %data41 = load double, double* %cast_data_ptr40, align 8
1509 %elem42 = load double, double* %elem2, align 8
1510 %tmp43 = fcmp oeq double %data41, %elem42
1511 br i1 %tmp43, label %then45, label %else47
1512
1513 merge44:                                   ; preds = %else47
1514 %retlist48 = load %list_item**, %list_item*** %retlist, align 8
1515 %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
1516 %l50 = load %list_item**, %list_item*** %l1, align 8
1517 %i151 = load %list_item*, %list_item** %l50, align 8
1518 %i52 = load i32, i32* %i, align 4
1519 %_result53 = call %list_item* @list_access(%list_item* %i151, i32 %i52)

```



```

1520 %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
    _result53, i32 0, i32 0
1521 %data_ptr55 = load i8*, i8* %data_ptr_ptr54, align 8
1522 %cast_data_ptr56 = bitcast i8* %data_ptr55 to double*
1523 %data57 = load double, double* %cast_data_ptr56, align 8
1524 %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
1525 %list_ptr_ptr59 = call %list_item** @insert_float(%list_item** %retlist48,
    double %data57, i32 %length58)
1526 store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
1527 %i60 = load i32, i32* %i, align 4
1528 %tmp61 = add i32 %i60, 1
1529 store i32 %tmp61, i32* %i, align 4
1530 br label %while31
1531
1532 then45:                                     ; preds = %while_body33
1533 %i46 = load i32, i32* %i, align 4
1534 store i32 %i46, i32* %remove_index, align 4
1535 br label %merge32
1536
1537 else47:                                     ; preds = %while_body33
1538 br label %merge44
1539
1540 merge68:                                   ; preds = %else108, %
    merge71
1541 br label %merge
1542
1543 then69:                                     ; preds = %merge32
1544 store i32 0, i32* %for_index, align 4
1545 store i32 0, i32* %index, align 4
1546 br label %while70
1547
1548 while70:                                   ; preds = %while_body72, %
    then69
1549 %for_index98 = load i32, i32* %for_index, align 4
1550 %remove_index99 = load i32, i32* %remove_index, align 4
1551 %tmp100 = add i32 %remove_index99, 1
1552 %len101 = load i32, i32* %len, align 4
1553 %mallocall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
1554 %head_ptr_ptr103 = bitcast i8* %mallocall102 to %list_item**
1555 store %list_item* null, %list_item** %head_ptr_ptr103, align 8
1556 %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
1557 %ilist105 = load %list_item*, %list_item** %range_list104, align 8
1558 %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
1559 %tmp107 = icmp slt i32 %for_index98, %length106
1560 br i1 %tmp107, label %while_body72, label %merge71
1561
1562 merge71:                                   ; preds = %while70
1563 br label %merge68
1564
1565 while_body72:                             ; preds = %while70
1566 %remove_index73 = load i32, i32* %remove_index, align 4
1567 %tmp74 = add i32 %remove_index73, 1
1568 %len75 = load i32, i32* %len, align 4
1569 %mallocall176 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
1570 %head_ptr_ptr = bitcast i8* %mallocall176 to %list_item**
1571 store %list_item* null, %list_item** %head_ptr_ptr, align 8

```

```

1572 %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
1573 %ilist77 = load %list_item*, %list_item** %range_list, align 8
1574 %for_index78 = load i32, i32* %for_index, align 4
1575 %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
    for_index78)
1576 %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
1577 %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
1578 %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
1579 %data83 = load i32, i32* %cast_data_ptr82, align 4
1580 store i32 %data83, i32* %index, align 4
1581 %for_index84 = load i32, i32* %for_index, align 4
1582 %tmp85 = add i32 %for_index84, 1
1583 store i32 %tmp85, i32* %for_index, align 4
1584 %retlist86 = load %list_item**, %list_item*** %retlist, align 8
1585 %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
1586 %l18 = load %list_item**, %list_item*** %l1, align 8
1587 %ilist89 = load %list_item*, %list_item** %l18, align 8
1588 %index90 = load i32, i32* %index, align 4
1589 %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
1590 %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
1591 %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
1592 %cast_data_ptr94 = bitcast i8* %data_ptr93 to double*
1593 %data95 = load double, double* %cast_data_ptr94, align 8
1594 %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
1595 %list_ptr_ptr97 = call %list_item** @insert_float(%list_item** %retlist86,
    double %data95, i32 %length96)
1596 store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
1597 br label %while70
1598
1599 else108:                                ; preds = %merge32
1600     br label %merge68
1601 }
1602
1603 define %list_item** @remove_int_int_int_bool(%list_item** %l, i32 %elem, i1
    %all) {
1604 entry:
1605     %index = alloca i32, align 4
1606     %for_index = alloca i32, align 4
1607     %remove_index = alloca i32, align 4
1608     %len = alloca i32, align 4
1609     %i = alloca i32, align 4
1610     %retlist = alloca %list_item**, align 8
1611     %l1 = alloca %list_item**, align 8
1612     store %list_item** %l, %list_item*** %l1, align 8
1613     %elem2 = alloca i32, align 4
1614     store i32 %elem, i32* %elem2, align 4
1615     %all3 = alloca i1, align 1
1616     store i1 %all, i1* %all3, align 1
1617     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))
1618     %list = bitcast i8* %mallocall to %list_item**
1619     store %list_item** null, %list_item** %list, align 8
1620     store %list_item** %list, %list_item*** %retlist, align 8
1621     store i32 0, i32* %i, align 4
1622     %l4 = load %list_item**, %list_item*** %l1, align 8

```

```

1623 %ilist = load %list_item*, %list_item** %l4, align 8
1624 %length = call i32 @list_length(%list_item* %ilist, i32 0)
1625 store i32 %length, i32* %len, align 4
1626 store i32 0, i32* %remove_index, align 4
1627 %all5 = load i1, i1* %all3, align 1
1628 br i1 %all5, label %then, label %else30
1629
1630 merge:                                ; preds = %merge68, %
    merge6
1631 %retlist109 = load %list_item**, %list_item*** %retlist, align 8
1632 ret %list_item** %retlist109
1633
1634 then:                                ; preds = %entry
    br label %while
1635
1636
1637 while:                                ; preds = %merge11, %
    then12, %then
1638 %i27 = load i32, i32* %i, align 4
1639 %len28 = load i32, i32* %len, align 4
1640 %tmp29 = icmp slt i32 %i27, %len28
1641 br i1 %tmp29, label %while_body, label %merge6
1642
1643 merge6:                                ; preds = %while
    br label %merge
1644
1645
1646 while_body:                            ; preds = %while
    %l7 = load %list_item**, %list_item*** %l1, align 8
1647 %ilist8 = load %list_item*, %list_item** %l7, align 8
1648 %i9 = load i32, i32* %i, align 4
1649 %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1650 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
1651     i32 0, i32 0
1652 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1653 %cast_data_ptr = bitcast i8* %data_ptr to i32*
1654 %data = load i32, i32* %cast_data_ptr, align 4
1655 %elem10 = load i32, i32* %elem2, align 4
1656 %tmp = icmp eq i32 %data, %elem10
1657 br i1 %tmp, label %then12, label %else
1658
1659 merge11:                                ; preds = %else
    %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1660 %list_ptr = load %list_item*, %list_item** %retlist15, align 8
1661 %l16 = load %list_item**, %list_item*** %l1, align 8
1662 %ilist17 = load %list_item*, %list_item** %l16, align 8
1663 %i18 = load i32, i32* %i, align 4
1664 %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
1665 %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
1666 %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
1667 %cast_data_ptr22 = bitcast i8* %data_ptr21 to i32*
1668 %data23 = load i32, i32* %cast_data_ptr22, align 4
1669 %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1670 %list_ptr_ptr = call %list_item** @insert_int(%list_item** %retlist15, i32
    %data23, i32 %length24)
1671 store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1672 %i25 = load i32, i32* %i, align 4
1673 %tmp26 = add i32 %i25, 1
1674 store i32 %tmp26, i32* %i, align 4
1675 br label %while
1676

```

```

1677
1678 then12:                                     ; preds = %while_body
1679     %i13 = load i32, i32* %i, align 4
1680     %tmp14 = add i32 %i13, 1
1681     store i32 %tmp14, i32* %i, align 4
1682     br label %while
1683
1684 else:   ; preds = %while_body
1685     br label %merge11
1686
1687 else30:                                       ; preds = %entry
1688     br label %while31
1689
1690 while31:                                     ; preds = %merge44, %
1691     else30
1692     %i62 = load i32, i32* %i, align 4
1693     %len63 = load i32, i32* %len, align 4
1694     %tmp64 = icmp slt i32 %i62, %len63
1695     br i1 %tmp64, label %while_body33, label %merge32
1696
1697 merge32:                                     ; preds = %while31, %
1698     then45
1699     %i65 = load i32, i32* %i, align 4
1700     %len66 = load i32, i32* %len, align 4
1701     %tmp67 = icmp ne i32 %i65, %len66
1702     br i1 %tmp67, label %then69, label %else108
1703
1704 while_body33:                               ; preds = %while31
1705     %l34 = load %list_item**, %list_item*** %l1, align 8
1706     %i135 = load %list_item*, %list_item** %l34, align 8
1707     %i36 = load i32, i32* %i, align 4
1708     %_result37 = call @list_access(%list_item* %i135, i32 %i36)
1709     %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
1710         _result37, i32 0, i32 0
1711     %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
1712     %cast_data_ptr40 = bitcast i8* %data_ptr39 to i32*
1713     %data41 = load i32, i32* %cast_data_ptr40, align 4
1714     %elem42 = load i32, i32* %elem2, align 4
1715     %tmp43 = icmp eq i32 %data41, %elem42
1716     br i1 %tmp43, label %then45, label %else47
1717
1718 merge44:                                     ; preds = %else47
1719     %retlist48 = load %list_item**, %list_item*** %retlist, align 8
1720     %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
1721     %l50 = load %list_item**, %list_item*** %l1, align 8
1722     %i151 = load %list_item*, %list_item** %l50, align 8
1723     %i52 = load i32, i32* %i, align 4
1724     %_result53 = call @list_access(%list_item* %i151, i32 %i52)
1725     %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
1726         _result53, i32 0, i32 0
1727     %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
1728     %cast_data_ptr56 = bitcast i8* %data_ptr55 to i32*
1729     %data57 = load i32, i32* %cast_data_ptr56, align 4
1730     %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
1731     %list_ptr_ptr59 = call %list_item** @insert_int(%list_item** %retlist48,
1732         i32 %data57, i32 %length58)
1733     store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
1734     %i60 = load i32, i32* %i, align 4
1735     %tmp61 = add i32 %i60, 1

```

```

1731 store i32 %tmp61, i32* %i, align 4
1732 br label %while31
1733
1734 then45:                                     ; preds = %while_body33
1735   %i46 = load i32, i32* %i, align 4
1736   store i32 %i46, i32* %remove_index, align 4
1737   br label %merge32
1738
1739 else47:                                     ; preds = %while_body33
1740   br label %merge44
1741
1742 merge68:                                   ; preds = %else108, %
1743   merge71
1744   br label %merge
1745
1746 then69:                                     ; preds = %merge32
1747   store i32 0, i32* %for_index, align 4
1748   store i32 0, i32* %index, align 4
1749   br label %while70
1750
1751 while70:                                   ; preds = %while_body72, %
1752   then69
1753   %for_index98 = load i32, i32* %for_index, align 4
1754   %remove_index99 = load i32, i32* %remove_index, align 4
1755   %tmp100 = add i32 %remove_index99, 1
1756   %len101 = load i32, i32* %len, align 4
1757   %mallocall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
1758     i1*, i1** null, i32 1) to i32))
1759   %head_ptr_ptr103 = bitcast i8* %mallocall102 to %list_item**
1760   store %list_item* null, %list_item** %head_ptr_ptr103, align 8
1761   %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
1762     len101, %list_item** %head_ptr_ptr103, i32 0)
1763   %ilist105 = load %list_item*, %list_item** %range_list104, align 8
1764   %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
1765   %tmp107 = icmp slt i32 %for_index98, %length106
1766   br i1 %tmp107, label %while_body72, label %merge71
1767
1768 merge71:                                   ; preds = %while70
1769   br label %merge68
1770
1771 while_body72:                               ; preds = %while70
1772   %remove_index73 = load i32, i32* %remove_index, align 4
1773   %tmp74 = add i32 %remove_index73, 1
1774   %len75 = load i32, i32* %len, align 4
1775   %mallocall176 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
1776     *, i1** null, i32 1) to i32))
1777   %head_ptr_ptr = bitcast i8* %mallocall176 to %list_item**
1778   store %list_item* null, %list_item** %head_ptr_ptr, align 8
1779   %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
1780     list_item** %head_ptr_ptr, i32 0)
1781   %ilist77 = load %list_item*, %list_item** %range_list, align 8
1782   %for_index78 = load i32, i32* %for_index, align 4
1783   %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
1784     for_index78)
1785   %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
1786     _result79, i32 0, i32 0
1787   %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
1788   %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
1789   %data83 = load i32, i32* %cast_data_ptr82, align 4

```

```

1782 store i32 %data83, i32* %index, align 4
1783 %for_index84 = load i32, i32* %for_index, align 4
1784 %tmp85 = add i32 %for_index84, 1
1785 store i32 %tmp85, i32* %for_index, align 4
1786 %retlist86 = load %list_item**, %list_item*** %retlist, align 8
1787 %list_ptr87 = load %list_item**, %list_item** %retlist86, align 8
1788 %l88 = load %list_item**, %list_item*** %l1, align 8
1789 %ilist89 = load %list_item**, %list_item** %l88, align 8
1790 %index90 = load i32, i32* %index, align 4
1791 %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
1792 %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
1793 %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
1794 %cast_data_ptr94 = bitcast i8* %data_ptr93 to i32*
1795 %data95 = load i32, i32* %cast_data_ptr94, align 4
1796 %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
1797 %list_ptr_ptr97 = call %list_item** @insert_int(%list_item** %retlist86,
    i32 %data95, i32 %length96)
1798 store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
1799 br label %while70
1800
1801 else108:                                ; preds = %merge32
1802 br label %merge68
1803 }
1804
1805 define i1 @contains_float_float(%list_item** %l, %list_item** %items) {
1806 entry:
1807 %r = alloca i1, align 1
1808 %for_index42 = alloca i32, align 4
1809 %ref = alloca double, align 8
1810 %for_index7 = alloca i32, align 4
1811 %item = alloca double, align 8
1812 %for_index = alloca i32, align 4
1813 %res = alloca %list_item**, align 8
1814 %l1 = alloca %list_item**, align 8
1815 store %list_item** %l, %list_item*** %l1, align 8
1816 %items2 = alloca %list_item**, align 8
1817 store %list_item** %items, %list_item*** %items2, align 8
1818 %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
1819 %list = bitcast i8* %mallocall to %list_item**
1820 store %list_item* null, %list_item** %list, align 8
1821 store %list_item** %list, %list_item*** %res, align 8
1822 store i32 0, i32* %for_index, align 4
1823 store double 0.000000e+00, double* %item, align 8
1824 br label %while
1825
1826 while:                                ; preds = %merge9, %entry
1827 %for_index37 = load i32, i32* %for_index, align 4
1828 %items38 = load %list_item**, %list_item*** %items2, align 8
1829 %ilist39 = load %list_item**, %list_item** %items38, align 8
1830 %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
1831 %tmp41 = icmp slt i32 %for_index37, %length40
1832 br i1 %tmp41, label %while_body, label %merge
1833
1834 merge:                                ; preds = %while
1835 store i32 0, i32* %for_index42, align 4
1836 store i1 false, i1* %r, align 1

```

```

1837     br label %while43
1838
1839 while_body:                                     ; preds = %while
1840     %items3 = load %list_item**, %list_item*** %items2, align 8
1841     %ilist = load %list_item*, %list_item** %items3, align 8
1842     %for_index4 = load i32, i32* %for_index, align 4
1843     %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
        for_index4)
1844     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
        i32 0, i32 0
1845     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1846     %cast_data_ptr = bitcast i8* %data_ptr to double*
1847     %data = load double, double* %cast_data_ptr, align 8
1848     store double %data, double* %item, align 8
1849     %for_index5 = load i32, i32* %for_index, align 4
1850     %tmp = add i32 %for_index5, 1
1851     store i32 %tmp, i32* %for_index, align 4
1852     %res6 = load %list_item**, %list_item*** %res, align 8
1853     %list_ptr = load %list_item*, %list_item** %res6, align 8
1854     %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
1855     %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
        false, i32 %length)
1856     store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
1857     store i32 0, i32* %for_index7, align 4
1858     store double 0.000000e+00, double* %ref, align 8
1859     br label %while8
1860
1861 while8:   ; preds = %merge24, %
        while_body
1862     %for_index32 = load i32, i32* %for_index7, align 4
1863     %l33 = load %list_item**, %list_item*** %l1, align 8
1864     %ilist34 = load %list_item*, %list_item** %l33, align 8
1865     %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
1866     %tmp36 = icmp slt i32 %for_index32, %length35
1867     br i1 %tmp36, label %while_body10, label %merge9
1868
1869 merge9:                                       ; preds = %while8
1870     br label %while
1871
1872 while_body10:                                 ; preds = %while8
1873     %l11 = load %list_item**, %list_item*** %l1, align 8
1874     %ilist12 = load %list_item*, %list_item** %l11, align 8
1875     %for_index13 = load i32, i32* %for_index7, align 4
1876     %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
        for_index13)
1877     %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
        _result14, i32 0, i32 0
1878     %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
1879     %cast_data_ptr17 = bitcast i8* %data_ptr16 to double*
1880     %data18 = load double, double* %cast_data_ptr17, align 8
1881     store double %data18, double* %ref, align 8
1882     %for_index19 = load i32, i32* %for_index7, align 4
1883     %tmp20 = add i32 %for_index19, 1
1884     store i32 %tmp20, i32* %for_index7, align 4
1885     %ref21 = load double, double* %ref, align 8
1886     %item22 = load double, double* %item, align 8
1887     %tmp23 = fcmp oeq double %ref21, %item22
1888     br i1 %tmp23, label %then, label %else
1889

```

```

1890 merge24:                                     ; preds = %else, %then
1891     br label %while8
1892
1893 then:   ; preds = %while_body10
1894     %res25 = load %list_item**, %list_item*** %res, align 8
1895     %ilist26 = load %list_item**, %list_item** %res25, align 8
1896     %res27 = load %list_item**, %list_item*** %res, align 8
1897     %ilist28 = load %list_item**, %list_item** %res27, align 8
1898     %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
1899     %tmp30 = sub i32 %length29, 1
1900     %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
1901     %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
1902         0, i32 0
1903     %allocaall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
1904         i1* null, i32 1) to i32))
1905     %copy_ptr = bitcast i8* %allocaall31 to i1*
1906     store i1 true, i1* %copy_ptr, align 1
1907     %ccopy = bitcast i1* %copy_ptr to i8*
1908     store i8* %ccopy, i8** %data_ptpt, align 8
1909     br label %merge24
1910
1911 else:   ; preds = %while_body10
1912     br label %merge24
1913
1914 while43:                                     ; preds = %merge58, %merge
1915     %for_index61 = load i32, i32* %for_index42, align 4
1916     %res62 = load %list_item**, %list_item*** %res, align 8
1917     %ilist63 = load %list_item**, %list_item** %res62, align 8
1918     %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
1919     %tmp65 = icmp slt i32 %for_index61, %length64
1920     br i1 %tmp65, label %while_body45, label %merge44
1921
1922 merge44:                                     ; preds = %while43
1923     ret i1 true
1924
1925 while_body45:                               ; preds = %while43
1926     %res46 = load %list_item**, %list_item*** %res, align 8
1927     %ilist47 = load %list_item**, %list_item** %res46, align 8
1928     %for_index48 = load i32, i32* %for_index42, align 4
1929     %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
1930         for_index48)
1931     %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
1932         _result49, i32 0, i32 0
1933     %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
1934     %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
1935     %data53 = load i1, i1* %cast_data_ptr52, align 1
1936     store i1 %data53, i1* %r, align 1
1937     %for_index54 = load i32, i32* %for_index42, align 4
1938     %tmp55 = add i32 %for_index54, 1
1939     store i32 %tmp55, i32* %for_index42, align 4
1940     %r56 = load i1, i1* %r, align 1
1941     %tmp57 = xor i1 %r56, true
1942     br i1 %tmp57, label %then59, label %else60
1943
1944 merge58:                                     ; preds = %else60
1945     br label %while43
1946
1947 then59:                                     ; preds = %while_body45
1948     ret i1 false

```



```

1945
1946 else60:                                     ; preds = %while_body45
1947     br label %merge58
1948 }
1949
1950 define i1 @contains_bool_bool(%list_item** %l, %list_item** %items) {
1951 entry:
1952     %r = alloca i1, align 1
1953     %for_index42 = alloca i32, align 4
1954     %ref = alloca i1, align 1
1955     %for_index7 = alloca i32, align 4
1956     %item = alloca i1, align 1
1957     %for_index = alloca i32, align 4
1958     %res = alloca %list_item**, align 8
1959     %l1 = alloca %list_item**, align 8
1960     store %list_item** %l, %list_item*** %l1, align 8
1961     %items2 = alloca %list_item**, align 8
1962     store %list_item** %items, %list_item*** %items2, align 8
1963     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
1964                                     i1** null, i32 1) to i32))
1965     %list = bitcast i8* %mallocall to %list_item**
1966     store %list_item* null, %list_item** %list, align 8
1967     store %list_item** %list, %list_item*** %res, align 8
1968     store i32 0, i32* %for_index, align 4
1969     store i1 false, i1* %item, align 1
1970     br label %while
1971
1972 while:                                     ; preds = %merge9, %entry
1973     %for_index37 = load i32, i32* %for_index, align 4
1974     %items38 = load %list_item**, %list_item*** %items2, align 8
1975     %ilist39 = load %list_item*, %list_item** %items38, align 8
1976     %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
1977     %tmp41 = icmp slt i32 %for_index37, %length40
1978     br i1 %tmp41, label %while_body, label %merge
1979
1980 merge:                                     ; preds = %while
1981     store i32 0, i32* %for_index42, align 4
1982     store i1 false, i1* %r, align 1
1983     br label %while43
1984
1985 while_body:                               ; preds = %while
1986     %items3 = load %list_item**, %list_item*** %items2, align 8
1987     %ilist = load %list_item*, %list_item** %items3, align 8
1988     %for_index4 = load i32, i32* %for_index, align 4
1989     %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
1990                                     for_index4)
1991     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
1992                                     i32 0, i32 0
1993     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1994     %cast_data_ptr = bitcast i8* %data_ptr to i1*
1995     %data = load i1, i1* %cast_data_ptr, align 1
1996     store i1 %data, i1* %item, align 1
1997     %for_index5 = load i32, i32* %for_index, align 4
1998     %tmp = add i32 %for_index5, 1
1999     store i32 %tmp, i32* %for_index, align 4
2000     %res6 = load %list_item**, %list_item*** %res, align 8
2001     %list_ptr = load %list_item*, %list_item** %res6, align 8
2002     %length = call i32 @list_length(%list_item* %list_ptr, i32 0)

```

```

2000 %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
      false, i32 %length)
2001 store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
2002 store i32 0, i32* %for_index7, align 4
2003 store i1 false, i1* %ref, align 1
2004 br label %while8
2005
2006 while8:                                     ; preds = %merge24, %
      while_body
2007 %for_index32 = load i32, i32* %for_index7, align 4
2008 %l33 = load %list_item**, %list_item*** %l1, align 8
2009 %ilist34 = load %list_item*, %list_item** %l33, align 8
2010 %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
2011 %tmp36 = icmp slt i32 %for_index32, %length35
2012 br i1 %tmp36, label %while_body10, label %merge9
2013
2014 merge9:                                     ; preds = %while8
      br label %while
2015
2016
2017 while_body10:                             ; preds = %while8
      %l11 = load %list_item**, %list_item*** %l1, align 8
2018 %ilist12 = load %list_item*, %list_item** %l11, align 8
2019 %for_index13 = load i32, i32* %for_index7, align 4
2020 %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
      for_index13)
2021 %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
      _result14, i32 0, i32 0
2022 %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
2023 %cast_data_ptr17 = bitcast i8* %data_ptr16 to i1*
2024 %data18 = load i1, i1* %cast_data_ptr17, align 1
2025 store i1 %data18, i1* %ref, align 1
2026 %for_index19 = load i32, i32* %for_index7, align 4
2027 %tmp20 = add i32 %for_index19, 1
2028 store i32 %tmp20, i32* %for_index7, align 4
2029 %ref21 = load i1, i1* %ref, align 1
2030 %item22 = load i1, i1* %item, align 1
2031 %tmp23 = icmp eq i1 %ref21, %item22
2032 br i1 %tmp23, label %then, label %else
2033
2034
2035 merge24:                                     ; preds = %else, %then
      br label %while8
2036
2037
2038 then:                                       ; preds = %while_body10
      %res25 = load %list_item**, %list_item*** %res, align 8
2039 %ilist26 = load %list_item*, %list_item** %res25, align 8
2040 %res27 = load %list_item**, %list_item*** %res, align 8
2041 %ilist28 = load %list_item*, %list_item** %res27, align 8
2042 %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
2043 %tmp30 = sub i32 %length29, 1
2044 %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
2045 %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
      0, i32 0
2046 %mallocall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
      i1* null, i32 1) to i32))
2047 %copy_ptr = bitcast i8* %mallocall31 to i1*
2048 store i1 true, i1* %copy_ptr, align 1
2049 %ccopy = bitcast i1* %copy_ptr to i8*
2050 store i8* %ccopy, i8** %data_ptpt, align 8
2051 br label %merge24
2052

```

```

2053
2054 else:                                     ; preds = %while_body10
2055     br label %merge24
2056
2057 while43:                                   ; preds = %merge58, %merge
2058     %for_index61 = load i32, i32* %for_index42, align 4
2059     %res62 = load %list_item**, %list_item*** %res, align 8
2060     %ilist63 = load %list_item*, %list_item** %res62, align 8
2061     %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
2062     %tmp65 = icmp slt i32 %for_index61, %length64
2063     br i1 %tmp65, label %while_body45, label %merge44
2064
2065 merge44:                                   ; preds = %while43
2066     ret i1 true
2067
2068 while_body45:                             ; preds = %while43
2069     %res46 = load %list_item**, %list_item*** %res, align 8
2070     %ilist47 = load %list_item*, %list_item** %res46, align 8
2071     %for_index48 = load i32, i32* %for_index42, align 4
2072     %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
        for_index48)
2073     %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
        _result49, i32 0, i32 0
2074     %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
2075     %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
2076     %data53 = load i1, i1* %cast_data_ptr52, align 1
2077     store i1 %data53, i1* %r, align 1
2078     %for_index54 = load i32, i32* %for_index42, align 4
2079     %tmp55 = add i32 %for_index54, 1
2080     store i32 %tmp55, i32* %for_index42, align 4
2081     %r56 = load i1, i1* %r, align 1
2082     %tmp57 = xor i1 %r56, true
2083     br i1 %tmp57, label %then59, label %else60
2084
2085 merge58:                                   ; preds = %else60
2086     br label %while43
2087
2088 then59:                                   ; preds = %while_body45
2089     ret i1 false
2090
2091 else60:                                   ; preds = %while_body45
2092     br label %merge58
2093 }
2094
2095 define i1 @contains_char_char(%list_item** %l, %list_item** %items) {
2096 entry:
2097     %r = alloca i1, align 1
2098     %for_index41 = alloca i32, align 4
2099     %ref = alloca i8, align 1
2100     %for_index7 = alloca i32, align 4
2101     %item = alloca i8, align 1
2102     %for_index = alloca i32, align 4
2103     %res = alloca %list_item**, align 8
2104     %l1 = alloca %list_item**, align 8
2105     store %list_item** %l, %list_item*** %l1, align 8
2106     %items2 = alloca %list_item**, align 8
2107     store %list_item** %items, %list_item*** %items2, align 8
2108     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))

```

```

2109 %list = bitcast i8* %allocacll to %list_item**
2110 store %list_item* null, %list_item** %list, align 8
2111 store %list_item** %list, %list_item*** %res, align 8
2112 store i32 0, i32* %for_index, align 4
2113 store i8 0, i8* %item, align 1
2114 br label %while
2115
2116 while:                                     ; preds = %merge9, %entry
2117 %for_index36 = load i32, i32* %for_index, align 4
2118 %items37 = load %list_item**, %list_item*** %items2, align 8
2119 %ilist38 = load %list_item*, %list_item** %items37, align 8
2120 %length39 = call i32 @list_length(%list_item* %ilist38, i32 0)
2121 %tmp40 = icmp slt i32 %for_index36, %length39
2122 br i1 %tmp40, label %while_body, label %merge
2123
2124 merge:                                     ; preds = %while
2125 store i32 0, i32* %for_index41, align 4
2126 store i1 false, i1* %r, align 1
2127 br label %while42
2128
2129 while_body:                               ; preds = %while
2130 %items3 = load %list_item**, %list_item*** %items2, align 8
2131 %ilist = load %list_item*, %list_item** %items3, align 8
2132 %for_index4 = load i32, i32* %for_index, align 4
2133 %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
2134 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
2135 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2136 %data = load i8, i8* %data_ptr, align 1
2137 store i8 %data, i8* %item, align 1
2138 %for_index5 = load i32, i32* %for_index, align 4
2139 %tmp = add i32 %for_index5, 1
2140 store i32 %tmp, i32* %for_index, align 4
2141 %res6 = load %list_item**, %list_item*** %res, align 8
2142 %list_ptr = load %list_item*, %list_item** %res6, align 8
2143 %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2144 %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
2145 store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
2146 store i32 0, i32* %for_index7, align 4
2147 store i8 0, i8* %ref, align 1
2148 br label %while8
2149
2150 while8:                                   ; preds = %merge23, %
    while_body
2151 %for_index31 = load i32, i32* %for_index7, align 4
2152 %l32 = load %list_item**, %list_item*** %l1, align 8
2153 %ilist33 = load %list_item*, %list_item** %l32, align 8
2154 %length34 = call i32 @list_length(%list_item* %ilist33, i32 0)
2155 %tmp35 = icmp slt i32 %for_index31, %length34
2156 br i1 %tmp35, label %while_body10, label %merge9
2157
2158 merge9:                                   ; preds = %while8
2159 br label %while
2160
2161 while_body10:                             ; preds = %while8
2162 %l11 = load %list_item**, %list_item*** %l1, align 8
2163 %ilist12 = load %list_item*, %list_item** %l11, align 8

```

```

2164 %for_index13 = load i32, i32* %for_index7, align 4
2165 %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
2166 %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
2167 %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
2168 %data17 = load i8, i8* %data_ptr16, align 1
2169 store i8 %data17, i8* %ref, align 1
2170 %for_index18 = load i32, i32* %for_index7, align 4
2171 %tmp19 = add i32 %for_index18, 1
2172 store i32 %tmp19, i32* %for_index7, align 4
2173 %ref20 = load i8, i8* %ref, align 1
2174 %item21 = load i8, i8* %item, align 1
2175 %tmp22 = icmp eq i8 %ref20, %item21
2176 br i1 %tmp22, label %then, label %else
2177
2178 merge23:                                ; preds = %else, %then
2179     br label %while8
2180
2181 then:                                    ; preds = %while_body10
2182 %res24 = load %list_item**, %list_item*** %res, align 8
2183 %ilist25 = load %list_item*, %list_item** %res24, align 8
2184 %res26 = load %list_item**, %list_item*** %res, align 8
2185 %ilist27 = load %list_item*, %list_item** %res26, align 8
2186 %length28 = call i32 @list_length(%list_item* %ilist27, i32 0)
2187 %tmp29 = sub i32 %length28, 1
2188 %result = call %list_item* @list_access(%list_item* %ilist25, i32 %tmp29)
2189 %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
2190 %allocaall30 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
    i1* null, i32 1) to i32))
2191 %copy_ptr = bitcast i8* %allocaall30 to i1*
2192 store i1 true, i1* %copy_ptr, align 1
2193 %ccopy = bitcast i1* %copy_ptr to i8*
2194 store i8* %ccopy, i8** %data_ptpt, align 8
2195 br label %merge23
2196
2197 else:                                    ; preds = %while_body10
2198     br label %merge23
2199
2200 while42:                                ; preds = %merge56, %merge
2201 %for_index59 = load i32, i32* %for_index41, align 4
2202 %res60 = load %list_item**, %list_item*** %res, align 8
2203 %ilist61 = load %list_item*, %list_item** %res60, align 8
2204 %length62 = call i32 @list_length(%list_item* %ilist61, i32 0)
2205 %tmp63 = icmp slt i32 %for_index59, %length62
2206 br i1 %tmp63, label %while_body44, label %merge43
2207
2208 merge43:                                ; preds = %while42
2209     ret i1 true
2210
2211 while_body44:                            ; preds = %while42
2212 %res45 = load %list_item**, %list_item*** %res, align 8
2213 %ilist46 = load %list_item*, %list_item** %res45, align 8
2214 %for_index47 = load i32, i32* %for_index41, align 4
2215 %_result48 = call %list_item* @list_access(%list_item* %ilist46, i32 %
    for_index47)
2216 %data_ptr_ptr49 = getelementptr inbounds %list_item, %list_item* %
    _result48, i32 0, i32 0

```

```

2217 %data_ptr50 = load i8*, i8** %data_ptr_ptr49, align 8
2218 %cast_data_ptr = bitcast i8* %data_ptr50 to i1*
2219 %data51 = load i1, i1* %cast_data_ptr, align 1
2220 store i1 %data51, i1* %r, align 1
2221 %for_index52 = load i32, i32* %for_index41, align 4
2222 %tmp53 = add i32 %for_index52, 1
2223 store i32 %tmp53, i32* %for_index41, align 4
2224 %r54 = load i1, i1* %r, align 1
2225 %tmp55 = xor i1 %r54, true
2226 br i1 %tmp55, label %then57, label %else58
2227
2228 merge56:                                ; preds = %else58
2229   br label %while42
2230
2231 then57:                                  ; preds = %while_body44
2232   ret i1 false
2233
2234 else58:                                  ; preds = %while_body44
2235   br label %merge56
2236 }
2237
2238 define i1 @contains_string_string(%list_item** %l, %list_item** %items) {
2239 entry:
2240   %r = alloca i1, align 1
2241   %for_index41 = alloca i32, align 4
2242   %ref = alloca i8*, align 8
2243   %for_index7 = alloca i32, align 4
2244   %item = alloca i8*, align 8
2245   %for_index = alloca i32, align 4
2246   %res = alloca %list_item**, align 8
2247   %l1 = alloca %list_item**, align 8
2248   store %list_item** %l, %list_item*** %l1, align 8
2249   %items2 = alloca %list_item**, align 8
2250   store %list_item** %items, %list_item*** %items2, align 8
2251   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
2252     i1** null, i32 1) to i32))
2252   %list = bitcast i8* %malloccall to %list_item**
2253   store %list_item* null, %list_item** %list, align 8
2254   store %list_item** %list, %list_item*** %res, align 8
2255   store i32 0, i32* %for_index, align 4
2256   store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.7, i32 0,
2257     i32 0), i8** %item, align 8
2257   br label %while
2258
2259 while:                                    ; preds = %merge9, %entry
2260   %for_index36 = load i32, i32* %for_index, align 4
2261   %items37 = load %list_item**, %list_item*** %items2, align 8
2262   %ilist38 = load %list_item*, %list_item** %items37, align 8
2263   %length39 = call i32 @list_length(%list_item* %ilist38, i32 0)
2264   %tmp40 = icmp slt i32 %for_index36, %length39
2265   br i1 %tmp40, label %while_body, label %merge
2266
2267 merge:                                    ; preds = %while
2268   store i32 0, i32* %for_index41, align 4
2269   store i1 false, i1* %r, align 1
2270   br label %while42
2271
2272 while_body:                              ; preds = %while
2273   %items3 = load %list_item**, %list_item*** %items2, align 8

```

```

2274 %ilist = load %list_item*, %list_item** %items3, align 8
2275 %for_index4 = load i32, i32* %for_index, align 4
2276 %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
2277 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
2278 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2279 %cast_data_ptr = bitcast i8* %data_ptr to i8**
2280 %data = load i8*, i8** %cast_data_ptr, align 8
2281 store i8* %data, i8** %item, align 8
2282 %for_index5 = load i32, i32* %for_index, align 4
2283 %tmp = add i32 %for_index5, 1
2284 store i32 %tmp, i32* %for_index, align 4
2285 %res6 = load %list_item**, %list_item*** %res, align 8
2286 %list_ptr = load %list_item*, %list_item** %res6, align 8
2287 %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2288 %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
2289 store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
2290 store i32 0, i32* %for_index7, align 4
2291 store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.8, i32 0,
    i32 0), i8** %ref, align 8
2292 br label %while8
2293
2294 while8:                                ; preds = %merge23, %
    while_body
2295 %for_index31 = load i32, i32* %for_index7, align 4
2296 %l32 = load %list_item**, %list_item*** %l1, align 8
2297 %ilist33 = load %list_item*, %list_item** %l32, align 8
2298 %length34 = call i32 @list_length(%list_item* %ilist33, i32 0)
2299 %tmp35 = icmp slt i32 %for_index31, %length34
2300 br i1 %tmp35, label %while_body10, label %merge9
2301
2302 merge9:                                ; preds = %while8
2303 br label %while
2304
2305 while_body10:                          ; preds = %while8
2306 %l11 = load %list_item**, %list_item*** %l1, align 8
2307 %ilist12 = load %list_item*, %list_item** %l11, align 8
2308 %for_index13 = load i32, i32* %for_index7, align 4
2309 %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
2310 %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
2311 %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
2312 %cast_data_ptr17 = bitcast i8* %data_ptr16 to i8**
2313 %data18 = load i8*, i8** %cast_data_ptr17, align 8
2314 store i8* %data18, i8** %ref, align 8
2315 %for_index19 = load i32, i32* %for_index7, align 4
2316 %tmp20 = add i32 %for_index19, 1
2317 store i32 %tmp20, i32* %for_index7, align 4
2318 %ref21 = load i8*, i8** %ref, align 8
2319 %item22 = load i8*, i8** %item, align 8
2320 %strcmp_eq = call i1 @strcmp_function(i8* %ref21, i8* %item22)
2321 br i1 %strcmp_eq, label %then, label %else
2322
2323 merge23:                              ; preds = %else, %then
2324 br label %while8
2325

```

```

2326 then:                                     ; preds = %while_body10
2327   %res24 = load %list_item**, %list_item*** %res, align 8
2328   %ilist25 = load %list_item*, %list_item** %res24, align 8
2329   %res26 = load %list_item**, %list_item*** %res, align 8
2330   %ilist27 = load %list_item*, %list_item** %res26, align 8
2331   %length28 = call i32 @list_length(%list_item* %ilist27, i32 0)
2332   %tmp29 = sub i32 %length28, 1
2333   %result = call %list_item* @list_access(%list_item* %ilist25, i32 %tmp29)
2334   %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
2335   0, i32 0
2336   %malloccall30 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
2337   i1* null, i32 1) to i32))
2338   %copy_ptr = bitcast i8* %malloccall30 to i1*
2339   store i1 true, i1* %copy_ptr, align 1
2340   %ccopy = bitcast i1* %copy_ptr to i8*
2341   store i8* %ccopy, i8** %data_ptpt, align 8
2342   br label %merge23
2343
2344 else:                                       ; preds = %while_body10
2345   br label %merge23
2346
2347 while42:                                   ; preds = %merge57, %merge
2348   %for_index60 = load i32, i32* %for_index41, align 4
2349   %res61 = load %list_item**, %list_item*** %res, align 8
2350   %ilist62 = load %list_item*, %list_item** %res61, align 8
2351   %length63 = call i32 @list_length(%list_item* %ilist62, i32 0)
2352   %tmp64 = icmp slt i32 %for_index60, %length63
2353   br i1 %tmp64, label %while_body44, label %merge43
2354
2355 merge43:                                   ; preds = %while42
2356   ret i1 true
2357
2358 while_body44:                             ; preds = %while42
2359   %res45 = load %list_item**, %list_item*** %res, align 8
2360   %ilist46 = load %list_item*, %list_item** %res45, align 8
2361   %for_index47 = load i32, i32* %for_index41, align 4
2362   %_result48 = call %list_item* @list_access(%list_item* %ilist46, i32 %
2363   for_index47)
2364   %data_ptr_ptr49 = getelementptr inbounds %list_item, %list_item* %
2365   _result48, i32 0, i32 0
2366   %data_ptr50 = load i8*, i8** %data_ptr_ptr49, align 8
2367   %cast_data_ptr51 = bitcast i8* %data_ptr50 to i1*
2368   %data52 = load i1, i1* %cast_data_ptr51, align 1
2369   store i1 %data52, i1* %r, align 1
2370   %for_index53 = load i32, i32* %for_index41, align 4
2371   %tmp54 = add i32 %for_index53, 1
2372   store i32 %tmp54, i32* %for_index41, align 4
2373   %r55 = load i1, i1* %r, align 1
2374   %tmp56 = xor i1 %r55, true
2375   br i1 %tmp56, label %then58, label %else59
2376
2377 merge57:                                   ; preds = %else59
2378   br label %while42
2379
2380 then58:                                   ; preds = %while_body44
2381   ret i1 false
2382
2383 else59:                                   ; preds = %while_body44
2384   br label %merge57

```



```

2381 }
2382
2383 define i1 @contains_int_int(%list_item** %l, %list_item** %items) {
2384 entry:
2385     %r = alloca i1, align 1
2386     %for_index42 = alloca i32, align 4
2387     %ref = alloca i32, align 4
2388     %for_index7 = alloca i32, align 4
2389     %item = alloca i32, align 4
2390     %for_index = alloca i32, align 4
2391     %res = alloca %list_item**, align 8
2392     %l1 = alloca %list_item**, align 8
2393     store %list_item** %l, %list_item*** %l1, align 8
2394     %items2 = alloca %list_item**, align 8
2395     store %list_item** %items, %list_item*** %items2, align 8
2396     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
2397         i1** null, i32 1) to i32))
2397     %list = bitcast i8* %mallocall to %list_item**
2398     store %list_item* null, %list_item** %list, align 8
2399     store %list_item** %list, %list_item*** %res, align 8
2400     store i32 0, i32* %for_index, align 4
2401     store i32 0, i32* %item, align 4
2402     br label %while
2403
2404 while:                                     ; preds = %merge9, %entry
2405     %for_index37 = load i32, i32* %for_index, align 4
2406     %items38 = load %list_item**, %list_item*** %items2, align 8
2407     %ilist39 = load %list_item*, %list_item** %items38, align 8
2408     %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
2409     %tmp41 = icmp slt i32 %for_index37, %length40
2410     br i1 %tmp41, label %while_body, label %merge
2411
2412 merge:                                     ; preds = %while
2413     store i32 0, i32* %for_index42, align 4
2414     store i1 false, i1* %r, align 1
2415     br label %while43
2416
2417 while_body:                               ; preds = %while
2418     %items3 = load %list_item**, %list_item*** %items2, align 8
2419     %ilist = load %list_item*, %list_item** %items3, align 8
2420     %for_index4 = load i32, i32* %for_index, align 4
2421     %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
2422         for_index4)
2422     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
2423         i32 0, i32 0
2423     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2424     %cast_data_ptr = bitcast i8* %data_ptr to i32*
2425     %data = load i32, i32* %cast_data_ptr, align 4
2426     store i32 %data, i32* %item, align 4
2427     %for_index5 = load i32, i32* %for_index, align 4
2428     %tmp = add i32 %for_index5, 1
2429     store i32 %tmp, i32* %for_index, align 4
2430     %res6 = load %list_item**, %list_item*** %res, align 8
2431     %list_ptr = load %list_item*, %list_item** %res6, align 8
2432     %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2433     %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
2434         false, i32 %length)
2434     store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
2435     store i32 0, i32* %for_index7, align 4

```

```

2436     store i32 0, i32* %ref, align 4
2437     br label %while8
2438
2439 while8:                                     ; preds = %merge24, %
    while_body
2440     %for_index32 = load i32, i32* %for_index7, align 4
2441     %l33 = load %list_item**, %list_item*** %l1, align 8
2442     %ilist34 = load %list_item*, %list_item** %l33, align 8
2443     %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
2444     %tmp36 = icmp slt i32 %for_index32, %length35
2445     br i1 %tmp36, label %while_body10, label %merge9
2446
2447 merge9:                                     ; preds = %while8
    br label %while
2448
2449
2450 while_body10:                             ; preds = %while8
    %l11 = load %list_item**, %list_item*** %l1, align 8
2451     %ilist12 = load %list_item*, %list_item** %l11, align 8
2452     %for_index13 = load i32, i32* %for_index7, align 4
2453     %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
2454     %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
2455     %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
2456     %cast_data_ptr17 = bitcast i8* %data_ptr16 to i32*
2457     %data18 = load i32, i32* %cast_data_ptr17, align 4
2458     store i32 %data18, i32* %ref, align 4
2459     %for_index19 = load i32, i32* %for_index7, align 4
2460     %tmp20 = add i32 %for_index19, 1
2461     store i32 %tmp20, i32* %for_index7, align 4
2462     %ref21 = load i32, i32* %ref, align 4
2463     %item22 = load i32, i32* %item, align 4
2464     %tmp23 = icmp eq i32 %ref21, %item22
2465     br i1 %tmp23, label %then, label %else
2466
2467
2468 merge24:                                   ; preds = %else, %then
    br label %while8
2469
2470
2471 then:                                       ; preds = %while_body10
    %res25 = load %list_item**, %list_item*** %res, align 8
2472     %ilist26 = load %list_item*, %list_item** %res25, align 8
2473     %res27 = load %list_item**, %list_item*** %res, align 8
2474     %ilist28 = load %list_item*, %list_item** %res27, align 8
2475     %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
2476     %tmp30 = sub i32 %length29, 1
2477     %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
2478     %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
2479     %malloccall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
    i1* null, i32 1) to i32))
2480     %copy_ptr = bitcast i8* %malloccall31 to i1*
2481     store i1 true, i1* %copy_ptr, align 1
2482     %ccopy = bitcast i1* %copy_ptr to i8*
2483     store i8* %ccopy, i8** %data_ptpt, align 8
2484     br label %merge24
2485
2486
2487 else:                                       ; preds = %while_body10
    br label %merge24
2488
2489

```

```

2490 while43:                                     ; preds = %merge58, %merge
2491   %for_index61 = load i32, i32* %for_index42, align 4
2492   %res62 = load %list_item**, %list_item*** %res, align 8
2493   %ilist63 = load %list_item*, %list_item** %res62, align 8
2494   %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
2495   %tmp65 = icmp slt i32 %for_index61, %length64
2496   br i1 %tmp65, label %while_body45, label %merge44
2497
2498 merge44:                                     ; preds = %while43
2499   ret i1 true
2500
2501 while_body45:                               ; preds = %while43
2502   %res46 = load %list_item**, %list_item*** %res, align 8
2503   %ilist47 = load %list_item*, %list_item** %res46, align 8
2504   %for_index48 = load i32, i32* %for_index42, align 4
2505   %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
    for_index48)
2506   %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
    _result49, i32 0, i32 0
2507   %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
2508   %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
2509   %data53 = load i1, i1* %cast_data_ptr52, align 1
2510   store i1 %data53, i1* %r, align 1
2511   %for_index54 = load i32, i32* %for_index42, align 4
2512   %tmp55 = add i32 %for_index54, 1
2513   store i32 %tmp55, i32* %for_index42, align 4
2514   %r56 = load i1, i1* %r, align 1
2515   %tmp57 = xor i1 %r56, true
2516   br i1 %tmp57, label %then59, label %else60
2517
2518 merge58:                                     ; preds = %else60
2519   br label %while43
2520
2521 then59:                                     ; preds = %while_body45
2522   ret i1 false
2523
2524 else60:                                     ; preds = %while_body45
2525   br label %merge58
2526 }
2527
2528 define i8* @join_string_string(%list_item** %text_list, i8* %connector) {
2529 entry:
2530   %index = alloca i32, align 4
2531   %for_index = alloca i32, align 4
2532   %list_length = alloca i32, align 4
2533   %res = alloca i8*, align 8
2534   %text_list1 = alloca %list_item**, align 8
2535   store %list_item** %text_list, %list_item*** %text_list1, align 8
2536   %connector2 = alloca i8*, align 8
2537   store i8* %connector, i8** %connector2, align 8
2538   store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.9, i32 0,
    i32 0), i8** %res, align 8
2539   %text_list3 = load %list_item**, %list_item*** %text_list1, align 8
2540   %ilist = load %list_item*, %list_item** %text_list3, align 8
2541   %length = call i32 @list_length(%list_item* %ilist, i32 0)
2542   store i32 %length, i32* %list_length, align 4
2543   store i32 0, i32* %for_index, align 4
2544   store i32 0, i32* %index, align 4
2545   br label %while

```

```

2546
2547 while:                                ; preds = %while_body, %
    entry
2548   %for_index31 = load i32, i32* %for_index, align 4
2549   %list_length32 = load i32, i32* %list_length, align 4
2550   %tmp33 = sub i32 %list_length32, 1
2551   %alloca134 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
2552   %head_ptr_ptr35 = bitcast i8* %alloca134 to %list_item**
2553   store %list_item* null, %list_item** %head_ptr_ptr35, align 8
2554   %range_list36 = call %list_item** @range_function(i32 0, i32 %tmp33, %
    list_item** %head_ptr_ptr35, i32 0)
2555   %i137 = load %list_item*, %list_item** %range_list36, align 8
2556   %length38 = call i32 @list_length(%list_item* %i137, i32 0)
2557   %tmp39 = icmp slt i32 %for_index31, %length38
2558   br i1 %tmp39, label %while_body, label %merge
2559
2560 merge:                                ; preds = %while
2561   %res40 = load i8*, i8** %res, align 8
2562   %text_list41 = load %list_item**, %list_item** %text_list1, align 8
2563   %i142 = load %list_item*, %list_item** %text_list41, align 8
2564   %list_length43 = load i32, i32* %list_length, align 4
2565   %tmp44 = sub i32 %list_length43, 1
2566   %_result45 = call %list_item* @list_access(%list_item* %i142, i32 %
    tmp44)
2567   %data_ptr_ptr46 = getelementptr inbounds %list_item, %list_item* %
    _result45, i32 0, i32 0
2568   %data_ptr47 = load i8*, i8** %data_ptr_ptr46, align 8
2569   %cast_data_ptr48 = bitcast i8* %data_ptr47 to i8**
2570   %data49 = load i8*, i8** %cast_data_ptr48, align 8
2571   %length50 = call i32 @string_length(i8* %res40, i32 0)
2572   %length51 = call i32 @string_length(i8* %data49, i32 0)
2573   %new_length52 = add i32 %length50, %length51
2574   %new_length_nul53 = add i32 %new_length52, 1
2575   %mallocsize54 = mul i32 %new_length_nul53, ptrtoint (i8* getelementptr (i8
    , i8* null, i32 1) to i32)
2576   %new_string56 = tail call i8* @malloc(i32 %mallocsize54)
2577   call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string56, i8* %res40, i32 %
    length50, i1 true)
2578   %new_spot57 = getelementptr i8, i8* %new_string56, i32 %length50
2579   call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot57, i8* %data49, i32 %
    length51, i1 true)
2580   %string_term58 = getelementptr i8, i8* %new_string56, i32 %new_length52
2581   store i8 0, i8* %string_term58, align 1
2582   store i8* %new_string56, i8** %res, align 8
2583   %res59 = load i8*, i8** %res, align 8
2584   ret i8* %res59
2585
2586 while_body:                            ; preds = %while
2587   %list_length4 = load i32, i32* %list_length, align 4
2588   %tmp = sub i32 %list_length4, 1
2589   %alloca11 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
2590   %head_ptr_ptr = bitcast i8* %alloca11 to %list_item**
2591   store %list_item* null, %list_item** %head_ptr_ptr, align 8
2592   %range_list = call %list_item** @range_function(i32 0, i32 %tmp, %
    list_item** %head_ptr_ptr, i32 0)
2593   %i15 = load %list_item*, %list_item** %range_list, align 8
2594   %for_index6 = load i32, i32* %for_index, align 4

```

```

2595 %_result = call %list_item* @list_access(%list_item* %ilist5, i32 %
      for_index6)
2596 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
      i32 0, i32 0
2597 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2598 %cast_data_ptr = bitcast i8* %data_ptr to i32*
2599 %data = load i32, i32* %cast_data_ptr, align 4
2600 store i32 %data, i32* %index, align 4
2601 %for_index7 = load i32, i32* %for_index, align 4
2602 %tmp8 = add i32 %for_index7, 1
2603 store i32 %tmp8, i32* %for_index, align 4
2604 %res9 = load i8*, i8** %res, align 8
2605 %text_list10 = load %list_item**, %list_item*** %text_list1, align 8
2606 %ilist11 = load %list_item*, %list_item** %text_list10, align 8
2607 %index12 = load i32, i32* %index, align 4
2608 %_result13 = call %list_item* @list_access(%list_item* %ilist11, i32 %
      index12)
2609 %data_ptr_ptr14 = getelementptr inbounds %list_item, %list_item* %
      _result13, i32 0, i32 0
2610 %data_ptr15 = load i8*, i8** %data_ptr_ptr14, align 8
2611 %cast_data_ptr16 = bitcast i8* %data_ptr15 to i8**
2612 %data17 = load i8*, i8** %cast_data_ptr16, align 8
2613 %length18 = call i32 @string_length(i8* %res9, i32 0)
2614 %length19 = call i32 @string_length(i8* %data17, i32 0)
2615 %new_length = add i32 %length18, %length19
2616 %new_length_nul = add i32 %new_length, 1
2617 %mallocsize = mul i32 %new_length_nul, ptrtoint (i8* getelementptr (i8, i8
      * null, i32 1) to i32)
2618 %new_string = tail call i8* @malloc(i32 %mallocsize)
2619 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %res9, i32 %
      length18, i1 true)
2620 %new_spot = getelementptr i8, i8* %new_string, i32 %length18
2621 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot, i8* %data17, i32 %
      length19, i1 true)
2622 %string_term = getelementptr i8, i8* %new_string, i32 %new_length
2623 store i8 0, i8* %string_term, align 1
2624 %connector21 = load i8*, i8** %connector2, align 8
2625 %length22 = call i32 @string_length(i8* %new_string, i32 0)
2626 %length23 = call i32 @string_length(i8* %connector21, i32 0)
2627 %new_length24 = add i32 %length22, %length23
2628 %new_length_nul25 = add i32 %new_length24, 1
2629 %mallocsize26 = mul i32 %new_length_nul25, ptrtoint (i8* getelementptr (i8
      , i8* null, i32 1) to i32)
2630 %new_string28 = tail call i8* @malloc(i32 %mallocsize26)
2631 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string28, i8* %new_string,
      i32 %length22, i1 true)
2632 %new_spot29 = getelementptr i8, i8* %new_string28, i32 %length22
2633 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot29, i8* %connector21,
      i32 %length23, i1 true)
2634 %string_term30 = getelementptr i8, i8* %new_string28, i32 %new_length24
2635 store i8 0, i8* %string_term30, align 1
2636 store i8* %new_string28, i8** %res, align 8
2637 br label %while
2638 }
2639
2640 define %list_item** @split(i8* %text, i8 %separator) {
2641 entry:
2642   %right = alloca i32, align 4
2643   %left = alloca i32, align 4

```

```

2644 %text_length = alloca i32, align 4
2645 %result = alloca %list_item**, align 8
2646 %text1 = alloca i8*, align 8
2647 store i8* %text, i8** %text1, align 8
2648 %separator2 = alloca i8, align 1
2649 store i8 %separator, i8* %separator2, align 1
2650 %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
2651 %list = bitcast i8* %malloccall to %list_item**
2652 store %list_item* null, %list_item** %list, align 8
2653 store %list_item** %list, %list_item*** %result, align 8
2654 %text3 = load i8*, i8** %text1, align 8
2655 %length = call i32 @string_length(i8* %text3, i32 0)
2656 store i32 %length, i32* %text_length, align 4
2657 store i32 0, i32* %left, align 4
2658 store i32 0, i32* %right, align 4
2659 br label %while
2660
2661 while:                                     ; preds = %merge7, %entry
2662 %right21 = load i32, i32* %right, align 4
2663 %text_length22 = load i32, i32* %text_length, align 4
2664 %tmp23 = icmp slt i32 %right21, %text_length22
2665 br i1 %tmp23, label %while_body, label %merge
2666
2667 merge:                                     ; preds = %while
2668 %result24 = load %list_item**, %list_item*** %result, align 8
2669 %list_ptr25 = load %list_item*, %list_item** %result24, align 8
2670 %text26 = load i8*, i8** %text1, align 8
2671 %left27 = load i32, i32* %left, align 4
2672 %get_char_ptr28 = getelementptr i8, i8* %text26, i32 %left27
2673 %left29 = load i32, i32* %left, align 4
2674 %right30 = load i32, i32* %right, align 4
2675 %subb31 = sub i32 %right30, %left29
2676 %length_w_nul32 = add i32 %subb31, 1
2677 %mallocsize33 = mul i32 %length_w_nul32, ptrtoint (i8* getelementptr (i8,
    i8* null, i32 1) to i32)
2678 %new_string35 = tail call i8* @malloc(i32 %mallocsize33)
2679 %string_term36 = getelementptr i8, i8* %new_string35, i32 %subb31
2680 store i8 0, i8* %string_term36, align 1
2681 call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string35, i8* %
    get_char_ptr28, i32 %subb31, i1 true)
2682 %length37 = call i32 @list_length(%list_item* %list_ptr25, i32 0)
2683 %list_ptr_ptr38 = call %list_item** @insert_string(%list_item** %result24,
    i8* %new_string35, i32 %length37)
2684 store %list_item** %list_ptr_ptr38, %list_item*** %result, align 8
2685 %result39 = load %list_item**, %list_item*** %result, align 8
2686 ret %list_item** %result39
2687
2688 while_body:                               ; preds = %while
2689 %text4 = load i8*, i8** %text1, align 8
2690 %right5 = load i32, i32* %right, align 4
2691 %get_char_ptr = getelementptr i8, i8* %text4, i32 %right5
2692 %get_char = load i8, i8* %get_char_ptr, align 1
2693 %separator6 = load i8, i8* %separator2, align 1
2694 %tmp = icmp eq i8 %get_char, %separator6
2695 br i1 %tmp, label %then, label %else
2696
2697 merge7:                                   ; preds = %else, %then
2698 br label %while

```

```

2699
2700 then:                                     ; preds = %while_body
2701   %result8 = load %list_item**, %list_item** %result, align 8
2702   %list_ptr = load %list_item*, %list_item** %result8, align 8
2703   %text9 = load i8*, i8** %text1, align 8
2704   %left10 = load i32, i32* %left, align 4
2705   %get_char_ptr11 = getelementptr i8, i8* %text9, i32 %left10
2706   %left12 = load i32, i32* %left, align 4
2707   %right13 = load i32, i32* %right, align 4
2708   %subb = sub i32 %right13, %left12
2709   %length_w_nul = add i32 %subb, 1
2710   %mallocsize = mul i32 %length_w_nul, ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32)
2711   %new_string = tail call i8* @malloc(i32 %mallocsize)
2712   %string_term = getelementptr i8, i8* %new_string, i32 %subb
2713   store i8 0, i8* %string_term, align 1
2714   call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %get_char_ptr11,
    i32 %subb, i1 true)
2715   %length15 = call i32 @list_length(%list_item* %list_ptr, i32 0)
2716   %list_ptr_ptr = call %list_item** @insert_string(%list_item** %result8, i8
    * %new_string, i32 %length15)
2717   store %list_item** %list_ptr_ptr, %list_item** %result, align 8
2718   %right16 = load i32, i32* %right, align 4
2719   %tmp17 = add i32 %right16, 1
2720   store i32 %tmp17, i32* %right, align 4
2721   %right18 = load i32, i32* %right, align 4
2722   store i32 %right18, i32* %left, align 4
2723   br label %merge7
2724
2725 else:                                     ; preds = %while_body
2726   %right19 = load i32, i32* %right, align 4
2727   %tmp20 = add i32 %right19, 1
2728   store i32 %tmp20, i32* %right, align 4
2729   br label %merge7
2730 }
2731
2732 define i8* @string_reverse(i8* %text) {
2733 entry:
2734   %index = alloca i32, align 4
2735   %string_length = alloca i32, align 4
2736   %res = alloca i8*, align 8
2737   %text1 = alloca i8*, align 8
2738   store i8* %text, i8** %text1, align 8
2739   store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.10, i32 0,
    i32 0), i8** %res, align 8
2740   %text2 = load i8*, i8** %text1, align 8
2741   %length = call i32 @string_length(i8* %text2, i32 0)
2742   store i32 %length, i32* %string_length, align 4
2743   %string_length3 = load i32, i32* %string_length, align 4
2744   %tmp = sub i32 %string_length3, 1
2745   store i32 %tmp, i32* %index, align 4
2746   br label %while
2747
2748 while:                                     ; preds = %while_body, %
    entry
2749   %index10 = load i32, i32* %index, align 4
2750   %tmp11 = icmp sge i32 %index10, 0
2751   br i1 %tmp11, label %while_body, label %merge
2752

```

```

2753 merge:                                     ; preds = %while
2754   %res12 = load i8*, i8** %res, align 8
2755   ret i8* %res12
2756
2757 while_body:                                 ; preds = %while
2758   %res4 = load i8*, i8** %res, align 8
2759   %text5 = load i8*, i8** %text1, align 8
2760   %index6 = load i32, i32* %index, align 4
2761   %get_char_ptr = getelementptr i8, i8* %text5, i32 %index6
2762   %get_char = load i8, i8* %get_char_ptr, align 1
2763   %length7 = call i32 @string_length(i8* %res4, i32 0)
2764   %new_length = add i32 %length7, 1
2765   %new_length_nul = add i32 %new_length, 1
2766   %mallocsize = mul i32 %new_length_nul, ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32)
2767   %new_string = tail call i8* @malloc(i32 %mallocsize)
2768   call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %res4, i32 %
    length7, i1 true)
2769   %new_spot = getelementptr i8, i8* %new_string, i32 %length7
2770   store i8 %get_char, i8* %new_spot, align 1
2771   %string_term = getelementptr i8, i8* %new_string, i32 %new_length
2772   store i8 0, i8* %string_term, align 1
2773   store i8* %new_string, i8** %res, align 8
2774   %index8 = load i32, i32* %index, align 4
2775   %tmp9 = sub i32 %index8, 1
2776   store i32 %tmp9, i32* %index, align 4
2777   br label %while
2778 }
2779
2780 define i1 @startswith(i8* %text, i8 %s) {
2781 entry:
2782   %text1 = alloca i8*, align 8
2783   store i8* %text, i8** %text1, align 8
2784   %s2 = alloca i8, align 1
2785   store i8 %s, i8* %s2, align 1
2786   %s3 = load i8, i8* %s2, align 1
2787   %text4 = load i8*, i8** %text1, align 8
2788   %get_char_ptr = getelementptr i8, i8* %text4, i32 0
2789   %get_char = load i8, i8* %get_char_ptr, align 1
2790   %tmp = icmp eq i8 %s3, %get_char
2791   ret i1 %tmp
2792 }
2793
2794 define i1 @endswith(i8* %text, i8 %e) {
2795 entry:
2796   %string_length = alloca i32, align 4
2797   %text1 = alloca i8*, align 8
2798   store i8* %text, i8** %text1, align 8
2799   %e2 = alloca i8, align 1
2800   store i8 %e, i8* %e2, align 1
2801   %text3 = load i8*, i8** %text1, align 8
2802   %length = call i32 @string_length(i8* %text3, i32 0)
2803   store i32 %length, i32* %string_length, align 4
2804   %e4 = load i8, i8* %e2, align 1
2805   %text5 = load i8*, i8** %text1, align 8
2806   %string_length6 = load i32, i32* %string_length, align 4
2807   %tmp = sub i32 %string_length6, 1
2808   %get_char_ptr = getelementptr i8, i8* %text5, i32 %tmp
2809   %get_char = load i8, i8* %get_char_ptr, align 1

```



```

2810 %tmp7 = icmp eq i8 %e4, %get_char
2811 ret i1 %tmp7
2812 }
2813
2814 define %list_item** @string_to_list(i8* %text) {
2815 entry:
2816 %c = alloca i8, align 1
2817 %for_index = alloca i32, align 4
2818 %result = alloca %list_item**, align 8
2819 %text1 = alloca i8*, align 8
2820 store i8* %text, i8** %text1, align 8
2821 %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
2822 %list = bitcast i8* %mallocall to %list_item**
2823 store %list_item* null, %list_item** %list, align 8
2824 store %list_item** %list, %list_item*** %result, align 8
2825 store i32 0, i32* %for_index, align 4
2826 store i8 0, i8* %c, align 1
2827 br label %while
2828
2829 while:                                     ; preds = %while_body, %
    entry
2830 %for_index7 = load i32, i32* %for_index, align 4
2831 %text8 = load i8*, i8** %text1, align 8
2832 %length9 = call i32 @string_length(i8* %text8, i32 0)
2833 %tmp10 = icmp slt i32 %for_index7, %length9
2834 br i1 %tmp10, label %while_body, label %merge
2835
2836 merge:                                     ; preds = %while
    %result11 = load %list_item**, %list_item*** %result, align 8
2837 ret %list_item** %result11
2838
2839 while_body:                               ; preds = %while
2840 %text2 = load i8*, i8** %text1, align 8
2841 %for_index3 = load i32, i32* %for_index, align 4
2842 %get_char_ptr = getelementptr i8, i8* %text2, i32 %for_index3
2843 %get_char = load i8, i8* %get_char_ptr, align 1
2844 store i8 %get_char, i8* %c, align 1
2845 %for_index4 = load i32, i32* %for_index, align 4
2846 %tmp = add i32 %for_index4, 1
2847 store i32 %tmp, i32* %for_index, align 4
2848 %result5 = load %list_item**, %list_item*** %result, align 8
2849 %list_ptr = load %list_item*, %list_item** %result5, align 8
2850 %c6 = load i8, i8* %c, align 1
2851 %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2852 %list_ptr_ptr = call %list_item** @insert_char(%list_item** %result5, i8 %
    c6, i32 %length)
2853 store %list_item** %list_ptr_ptr, %list_item*** %result, align 8
2854 br label %while
2855 }
2856
2857
2858 define i8 @lower(i8 %c) {
2859 entry:
2860 %c_ref = alloca i8, align 1
2861 %for_index = alloca i32, align 4
2862 %index = alloca i32, align 4
2863 %c1 = alloca i8, align 1
2864 store i8 %c, i8* %c1, align 1
2865 store i32 -1, i32* %index, align 4

```

```

2866 store i32 0, i32* %for_index, align 4
2867 store i8 0, i8* %c_ref, align 1
2868 br label %while
2869
2870 while:                                     ; preds = %merge9, %entry
2871 %for_index24 = load i32, i32* %for_index, align 4
2872 %ASCII25 = load %list_item**, %list_item*** @ASCII, align 8
2873 %ilist26 = load %list_item*, %list_item** %ASCII25, align 8
2874 %length = call i32 @list_length(%list_item* %ilist26, i32 0)
2875 %tmp27 = icmp slt i32 %for_index24, %length
2876 br i1 %tmp27, label %while_body, label %merge
2877
2878 merge:                                     ; preds = %while
2879 %c28 = load i8, i8* %c1, align 1
2880 ret i8 %c28
2881
2882 while_body:                               ; preds = %while
2883 %ASCII = load %list_item**, %list_item*** @ASCII, align 8
2884 %ilist = load %list_item*, %list_item** %ASCII, align 8
2885 %for_index2 = load i32, i32* %for_index, align 4
2886 %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index2)
2887 %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
2888 %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2889 %data = load i8, i8* %data_ptr, align 1
2890 store i8 %data, i8* %c_ref, align 1
2891 %for_index3 = load i32, i32* %for_index, align 4
2892 %tmp = add i32 %for_index3, 1
2893 store i32 %tmp, i32* %for_index, align 4
2894 %index4 = load i32, i32* %index, align 4
2895 %tmp5 = add i32 %index4, 1
2896 store i32 %tmp5, i32* %index, align 4
2897 %c6 = load i8, i8* %c1, align 1
2898 %c_ref7 = load i8, i8* %c_ref, align 1
2899 %tmp8 = icmp eq i8 %c6, %c_ref7
2900 br i1 %tmp8, label %then, label %else23
2901
2902 merge9:                                   ; preds = %else23, %
    merge12
2903 br label %while
2904
2905 then:                                     ; preds = %while_body
2906 %index10 = load i32, i32* %index, align 4
2907 %tmp11 = icmp slt i32 %index10, 26
2908 br i1 %tmp11, label %then13, label %else
2909
2910 merge12:                                 ; No predecessors!
2911 br label %merge9
2912
2913 then13:                                   ; preds = %then
2914 %c14 = load i8, i8* %c1, align 1
2915 ret i8 %c14
2916
2917 else:                                     ; preds = %then
2918 %ASCII15 = load %list_item**, %list_item*** @ASCII, align 8
2919 %ilist16 = load %list_item*, %list_item** %ASCII15, align 8
2920 %index17 = load i32, i32* %index, align 4
2921 %tmp18 = sub i32 %index17, 26

```

```

2922  %_result19 = call %list_item* @list_access(%list_item* %ilist16, i32 %
      tmp18)
2923  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
      _result19, i32 0, i32 0
2924  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
2925  %data22 = load i8, i8* %data_ptr21, align 1
2926  ret i8 %data22
2927
2928 else23:                                     ; preds = %while_body
2929     br label %merge9
2930 }
2931
2932 define i8 @upper(i8 %c) {
2933 entry:
2934     %c_ref = alloca i8, align 1
2935     %for_index = alloca i32, align 4
2936     %index = alloca i32, align 4
2937     %c1 = alloca i8, align 1
2938     store i8 %c, i8* %c1, align 1
2939     store i32 -1, i32* %index, align 4
2940     store i32 0, i32* %for_index, align 4
2941     store i8 0, i8* %c_ref, align 1
2942     br label %while
2943
2944 while:                                     ; preds = %merge9, %entry
2945     %for_index24 = load i32, i32* %for_index, align 4
2946     %ASCII25 = load %list_item**, %list_item*** @ASCII, align 8
2947     %ilist26 = load %list_item*, %list_item** %ASCII25, align 8
2948     %length = call i32 @list_length(%list_item* %ilist26, i32 0)
2949     %tmp27 = icmp slt i32 %for_index24, %length
2950     br i1 %tmp27, label %while_body, label %merge
2951
2952 merge:                                     ; preds = %while
2953     %c28 = load i8, i8* %c1, align 1
2954     ret i8 %c28
2955
2956 while_body:                               ; preds = %while
2957     %ASCII = load %list_item**, %list_item*** @ASCII, align 8
2958     %ilist = load %list_item*, %list_item** %ASCII, align 8
2959     %for_index2 = load i32, i32* %for_index, align 4
2960     %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
      for_index2)
2961     %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
      i32 0, i32 0
2962     %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2963     %data = load i8, i8* %data_ptr, align 1
2964     store i8 %data, i8* %c_ref, align 1
2965     %for_index3 = load i32, i32* %for_index, align 4
2966     %tmp = add i32 %for_index3, 1
2967     store i32 %tmp, i32* %for_index, align 4
2968     %index4 = load i32, i32* %index, align 4
2969     %tmp5 = add i32 %index4, 1
2970     store i32 %tmp5, i32* %index, align 4
2971     %c6 = load i8, i8* %c1, align 1
2972     %c_ref7 = load i8, i8* %c_ref, align 1
2973     %tmp8 = icmp eq i8 %c6, %c_ref7
2974     br i1 %tmp8, label %then, label %else23
2975

```

```

2976 merge9:                                     ; preds = %else23, %
      merge12
2977   br label %while
2978
2979 then:   ; preds = %while_body
2980   %index10 = load i32, i32* %index, align 4
2981   %tmp11 = icmp sgt i32 %index10, 25
2982   br i1 %tmp11, label %then13, label %else
2983
2984 merge12:                                     ; No predecessors!
2985   br label %merge9
2986
2987 then13:                                     ; preds = %then
2988   %c14 = load i8, i8* %c1, align 1
2989   ret i8 %c14
2990
2991 else:   ; preds = %then
2992   %ASCI15 = load %list_item*, %list_item** @ASCI1, align 8
2993   %i16 = load %list_item*, %list_item** %ASCI15, align 8
2994   %index17 = load i32, i32* %index, align 4
2995   %tmp18 = add i32 %index17, 26
2996   %_result19 = call %list_item* @list_access(%list_item* %i16, i32 %
      tmp18)
2997   %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
      _result19, i32 0, i32 0
2998   %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
2999   %data22 = load i8, i8* %data_ptr21, align 1
3000   ret i8 %data22
3001
3002 else23:                                     ; preds = %while_body
3003   br label %merge9
3004 }
3005
3006 define i1 @strcmp(i8* %str1, i8* %str2) {
3007 entry:
3008   %c2 = alloca i8, align 1
3009   %c1 = alloca i8, align 1
3010   %i = alloca i32, align 4
3011   %str11 = alloca i8*, align 8
3012   store i8* %str1, i8** %str11, align 8
3013   %str22 = alloca i8*, align 8
3014   store i8* %str2, i8** %str22, align 8
3015   %str13 = load i8*, i8** %str11, align 8
3016   %length = call i32 @string_length(i8* %str13, i32 0)
3017   %str24 = load i8*, i8** %str22, align 8
3018   %length5 = call i32 @string_length(i8* %str24, i32 0)
3019   %tmp = icmp ne i32 %length, %length5
3020   br i1 %tmp, label %then, label %else
3021
3022 merge:                                       ; preds = %else
3023   store i32 0, i32* %i, align 4
3024   br label %while
3025
3026 then:                                       ; preds = %entry
3027   ret i1 false
3028
3029 else:                                       ; preds = %entry
3030   br label %merge
3031

```

```

3032 while:                                     ; preds = %merge16, %merge
3033     %i21 = load i32, i32* %i, align 4
3034     %str122 = load i8*, i8** %str11, align 8
3035     %length23 = call i32 @string_length(i8* %str122, i32 0)
3036     %tmp24 = icmp slt i32 %i21, %length23
3037     br i1 %tmp24, label %while_body, label %merge6
3038
3039 merge6:                                     ; preds = %while
3040     ret i1 true
3041
3042 while_body:                                ; preds = %while
3043     %str17 = load i8*, i8** %str11, align 8
3044     %i8 = load i32, i32* %i, align 4
3045     %get_char_ptr = getelementptr i8, i8* %str17, i32 %i8
3046     %get_char = load i8, i8* %get_char_ptr, align 1
3047     store i8 %get_char, i8* %c1, align 1
3048     %str29 = load i8*, i8** %str22, align 8
3049     %i10 = load i32, i32* %i, align 4
3050     %get_char_ptr11 = getelementptr i8, i8* %str29, i32 %i10
3051     %get_char12 = load i8, i8* %get_char_ptr11, align 1
3052     store i8 %get_char12, i8* %c2, align 1
3053     %c113 = load i8, i8* %c1, align 1
3054     %c214 = load i8, i8* %c2, align 1
3055     %tmp15 = icmp ne i8 %c113, %c214
3056     br i1 %tmp15, label %then17, label %else18
3057
3058 merge16:                                   ; preds = %else18
3059     %i19 = load i32, i32* %i, align 4
3060     %tmp20 = add i32 %i19, 1
3061     store i32 %tmp20, i32* %i, align 4
3062     br label %while
3063
3064 then17:                                    ; preds = %while_body
3065     ret i1 false
3066
3067 else18:                                    ; preds = %while_body
3068     br label %merge16
3069 }
3070
3071 declare noalias i8* @malloc(i32)
3072
3073 define i32 @list_length(%list_item* %0, i32 %1) {
3074 entry:
3075     %ptr_is_null = icmp eq %list_item* %0, null
3076     br i1 %ptr_is_null, label %then, label %else
3077
3078 then:                                       ; preds = %entry
3079     ret i32 %1
3080
3081 else:                                       ; preds = %entry
3082     %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
        1
3083     %next = load %list_item*, %list_item** %next_ptr, align 8
3084     %add = add i32 %1, 1
3085     %result = call i32 @list_length(%list_item* %next, i32 %add)
3086     ret i32 %result
3087 }
3088
3089 define %list_item* @list_access(%list_item* %0, i32 %1) {

```

```

3090 entry:
3091     %is_zero = icmp eq i32 %1, 0
3092     br i1 %is_zero, label %then, label %else
3093
3094 then:                                     ; preds = %entry
3095     ret %list_item* %0
3096
3097 else:                                     ; preds = %entry
3098     %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
3099     1
3100     %next = load %list_item*, %list_item** %next_ptr, align 8
3101     %sub = sub i32 %1, 1
3102     %result = call %list_item* @list_access(%list_item* %next, i32 %sub)
3103     ret %list_item* %result
3104 }
3105
3106 define %list_item** @insert_string(%list_item** %0, i8* %1, i32 %2) {
3107 entry:
3108     %list_ptr = load %list_item*, %list_item** %0, align 8
3109     %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
3110     i1** null, i32 1) to i32))
3111     %new_list_ptr_ptr = bitcast i8* %mallocall to %list_item**
3112     store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3113     %last_node_ptr_ptr = call %list_item** @list_copy_string(%list_item* %
3114     list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3115     %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3116     %temp = alloca %list_item, align 8
3117     %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3118     store %list_item* %new_list_ptr, %list_item** %next, align 8
3119     %mallocall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
3120     getelementptr (%list_item, %list_item* null, i32 1) to i32))
3121     %data_node = bitcast i8* %mallocall1 to %list_item*
3122     %mallocall2 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
3123     *, i1** null, i32 1) to i32))
3124     %data = bitcast i8* %mallocall2 to i8**
3125     store i8* %1, i8** %data, align 8
3126     %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3127     i32 0
3128     %cast = bitcast i8** %data to i8*
3129     store i8* %cast, i8** %dat, align 8
3130     %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3131     %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
3132     1
3133     %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3134     i32 1
3135     %temp4 = load %list_item*, %list_item** %test, align 8
3136     store %list_item* %temp4, %list_item** %dat3, align 8
3137     store %list_item* %data_node, %list_item** %test, align 8
3138     %temp5 = load %list_item*, %list_item** %next, align 8
3139     store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3140     ret %list_item** %new_list_ptr_ptr
3141 }
3142
3143 define %list_item** @list_copy_string(%list_item* %0, i32 %1, %list_item**
3144     %2) {
3145 entry:
3146     %is_zero = icmp eq i32 %1, 0
3147     %ptr_is_null = icmp eq %list_item* %0, null
3148     %or_conds = or i1 %is_zero, %ptr_is_null

```

```

3140     br i1 %or_conds, label %then, label %else
3141
3142 then:                                     ; preds = %entry
3143     ret %list_item** %2
3144
3145 else:                                     ; preds = %entry
3146     %allocacll = tail call i8* @malloc(i32 ptrtoint (%list_item*
3147         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3148     %new_struct_ptr = bitcast i8* %allocacll to %list_item*
3149     %store_list_item_zeroinitializer, %list_item* %new_struct_ptr, align 1
3150     %allocacll1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
3151         *, i1** null, i32 1) to i32))
3152     %ltyp = bitcast i8* %allocacll1 to i8**
3153     %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
3154         0, i32 0
3155     %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3156     %cast_old_data_ptr = bitcast i8* %old_data_ptr to i8**
3157     %old_data = load i8*, i8** %cast_old_data_ptr, align 8
3158     %store_i8* %old_data, i8** %ltyp, align 8
3159     %data_ptr_cast = bitcast i8** %ltyp to i8*
3160     %store_new_data = getelementptr inbounds %list_item, %list_item* %
3161         new_struct_ptr, i32 0, i32 0
3162     %store_i8* %data_ptr_cast, i8** %store_new_data, align 8
3163     %store %list_item* %new_struct_ptr, %list_item** %2, align 8
3164     %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
3165         i32 0, i32 1
3166     %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
3167         1
3168     %next2 = load %list_item*, %list_item** %next_ptr, align 8
3169     %sub = sub i32 %1, 1
3170     %3 = call %list_item** @list_copy_string(%list_item* %next2, i32 %sub, %
3171         list_item** %next)
3172     ret %list_item** %3
3173 }
3174
3175 define %list_item** @range_function(i32 %0, i32 %1, %list_item** %2, i32 %3)
3176 {
3177 entry:
3178     %is_last = icmp eq i32 %0, %1
3179     br i1 %is_last, label %then, label %else
3180
3181 then:                                     ; preds = %entry
3182     ret %list_item** %2
3183
3184 else:                                     ; preds = %entry
3185     %head_ptr_ptr = call %list_item** @insert_int(%list_item** %2, i32 %0, i32
3186         %3)
3187     %next_s = add i32 1, %0
3188     %next_length = add i32 1, %3
3189     %4 = call %list_item** @range_function(i32 %next_s, i32 %1, %list_item** %
3190         head_ptr_ptr, i32 %next_length)
3191     ret %list_item** %4
3192 }
3193
3194 define %list_item** @insert_int(%list_item** %0, i32 %1, i32 %2) {
3195 entry:
3196     %list_ptr = load %list_item*, %list_item** %0, align 8
3197     %allocacll = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
3198         i1** null, i32 1) to i32))

```

```

3188 %new_list_ptr_ptr = bitcast i8* %alloca1 to %list_item**
3189 store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3190 %last_node_ptr_ptr = call %list_item** @list_copy_int(%list_item* %
    list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3191 %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3192 %temp = alloca %list_item, align 8
3193 %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3194 store %list_item* %new_list_ptr, %list_item** %next, align 8
3195 %alloca11 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
3196 %data_node = bitcast i8* %alloca11 to %list_item*
3197 %alloca12 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
    , i32* null, i32 1) to i32))
3198 %data = bitcast i8* %alloca12 to i32*
3199 store i32 %1, i32* %data, align 4
3200 %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 0
3201 %cast = bitcast i32* %data to i8*
3202 store i8* %cast, i8** %dat, align 8
3203 %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3204 %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
    1
3205 %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 1
3206 %temp4 = load %list_item*, %list_item** %test, align 8
3207 store %list_item* %temp4, %list_item** %dat3, align 8
3208 store %list_item* %data_node, %list_item** %test, align 8
3209 %temp5 = load %list_item*, %list_item** %next, align 8
3210 store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3211 ret %list_item** %new_list_ptr_ptr
3212 }
3213
3214 define %list_item** @list_copy_int(%list_item* %0, i32 %1, %list_item** %2)
    {
3215 entry:
3216     %is_zero = icmp eq i32 %1, 0
3217     %ptr_is_null = icmp eq %list_item* %0, null
3218     %or_conds = or i1 %is_zero, %ptr_is_null
3219     br i1 %or_conds, label %then, label %else
3220
3221 then:
3222     ret %list_item** %2
3223
3224 else:
3225     %alloca1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
3226     %new_struct_ptr = bitcast i8* %alloca1 to %list_item*
3227     store %list_item* zeroinitializer, %list_item* %new_struct_ptr, align 1
3228     %alloca11 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
        , i32* null, i32 1) to i32))
3229     %ltyp = bitcast i8* %alloca11 to i32*
3230     %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
        0, i32 0
3231     %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3232     %cast_old_data_ptr = bitcast i8* %old_data_ptr to i32*
3233     %old_data = load i32, i32* %cast_old_data_ptr, align 4
3234     store i32 %old_data, i32* %ltyp, align 4
3235     %data_ptr_cast = bitcast i32* %ltyp to i8*

```



```

3236 %store_new_data = getelementptr inbounds %list_item, %list_item* %
    new_struct_ptr, i32 0, i32 0
3237 store i8* %data_ptr_cast, i8** %store_new_data, align 8
3238 store %list_item* %new_struct_ptr, %list_item** %2, align 8
3239 %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
    i32 0, i32 1
3240 %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
3241 %next2 = load %list_item*, %list_item** %next_ptr, align 8
3242 %sub = sub i32 %1, 1
3243 %3 = call %list_item** @list_copy_int(%list_item* %next2, i32 %sub, %
    list_item** %next)
3244 ret %list_item** %3
3245 }
3246
3247 define %list_item** @insert_char(%list_item** %0, i8 %1, i32 %2) {
3248 entry:
3249 %list_ptr = load %list_item*, %list_item** %0, align 8
3250 %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
3251 %new_list_ptr_ptr = bitcast i8* %malloccall to %list_item**
3252 store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3253 %last_node_ptr_ptr = call %list_item** @list_copy_char(%list_item* %
    list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3254 %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3255 %temp = alloca %list_item, align 8
3256 %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3257 store %list_item* %new_list_ptr, %list_item** %next, align 8
3258 %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
3259 %data_node = bitcast i8* %malloccall1 to %list_item*
3260 %data = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
    null, i32 1) to i32))
3261 store i8 %1, i8* %data, align 1
3262 %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 0
3263 store i8* %data, i8** %dat, align 8
3264 %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3265 %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
    1
3266 %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 1
3267 %temp4 = load %list_item*, %list_item** %test, align 8
3268 store %list_item* %temp4, %list_item** %dat3, align 8
3269 store %list_item* %data_node, %list_item** %test, align 8
3270 %temp5 = load %list_item*, %list_item** %next, align 8
3271 store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3272 ret %list_item** %new_list_ptr_ptr
3273 }
3274
3275 define %list_item** @list_copy_char(%list_item* %0, i32 %1, %list_item** %2)
    {
3276 entry:
3277 %is_zero = icmp eq i32 %1, 0
3278 %ptr_is_null = icmp eq %list_item* %0, null
3279 %or_conds = or i1 %is_zero, %ptr_is_null
3280 br i1 %or_conds, label %then, label %else
3281
3282 then:                                     ; preds = %entry

```

```

3283     ret %list_item** %2
3284
3285 else:
3286     %alloca1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
3287         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3288     %new_struct_ptr = bitcast i8* %alloca1 to %list_item*
3289     store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3290     %ltyp = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
3291         null, i32 1) to i32))
3292     %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
3293         0, i32 0
3294     %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3295     %old_data = load i8, i8* %old_data_ptr, align 1
3296     store i8 %old_data, i8* %ltyp, align 1
3297     %store_new_data = getelementptr inbounds %list_item, %list_item* %
3298         new_struct_ptr, i32 0, i32 0
3299     store i8* %ltyp, i8** %store_new_data, align 8
3300     store %list_item* %new_struct_ptr, %list_item** %2, align 8
3301     %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
3302         i32 0, i32 1
3303     %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
3304         1
3305     %next2 = load %list_item*, %list_item** %next_ptr, align 8
3306     %sub = sub i32 %1, 1
3307     %3 = call %list_item** @list_copy_char(%list_item* %next2, i32 %sub, %
3308         list_item** %next)
3309     ret %list_item** %3
3310 }
3311
3312 define %list_item** @insert_bool(%list_item** %0, i1 %1, i32 %2) {
3313 entry:
3314     %list_ptr = load %list_item*, %list_item** %0, align 8
3315     %alloca1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
3316         i1** null, i32 1) to i32))
3317     %new_list_ptr_ptr = bitcast i8* %alloca1 to %list_item**
3318     store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3319     %last_node_ptr_ptr = call %list_item** @list_copy_bool(%list_item* %
3320         list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3321     %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3322     %temp = alloca %list_item, align 8
3323     %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3324     store %list_item* %new_list_ptr, %list_item** %next, align 8
3325     %alloca11 = tail call i8* @malloc(i32 ptrtoint (%list_item*
3326         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3327     %data_node = bitcast i8* %alloca11 to %list_item*
3328     %alloca12 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
3329         i1* null, i32 1) to i32))
3330     %data = bitcast i8* %alloca12 to i1*
3331     store i1 %1, i1* %data, align 1
3332     %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3333         i32 0
3334     %cast = bitcast i1* %data to i8*
3335     store i8* %cast, i8** %dat, align 8
3336     %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3337     %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
3338         1
3339     %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
3340         i32 1
3341     %temp4 = load %list_item*, %list_item** %test, align 8

```

```

3328 store %list_item* %temp4, %list_item** %dat3, align 8
3329 store %list_item* %data_node, %list_item** %test, align 8
3330 %temp5 = load %list_item*, %list_item** %next, align 8
3331 store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3332 ret %list_item** %new_list_ptr_ptr
3333 }
3334
3335 define %list_item** @list_copy_bool(%list_item* %0, i32 %1, %list_item** %2)
3336 {
3337 entry:
3338   %is_zero = icmp eq i32 %1, 0
3339   %ptr_is_null = icmp eq %list_item* %0, null
3340   %or_conds = or i1 %is_zero, %ptr_is_null
3341   br i1 %or_conds, label %then, label %else
3342
3343 then:                                     ; preds = %entry
3344   ret %list_item** %2
3345
3346 else:                                     ; preds = %entry
3347   %mallocall = tail call i8* @malloc(i32 ptrtoint (%list_item*
3348     %getelementptr (%list_item, %list_item* null, i32 1) to i32))
3349   %new_struct_ptr = bitcast i8* %mallocall to %list_item*
3350   store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3351   %mallocall1 = tail call i8* @malloc(i32 ptrtoint (i1* %getelementptr (i1,
3352     i1* null, i32 1) to i32))
3353   %ltyp = bitcast i8* %mallocall1 to i1*
3354   %old_data_ptr_ptr = %getelementptr inbounds %list_item, %list_item* %0, i32
3355     0, i32 0
3356   %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3357   %cast_old_data_ptr = bitcast i8* %old_data_ptr to i1*
3358   %old_data = load i1, i1* %cast_old_data_ptr, align 1
3359   store i1 %old_data, i1* %ltyp, align 1
3360   %data_ptr_cast = bitcast i1* %ltyp to i8*
3361   %store_new_data = %getelementptr inbounds %list_item, %list_item* %
3362     new_struct_ptr, i32 0, i32 0
3363   store i8* %data_ptr_cast, i8** %store_new_data, align 8
3364   store %list_item* %new_struct_ptr, %list_item** %2, align 8
3365   %next = %getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
3366     i32 0, i32 1
3367   %next_ptr = %getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
3368     1
3369   %next2 = load %list_item*, %list_item** %next_ptr, align 8
3370   %sub = sub i32 %1, 1
3371   %3 = call %list_item** @list_copy_bool(%list_item* %next2, i32 %sub, %
3372     list_item** %next)
3373   ret %list_item** %3
3374 }
3375
3376 define %list_item** @insert_float(%list_item** %0, double %1, i32 %2) {
3377 entry:
3378   %list_ptr = load %list_item*, %list_item** %0, align 8
3379   %mallocall = tail call i8* @malloc(i32 ptrtoint (i1** %getelementptr (i1*,
3380     i1** null, i32 1) to i32))
3381   %new_list_ptr_ptr = bitcast i8* %mallocall to %list_item**
3382   store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3383   %last_node_ptr_ptr = call %list_item** @list_copy_float(%list_item* %
3384     list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3385   %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3386   %temp = alloca %list_item, align 8

```

```

3377 %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3378 store %list_item* %new_list_ptr, %list_item** %next, align 8
3379 %allocacll1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
3380 %data_node = bitcast i8* %allocacll1 to %list_item*
3381 %allocacll2 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (
    double, double* null, i32 1) to i32))
3382 %data = bitcast i8* %allocacll2 to double*
3383 store double %1, double* %data, align 8
3384 %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 0
3385 %cast = bitcast double* %data to i8*
3386 store i8* %cast, i8** %dat, align 8
3387 %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3388 %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
    1
3389 %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 1
3390 %temp4 = load %list_item*, %list_item** %test, align 8
3391 store %list_item* %temp4, %list_item** %dat3, align 8
3392 store %list_item* %data_node, %list_item** %test, align 8
3393 %temp5 = load %list_item*, %list_item** %next, align 8
3394 store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3395 ret %list_item** %new_list_ptr_ptr
3396 }
3397
3398 define %list_item** @list_copy_float(%list_item* %0, i32 %1, %list_item**
    %2) {
3399 entry:
3400     %is_zero = icmp eq i32 %1, 0
3401     %ptr_is_null = icmp eq %list_item* %0, null
3402     %or_conds = or i1 %is_zero, %ptr_is_null
3403     br i1 %or_conds, label %then, label %else
3404
3405 then:                                     ; preds = %entry
3406     ret %list_item** %2
3407
3408 else:                                     ; preds = %entry
3409     %allocacll = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
3410     %new_struct_ptr = bitcast i8* %allocacll to %list_item*
3411     store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3412     %allocacll1 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (
        double, double* null, i32 1) to i32))
3413     %ltyp = bitcast i8* %allocacll1 to double*
3414     %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
        0, i32 0
3415     %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3416     %cast_old_data_ptr = bitcast i8* %old_data_ptr to double*
3417     %old_data = load double, double* %cast_old_data_ptr, align 8
3418     store double %old_data, double* %ltyp, align 8
3419     %data_ptr_cast = bitcast double* %ltyp to i8*
3420     %store_new_data = getelementptr inbounds %list_item, %list_item* %
        new_struct_ptr, i32 0, i32 0
3421     store i8* %data_ptr_cast, i8** %store_new_data, align 8
3422     store %list_item* %new_struct_ptr, %list_item** %2, align 8
3423     %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
        i32 0, i32 1

```

```

3424 %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
      1
3425 %next2 = load %list_item*, %list_item** %next_ptr, align 8
3426 %sub = sub i32 %1, 1
3427 %3 = call %list_item** @list_copy_float(%list_item* %next2, i32 %sub, %
      list_item** %next)
3428 ret %list_item** %3
3429 }
3430
3431 define i1 @strcmp_function(i8* %0, i8* %1) {
3432 entry:
3433   %length = call i32 @string_length(i8* %0, i32 0)
3434   %length1 = call i32 @string_length(i8* %1, i32 0)
3435   %same_length = icmp ne i32 %length, %length1
3436   br i1 %same_length, label %then, label %else
3437
3438 then:                                     ; preds = %entry
3439   ret i1 false
3440
3441 else:                                     ; preds = %entry
3442   %last_index = sub i32 %length, 1
3443   %res = call i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %last_index,
      i32 0)
3444   ret i1 %res
3445 }
3446
3447 define i32 @string_length(i8* %0, i32 %1) {
3448 entry:
3449   %char = load i8, i8* %0, align 1
3450   %ptr_is_null = icmp eq i8 %char, 0
3451   br i1 %ptr_is_null, label %then, label %else
3452
3453 then:                                     ; preds = %entry
3454   ret i32 %1
3455
3456 else:                                     ; preds = %entry
3457   %next_ptr = getelementptr i8, i8* %0, i32 1
3458   %add = add i32 %1, 1
3459   %result = call i32 @string_length(i8* %next_ptr, i32 %add)
3460   ret i32 %result
3461 }
3462
3463 define i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %2, i32 %3) {
3464 entry:
3465   %charA_ptr = getelementptr i8, i8* %0, i32 %3
3466   %charA = load i8, i8* %charA_ptr, align 1
3467   %charB_ptr = getelementptr i8, i8* %1, i32 %3
3468   %charB = load i8, i8* %charB_ptr, align 1
3469   %not_same = icmp ne i8 %charA, %charB
3470   br i1 %not_same, label %then_not_same, label %else_same
3471
3472 then_not_same:                           ; preds = %entry
3473   ret i1 false
3474
3475 else_same:                               ; preds = %entry
3476   %last_char = icmp eq i32 %3, %2
3477   br i1 %last_char, label %then_end, label %else_not_end
3478
3479 then_end:                               ; preds = %else_same

```

```

3480     ret i1 true
3481
3482 else_not_end:                                ; preds = %else_same
3483     %next_index = add i32 1, %3
3484     %res = call i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %2, i32 %
        next_index)
3485     ret i1 %res
3486 }
3487
3488 ; Function Attrs: argmemonly nounwind willreturn
3489 declare void @llvm.memcpy.p0i8.p0i8.i32(i8* noalias nocapture writeonly, i8*
        noalias nocapture readonly, i32, i1 immarg) #0
3490
3491 attributes #0 = { argmemonly nounwind willreturn }

```