A Text Extraction and Manipulation Language

TEAM LANGUAGE FINAL REPORT

May 12, 2021

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

Contents

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:

```
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:

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

Compiling and executing the above code outputs:

```
4
Team
3.222000
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:

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

Compiling and executing the above code outputs:

```
4
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:

```
int calcSum(list data):
    int count = 0;
    for num in data:
        count += num;
    end
    return count;
end

list nums = [4,6,10,11,7,3];
int sum = calcSum(nums);
print("%d\n", sum);
```

Compiling and executing the above code outputs:

```
41
```

2.3.5 File I/O

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

```
file input = open("example.txt", "r");
string currline;
while ((currline = readline(intput)) != "")):
    print(currline);
end

close(input);
```

Here is a simple program to write to a file:

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

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 overwitten 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:

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

Team also supports list slicing with the following syntax:

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

```
// Function that calculates the average of a list of integers, after
    removing outliers
int computeAverage((list)->int average, list data):
    list validData = []
    for num in data:
        if (num < 95):
            validData = append(validData, num);
        end
    end
    return average(validData);
end

// Assume the function average takes in a list and returns an int
list data = [40, 50, 34, 89, 99, 57];
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:

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

Here is the grammar for all expressions:

```
expr -> unary_expr
expr -> expr + expr
expr -> expr - expr
expr -> expr * expr
expr -> expr / expr
expr -> expr ^ expr
expr -> expr % expr
expr -> expr == expr
expr -> expr != expr
expr -> expr <  expr
expr -> expr <= expr
expr -> expr >  expr
expr -> expr >= expr
expr -> expr and expr
expr -> expr or expr
expr -> not expr
expr -> expr ..  expr
expr -> expr = expr
expr -> expr += expr
expr -> expr -= expr
expr -> expr *= expr
expr -> expr /= expr
expr -> expr %= expr

unary_expr -> bracket_expr
unary_expr -> - unary_expr
unary_expr -> not unary_expr

bracket_expr -> primary
bracket_expr -> bracket_expr [ index ]
bracket_expr -> bracket_expr ( args_opt )

primary -> LITERAL (integer literal)
primary -> BLIT    (boolean literal)
primary -> FLIT    (float literal)
primary -> CLIT    (char literal)
primary -> SLIT    (string literal)
primary -> ID
primary -> [ list_literal ]
primary -> ( expr )

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

index -> expr
index -> expr : expr
index -> : expr
index -> expr :
index -> :

list_literal -> NOTHING
list_literal -> expr
list_literal -> list_literal : expr

args_opt ->
args_opt -> args_list

args_list -> expr
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:

```
expr -> expr + expr
expr -> expr - expr
expr -> expr * expr
expr -> expr / expr
expr -> expr ^ expr
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.

```
int x;
x = 5;
float y = 5.9;
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:

```
expr -> expr += expr
expr -> expr -= expr
expr -> expr *= expr
expr -> expr /= expr
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:

```
expr -> expr == expr
expr -> expr != expr
expr -> expr <  expr
expr -> expr <= expr
expr -> expr >  expr
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:

```
expr -> expr and expr
expr -> expr or expr
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:

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

The grammar for the range operator is:

```
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.

```
string greeting = "hello";
print("%s", greeting[0]); // Prints "h"

list classes = ["Compilers", "Security", "Algorithms", "Entrepreneurship", "
    Nutrition"];
print("%s", classes[0]); // Prints "Compilers"
```

Here is the grammar for index expressions:

```
bracket_expr -> bracket_expr [ index ]

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:

```
string greeting = "hello";
print("%s", greeting[1:5]); // Prints "ello"

string state = "Massachusetts";
print("%s", state[:4]); // Prints "Mass"

list classes = ["Compilers", "Security", "Algorithms", "Entrepreneurship", "
    Nutrition"];
list csClasses = classes[:3];

for class in csClasses:
    print("%s ", class); // Prints "Compilers Security Algorithms"
end
```

Here is the grammar for slice expressions:

```
expr -> unary_expr

unary_expr -> bracket_expr

bracket_expr -> bracket_expr [ index ]

index -> expr : expr
index -> : expr
index -> expr :
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:

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

Here is the grammar for function call:

```
expr -> unary_expr

unary_expr -> bracket_expr

bracket_expr -> bracket_expr ( args_opt )

args_opt ->
args_opt -> args_list

args_list -> expr
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.

```
int my_favorite_number = 43;
string university = "Tufts";
```

Here is the grammar for literals:

```
expr -> unary_expr

unary_expr -> bracket_expr

bracket_expr -> primary

primary -> LITERAL (integer literal)
primary -> BLIT (boolean literal)
primary -> FLIT (float literal)
primary -> CLIT (char literal)
primary -> SLIT (string literal)
primary -> ID
primary -> [ list_literal ]

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:

```
bool graduated;
string class = "Compilers";
```

Here is the grammar for variables:

```
expr -> unary_expr

unary_expr -> bracket_expr

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:

```
stmt -> vdecl ;
stmt -> expr ;
stmt -> return expr_opt ;
stmt -> if internal_if
stmt -> for ID in expr : stmt_list end
stmt ->  while expr : stmt_list end
stmt -> break ;
stmt -> continue ;

vdecl -> typ ID = expr
vdecl -> typ ID

internal_if -> expr : stmt_list else_list end

stmt_list -> NOTHING
stmt_list -> stmt_list stmt

expr_opt -> NOTHING
expr_opt -> expr

else_list -> NOTHING
else_list -> elif expr : stmt_list else_list
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^{th} iteration, the variable is assigned the i^{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.

```
int add(int x, int y):
    return x + y;
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:

```
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:

```
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:

```
void add_one(int x)
    x += 1;
end

int x = 2;
add_one(x);
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:

```
void change_first_element(list x)
    x[0] = 2;
end

list x = [1];
change_first_element(x);
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.

```
string shout(string text):
    string res = "";
    for c in text:
        res += upper(c);
    end

    return res;
end

string whisper(string text):
    string res = "";
    for c in text:
        res += lower(c);
    end

    return res;
end

/*
greet takes a function that takes a string
and returns a string (string->string)
and returns nothing (void)
*/
void greet((string)->string func):
    string greeting = func("Hello");
    print("%s", greeting);
end

/*
shout function is assigned to a function
variable named shout_var
*/
(string)->string shout_var = shout;
greet(shout_var); // prints "HELLO"
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

```
int y = 5;
print("%d", y); // y is visible here

void local():
```

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

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

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

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

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

- Formal parameters are only visible inside its respective function

```
void local(int j):
    print("%d", j); // j is visible here
end
// 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

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

```
list courses_taken = [11, 15, 40, 61];
list courses_to_take = [105, 160, 170];

if contains(courses_taken, 160):
    print("This kid knows algorithms\n");
else:
    print("This kid does know algorithms\n");
end

list courses_required = courses_taken + courses_to_take;
// courses_required is [11, 15, 40, 61, 105, 160, 170]
// courses_taken is still [11, 15, 40, 61]
// courses_to_take is still [105, 160, 170]

```

```
list more_cs = append(courses_taken, 107);
// more_cs is [11, 15, 40, 61, 107];

list more_math = insert(courses_taken, 34, 0);
// 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:

```
int i = 5;
float f = 3.14159;
char c = 'a';
print("%d", i); // displays 5
print("%f", f); // displays 3.14
print("%c", c); // displays a
print("%s", true); // displays true
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.

```
list roster = ["Lulu", "Yingjie", "Saurav", "Naoki"];

file output = open("output.txt", "w");

for name in roster:
  write(output, name + "\n");
end

close(output);

file input = open("output.txt", "r");
string currline;

while ((currline = readline(input)) != ""):
  print(currline);
end

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.

```
list courses_taken = [11, 15, 40, 61];
list courses_to_take = [105, 160, 170];
list courses_required = courses_taken + courses_to_take;

if contains(courses_taken, [160]):
    print("This kid knows algorithms\n");
else:
    print("This kid does not know algorithms\n");
end

list cs_minor = remove_int(remove_int(courses_required, 160, true), 105,
    true);
// courses_required is still [11, 15, 40, 61, 105, 160, 170]
// 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 delimeter and returns a list of substrings divided by the delimeter.
  - Parameter:
    - `str` (`string`): a string
    - `delim` (`string`): a string
  - Returns: a list (`list`) of strings that are separated by the delimeter

```
list l = split("Hello,World", ",");
// 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`)

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

```
bool b = startswith("hello world", 'h');
// b = true
bool c = startswith("hello world", 'i');
// 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`)

```
bool b = endswith("chocolate", 'e');
// b = true
bool b = endswith("chocolate", 'm');
// 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`)

```
char c = lower('H');
// 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`)

```
char c = upper('h');
// 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.

```
// One or more digits between 0 and 9, inclusive
string digits = "[0-9]+";

// Any string that starts with "chachacha"
string echos = "^cha{3}[.]*";

// Any string that ends with "a" or "b"
string ab = "(a|b)$";
```

6.5 Built-in Functions

- `match(target, regex)`: takes and 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

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

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

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

```
string a = replaceall("u", "a", "jumbo tufts");
// 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`.

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

7 Project Plan

7.1 Planning, Specification, and Development Process

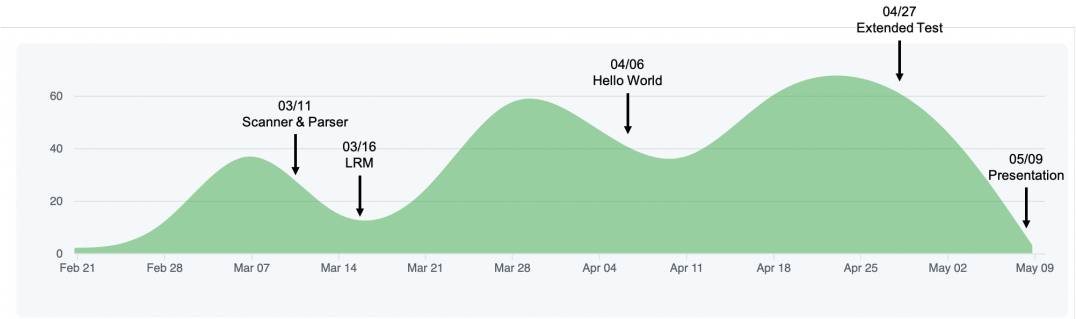
At the very beginning of the project, we each came up with our own language and voted on the one the one we thought was the best to implement. We picked the area of text manipulation because we realized how cumbersome it is to perform some simple text related tasks with general purpose languages. After 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.

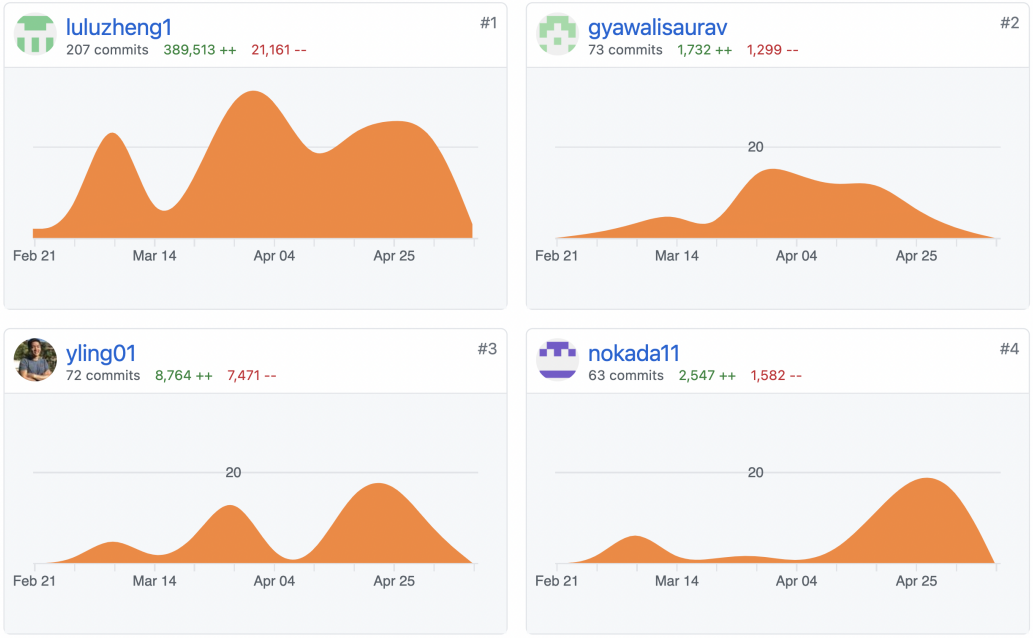
7.4 Development Tools

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.

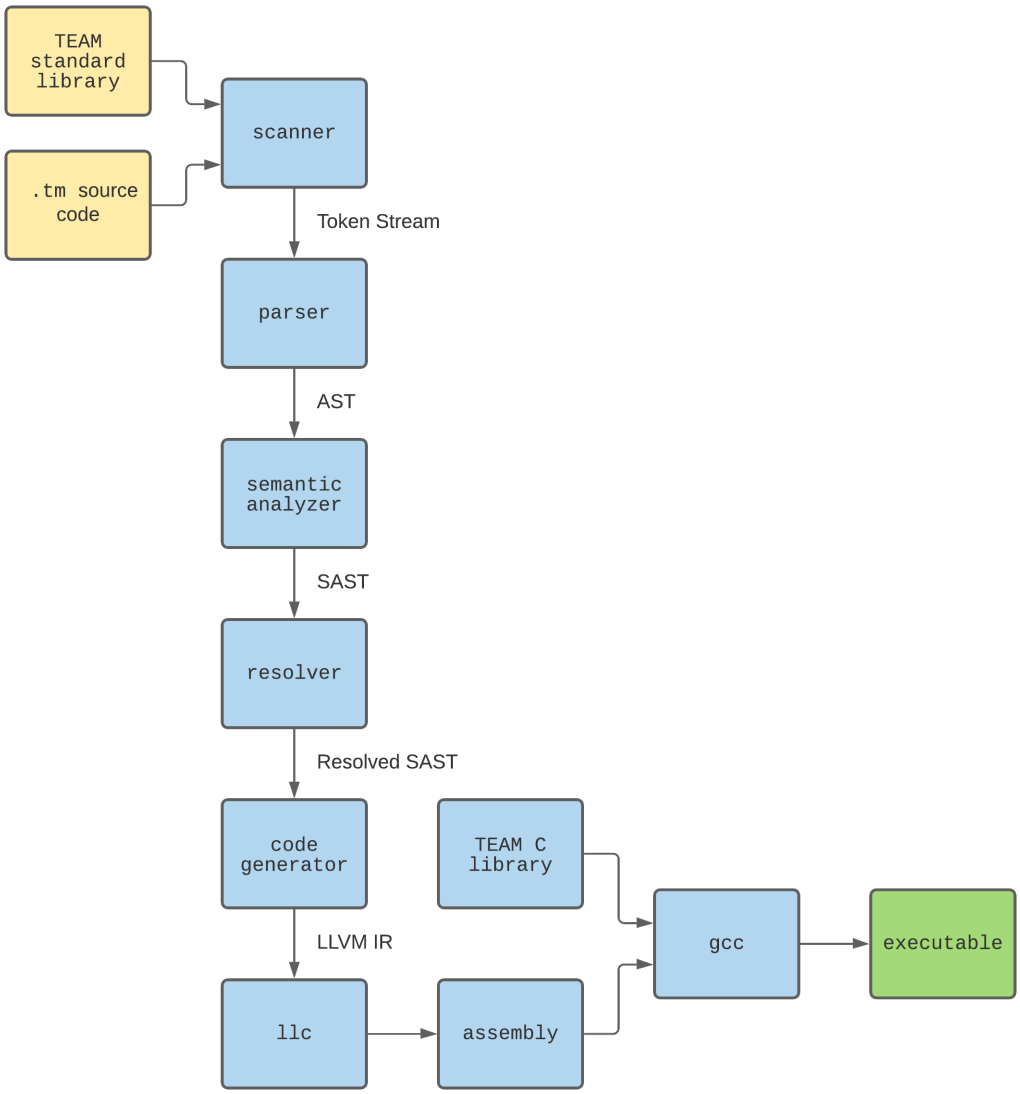


8 Architectural Design

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.

```
all: team.native fileio regex
team.native : ./src/parser.mly ./src/scanner.mll ./src/codegen.ml ./src/
    semant.ml ./src/resolve.ml ./src/team.ml
  opam config exec -- \
  ocamlbuild -use-ocamlfind ./src/team.native
# For built-in functions
.PHONY: fileio
fileio: ./c_library/fileio.c
  gcc -c -Wall -g ./c_library/fileio.c
  gcc -g -o fileio -DBUILD_TEST ./c_library/fileio.c

.PHONY: regex
regex : ./c_library/regex.c
  gcc -c -Wall -g ./c_library/regex.c
  gcc -g  -lpcreposix -lpcre2-8 -o regex -DBUILD_TEST ./c_library/regex.c

.PHONY : clean
clean :
  ocamlbuild -clean
  rm *.o
  rm regex
  rm fileio
  rm -r *.dSYM/
```

9.2.2 `runtests.py`

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

```
#!/usr/bin/env python3
import sys
import os
import subprocess
import optparse

class bcolors:
    OKGREEN = '\033[92m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    WARNING = '\033[93m'
    UNDERLINE = '\033[4m'

SCANNER_PARSER_DIR = ("tests/ast_tests", "ref/ast_ref")
SEMANT_DIR = ("tests/sast_tests", "ref/sast_ref")
CODEGEN_DIR = ("tests/codegen_tests", "ref/codegen_ref")
EXTENDED_DIR = ("tests/extended_tests", "ref/extended_ref")

def getCompiledFiles():
    fileList = [f for f in os.listdir(".") if ".o" in f]
    return " ".join(fileList)

def sortingKey(fileName):
    return fileName.split(".")[0]

def runTests(testMode):
    if testMode == "sast":
        dirTuple = SEMANT_DIR
    elif testMode == "ast":
        dirTuple = SCANNER_PARSER_DIR
    elif testMode == "codegen":
        dirTuple = CODEGEN_DIR
    elif testMode == "extended":
        dirTuple = EXTENDED_DIR
    else:
        print(bcolors.FAIL + "Test mode: {} not supported".format(testMode)
    + bcolors.ENDC)
        sys.exit()

    testFileDir, refFileDir = dirTuple
    testFiles = list(map(lambda x: "/".join((testFileDir, x)), sorted(filter
    (lambda x: "tm" in x, os.listdir(testFileDir)), key=sortingKey)))
    refFiles = list(map(lambda x: "/".join((refFileDir, x)), sorted(filter(
    lambda x: "log" in x, os.listdir(refFileDir)), key=sortingKey)))

    assert len(testFiles) == len(refFiles)

    for i in range(len(testFiles)):
        testFile, refFile = testFiles[i], refFiles[i]

        logFile = runFile(testFile, testMode)
        checkResults(logFile, refFile)

def printFailedTestMessage(f_generated, f_reference):
    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,
```

```
                            stderr=subprocess.PIPE)

    if userInput:
        dir_prefix = "user_log"
    else:
        dir_prefix = "log/" + "_".join((testMode, "log"))

    if not os.path.exists(dir_prefix):
        os.makedirs(dir_prefix)
    filename = '{}/{}.log'.format(dir_prefix, fileName.split('/')[-1].split(
    '.')[0])
    with open(filename, 'w+') as fo:
        toWrite = stdout if stdout else stderr
        fo.write(toWrite.decode('utf-8'))
    return filename


def interpretCommand(command):
    return True if command[0].upper() == "T" else False

def compile(topLevel):
    process = subprocess.Popen(['ocamlbuild', '-use-ocamlfind', topLevel],
                               stdout=subprocess.PIPE,
                               stderr=subprocess.PIPE)
    stdout, stderr = process.communicate()
    if 'failed' in stdout.decode('utf-8') or "Error" in stdout.decode('utf-8
    '):
        print(bcolors.FAIL + "Error detected when compiling {}. See message
    below.".format(topLevel) + bcolors.ENDC)
        print(stdout.decode('utf-8'))
        sys.exit()

def clean(target):
    if target == "ocaml":
        process = subprocess.Popen(['ocamlbuild', '-clean'],
                                   stdout=subprocess.PIPE,
                                   stderr=subprocess.PIPE)

if __name__ == "__main__":
    parser = optparse.OptionParser()
    parser.add_option('-c', '--recompile',
                      dest='recompile', default='False',
                      help='True to recompile top level.')
    parser.add_option('-t', '--testFile',
                      dest='testFile', default='',
                      help='Specify one test file. AST is written.')
    parser.add_option('-r', '--reference',
                      dest='reference', default='',
                      help='The reference to compare generated AST.')
    parser.add_option('-l', '--topLevel',
                      dest='topLevel', default='team.native',
                      help='Name of the top level')
    parser.add_option('-m', '--testMode',
                      dest='testMode', default='extended',
                      help='ast, sast, codegen, extended, all')

    options, args = parser.parse_args()
    recompile = interpretCommand(options.recompile)
    testFile = options.testFile
```

```
    topLevel = options.topLevel
    reference = options.reference
    testMode = options.testMode

    if reference != '' and testFile == '':
        print("Error detected!\nReference specified with no test file specified.\nExiting...\n")
        sys.exit()

    if recompile or topLevel not in os.listdir('/'):
        compile(topLevel)

    if testFile != '':
        if testMode == "all":
            print("Error detected!\n Test mode cannot be all when testing a single file")
            sys.exit()
        logFile = runFile(testFile, testMode, True)
        checkResults(logFile, reference)
    else:
        testMode = [testMode] if testMode != "all" else ["ast", "sast", "codegen", "extended"]
        for m in testMode:
            print(bcolors.WARNING + bcolors.UNDERLINE + "\nTest mode: {}\n".format(m) + bcolors.ENDC)
            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:

```
/* This program maximizes profit by choosing a single day to buy
 * one stock and choosing a different day in the future to sell
 * that stock.
 * Return the maximum profit one can achieve from this transaction.
 * If no profits can be achieved, return 0. */

int max_profit(list stock_prices):
    int min_price = stock_prices[0];
    int max_profit = 0;

    for i in 1..length(stock_prices)-1:
        int current_price = stock_prices[i];
        int next_price = stock_prices[i+1];
        if current_price > next_price:
            int potential_profit = current_price - min_price;
            max_profit += potential_profit;
            min_price = stock_prices[i+1];
        else:
            if min_price > current_price:
                min_price = current_price;
```

```
            end
        end
    end
    return max_profit;
end

print("%d\n", max_profit([5,2,6,9,3]));
```

The program is expected to have the following output:

```
7
```

Below is the LLVM generated by the compiler:

```
; ModuleID = 'TEAM'
source_filename = "TEAM"

%list_item = type <{ i8*, %list_item* }>

@ASCII = global %list_item** null
@string = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@string.1 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
@string.2 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
@string.3 = private unnamed_addr constant [6 x i8] c"wolrd\00", align 1
@string.4 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
@string.5 = private unnamed_addr constant [5 x i8] c"what\00", align 1
@string.6 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
@string.7 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@string.8 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@string.9 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@string.10 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@string.11 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1

declare i32 @printf(i8*, ...)

declare double @pow(double, double)

declare i8* @fopen(i8*, i8*)

declare i32 @close(i8*)

declare i8* @readline(i8*)

declare i8* @write(i8*, i8*)

declare i1 @match(i8*, i8*)

declare i8* @find(i8*, i8*)

declare i8* @replace(i8*, i8*, i8*, i32)

declare i8* @replace_all(i8*, i8*, i8*)

declare %list_item** @find_all(i8*, i8*)

define i32 @main() {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
```

```
%malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item = bitcast i8* %malloccall1 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item, align 1
%copied = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
  null, i32 1) to i32))
store i8 90, i8* %copied, align 1
%data_ptr_container = getelementptr inbounds %list_item, %list_item* %
  list_item, i32 0, i32 0
store i8* %copied, i8** %data_ptr_container, align 8
%next = getelementptr inbounds %list_item, %list_item* %list_item, i32 0,
  i32 1
store %list_item* null, %list_item** %next, align 8
%malloccall3 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item4 = bitcast i8* %malloccall3 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item4, align 1
%copied6 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
  null, i32 1) to i32))
store i8 89, i8* %copied6, align 1
%data_ptr_container7 = getelementptr inbounds %list_item, %list_item* %
  list_item4, i32 0, i32 0
store i8* %copied6, i8** %data_ptr_container7, align 8
%next8 = getelementptr inbounds %list_item, %list_item* %list_item4, i32
  0, i32 1
store %list_item* %list_item, %list_item** %next8, align 8
%malloccall9 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item10 = bitcast i8* %malloccall9 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item10, align 1
%copied12 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
   null, i32 1) to i32))
store i8 88, i8* %copied12, align 1
%data_ptr_container13 = getelementptr inbounds %list_item, %list_item* %
  list_item10, i32 0, i32 0
store i8* %copied12, i8** %data_ptr_container13, align 8
%next14 = getelementptr inbounds %list_item, %list_item* %list_item10, i32
   0, i32 1
store %list_item* %list_item4, %list_item** %next14, align 8
%malloccall15 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item16 = bitcast i8* %malloccall15 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item16, align 1
%copied18 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
   null, i32 1) to i32))
store i8 87, i8* %copied18, align 1
%data_ptr_container19 = getelementptr inbounds %list_item, %list_item* %
  list_item16, i32 0, i32 0
store i8* %copied18, i8** %data_ptr_container19, align 8
%next20 = getelementptr inbounds %list_item, %list_item* %list_item16, i32
   0, i32 1
store %list_item* %list_item10, %list_item** %next20, align 8
%malloccall21 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item22 = bitcast i8* %malloccall21 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item22, align 1
%copied24 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
   null, i32 1) to i32))
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  %malloccall27 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
92  %list_item28 = bitcast i8* %malloccall27 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 %malloccall33 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
101 %list_item34 = bitcast i8* %malloccall33 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 %malloccall39 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
110 %list_item40 = bitcast i8* %malloccall39 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 %malloccall45 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
119 %list_item46 = bitcast i8* %malloccall45 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 %malloccall51 = tail call i8* @malloc(i32 ptrtoint (%list_item*
```

```
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item52 = bitcast i8* %malloccall51 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item52, align 1
  %copied54 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 81, i8* %copied54, align 1
  %data_ptr_container55 = getelementptr inbounds %list_item, %list_item* %
    list_item52, i32 0, i32 0
  store i8* %copied54, i8** %data_ptr_container55, align 8
  %next56 = getelementptr inbounds %list_item, %list_item* %list_item52, i32
     0, i32 1
  store %list_item* %list_item46, %list_item** %next56, align 8
  %malloccall57 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item58 = bitcast i8* %malloccall57 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item58, align 1
  %copied60 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 80, i8* %copied60, align 1
  %data_ptr_container61 = getelementptr inbounds %list_item, %list_item* %
    list_item58, i32 0, i32 0
  store i8* %copied60, i8** %data_ptr_container61, align 8
  %next62 = getelementptr inbounds %list_item, %list_item* %list_item58, i32
     0, i32 1
  store %list_item* %list_item52, %list_item** %next62, align 8
  %malloccall63 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item64 = bitcast i8* %malloccall63 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item64, align 1
  %copied66 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 79, i8* %copied66, align 1
  %data_ptr_container67 = getelementptr inbounds %list_item, %list_item* %
    list_item64, i32 0, i32 0
  store i8* %copied66, i8** %data_ptr_container67, align 8
  %next68 = getelementptr inbounds %list_item, %list_item* %list_item64, i32
     0, i32 1
  store %list_item* %list_item58, %list_item** %next68, align 8
  %malloccall69 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item70 = bitcast i8* %malloccall69 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item70, align 1
  %copied72 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 78, i8* %copied72, align 1
  %data_ptr_container73 = getelementptr inbounds %list_item, %list_item* %
    list_item70, i32 0, i32 0
  store i8* %copied72, i8** %data_ptr_container73, align 8
  %next74 = getelementptr inbounds %list_item, %list_item* %list_item70, i32
     0, i32 1
  store %list_item* %list_item64, %list_item** %next74, align 8
  %malloccall75 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item76 = bitcast i8* %malloccall75 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item76, align 1
  %copied78 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 77, i8* %copied78, align 1
  %data_ptr_container79 = getelementptr inbounds %list_item, %list_item* %
```

```
    list_item76, i32 0, i32 0
  store i8* %copied78, i8** %data_ptr_container79, align 8
  %next80 = getelementptr inbounds %list_item, %list_item* %list_item76, i32
     0, i32 1
  store %list_item* %list_item70, %list_item** %next80, align 8
  %malloccall81 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item82 = bitcast i8* %malloccall81 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item82, align 1
  %copied84 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 76, i8* %copied84, align 1
  %data_ptr_container85 = getelementptr inbounds %list_item, %list_item* %
    list_item82, i32 0, i32 0
  store i8* %copied84, i8** %data_ptr_container85, align 8
  %next86 = getelementptr inbounds %list_item, %list_item* %list_item82, i32
     0, i32 1
  store %list_item* %list_item76, %list_item** %next86, align 8
  %malloccall87 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item88 = bitcast i8* %malloccall87 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item88, align 1
  %copied90 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 75, i8* %copied90, align 1
  %data_ptr_container91 = getelementptr inbounds %list_item, %list_item* %
    list_item88, i32 0, i32 0
  store i8* %copied90, i8** %data_ptr_container91, align 8
  %next92 = getelementptr inbounds %list_item, %list_item* %list_item88, i32
     0, i32 1
  store %list_item* %list_item82, %list_item** %next92, align 8
  %malloccall93 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item94 = bitcast i8* %malloccall93 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item94, align 1
  %copied96 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 74, i8* %copied96, align 1
  %data_ptr_container97 = getelementptr inbounds %list_item, %list_item* %
    list_item94, i32 0, i32 0
  store i8* %copied96, i8** %data_ptr_container97, align 8
  %next98 = getelementptr inbounds %list_item, %list_item* %list_item94, i32
     0, i32 1
  store %list_item* %list_item88, %list_item** %next98, align 8
  %malloccall99 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item100 = bitcast i8* %malloccall99 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item100, align 1
  %copied102 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 73, i8* %copied102, align 1
  %data_ptr_container103 = getelementptr inbounds %list_item, %list_item* %
    list_item100, i32 0, i32 0
  store i8* %copied102, i8** %data_ptr_container103, align 8
  %next104 = getelementptr inbounds %list_item, %list_item* %list_item100,
    i32 0, i32 1
  store %list_item* %list_item94, %list_item** %next104, align 8
  %malloccall105 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
```

```
209   %list_item106 = bitcast i8* %malloccall105 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   %malloccall111 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
218   %list_item112 = bitcast i8* %malloccall111 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   %malloccall117 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
227   %list_item118 = bitcast i8* %malloccall117 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   %malloccall123 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
236   %list_item124 = bitcast i8* %malloccall123 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   %malloccall129 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
245   %list_item130 = bitcast i8* %malloccall129 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
```

```
  store i8* %copied132, i8** %data_ptr_container133, align 8
  %next134 = getelementptr inbounds %list_item, %list_item* %list_item130,
    i32 0, i32 1
  store %list_item* %list_item124, %list_item** %next134, align 8
  %malloccall135 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item136 = bitcast i8* %malloccall135 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item136, align 1
  %copied138 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 67, i8* %copied138, align 1
  %data_ptr_container139 = getelementptr inbounds %list_item, %list_item* %
    list_item136, i32 0, i32 0
  store i8* %copied138, i8** %data_ptr_container139, align 8
  %next140 = getelementptr inbounds %list_item, %list_item* %list_item136,
    i32 0, i32 1
  store %list_item* %list_item130, %list_item** %next140, align 8
  %malloccall141 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item142 = bitcast i8* %malloccall141 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item142, align 1
  %copied144 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 66, i8* %copied144, align 1
  %data_ptr_container145 = getelementptr inbounds %list_item, %list_item* %
    list_item142, i32 0, i32 0
  store i8* %copied144, i8** %data_ptr_container145, align 8
  %next146 = getelementptr inbounds %list_item, %list_item* %list_item142,
    i32 0, i32 1
  store %list_item* %list_item136, %list_item** %next146, align 8
  %malloccall147 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item148 = bitcast i8* %malloccall147 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item148, align 1
  %copied150 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 65, i8* %copied150, align 1
  %data_ptr_container151 = getelementptr inbounds %list_item, %list_item* %
    list_item148, i32 0, i32 0
  store i8* %copied150, i8** %data_ptr_container151, align 8
  %next152 = getelementptr inbounds %list_item, %list_item* %list_item148,
    i32 0, i32 1
  store %list_item* %list_item142, %list_item** %next152, align 8
  %malloccall153 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item154 = bitcast i8* %malloccall153 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item154, align 1
  %copied156 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 122, i8* %copied156, align 1
  %data_ptr_container157 = getelementptr inbounds %list_item, %list_item* %
    list_item154, i32 0, i32 0
  store i8* %copied156, i8** %data_ptr_container157, align 8
  %next158 = getelementptr inbounds %list_item, %list_item* %list_item154,
    i32 0, i32 1
  store %list_item* %list_item148, %list_item** %next158, align 8
  %malloccall159 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item160 = bitcast i8* %malloccall159 to %list_item*
```

```
291   store %list_item zeroinitializer, %list_item* %list_item160, align 1
292   %copied162 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
        * null, i32 1) to i32))
293   store i8 121, i8* %copied162, align 1
294   %data_ptr_container163 = getelementptr inbounds %list_item, %list_item* %
        list_item160, i32 0, i32 0
295   store i8* %copied162, i8** %data_ptr_container163, align 8
296   %next164 = getelementptr inbounds %list_item, %list_item* %list_item160,
        i32 0, i32 1
297   store %list_item* %list_item154, %list_item** %next164, align 8
298   %malloccall165 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
299   %list_item166 = bitcast i8* %malloccall165 to %list_item*
300   store %list_item zeroinitializer, %list_item* %list_item166, align 1
301   %copied168 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
        * null, i32 1) to i32))
302   store i8 120, i8* %copied168, align 1
303   %data_ptr_container169 = getelementptr inbounds %list_item, %list_item* %
        list_item166, i32 0, i32 0
304   store i8* %copied168, i8** %data_ptr_container169, align 8
305   %next170 = getelementptr inbounds %list_item, %list_item* %list_item166,
        i32 0, i32 1
306   store %list_item* %list_item160, %list_item** %next170, align 8
307   %malloccall171 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
308   %list_item172 = bitcast i8* %malloccall171 to %list_item*
309   store %list_item zeroinitializer, %list_item* %list_item172, align 1
310   %copied174 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
        * null, i32 1) to i32))
311   store i8 119, i8* %copied174, align 1
312   %data_ptr_container175 = getelementptr inbounds %list_item, %list_item* %
        list_item172, i32 0, i32 0
313   store i8* %copied174, i8** %data_ptr_container175, align 8
314   %next176 = getelementptr inbounds %list_item, %list_item* %list_item172,
        i32 0, i32 1
315   store %list_item* %list_item166, %list_item** %next176, align 8
316   %malloccall177 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
317   %list_item178 = bitcast i8* %malloccall177 to %list_item*
318   store %list_item zeroinitializer, %list_item* %list_item178, align 1
319   %copied180 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
        * null, i32 1) to i32))
320   store i8 118, i8* %copied180, align 1
321   %data_ptr_container181 = getelementptr inbounds %list_item, %list_item* %
        list_item178, i32 0, i32 0
322   store i8* %copied180, i8** %data_ptr_container181, align 8
323   %next182 = getelementptr inbounds %list_item, %list_item* %list_item178,
        i32 0, i32 1
324   store %list_item* %list_item172, %list_item** %next182, align 8
325   %malloccall183 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
326   %list_item184 = bitcast i8* %malloccall183 to %list_item*
327   store %list_item zeroinitializer, %list_item* %list_item184, align 1
328   %copied186 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
        * null, i32 1) to i32))
329   store i8 117, i8* %copied186, align 1
330   %data_ptr_container187 = getelementptr inbounds %list_item, %list_item* %
        list_item184, i32 0, i32 0
331   store i8* %copied186, i8** %data_ptr_container187, 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   %malloccall189 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
335   %list_item190 = bitcast i8* %malloccall189 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   %malloccall195 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
344   %list_item196 = bitcast i8* %malloccall195 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   %malloccall201 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
353   %list_item202 = bitcast i8* %malloccall201 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   %malloccall207 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
362   %list_item208 = bitcast i8* %malloccall207 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   %malloccall213 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
371   %list_item214 = bitcast i8* %malloccall213 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   %malloccall219 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
380   %list_item220 = bitcast i8* %malloccall219 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   %malloccall225 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
389   %list_item226 = bitcast i8* %malloccall225 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   %malloccall231 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
398   %list_item232 = bitcast i8* %malloccall231 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   %malloccall237 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
407   %list_item238 = bitcast i8* %malloccall237 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
store %list_item* %list_item232, %list_item** %next242, align 8
%malloccall243 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item244 = bitcast i8* %malloccall243 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item244, align 1
%copied246 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
  * null, i32 1) to i32))
store i8 107, i8* %copied246, align 1
%data_ptr_container247 = getelementptr inbounds %list_item, %list_item* %
  list_item244, i32 0, i32 0
store i8* %copied246, i8** %data_ptr_container247, align 8
%next248 = getelementptr inbounds %list_item, %list_item* %list_item244,
  i32 0, i32 1
store %list_item* %list_item238, %list_item** %next248, align 8
%malloccall249 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item250 = bitcast i8* %malloccall249 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item250, align 1
%copied252 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
  * null, i32 1) to i32))
store i8 106, i8* %copied252, align 1
%data_ptr_container253 = getelementptr inbounds %list_item, %list_item* %
  list_item250, i32 0, i32 0
store i8* %copied252, i8** %data_ptr_container253, align 8
%next254 = getelementptr inbounds %list_item, %list_item* %list_item250,
  i32 0, i32 1
store %list_item* %list_item244, %list_item** %next254, align 8
%malloccall255 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item256 = bitcast i8* %malloccall255 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item256, align 1
%copied258 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
  * null, i32 1) to i32))
store i8 105, i8* %copied258, align 1
%data_ptr_container259 = getelementptr inbounds %list_item, %list_item* %
  list_item256, i32 0, i32 0
store i8* %copied258, i8** %data_ptr_container259, align 8
%next260 = getelementptr inbounds %list_item, %list_item* %list_item256,
  i32 0, i32 1
store %list_item* %list_item250, %list_item** %next260, align 8
%malloccall261 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item262 = bitcast i8* %malloccall261 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item262, align 1
%copied264 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
  * null, i32 1) to i32))
store i8 104, i8* %copied264, align 1
%data_ptr_container265 = getelementptr inbounds %list_item, %list_item* %
  list_item262, i32 0, i32 0
store i8* %copied264, i8** %data_ptr_container265, align 8
%next266 = getelementptr inbounds %list_item, %list_item* %list_item262,
  i32 0, i32 1
store %list_item* %list_item256, %list_item** %next266, align 8
%malloccall267 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item268 = bitcast i8* %malloccall267 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item268, align 1
%copied270 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
```

```
  * null, i32 1) to i32))
  store i8 103, i8* %copied270, align 1
  %data_ptr_container271 = getelementptr inbounds %list_item, %list_item* %
    list_item268, i32 0, i32 0
  store i8* %copied270, i8** %data_ptr_container271, align 8
  %next272 = getelementptr inbounds %list_item, %list_item* %list_item268,
    i32 0, i32 1
  store %list_item* %list_item262, %list_item** %next272, align 8
  %malloccall273 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item274 = bitcast i8* %malloccall273 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item274, align 1
  %copied276 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 102, i8* %copied276, align 1
  %data_ptr_container277 = getelementptr inbounds %list_item, %list_item* %
    list_item274, i32 0, i32 0
  store i8* %copied276, i8** %data_ptr_container277, align 8
  %next278 = getelementptr inbounds %list_item, %list_item* %list_item274,
    i32 0, i32 1
  store %list_item* %list_item268, %list_item** %next278, align 8
  %malloccall279 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item280 = bitcast i8* %malloccall279 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item280, align 1
  %copied282 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 101, i8* %copied282, align 1
  %data_ptr_container283 = getelementptr inbounds %list_item, %list_item* %
    list_item280, i32 0, i32 0
  store i8* %copied282, i8** %data_ptr_container283, align 8
  %next284 = getelementptr inbounds %list_item, %list_item* %list_item280,
    i32 0, i32 1
  store %list_item* %list_item274, %list_item** %next284, align 8
  %malloccall285 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item286 = bitcast i8* %malloccall285 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item286, align 1
  %copied288 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 100, i8* %copied288, align 1
  %data_ptr_container289 = getelementptr inbounds %list_item, %list_item* %
    list_item286, i32 0, i32 0
  store i8* %copied288, i8** %data_ptr_container289, align 8
  %next290 = getelementptr inbounds %list_item, %list_item* %list_item286,
    i32 0, i32 1
  store %list_item* %list_item280, %list_item** %next290, align 8
  %malloccall291 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item292 = bitcast i8* %malloccall291 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item292, align 1
  %copied294 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 99, i8* %copied294, align 1
  %data_ptr_container295 = getelementptr inbounds %list_item, %list_item* %
    list_item292, i32 0, i32 0
  store i8* %copied294, i8** %data_ptr_container295, align 8
  %next296 = getelementptr inbounds %list_item, %list_item* %list_item292,
    i32 0, i32 1
```

```
  store %list_item* %list_item286, %list_item** %next296, align 8
  %malloccall297 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item298 = bitcast i8* %malloccall297 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item298, align 1
  %copied300 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 98, i8* %copied300, align 1
  %data_ptr_container301 = getelementptr inbounds %list_item, %list_item* %
    list_item298, i32 0, i32 0
  store i8* %copied300, i8** %data_ptr_container301, align 8
  %next302 = getelementptr inbounds %list_item, %list_item* %list_item298,
    i32 0, i32 1
  store %list_item* %list_item292, %list_item** %next302, align 8
  %malloccall303 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item304 = bitcast i8* %malloccall303 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item304, align 1
  %copied306 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 97, i8* %copied306, align 1
  %data_ptr_container307 = getelementptr inbounds %list_item, %list_item* %
    list_item304, i32 0, i32 0
  store i8* %copied306, i8** %data_ptr_container307, align 8
  %next308 = getelementptr inbounds %list_item, %list_item* %list_item304,
    i32 0, i32 1
  store %list_item* %list_item298, %list_item** %next308, align 8
  store %list_item* %list_item304, %list_item** %list, align 8
  store %list_item** %list, %list_item*** @ASCII, align 8
  %malloccall309 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list310 = bitcast i8* %malloccall309 to %list_item**
  %malloccall311 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item312 = bitcast i8* %malloccall311 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item312, align 1
  %malloccall313 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %copied314 = bitcast i8* %malloccall313 to i8**
  store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.1, i32 0,
    i32 0), i8** %copied314, align 8
  %cast_ptr = bitcast i8** %copied314 to i8*
  %data_ptr_container315 = getelementptr inbounds %list_item, %list_item* %
    list_item312, i32 0, i32 0
  store i8* %cast_ptr, i8** %data_ptr_container315, align 8
  %next316 = getelementptr inbounds %list_item, %list_item* %list_item312,
    i32 0, i32 1
  store %list_item* null, %list_item** %next316, align 8
  store %list_item* %list_item312, %list_item** %list310, align 8
  %_result = call i8* @join_string_string(%list_item** %list310, i8*
    getelementptr inbounds ([1 x i8], [1 x i8]* @string, i32 0, i32 0))
  %malloccall317 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list318 = bitcast i8* %malloccall317 to %list_item**
  %malloccall319 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item320 = bitcast i8* %malloccall319 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item320, align 1
  %malloccall321 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
```

```
    i32, i32* null, i32 1) to i32))
  %copied322 = bitcast i8* %malloccall321 to i32*
  store i32 1, i32* %copied322, align 4
  %cast_ptr323 = bitcast i32* %copied322 to i8*
  %data_ptr_container324 = getelementptr inbounds %list_item, %list_item* %
    list_item320, i32 0, i32 0
  store i8* %cast_ptr323, i8** %data_ptr_container324, align 8
  %next325 = getelementptr inbounds %list_item, %list_item* %list_item320,
    i32 0, i32 1
  store %list_item* null, %list_item** %next325, align 8
  store %list_item* %list_item320, %list_item** %list318, align 8
  %malloccall326 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list327 = bitcast i8* %malloccall326 to %list_item**
  %malloccall328 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item329 = bitcast i8* %malloccall328 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item329, align 1
  %malloccall330 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
  %copied331 = bitcast i8* %malloccall330 to i32*
  store i32 2, i32* %copied331, align 4
  %cast_ptr332 = bitcast i32* %copied331 to i8*
  %data_ptr_container333 = getelementptr inbounds %list_item, %list_item* %
    list_item329, i32 0, i32 0
  store i8* %cast_ptr332, i8** %data_ptr_container333, align 8
  %next334 = getelementptr inbounds %list_item, %list_item* %list_item329,
    i32 0, i32 1
  store %list_item* null, %list_item** %next334, align 8
  %malloccall335 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item336 = bitcast i8* %malloccall335 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item336, align 1
  %malloccall337 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
  %copied338 = bitcast i8* %malloccall337 to i32*
  store i32 1, i32* %copied338, align 4
  %cast_ptr339 = bitcast i32* %copied338 to i8*
  %data_ptr_container340 = getelementptr inbounds %list_item, %list_item* %
    list_item336, i32 0, i32 0
  store i8* %cast_ptr339, i8** %data_ptr_container340, align 8
  %next341 = getelementptr inbounds %list_item, %list_item* %list_item336,
    i32 0, i32 1
  store %list_item* %list_item329, %list_item** %next341, align 8
  store %list_item* %list_item336, %list_item** %list327, align 8
  %_result342 = call i1 @contains_int_int(%list_item** %list327, %list_item
    ** %list318)
  %malloccall343 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list344 = bitcast i8* %malloccall343 to %list_item**
  %malloccall345 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item346 = bitcast i8* %malloccall345 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item346, align 1
  %malloccall347 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %copied348 = bitcast i8* %malloccall347 to i8**
  store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.2, i32 0,
    i32 0), i8** %copied348, align 8
```

```
  %cast_ptr349 = bitcast i8** %copied348 to i8*
  %data_ptr_container350 = getelementptr inbounds %list_item, %list_item* %
    list_item346, i32 0, i32 0
  store i8* %cast_ptr349, i8** %data_ptr_container350, align 8
  %next351 = getelementptr inbounds %list_item, %list_item* %list_item346,
    i32 0, i32 1
  store %list_item* null, %list_item** %next351, align 8
  store %list_item* %list_item346, %list_item** %list344, align 8
  %malloccall352 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list353 = bitcast i8* %malloccall352 to %list_item**
  %malloccall354 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item355 = bitcast i8* %malloccall354 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item355, align 1
  %malloccall356 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %copied357 = bitcast i8* %malloccall356 to i8**
  store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.3, i32 0,
    i32 0), i8** %copied357, align 8
  %cast_ptr358 = bitcast i8** %copied357 to i8*
  %data_ptr_container359 = getelementptr inbounds %list_item, %list_item* %
    list_item355, i32 0, i32 0
  store i8* %cast_ptr358, i8** %data_ptr_container359, align 8
  %next360 = getelementptr inbounds %list_item, %list_item* %list_item355,
    i32 0, i32 1
  store %list_item* null, %list_item** %next360, align 8
  %malloccall361 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item362 = bitcast i8* %malloccall361 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item362, align 1
  %malloccall363 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %copied364 = bitcast i8* %malloccall363 to i8**
  store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.4, i32 0,
    i32 0), i8** %copied364, align 8
  %cast_ptr365 = bitcast i8** %copied364 to i8*
  %data_ptr_container366 = getelementptr inbounds %list_item, %list_item* %
    list_item362, i32 0, i32 0
  store i8* %cast_ptr365, i8** %data_ptr_container366, align 8
  %next367 = getelementptr inbounds %list_item, %list_item* %list_item362,
    i32 0, i32 1
  store %list_item* %list_item355, %list_item** %next367, align 8
  store %list_item* %list_item362, %list_item** %list353, align 8
  %_result368 = call i1 @contains_string_string(%list_item** %list353, %
    list_item** %list344)
  %malloccall369 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list370 = bitcast i8* %malloccall369 to %list_item**
  %malloccall371 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item372 = bitcast i8* %malloccall371 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item372, align 1
  %copied374 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 104, i8* %copied374, align 1
  %data_ptr_container375 = getelementptr inbounds %list_item, %list_item* %
    list_item372, i32 0, i32 0
  store i8* %copied374, i8** %data_ptr_container375, align 8
```

```
  %next376 = getelementptr inbounds %list_item, %list_item* %list_item372,
    i32 0, i32 1
  store %list_item* null, %list_item** %next376, align 8
  store %list_item* %list_item372, %list_item** %list370, align 8
  %malloccall377 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list378 = bitcast i8* %malloccall377 to %list_item**
  %malloccall379 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item380 = bitcast i8* %malloccall379 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item380, align 1
  %copied382 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 101, i8* %copied382, align 1
  %data_ptr_container383 = getelementptr inbounds %list_item, %list_item* %
    list_item380, i32 0, i32 0
  store i8* %copied382, i8** %data_ptr_container383, align 8
  %next384 = getelementptr inbounds %list_item, %list_item* %list_item380,
    i32 0, i32 1
  store %list_item* null, %list_item** %next384, align 8
  %malloccall385 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item386 = bitcast i8* %malloccall385 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item386, align 1
  %copied388 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 104, i8* %copied388, align 1
  %data_ptr_container389 = getelementptr inbounds %list_item, %list_item* %
    list_item386, i32 0, i32 0
  store i8* %copied388, i8** %data_ptr_container389, align 8
  %next390 = getelementptr inbounds %list_item, %list_item* %list_item386,
    i32 0, i32 1
  store %list_item* %list_item380, %list_item** %next390, align 8
  store %list_item* %list_item386, %list_item** %list378, align 8
  %_result391 = call i1 @contains_char_char(%list_item** %list378, %
    list_item** %list370)
  %malloccall392 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list393 = bitcast i8* %malloccall392 to %list_item**
  %malloccall394 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item395 = bitcast i8* %malloccall394 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item395, align 1
  %malloccall396 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
  %copied397 = bitcast i8* %malloccall396 to i1*
  store i1 true, i1* %copied397, align 1
  %cast_ptr398 = bitcast i1* %copied397 to i8*
  %data_ptr_container399 = getelementptr inbounds %list_item, %list_item* %
    list_item395, i32 0, i32 0
  store i8* %cast_ptr398, i8** %data_ptr_container399, align 8
  %next400 = getelementptr inbounds %list_item, %list_item* %list_item395,
    i32 0, i32 1
  store %list_item* null, %list_item** %next400, align 8
  store %list_item* %list_item395, %list_item** %list393, align 8
  %malloccall401 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list402 = bitcast i8* %malloccall401 to %list_item**
  %malloccall403 = tail call i8* @malloc(i32 ptrtoint (%list_item*
```

```
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item404 = bitcast i8* %malloccall403 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item404, align 1
  %malloccall405 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
  %copied406 = bitcast i8* %malloccall405 to i1*
  store i1 false, i1* %copied406, align 1
  %cast_ptr407 = bitcast i1* %copied406 to i8*
  %data_ptr_container408 = getelementptr inbounds %list_item, %list_item* %
    list_item404, i32 0, i32 0
  store i8* %cast_ptr407, i8** %data_ptr_container408, align 8
  %next409 = getelementptr inbounds %list_item, %list_item* %list_item404,
    i32 0, i32 1
  store %list_item* null, %list_item** %next409, align 8
  %malloccall410 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item411 = bitcast i8* %malloccall410 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item411, align 1
  %malloccall412 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
  %copied413 = bitcast i8* %malloccall412 to i1*
  store i1 true, i1* %copied413, align 1
  %cast_ptr414 = bitcast i1* %copied413 to i8*
  %data_ptr_container415 = getelementptr inbounds %list_item, %list_item* %
    list_item411, i32 0, i32 0
  store i8* %cast_ptr414, i8** %data_ptr_container415, align 8
  %next416 = getelementptr inbounds %list_item, %list_item* %list_item411,
    i32 0, i32 1
  store %list_item* %list_item404, %list_item** %next416, align 8
  store %list_item* %list_item411, %list_item** %list402, align 8
  %_result417 = call i1 @contains_bool_bool(%list_item** %list402, %
    list_item** %list393)
  %malloccall418 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list419 = bitcast i8* %malloccall418 to %list_item**
  %malloccall420 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item421 = bitcast i8* %malloccall420 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item421, align 1
  %malloccall422 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
     (double, double* null, i32 1) to i32))
  %copied423 = bitcast i8* %malloccall422 to double*
  store double 1.100000e+00, double* %copied423, align 8
  %cast_ptr424 = bitcast double* %copied423 to i8*
  %data_ptr_container425 = getelementptr inbounds %list_item, %list_item* %
    list_item421, i32 0, i32 0
  store i8* %cast_ptr424, i8** %data_ptr_container425, align 8
  %next426 = getelementptr inbounds %list_item, %list_item* %list_item421,
    i32 0, i32 1
  store %list_item* null, %list_item** %next426, align 8
  store %list_item* %list_item421, %list_item** %list419, align 8
  %malloccall427 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list428 = bitcast i8* %malloccall427 to %list_item**
  %malloccall429 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item430 = bitcast i8* %malloccall429 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item430, align 1
  %malloccall431 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
```

```
    (double, double* null, i32 1) to i32))
%copied432 = bitcast i8* %malloccall431 to double*
store double 1.200000e+00, double* %copied432, align 8
%cast_ptr433 = bitcast double* %copied432 to i8*
%data_ptr_container434 = getelementptr inbounds %list_item, %list_item* %
  list_item430, i32 0, i32 0
store i8* %cast_ptr433, i8** %data_ptr_container434, align 8
%next435 = getelementptr inbounds %list_item, %list_item* %list_item430,
  i32 0, i32 1
store %list_item* null, %list_item** %next435, align 8
%malloccall436 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item437 = bitcast i8* %malloccall436 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item437, align 1
%malloccall438 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
    (double, double* null, i32 1) to i32))
%copied439 = bitcast i8* %malloccall438 to double*
store double 1.100000e+00, double* %copied439, align 8
%cast_ptr440 = bitcast double* %copied439 to i8*
%data_ptr_container441 = getelementptr inbounds %list_item, %list_item* %
  list_item437, i32 0, i32 0
store i8* %cast_ptr440, i8** %data_ptr_container441, align 8
%next442 = getelementptr inbounds %list_item, %list_item* %list_item437,
  i32 0, i32 1
store %list_item* %list_item430, %list_item** %next442, align 8
store %list_item* %list_item437, %list_item** %list428, align 8
%_result443 = call i1 @contains_float_float(%list_item** %list428, %
  list_item** %list419)
%malloccall444 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
  i1*, i1** null, i32 1) to i32))
%list445 = bitcast i8* %malloccall444 to %list_item**
%malloccall446 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item447 = bitcast i8* %malloccall446 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item447, align 1
%malloccall448 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
  i32, i32* null, i32 1) to i32))
%copied449 = bitcast i8* %malloccall448 to i32*
store i32 1, i32* %copied449, align 4
%cast_ptr450 = bitcast i32* %copied449 to i8*
%data_ptr_container451 = getelementptr inbounds %list_item, %list_item* %
  list_item447, i32 0, i32 0
store i8* %cast_ptr450, i8** %data_ptr_container451, align 8
%next452 = getelementptr inbounds %list_item, %list_item* %list_item447,
  i32 0, i32 1
store %list_item* null, %list_item** %next452, align 8
store %list_item* %list_item447, %list_item** %list445, align 8
%_result453 = call %list_item** @remove_int_int_int_bool(%list_item** %
  list445, i32 1, i1 true)
%malloccall454 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
  i1*, i1** null, i32 1) to i32))
%list455 = bitcast i8* %malloccall454 to %list_item**
%malloccall456 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item457 = bitcast i8* %malloccall456 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item457, align 1
%malloccall458 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr
    (double, double* null, i32 1) to i32))
%copied459 = bitcast i8* %malloccall458 to double*
```

```
  store double 1.500000e+00, double* %copied459, align 8
  %cast_ptr460 = bitcast double* %copied459 to i8*
  %data_ptr_container461 = getelementptr inbounds %list_item, %list_item* %
    list_item457, i32 0, i32 0
  store i8* %cast_ptr460, i8** %data_ptr_container461, align 8
  %next462 = getelementptr inbounds %list_item, %list_item* %list_item457,
    i32 0, i32 1
  store %list_item* null, %list_item** %next462, align 8
  store %list_item* %list_item457, %list_item** %list455, align 8
  %_result463 = call %list_item** @remove_float_float_float_bool(%list_item
    ** %list455, double 1.500000e+00, i1 false)
  %malloccall464 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list465 = bitcast i8* %malloccall464 to %list_item**
  %malloccall466 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item467 = bitcast i8* %malloccall466 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item467, align 1
  %malloccall468 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1
    , i1* null, i32 1) to i32))
  %copied469 = bitcast i8* %malloccall468 to i1*
  store i1 true, i1* %copied469, align 1
  %cast_ptr470 = bitcast i1* %copied469 to i8*
  %data_ptr_container471 = getelementptr inbounds %list_item, %list_item* %
    list_item467, i32 0, i32 0
  store i8* %cast_ptr470, i8** %data_ptr_container471, align 8
  %next472 = getelementptr inbounds %list_item, %list_item* %list_item467,
    i32 0, i32 1
  store %list_item* null, %list_item** %next472, align 8
  store %list_item* %list_item467, %list_item** %list465, align 8
  %_result473 = call %list_item** @remove_bool_bool_bool_bool(%list_item** %
    list465, i1 true, i1 true)
  %malloccall474 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list475 = bitcast i8* %malloccall474 to %list_item**
  %malloccall476 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item477 = bitcast i8* %malloccall476 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item477, align 1
  %copied479 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 97, i8* %copied479, align 1
  %data_ptr_container480 = getelementptr inbounds %list_item, %list_item* %
    list_item477, i32 0, i32 0
  store i8* %copied479, i8** %data_ptr_container480, align 8
  %next481 = getelementptr inbounds %list_item, %list_item* %list_item477,
    i32 0, i32 1
  store %list_item* null, %list_item** %next481, align 8
  store %list_item* %list_item477, %list_item** %list475, align 8
  %_result482 = call %list_item** @remove_char_char_char_bool(%list_item** %
    list475, i8 97, i1 true)
  %malloccall483 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list484 = bitcast i8* %malloccall483 to %list_item**
  %malloccall485 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item486 = bitcast i8* %malloccall485 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item486, align 1
  %malloccall487 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
```

```
  i1*, i1** null, i32 1) to i32))
%copied488 = bitcast i8* %malloccall487 to i8**
store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.6, i32 0,
  i32 0), i8** %copied488, align 8
%cast_ptr489 = bitcast i8** %copied488 to i8*
%data_ptr_container490 = getelementptr inbounds %list_item, %list_item* %
  list_item486, i32 0, i32 0
store i8* %cast_ptr489, i8** %data_ptr_container490, align 8
%next491 = getelementptr inbounds %list_item, %list_item* %list_item486,
  i32 0, i32 1
store %list_item* null, %list_item** %next491, align 8
store %list_item* %list_item486, %list_item** %list484, align 8
%_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)
%malloccall493 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
  i1*, i1** null, i32 1) to i32))
%list494 = bitcast i8* %malloccall493 to %list_item**
%malloccall495 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item496 = bitcast i8* %malloccall495 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item496, align 1
%malloccall497 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
  i32, i32* null, i32 1) to i32))
%copied498 = bitcast i8* %malloccall497 to i32*
store i32 3, i32* %copied498, align 4
%cast_ptr499 = bitcast i32* %copied498 to i8*
%data_ptr_container500 = getelementptr inbounds %list_item, %list_item* %
  list_item496, i32 0, i32 0
store i8* %cast_ptr499, i8** %data_ptr_container500, align 8
%next501 = getelementptr inbounds %list_item, %list_item* %list_item496,
  i32 0, i32 1
store %list_item* null, %list_item** %next501, align 8
%malloccall502 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item503 = bitcast i8* %malloccall502 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item503, align 1
%malloccall504 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
  i32, i32* null, i32 1) to i32))
%copied505 = bitcast i8* %malloccall504 to i32*
store i32 9, i32* %copied505, align 4
%cast_ptr506 = bitcast i32* %copied505 to i8*
%data_ptr_container507 = getelementptr inbounds %list_item, %list_item* %
  list_item503, i32 0, i32 0
store i8* %cast_ptr506, i8** %data_ptr_container507, align 8
%next508 = getelementptr inbounds %list_item, %list_item* %list_item503,
  i32 0, i32 1
store %list_item* %list_item496, %list_item** %next508, align 8
%malloccall509 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item510 = bitcast i8* %malloccall509 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item510, align 1
%malloccall511 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
  i32, i32* null, i32 1) to i32))
%copied512 = bitcast i8* %malloccall511 to i32*
store i32 6, i32* %copied512, align 4
%cast_ptr513 = bitcast i32* %copied512 to i8*
%data_ptr_container514 = getelementptr inbounds %list_item, %list_item* %
  list_item510, i32 0, i32 0
```

```
  store i8* %cast_ptr513, i8** %data_ptr_container514, align 8
  %next515 = getelementptr inbounds %list_item, %list_item* %list_item510,
    i32 0, i32 1
  store %list_item* %list_item503, %list_item** %next515, align 8
  %malloccall516 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item517 = bitcast i8* %malloccall516 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item517, align 1
  %malloccall518 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
  %copied519 = bitcast i8* %malloccall518 to i32*
  store i32 2, i32* %copied519, align 4
  %cast_ptr520 = bitcast i32* %copied519 to i8*
  %data_ptr_container521 = getelementptr inbounds %list_item, %list_item* %
    list_item517, i32 0, i32 0
  store i8* %cast_ptr520, i8** %data_ptr_container521, align 8
  %next522 = getelementptr inbounds %list_item, %list_item* %list_item517,
    i32 0, i32 1
  store %list_item* %list_item510, %list_item** %next522, align 8
  %malloccall523 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item524 = bitcast i8* %malloccall523 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item524, align 1
  %malloccall525 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
  %copied526 = bitcast i8* %malloccall525 to i32*
  store i32 5, i32* %copied526, align 4
  %cast_ptr527 = bitcast i32* %copied526 to i8*
  %data_ptr_container528 = getelementptr inbounds %list_item, %list_item* %
    list_item524, i32 0, i32 0
  store i8* %cast_ptr527, i8** %data_ptr_container528, align 8
  %next529 = getelementptr inbounds %list_item, %list_item* %list_item524,
    i32 0, i32 1
  store %list_item* %list_item517, %list_item** %next529, align 8
  store %list_item* %list_item524, %list_item** %list494, align 8
  %_result530 = call i32 @max_profit_int(%list_item** %list494)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8
    ], [4 x i8]* @string.7, i32 0, i32 0), i32 %_result530)
  ret i32 0
}

define i32 @max_profit_int(%list_item** %stock_prices) {
entry:
  %potential_profit = alloca i32, align 4
  %next_price = alloca i32, align 4
  %current_price = alloca i32, align 4
  %i = alloca i32, align 4
  %for_index = alloca i32, align 4
  %max_profit = alloca i32, align 4
  %min_price = alloca i32, align 4
  %stock_prices1 = alloca %list_item**, align 8
  store %list_item** %stock_prices, %list_item*** %stock_prices1, align 8
  %stock_prices2 = load %list_item**, %list_item*** %stock_prices1, align 8
  %ilist = load %list_item*, %list_item** %stock_prices2, align 8
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 0)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %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   %malloccall62 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
        *, i1** null, i32 1) to i32))
892   %head_ptr_ptr63 = bitcast i8* %malloccall62 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   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))
910   %head_ptr_ptr = bitcast i8* %malloccall 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,
```

```
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i8**
  %data = load i8*, i8** %cast_data_ptr, align 8
  %_result11 = call i1 @strcmp(i8* %data, i8* %elem7)
  br i1 %_result11, label %then13, label %else

merge12:                                          ; preds = %else
  %retlist15 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr = load %list_item*, %list_item** %retlist15, align 8
  %l16 = load %list_item**, %list_item*** %l1, align 8
  %ilist17 = load %list_item*, %list_item** %l16, align 8
  %i18 = load i32, i32* %i, align 4
  %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %cast_data_ptr22 = bitcast i8* %data_ptr21 to i8**
  %data23 = load i8*, i8** %cast_data_ptr22, align 8
  %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_string(%list_item** %retlist15,
    i8* %data23, i32 %length24)
  store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
  %i25 = load i32, i32* %i, align 4
  %tmp26 = add i32 %i25, 1
  store i32 %tmp26, i32* %i, align 4
  br label %while

then13:                                           ; preds = %while_body
  %i14 = load i32, i32* %i, align 4
  %tmp = add i32 %i14, 1
  store i32 %tmp, i32* %i, align 4
  br label %while

else:                                             ; preds = %while_body
  br label %merge12

else30:                                           ; preds = %entry
  br label %while31

while31:                                          ; preds = %merge44, %
    else30
  %i62 = load i32, i32* %i, align 4
  %len63 = load i32, i32* %len, align 4
  %tmp64 = icmp slt i32 %i62, %len63
  br i1 %tmp64, label %while_body33, label %merge32

merge32:                                          ; preds = %while31, %
    then45
  %i65 = load i32, i32* %i, align 4
  %len66 = load i32, i32* %len, align 4
  %tmp67 = icmp ne i32 %i65, %len66
  br i1 %tmp67, label %then69, label %else108

while_body33:                                     ; preds = %while31
  %elem34 = load i8*, i8** %elem2, align 8
  %l35 = load %list_item**, %list_item*** %l1, align 8
  %ilist36 = load %list_item*, %list_item** %l35, align 8
  %i37 = load i32, i32* %i, align 4
```

```
  %_result38 = call %list_item* @list_access(%list_item* %ilist36, i32 %i37)
  %data_ptr_ptr39 = getelementptr inbounds %list_item, %list_item* %
    _result38, i32 0, i32 0
  %data_ptr40 = load i8*, i8** %data_ptr_ptr39, align 8
  %cast_data_ptr41 = bitcast i8* %data_ptr40 to i8**
  %data42 = load i8*, i8** %cast_data_ptr41, align 8
  %_result43 = call i1 @strcmp(i8* %data42, i8* %elem34)
  br i1 %_result43, label %then45, label %else47

merge44:                                          ; preds = %else47
  %retlist48 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
  %l50 = load %list_item**, %list_item*** %l1, align 8
  %ilist51 = load %list_item*, %list_item** %l50, align 8
  %i52 = load i32, i32* %i, align 4
  %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
  %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
    _result53, i32 0, i32 0
  %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
  %cast_data_ptr56 = bitcast i8* %data_ptr55 to i8**
  %data57 = load i8*, i8** %cast_data_ptr56, align 8
  %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
  %list_ptr_ptr59 = call %list_item** @insert_string(%list_item** %retlist48
    , i8* %data57, i32 %length58)
  store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
  %i60 = load i32, i32* %i, align 4
  %tmp61 = add i32 %i60, 1
  store i32 %tmp61, i32* %i, align 4
  br label %while31

then45:                                           ; preds = %while_body33
  %i46 = load i32, i32* %i, align 4
  store i32 %i46, i32* %remove_index, align 4
  br label %merge32

else47:                                           ; preds = %while_body33
  br label %merge44

merge68:                                          ; preds = %else108, %
    merge71
  br label %merge

then69:                                           ; preds = %merge32
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while70

while70:                                          ; preds = %while_body72, %
    then69
  %for_index98 = load i32, i32* %for_index, align 4
  %remove_index99 = load i32, i32* %remove_index, align 4
  %tmp100 = add i32 %remove_index99, 1
  %len101 = load i32, i32* %len, align 4
  %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %head_ptr_ptr103 = bitcast i8* %malloccall102 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr103, align 8
  %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
```

```
  %ilist105 = load %list_item*, %list_item** %range_list104, align 8
  %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
  %tmp107 = icmp slt i32 %for_index98, %length106
  br i1 %tmp107, label %while_body72, label %merge71

merge71:                                          ; preds = %while70
  br label %merge68

while_body72:                                     ; preds = %while70
  %remove_index73 = load i32, i32* %remove_index, align 4
  %tmp74 = add i32 %remove_index73, 1
  %len75 = load i32, i32* %len, align 4
  %malloccall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall76 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
  %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist77 = load %list_item*, %list_item** %range_list, align 8
  %for_index78 = load i32, i32* %for_index, align 4
  %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
    for_index78)
  %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
  %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
  %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
  %data83 = load i32, i32* %cast_data_ptr82, align 4
  store i32 %data83, i32* %index, align 4
  %for_index84 = load i32, i32* %for_index, align 4
  %tmp85 = add i32 %for_index84, 1
  store i32 %tmp85, i32* %for_index, align 4
  %retlist86 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
  %l88 = load %list_item**, %list_item*** %l1, align 8
  %ilist89 = load %list_item*, %list_item** %l88, align 8
  %index90 = load i32, i32* %index, align 4
  %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
  %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
  %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
  %cast_data_ptr94 = bitcast i8* %data_ptr93 to i8**
  %data95 = load i8*, i8** %cast_data_ptr94, align 8
  %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
  %list_ptr_ptr97 = call %list_item** @insert_string(%list_item** %retlist86
    , i8* %data95, i32 %length96)
  store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
  br label %while70

else108:                                          ; preds = %merge32
  br label %merge68
}

define %list_item** @remove_char_char_char_bool(%list_item** %l, i8 %elem,
    i1 %all) {
entry:
  %index = alloca i32, align 4
  %for_index = alloca i32, align 4
  %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
1224   br label %while
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
1233   br label %merge
1234
1235 while_body:                                       ; preds = %while
1236   %l7 = load %list_item**, %list_item*** %l1, align 8
1237   %ilist8 = load %list_item*, %list_item** %l7, align 8
1238   %i9 = load i32, i32* %i, align 4
1239   %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1240   %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
        i32 0, i32 0
1241   %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1242   %data = load i8, i8* %data_ptr, align 1
1243   %elem10 = load i8, i8* %elem2, align 1
1244   %tmp = icmp eq i8 %data, %elem10
1245   br i1 %tmp, label %then12, label %else
1246
1247 merge11:                                          ; preds = %else
1248   %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1249   %list_ptr = load %list_item*, %list_item** %retlist15, align 8
1250   %l16 = load %list_item**, %list_item*** %l1, align 8
1251   %ilist17 = load %list_item*, %list_item** %l16, align 8
```

```
  %i18 = load i32, i32* %i, align 4
  %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %data22 = load i8, i8* %data_ptr21, align 1
  %length23 = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_char(%list_item** %retlist15, i8
     %data22, i32 %length23)
  store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
  %i24 = load i32, i32* %i, align 4
  %tmp25 = add i32 %i24, 1
  store i32 %tmp25, i32* %i, align 4
  br label %while

then12:                                           ; preds = %while_body
  %i13 = load i32, i32* %i, align 4
  %tmp14 = add i32 %i13, 1
  store i32 %tmp14, i32* %i, align 4
  br label %while

else:                                             ; preds = %while_body
  br label %merge11

else29:                                           ; preds = %entry
  br label %while30

while30:                                          ; preds = %merge42, %
    else29
  %i59 = load i32, i32* %i, align 4
  %len60 = load i32, i32* %len, align 4
  %tmp61 = icmp slt i32 %i59, %len60
  br i1 %tmp61, label %while_body32, label %merge31

merge31:                                          ; preds = %while30, %
    then43
  %i62 = load i32, i32* %i, align 4
  %len63 = load i32, i32* %len, align 4
  %tmp64 = icmp ne i32 %i62, %len63
  br i1 %tmp64, label %then66, label %else103

while_body32:                                     ; preds = %while30
  %l33 = load %list_item**, %list_item*** %l1, align 8
  %ilist34 = load %list_item*, %list_item** %l33, align 8
  %i35 = load i32, i32* %i, align 4
  %_result36 = call %list_item* @list_access(%list_item* %ilist34, i32 %i35)
  %data_ptr_ptr37 = getelementptr inbounds %list_item, %list_item* %
    _result36, i32 0, i32 0
  %data_ptr38 = load i8*, i8** %data_ptr_ptr37, align 8
  %data39 = load i8, i8* %data_ptr38, align 1
  %elem40 = load i8, i8* %elem2, align 1
  %tmp41 = icmp eq i8 %data39, %elem40
  br i1 %tmp41, label %then43, label %else45

merge42:                                          ; preds = %else45
  %retlist46 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr47 = load %list_item*, %list_item** %retlist46, align 8
  %l48 = load %list_item**, %list_item*** %l1, align 8
  %ilist49 = load %list_item*, %list_item** %l48, align 8
```

```
  %i50 = load i32, i32* %i, align 4
  %_result51 = call %list_item* @list_access(%list_item* %ilist49, i32 %i50)
  %data_ptr_ptr52 = getelementptr inbounds %list_item, %list_item* %
    _result51, i32 0, i32 0
  %data_ptr53 = load i8*, i8** %data_ptr_ptr52, align 8
  %data54 = load i8, i8* %data_ptr53, align 1
  %length55 = call i32 @list_length(%list_item* %list_ptr47, i32 0)
  %list_ptr_ptr56 = call %list_item** @insert_char(%list_item** %retlist46,
    i8 %data54, i32 %length55)
  store %list_item** %list_ptr_ptr56, %list_item*** %retlist, align 8
  %i57 = load i32, i32* %i, align 4
  %tmp58 = add i32 %i57, 1
  store i32 %tmp58, i32* %i, align 4
  br label %while30

then43:                                           ; preds = %while_body32
  %i44 = load i32, i32* %i, align 4
  store i32 %i44, i32* %remove_index, align 4
  br label %merge31

else45:                                           ; preds = %while_body32
  br label %merge42

merge65:                                          ; preds = %else103, %
    merge68
  br label %merge

then66:                                           ; preds = %merge31
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while67

while67:                                          ; preds = %while_body69, %
    then66
  %for_index93 = load i32, i32* %for_index, align 4
  %remove_index94 = load i32, i32* %remove_index, align 4
  %tmp95 = add i32 %remove_index94, 1
  %len96 = load i32, i32* %len, align 4
  %malloccall97 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr98 = bitcast i8* %malloccall97 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr98, align 8
  %range_list99 = call %list_item** @range_function(i32 %tmp95, i32 %len96,
    %list_item** %head_ptr_ptr98, i32 0)
  %ilist100 = load %list_item*, %list_item** %range_list99, align 8
  %length101 = call i32 @list_length(%list_item* %ilist100, i32 0)
  %tmp102 = icmp slt i32 %for_index93, %length101
  br i1 %tmp102, label %while_body69, label %merge68

merge68:                                          ; preds = %while67
  br label %merge65

while_body69:                                     ; preds = %while67
  %remove_index70 = load i32, i32* %remove_index, align 4
  %tmp71 = add i32 %remove_index70, 1
  %len72 = load i32, i32* %len, align 4
  %malloccall73 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall73 to %list_item**
```

```
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
  %range_list = call %list_item** @range_function(i32 %tmp71, i32 %len72, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist74 = load %list_item*, %list_item** %range_list, align 8
  %for_index75 = load i32, i32* %for_index, align 4
  %_result76 = call %list_item* @list_access(%list_item* %ilist74, i32 %
    for_index75)
  %data_ptr_ptr77 = getelementptr inbounds %list_item, %list_item* %
    _result76, i32 0, i32 0
  %data_ptr78 = load i8*, i8** %data_ptr_ptr77, align 8
  %cast_data_ptr = bitcast i8* %data_ptr78 to i32*
  %data79 = load i32, i32* %cast_data_ptr, align 4
  store i32 %data79, i32* %index, align 4
  %for_index80 = load i32, i32* %for_index, align 4
  %tmp81 = add i32 %for_index80, 1
  store i32 %tmp81, i32* %for_index, align 4
  %retlist82 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr83 = load %list_item*, %list_item** %retlist82, align 8
  %l84 = load %list_item**, %list_item*** %l1, align 8
  %ilist85 = load %list_item*, %list_item** %l84, align 8
  %index86 = load i32, i32* %index, align 4
  %_result87 = call %list_item* @list_access(%list_item* %ilist85, i32 %
    index86)
  %data_ptr_ptr88 = getelementptr inbounds %list_item, %list_item* %
    _result87, i32 0, i32 0
  %data_ptr89 = load i8*, i8** %data_ptr_ptr88, align 8
  %data90 = load i8, i8* %data_ptr89, align 1
  %length91 = call i32 @list_length(%list_item* %list_ptr83, i32 0)
  %list_ptr_ptr92 = call %list_item** @insert_char(%list_item** %retlist82,
    i8 %data90, i32 %length91)
  store %list_item** %list_ptr_ptr92, %list_item*** %retlist, align 8
  br label %while67

else103:                                          ; preds = %merge31
  br label %merge65
}

define %list_item** @remove_bool_bool_bool_bool(%list_item** %l, i1 %elem,
    i1 %all) {
entry:
  %index = alloca i32, align 4
  %for_index = alloca i32, align 4
  %remove_index = alloca i32, align 4
  %len = alloca i32, align 4
  %i = alloca i32, align 4
  %retlist = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %elem2 = alloca i1, align 1
  store i1 %elem, i1* %elem2, align 1
  %all3 = alloca i1, align 1
  store i1 %all, i1* %all3, align 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %retlist, align 8
  store i32 0, i32* %i, align 4
  %l4 = load %list_item**, %list_item*** %l1, align 8
```

```
  %ilist = load %list_item*, %list_item** %l4, align 8
  %length = call i32 @list_length(%list_item* %ilist, i32 0)
  store i32 %length, i32* %len, align 4
  store i32 0, i32* %remove_index, align 4
  %all5 = load i1, i1* %all3, align 1
  br i1 %all5, label %then, label %else30

merge:                                            ; preds = %merge68, %
    merge6
  %retlist109 = load %list_item**, %list_item*** %retlist, align 8
  ret %list_item** %retlist109

then:                                             ; preds = %entry
  br label %while

while:                                            ; preds = %merge11, %
    then12, %then
  %i27 = load i32, i32* %i, align 4
  %len28 = load i32, i32* %len, align 4
  %tmp29 = icmp slt i32 %i27, %len28
  br i1 %tmp29, label %while_body, label %merge6

merge6:                                           ; preds = %while
  br label %merge

while_body:                                       ; preds = %while
  %l7 = load %list_item**, %list_item*** %l1, align 8
  %ilist8 = load %list_item*, %list_item** %l7, align 8
  %i9 = load i32, i32* %i, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i1*
  %data = load i1, i1* %cast_data_ptr, align 1
  %elem10 = load i1, i1* %elem2, align 1
  %tmp = icmp eq i1 %data, %elem10
  br i1 %tmp, label %then12, label %else

merge11:                                          ; preds = %else
  %retlist15 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr = load %list_item*, %list_item** %retlist15, align 8
  %l16 = load %list_item**, %list_item*** %l1, align 8
  %ilist17 = load %list_item*, %list_item** %l16, align 8
  %i18 = load i32, i32* %i, align 4
  %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %cast_data_ptr22 = bitcast i8* %data_ptr21 to i1*
  %data23 = load i1, i1* %cast_data_ptr22, align 1
  %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %retlist15, i1
     %data23, i32 %length24)
  store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
  %i25 = load i32, i32* %i, align 4
  %tmp26 = add i32 %i25, 1
  store i32 %tmp26, i32* %i, align 4
  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, %
       else30
1477   %i62 = load i32, i32* %i, align 4
1478   %len63 = load i32, i32* %len, align 4
1479   %tmp64 = icmp slt i32 %i62, %len63
1480   br i1 %tmp64, label %while_body33, label %merge32
1481
1482 merge32:                                          ; preds = %while31, %
       then45
1483   %i65 = load i32, i32* %i, align 4
1484   %len66 = load i32, i32* %len, align 4
1485   %tmp67 = icmp ne i32 %i65, %len66
1486   br i1 %tmp67, label %then69, label %else108
1487
1488 while_body33:                                     ; preds = %while31
1489   %l34 = load %list_item**, %list_item*** %l1, align 8
1490   %ilist35 = load %list_item*, %list_item** %l34, align 8
1491   %i36 = load i32, i32* %i, align 4
1492   %_result37 = call %list_item* @list_access(%list_item* %ilist35, i32 %i36)
1493   %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
       _result37, i32 0, i32 0
1494   %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
1495   %cast_data_ptr40 = bitcast i8* %data_ptr39 to i1*
1496   %data41 = load i1, i1* %cast_data_ptr40, align 1
1497   %elem42 = load i1, i1* %elem2, align 1
1498   %tmp43 = icmp eq i1 %data41, %elem42
1499   br i1 %tmp43, label %then45, label %else47
1500
1501 merge44:                                          ; preds = %else47
1502   %retlist48 = load %list_item**, %list_item*** %retlist, align 8
1503   %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
1504   %l50 = load %list_item**, %list_item*** %l1, align 8
1505   %ilist51 = load %list_item*, %list_item** %l50, align 8
1506   %i52 = load i32, i32* %i, align 4
1507   %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
1508   %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
       _result53, i32 0, i32 0
1509   %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
1510   %cast_data_ptr56 = bitcast i8* %data_ptr55 to i1*
1511   %data57 = load i1, i1* %cast_data_ptr56, align 1
1512   %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
1513   %list_ptr_ptr59 = call %list_item** @insert_bool(%list_item** %retlist48,
       i1 %data57, i32 %length58)
1514   store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
1515   %i60 = load i32, i32* %i, align 4
1516   %tmp61 = add i32 %i60, 1
```

```
  store i32 %tmp61, i32* %i, align 4
  br label %while31

then45:                                           ; preds = %while_body33
  %i46 = load i32, i32* %i, align 4
  store i32 %i46, i32* %remove_index, align 4
  br label %merge32

else47:                                           ; preds = %while_body33
  br label %merge44

merge68:                                          ; preds = %else108, %
    merge71
  br label %merge

then69:                                           ; preds = %merge32
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while70

while70:                                          ; preds = %while_body72, %
    then69
  %for_index98 = load i32, i32* %for_index, align 4
  %remove_index99 = load i32, i32* %remove_index, align 4
  %tmp100 = add i32 %remove_index99, 1
  %len101 = load i32, i32* %len, align 4
  %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %head_ptr_ptr103 = bitcast i8* %malloccall102 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr103, align 8
  %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
  %ilist105 = load %list_item*, %list_item** %range_list104, align 8
  %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
  %tmp107 = icmp slt i32 %for_index98, %length106
  br i1 %tmp107, label %while_body72, label %merge71

merge71:                                          ; preds = %while70
  br label %merge68

while_body72:                                     ; preds = %while70
  %remove_index73 = load i32, i32* %remove_index, align 4
  %tmp74 = add i32 %remove_index73, 1
  %len75 = load i32, i32* %len, align 4
  %malloccall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall76 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
  %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist77 = load %list_item*, %list_item** %range_list, align 8
  %for_index78 = load i32, i32* %for_index, align 4
  %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
    for_index78)
  %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
  %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
  %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
  %data83 = load i32, i32* %cast_data_ptr82, align 4
```

```
  store i32 %data83, i32* %index, align 4
  %for_index84 = load i32, i32* %for_index, align 4
  %tmp85 = add i32 %for_index84, 1
  store i32 %tmp85, i32* %for_index, align 4
  %retlist86 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
  %l88 = load %list_item**, %list_item*** %l1, align 8
  %ilist89 = load %list_item*, %list_item** %l88, align 8
  %index90 = load i32, i32* %index, align 4
  %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
  %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
  %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
  %cast_data_ptr94 = bitcast i8* %data_ptr93 to i1*
  %data95 = load i1, i1* %cast_data_ptr94, align 1
  %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
  %list_ptr_ptr97 = call %list_item** @insert_bool(%list_item** %retlist86,
    i1 %data95, i32 %length96)
  store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
  br label %while70

else108:                                          ; preds = %merge32
  br label %merge68
}

define %list_item** @remove_float_float_float_bool(%list_item** %l, double %
    elem, i1 %all) {
entry:
  %index = alloca i32, align 4
  %for_index = alloca i32, align 4
  %remove_index = alloca i32, align 4
  %len = alloca i32, align 4
  %i = alloca i32, align 4
  %retlist = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %elem2 = alloca double, align 8
  store double %elem, double* %elem2, align 8
  %all3 = alloca i1, align 1
  store i1 %all, i1* %all3, align 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %retlist, align 8
  store i32 0, i32* %i, align 4
  %l4 = load %list_item**, %list_item*** %l1, align 8
  %ilist = load %list_item*, %list_item** %l4, align 8
  %length = call i32 @list_length(%list_item* %ilist, i32 0)
  store i32 %length, i32* %len, align 4
  store i32 0, i32* %remove_index, align 4
  %all5 = load i1, i1* %all3, align 1
  br i1 %all5, label %then, label %else30

merge:                                            ; preds = %merge68, %
    merge6
  %retlist109 = load %list_item**, %list_item*** %retlist, align 8
  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
```

```
merge68:                                          ; preds = %else108, %
    merge71
  br label %merge

then69:                                           ; preds = %merge32
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while70

while70:                                          ; preds = %while_body72, %
    then69
  %for_index98 = load i32, i32* %for_index, align 4
  %remove_index99 = load i32, i32* %remove_index, align 4
  %tmp100 = add i32 %remove_index99, 1
  %len101 = load i32, i32* %len, align 4
  %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %head_ptr_ptr103 = bitcast i8* %malloccall102 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr103, align 8
  %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
  %ilist105 = load %list_item*, %list_item** %range_list104, align 8
  %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
  %tmp107 = icmp slt i32 %for_index98, %length106
  br i1 %tmp107, label %while_body72, label %merge71

merge71:                                          ; preds = %while70
  br label %merge68

while_body72:                                     ; preds = %while70
  %remove_index73 = load i32, i32* %remove_index, align 4
  %tmp74 = add i32 %remove_index73, 1
  %len75 = load i32, i32* %len, align 4
  %malloccall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall76 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
  %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist77 = load %list_item*, %list_item** %range_list, align 8
  %for_index78 = load i32, i32* %for_index, align 4
  %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
    for_index78)
  %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
  %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
  %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
  %data83 = load i32, i32* %cast_data_ptr82, align 4
  store i32 %data83, i32* %index, align 4
  %for_index84 = load i32, i32* %for_index, align 4
  %tmp85 = add i32 %for_index84, 1
  store i32 %tmp85, i32* %for_index, align 4
  %retlist86 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
  %l88 = load %list_item**, %list_item*** %l1, align 8
  %ilist89 = load %list_item*, %list_item** %l88, align 8
  %index90 = load i32, i32* %index, align 4
  %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
```

```
  %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
  %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
  %cast_data_ptr94 = bitcast i8* %data_ptr93 to double*
  %data95 = load double, double* %cast_data_ptr94, align 8
  %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
  %list_ptr_ptr97 = call %list_item** @insert_float(%list_item** %retlist86,
     double %data95, i32 %length96)
  store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
  br label %while70

else108:                                          ; preds = %merge32
  br label %merge68
}

define %list_item** @remove_int_int_int_bool(%list_item** %l, i32 %elem, i1
    %all) {
entry:
  %index = alloca i32, align 4
  %for_index = alloca i32, align 4
  %remove_index = alloca i32, align 4
  %len = alloca i32, align 4
  %i = alloca i32, align 4
  %retlist = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %elem2 = alloca i32, align 4
  store i32 %elem, i32* %elem2, align 4
  %all3 = alloca i1, align 1
  store i1 %all, i1* %all3, align 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %retlist, align 8
  store i32 0, i32* %i, align 4
  %l4 = load %list_item**, %list_item*** %l1, align 8
  %ilist = load %list_item*, %list_item** %l4, align 8
  %length = call i32 @list_length(%list_item* %ilist, i32 0)
  store i32 %length, i32* %len, align 4
  store i32 0, i32* %remove_index, align 4
  %all5 = load i1, i1* %all3, align 1
  br i1 %all5, label %then, label %else30

merge:                                            ; preds = %merge68, %
    merge6
  %retlist109 = load %list_item**, %list_item*** %retlist, align 8
  ret %list_item** %retlist109

then:                                             ; preds = %entry
  br label %while

while:                                            ; preds = %merge11, %
    then12, %then
  %i27 = load i32, i32* %i, align 4
  %len28 = load i32, i32* %len, align 4
  %tmp29 = icmp slt i32 %i27, %len28
  br i1 %tmp29, label %while_body, label %merge6

```

```
merge6:                                           ; preds = %while
  br label %merge

while_body:                                       ; preds = %while
  %l7 = load %list_item**, %list_item*** %l1, align 8
  %ilist8 = load %list_item*, %list_item** %l7, align 8
  %i9 = load i32, i32* %i, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i32*
  %data = load i32, i32* %cast_data_ptr, align 4
  %elem10 = load i32, i32* %elem2, align 4
  %tmp = icmp eq i32 %data, %elem10
  br i1 %tmp, label %then12, label %else

merge11:                                          ; preds = %else
  %retlist15 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr = load %list_item*, %list_item** %retlist15, align 8
  %l16 = load %list_item**, %list_item*** %l1, align 8
  %ilist17 = load %list_item*, %list_item** %l16, align 8
  %i18 = load i32, i32* %i, align 4
  %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %cast_data_ptr22 = bitcast i8* %data_ptr21 to i32*
  %data23 = load i32, i32* %cast_data_ptr22, align 4
  %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_int(%list_item** %retlist15, i32
     %data23, i32 %length24)
  store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
  %i25 = load i32, i32* %i, align 4
  %tmp26 = add i32 %i25, 1
  store i32 %tmp26, i32* %i, align 4
  br label %while

then12:                                           ; preds = %while_body
  %i13 = load i32, i32* %i, align 4
  %tmp14 = add i32 %i13, 1
  store i32 %tmp14, i32* %i, align 4
  br label %while

else:                                             ; preds = %while_body
  br label %merge11

else30:                                           ; preds = %entry
  br label %while31

while31:                                          ; preds = %merge44, %
    else30
  %i62 = load i32, i32* %i, align 4
  %len63 = load i32, i32* %len, align 4
  %tmp64 = icmp slt i32 %i62, %len63
  br i1 %tmp64, label %while_body33, label %merge32

merge32:                                          ; preds = %while31, %
    then45
```

```
  %i65 = load i32, i32* %i, align 4
  %len66 = load i32, i32* %len, align 4
  %tmp67 = icmp ne i32 %i65, %len66
  br i1 %tmp67, label %then69, label %else108

while_body33:                                     ; preds = %while31
  %l34 = load %list_item**, %list_item*** %l1, align 8
  %ilist35 = load %list_item*, %list_item** %l34, align 8
  %i36 = load i32, i32* %i, align 4
  %_result37 = call %list_item* @list_access(%list_item* %ilist35, i32 %i36)
  %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
    _result37, i32 0, i32 0
  %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
  %cast_data_ptr40 = bitcast i8* %data_ptr39 to i32*
  %data41 = load i32, i32* %cast_data_ptr40, align 4
  %elem42 = load i32, i32* %elem2, align 4
  %tmp43 = icmp eq i32 %data41, %elem42
  br i1 %tmp43, label %then45, label %else47

merge44:                                          ; preds = %else47
  %retlist48 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
  %l50 = load %list_item**, %list_item*** %l1, align 8
  %ilist51 = load %list_item*, %list_item** %l50, align 8
  %i52 = load i32, i32* %i, align 4
  %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
  %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
    _result53, i32 0, i32 0
  %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
  %cast_data_ptr56 = bitcast i8* %data_ptr55 to i32*
  %data57 = load i32, i32* %cast_data_ptr56, align 4
  %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
  %list_ptr_ptr59 = call %list_item** @insert_int(%list_item** %retlist48,
    i32 %data57, i32 %length58)
  store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
  %i60 = load i32, i32* %i, align 4
  %tmp61 = add i32 %i60, 1
  store i32 %tmp61, i32* %i, align 4
  br label %while31

then45:                                           ; preds = %while_body33
  %i46 = load i32, i32* %i, align 4
  store i32 %i46, i32* %remove_index, align 4
  br label %merge32

else47:                                           ; preds = %while_body33
  br label %merge44

merge68:                                          ; preds = %else108, %
    merge71
  br label %merge

then69:                                           ; preds = %merge32
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while70

while70:                                          ; preds = %while_body72, %
    then69
```

```
  %for_index98 = load i32, i32* %for_index, align 4
  %remove_index99 = load i32, i32* %remove_index, align 4
  %tmp100 = add i32 %remove_index99, 1
  %len101 = load i32, i32* %len, align 4
  %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %head_ptr_ptr103 = bitcast i8* %malloccall102 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr103, align 8
  %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
  %ilist105 = load %list_item*, %list_item** %range_list104, align 8
  %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
  %tmp107 = icmp slt i32 %for_index98, %length106
  br i1 %tmp107, label %while_body72, label %merge71

merge71:                                          ; preds = %while70
  br label %merge68

while_body72:                                     ; preds = %while70
  %remove_index73 = load i32, i32* %remove_index, align 4
  %tmp74 = add i32 %remove_index73, 1
  %len75 = load i32, i32* %len, align 4
  %malloccall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall76 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
  %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist77 = load %list_item*, %list_item** %range_list, align 8
  %for_index78 = load i32, i32* %for_index, align 4
  %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
    for_index78)
  %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
  %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
  %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
  %data83 = load i32, i32* %cast_data_ptr82, align 4
  store i32 %data83, i32* %index, align 4
  %for_index84 = load i32, i32* %for_index, align 4
  %tmp85 = add i32 %for_index84, 1
  store i32 %tmp85, i32* %for_index, align 4
  %retlist86 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
  %l88 = load %list_item**, %list_item*** %l1, align 8
  %ilist89 = load %list_item*, %list_item** %l88, align 8
  %index90 = load i32, i32* %index, align 4
  %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
  %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
  %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
  %cast_data_ptr94 = bitcast i8* %data_ptr93 to i32*
  %data95 = load i32, i32* %cast_data_ptr94, align 4
  %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
  %list_ptr_ptr97 = call %list_item** @insert_int(%list_item** %retlist86,
    i32 %data95, i32 %length96)
  store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
  br label %while70

```

```
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   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
         i1** null, i32 1) to i32))
2009   %list = bitcast i8* %malloccall 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 %
        for_index4)
2034   %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
        i32 0, i32 0
2035   %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
2036   %cast_data_ptr = bitcast i8* %data_ptr to double*
2037   %data = load double, double* %cast_data_ptr, align 8
2038   store double %data, double* %item, align 8
2039   %for_index5 = load i32, i32* %for_index, align 4
2040   %tmp = add i32 %for_index5, 1
2041   store i32 %tmp, i32* %for_index, align 4
2042   %res6 = load %list_item**, %list_item*** %res, align 8
2043   %list_ptr = load %list_item*, %list_item** %res6, align 8
2044   %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
2045   %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
        false, i32 %length)
```

```
  store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
  store i32 0, i32* %for_index7, align 4
  store double 0.000000e+00, double* %ref, align 8
  br label %while8

while8:                                           ; preds = %merge24, %
    while_body
  %for_index32 = load i32, i32* %for_index7, align 4
  %l33 = load %list_item**, %list_item*** %l1, align 8
  %ilist34 = load %list_item*, %list_item** %l33, align 8
  %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
  %tmp36 = icmp slt i32 %for_index32, %length35
  br i1 %tmp36, label %while_body10, label %merge9

merge9:                                           ; preds = %while8
  br label %while

while_body10:                                     ; preds = %while8
  %l11 = load %list_item**, %list_item*** %l1, align 8
  %ilist12 = load %list_item*, %list_item** %l11, align 8
  %for_index13 = load i32, i32* %for_index7, align 4
  %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
  %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
  %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
  %cast_data_ptr17 = bitcast i8* %data_ptr16 to double*
  %data18 = load double, double* %cast_data_ptr17, align 8
  store double %data18, double* %ref, align 8
  %for_index19 = load i32, i32* %for_index7, align 4
  %tmp20 = add i32 %for_index19, 1
  store i32 %tmp20, i32* %for_index7, align 4
  %ref21 = load double, double* %ref, align 8
  %item22 = load double, double* %item, align 8
  %tmp23 = fcmp oeq double %ref21, %item22
  br i1 %tmp23, label %then, label %else

merge24:                                          ; preds = %else, %then
  br label %while8

then:                                             ; preds = %while_body10
  %res25 = load %list_item**, %list_item*** %res, align 8
  %ilist26 = load %list_item*, %list_item** %res25, align 8
  %res27 = load %list_item**, %list_item*** %res, align 8
  %ilist28 = load %list_item*, %list_item** %res27, align 8
  %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
  %tmp30 = sub i32 %length29, 1
  %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
  %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
  %malloccall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
     i1* null, i32 1) to i32))
  %copy_ptr = bitcast i8* %malloccall31 to i1*
  store i1 true, i1* %copy_ptr, align 1
  %ccopy = bitcast i1* %copy_ptr to i8*
  store i8* %ccopy, i8** %data_ptpt, align 8
  br label %merge24

else:                                             ; preds = %while_body10
```

```
  br label %merge24

while43:                                          ; preds = %merge58, %merge
  %for_index61 = load i32, i32* %for_index42, align 4
  %res62 = load %list_item**, %list_item*** %res, align 8
  %ilist63 = load %list_item*, %list_item** %res62, align 8
  %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
  %tmp65 = icmp slt i32 %for_index61, %length64
  br i1 %tmp65, label %while_body45, label %merge44

merge44:                                          ; preds = %while43
  ret i1 true

while_body45:                                     ; preds = %while43
  %res46 = load %list_item**, %list_item*** %res, align 8
  %ilist47 = load %list_item*, %list_item** %res46, align 8
  %for_index48 = load i32, i32* %for_index42, align 4
  %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
    for_index48)
  %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
    _result49, i32 0, i32 0
  %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
  %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
  %data53 = load i1, i1* %cast_data_ptr52, align 1
  store i1 %data53, i1* %r, align 1
  %for_index54 = load i32, i32* %for_index42, align 4
  %tmp55 = add i32 %for_index54, 1
  store i32 %tmp55, i32* %for_index42, align 4
  %r56 = load i1, i1* %r, align 1
  %tmp57 = xor i1 %r56, true
  br i1 %tmp57, label %then59, label %else60

merge58:                                          ; preds = %else60
  br label %while43

then59:                                           ; preds = %while_body45
  ret i1 false

else60:                                           ; preds = %while_body45
  br label %merge58
}

define i1 @contains_bool_bool(%list_item** %l, %list_item** %items) {
entry:
  %r = alloca i1, align 1
  %for_index42 = alloca i32, align 4
  %ref = alloca i1, align 1
  %for_index7 = alloca i32, align 4
  %item = alloca i1, align 1
  %for_index = alloca i32, align 4
  %res = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %items2 = alloca %list_item**, align 8
  store %list_item** %items, %list_item*** %items2, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
```

```
  store %list_item** %list, %list_item*** %res, align 8
  store i32 0, i32* %for_index, align 4
  store i1 false, i1* %item, align 1
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index37 = load i32, i32* %for_index, align 4
  %items38 = load %list_item**, %list_item*** %items2, align 8
  %ilist39 = load %list_item*, %list_item** %items38, align 8
  %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
  %tmp41 = icmp slt i32 %for_index37, %length40
  br i1 %tmp41, label %while_body, label %merge

merge:                                            ; preds = %while
  store i32 0, i32* %for_index42, align 4
  store i1 false, i1* %r, align 1
  br label %while43

while_body:                                       ; preds = %while
  %items3 = load %list_item**, %list_item*** %items2, align 8
  %ilist = load %list_item*, %list_item** %items3, align 8
  %for_index4 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i1*
  %data = load i1, i1* %cast_data_ptr, align 1
  store i1 %data, i1* %item, align 1
  %for_index5 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index5, 1
  store i32 %tmp, i32* %for_index, align 4
  %res6 = load %list_item**, %list_item*** %res, align 8
  %list_ptr = load %list_item*, %list_item** %res6, align 8
  %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
  store i32 0, i32* %for_index7, align 4
  store i1 false, i1* %ref, align 1
  br label %while8

while8:                                           ; preds = %merge24, %
    while_body
  %for_index32 = load i32, i32* %for_index7, align 4
  %l33 = load %list_item**, %list_item*** %l1, align 8
  %ilist34 = load %list_item*, %list_item** %l33, align 8
  %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
  %tmp36 = icmp slt i32 %for_index32, %length35
  br i1 %tmp36, label %while_body10, label %merge9

merge9:                                           ; preds = %while8
  br label %while

while_body10:                                     ; preds = %while8
  %l11 = load %list_item**, %list_item*** %l1, align 8
  %ilist12 = load %list_item*, %list_item** %l11, align 8
  %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   %malloccall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
         i1* null, i32 1) to i32))
2238   %copy_ptr = bitcast i8* %malloccall31 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
```

```
  %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
  %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
  %data53 = load i1, i1* %cast_data_ptr52, align 1
  store i1 %data53, i1* %r, align 1
  %for_index54 = load i32, i32* %for_index42, align 4
  %tmp55 = add i32 %for_index54, 1
  store i32 %tmp55, i32* %for_index42, align 4
  %r56 = load i1, i1* %r, align 1
  %tmp57 = xor i1 %r56, true
  br i1 %tmp57, label %then59, label %else60

merge58:                                          ; preds = %else60
  br label %while43

then59:                                           ; preds = %while_body45
  ret i1 false

else60:                                           ; preds = %while_body45
  br label %merge58
}

define i1 @contains_char_char(%list_item** %l, %list_item** %items) {
entry:
  %r = alloca i1, align 1
  %for_index41 = alloca i32, align 4
  %ref = alloca i8, align 1
  %for_index7 = alloca i32, align 4
  %item = alloca i8, align 1
  %for_index = alloca i32, align 4
  %res = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %items2 = alloca %list_item**, align 8
  store %list_item** %items, %list_item*** %items2, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %res, align 8
  store i32 0, i32* %for_index, align 4
  store i8 0, i8* %item, align 1
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index36 = load i32, i32* %for_index, align 4
  %items37 = load %list_item**, %list_item*** %items2, align 8
  %ilist38 = load %list_item*, %list_item** %items37, align 8
  %length39 = call i32 @list_length(%list_item* %ilist38, i32 0)
  %tmp40 = icmp slt i32 %for_index36, %length39
  br i1 %tmp40, label %while_body, label %merge

merge:                                            ; preds = %while
  store i32 0, i32* %for_index41, align 4
  store i1 false, i1* %r, align 1
  br label %while42

while_body:                                       ; preds = %while
  %items3 = load %list_item**, %list_item*** %items2, align 8
  %ilist = load %list_item*, %list_item** %items3, align 8
```

```
  %for_index4 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %data = load i8, i8* %data_ptr, align 1
  store i8 %data, i8* %item, align 1
  %for_index5 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index5, 1
  store i32 %tmp, i32* %for_index, align 4
  %res6 = load %list_item**, %list_item*** %res, align 8
  %list_ptr = load %list_item*, %list_item** %res6, align 8
  %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
  store i32 0, i32* %for_index7, align 4
  store i8 0, i8* %ref, align 1
  br label %while8

while8:                                           ; preds = %merge23, %
    while_body
  %for_index31 = load i32, i32* %for_index7, align 4
  %l32 = load %list_item**, %list_item*** %l1, align 8
  %ilist33 = load %list_item*, %list_item** %l32, align 8
  %length34 = call i32 @list_length(%list_item* %ilist33, i32 0)
  %tmp35 = icmp slt i32 %for_index31, %length34
  br i1 %tmp35, label %while_body10, label %merge9

merge9:                                           ; preds = %while8
  br label %while

while_body10:                                     ; preds = %while8
  %l11 = load %list_item**, %list_item*** %l1, align 8
  %ilist12 = load %list_item*, %list_item** %l11, align 8
  %for_index13 = load i32, i32* %for_index7, align 4
  %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
  %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
  %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
  %data17 = load i8, i8* %data_ptr16, align 1
  store i8 %data17, i8* %ref, align 1
  %for_index18 = load i32, i32* %for_index7, align 4
  %tmp19 = add i32 %for_index18, 1
  store i32 %tmp19, i32* %for_index7, align 4
  %ref20 = load i8, i8* %ref, align 1
  %item21 = load i8, i8* %item, align 1
  %tmp22 = icmp eq i8 %ref20, %item21
  br i1 %tmp22, label %then, label %else

merge23:                                          ; preds = %else, %then
  br label %while8

then:                                             ; preds = %while_body10
  %res24 = load %list_item**, %list_item*** %res, align 8
  %ilist25 = load %list_item*, %list_item** %res24, align 8
  %res26 = load %list_item**, %list_item*** %res, align 8
```

```
  %ilist27 = load %list_item*, %list_item** %res26, align 8
  %length28 = call i32 @list_length(%list_item* %ilist27, i32 0)
  %tmp29 = sub i32 %length28, 1
  %result = call %list_item* @list_access(%list_item* %ilist25, i32 %tmp29)
  %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
  %malloccall30 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
     i1* null, i32 1) to i32))
  %copy_ptr = bitcast i8* %malloccall30 to i1*
  store i1 true, i1* %copy_ptr, align 1
  %ccopy = bitcast i1* %copy_ptr to i8*
  store i8* %ccopy, i8** %data_ptpt, align 8
  br label %merge23

else:                                             ; preds = %while_body10
  br label %merge23

while42:                                          ; preds = %merge56, %merge
  %for_index59 = load i32, i32* %for_index41, align 4
  %res60 = load %list_item**, %list_item*** %res, align 8
  %ilist61 = load %list_item*, %list_item** %res60, align 8
  %length62 = call i32 @list_length(%list_item* %ilist61, i32 0)
  %tmp63 = icmp slt i32 %for_index59, %length62
  br i1 %tmp63, label %while_body44, label %merge43

merge43:                                          ; preds = %while42
  ret i1 true

while_body44:                                     ; preds = %while42
  %res45 = load %list_item**, %list_item*** %res, align 8
  %ilist46 = load %list_item*, %list_item** %res45, align 8
  %for_index47 = load i32, i32* %for_index41, align 4
  %_result48 = call %list_item* @list_access(%list_item* %ilist46, i32 %
    for_index47)
  %data_ptr_ptr49 = getelementptr inbounds %list_item, %list_item* %
    _result48, i32 0, i32 0
  %data_ptr50 = load i8*, i8** %data_ptr_ptr49, align 8
  %cast_data_ptr = bitcast i8* %data_ptr50 to i1*
  %data51 = load i1, i1* %cast_data_ptr, align 1
  store i1 %data51, i1* %r, align 1
  %for_index52 = load i32, i32* %for_index41, align 4
  %tmp53 = add i32 %for_index52, 1
  store i32 %tmp53, i32* %for_index41, align 4
  %r54 = load i1, i1* %r, align 1
  %tmp55 = xor i1 %r54, true
  br i1 %tmp55, label %then57, label %else58

merge56:                                          ; preds = %else58
  br label %while42

then57:                                           ; preds = %while_body44
  ret i1 false

else58:                                           ; preds = %while_body44
  br label %merge56
}

define i1 @contains_string_string(%list_item** %l, %list_item** %items) {
entry:
```

```
  %r = alloca i1, align 1
  %for_index41 = alloca i32, align 4
  %ref = alloca i8*, align 8
  %for_index7 = alloca i32, align 4
  %item = alloca i8*, align 8
  %for_index = alloca i32, align 4
  %res = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %items2 = alloca %list_item**, align 8
  store %list_item** %items, %list_item*** %items2, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %res, align 8
  store i32 0, i32* %for_index, align 4
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.8, i32 0,
    i32 0), i8** %item, align 8
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index36 = load i32, i32* %for_index, align 4
  %items37 = load %list_item**, %list_item*** %items2, align 8
  %ilist38 = load %list_item*, %list_item** %items37, align 8
  %length39 = call i32 @list_length(%list_item* %ilist38, i32 0)
  %tmp40 = icmp slt i32 %for_index36, %length39
  br i1 %tmp40, label %while_body, label %merge

merge:                                            ; preds = %while
  store i32 0, i32* %for_index41, align 4
  store i1 false, i1* %r, align 1
  br label %while42

while_body:                                       ; preds = %while
  %items3 = load %list_item**, %list_item*** %items2, align 8
  %ilist = load %list_item*, %list_item** %items3, align 8
  %for_index4 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i8**
  %data = load i8*, i8** %cast_data_ptr, align 8
  store i8* %data, i8** %item, align 8
  %for_index5 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index5, 1
  store i32 %tmp, i32* %for_index, align 4
  %res6 = load %list_item**, %list_item*** %res, align 8
  %list_ptr = load %list_item*, %list_item** %res6, align 8
  %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
  store i32 0, i32* %for_index7, align 4
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.9, i32 0,
    i32 0), i8** %ref, align 8
  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   %malloccall30 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
         i1* null, i32 1) to i32))
2526   %copy_ptr = bitcast i8* %malloccall30 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
```

```
  %res61 = load %list_item**, %list_item*** %res, align 8
  %ilist62 = load %list_item*, %list_item** %res61, align 8
  %length63 = call i32 @list_length(%list_item* %ilist62, i32 0)
  %tmp64 = icmp slt i32 %for_index60, %length63
  br i1 %tmp64, label %while_body44, label %merge43

merge43:                                          ; preds = %while42
  ret i1 true

while_body44:                                     ; preds = %while42
  %res45 = load %list_item**, %list_item*** %res, align 8
  %ilist46 = load %list_item*, %list_item** %res45, align 8
  %for_index47 = load i32, i32* %for_index41, align 4
  %_result48 = call %list_item* @list_access(%list_item* %ilist46, i32 %
    for_index47)
  %data_ptr_ptr49 = getelementptr inbounds %list_item, %list_item* %
    _result48, i32 0, i32 0
  %data_ptr50 = load i8*, i8** %data_ptr_ptr49, align 8
  %cast_data_ptr51 = bitcast i8* %data_ptr50 to i1*
  %data52 = load i1, i1* %cast_data_ptr51, align 1
  store i1 %data52, i1* %r, align 1
  %for_index53 = load i32, i32* %for_index41, align 4
  %tmp54 = add i32 %for_index53, 1
  store i32 %tmp54, i32* %for_index41, align 4
  %r55 = load i1, i1* %r, align 1
  %tmp56 = xor i1 %r55, true
  br i1 %tmp56, label %then58, label %else59

merge57:                                          ; preds = %else59
  br label %while42

then58:                                           ; preds = %while_body44
  ret i1 false

else59:                                           ; preds = %while_body44
  br label %merge57
}

define i1 @contains_int_int(%list_item** %l, %list_item** %items) {
entry:
  %r = alloca i1, align 1
  %for_index42 = alloca i32, align 4
  %ref = alloca i32, align 4
  %for_index7 = alloca i32, align 4
  %item = alloca i32, align 4
  %for_index = alloca i32, align 4
  %res = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %items2 = alloca %list_item**, align 8
  store %list_item** %items, %list_item*** %items2, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %res, align 8
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %item, align 4
  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
```

```
  %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
  %cast_data_ptr17 = bitcast i8* %data_ptr16 to i32*
  %data18 = load i32, i32* %cast_data_ptr17, align 4
  store i32 %data18, i32* %ref, align 4
  %for_index19 = load i32, i32* %for_index7, align 4
  %tmp20 = add i32 %for_index19, 1
  store i32 %tmp20, i32* %for_index7, align 4
  %ref21 = load i32, i32* %ref, align 4
  %item22 = load i32, i32* %item, align 4
  %tmp23 = icmp eq i32 %ref21, %item22
  br i1 %tmp23, label %then, label %else

merge24:                                          ; preds = %else, %then
  br label %while8

then:                                             ; preds = %while_body10
  %res25 = load %list_item**, %list_item*** %res, align 8
  %ilist26 = load %list_item*, %list_item** %res25, align 8
  %res27 = load %list_item**, %list_item*** %res, align 8
  %ilist28 = load %list_item*, %list_item** %res27, align 8
  %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
  %tmp30 = sub i32 %length29, 1
  %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
  %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
  %malloccall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
     i1* null, i32 1) to i32))
  %copy_ptr = bitcast i8* %malloccall31 to i1*
  store i1 true, i1* %copy_ptr, align 1
  %ccopy = bitcast i1* %copy_ptr to i8*
  store i8* %ccopy, i8** %data_ptpt, align 8
  br label %merge24

else:                                             ; preds = %while_body10
  br label %merge24

while43:                                          ; preds = %merge58, %merge
  %for_index61 = load i32, i32* %for_index42, align 4
  %res62 = load %list_item**, %list_item*** %res, align 8
  %ilist63 = load %list_item*, %list_item** %res62, align 8
  %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
  %tmp65 = icmp slt i32 %for_index61, %length64
  br i1 %tmp65, label %while_body45, label %merge44

merge44:                                          ; preds = %while43
  ret i1 true

while_body45:                                     ; preds = %while43
  %res46 = load %list_item**, %list_item*** %res, align 8
  %ilist47 = load %list_item*, %list_item** %res46, align 8
  %for_index48 = load i32, i32* %for_index42, align 4
  %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
    for_index48)
  %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
    _result49, i32 0, i32 0
  %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
  %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
  %data53 = load i1, i1* %cast_data_ptr52, align 1
  store i1 %data53, i1* %r, align 1
```

```
  %for_index54 = load i32, i32* %for_index42, align 4
  %tmp55 = add i32 %for_index54, 1
  store i32 %tmp55, i32* %for_index42, align 4
  %r56 = load i1, i1* %r, align 1
  %tmp57 = xor i1 %r56, true
  br i1 %tmp57, label %then59, label %else60

merge58:                                          ; preds = %else60
  br label %while43

then59:                                           ; preds = %while_body45
  ret i1 false

else60:                                           ; preds = %while_body45
  br label %merge58
}

define i8* @join_string_string(%list_item** %text_list, i8* %connector) {
entry:
  %index = alloca i32, align 4
  %for_index = alloca i32, align 4
  %list_length = alloca i32, align 4
  %res = alloca i8*, align 8
  %text_list1 = alloca %list_item**, align 8
  store %list_item** %text_list, %list_item*** %text_list1, align 8
  %connector2 = alloca i8*, align 8
  store i8* %connector, i8** %connector2, align 8
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.10, i32 0,
    i32 0), i8** %res, align 8
  %text_list3 = load %list_item**, %list_item*** %text_list1, align 8
  %ilist = load %list_item*, %list_item** %text_list3, align 8
  %length = call i32 @list_length(%list_item* %ilist, i32 0)
  store i32 %length, i32* %list_length, align 4
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while

while:                                            ; preds = %while_body, %
    entry
  %for_index31 = load i32, i32* %for_index, align 4
  %list_length32 = load i32, i32* %list_length, align 4
  %tmp33 = sub i32 %list_length32, 1
  %malloccall34 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr35 = bitcast i8* %malloccall34 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr35, align 8
  %range_list36 = call %list_item** @range_function(i32 0, i32 %tmp33, %
    list_item** %head_ptr_ptr35, i32 0)
  %ilist37 = load %list_item*, %list_item** %range_list36, align 8
  %length38 = call i32 @list_length(%list_item* %ilist37, i32 0)
  %tmp39 = icmp slt i32 %for_index31, %length38
  br i1 %tmp39, label %while_body, label %merge

merge:                                            ; preds = %while
  %res40 = load i8*, i8** %res, align 8
  %text_list41 = load %list_item**, %list_item*** %text_list1, align 8
  %ilist42 = load %list_item*, %list_item** %text_list41, align 8
  %list_length43 = load i32, i32* %list_length, align 4
  %tmp44 = sub i32 %list_length43, 1
```

```
  %_result45 = call %list_item* @list_access(%list_item* %ilist42, i32 %
    tmp44)
  %data_ptr_ptr46 = getelementptr inbounds %list_item, %list_item* %
    _result45, i32 0, i32 0
  %data_ptr47 = load i8*, i8** %data_ptr_ptr46, align 8
  %cast_data_ptr48 = bitcast i8* %data_ptr47 to i8**
  %data49 = load i8*, i8** %cast_data_ptr48, align 8
  %length50 = call i32 @string_length(i8* %res40, i32 0)
  %length51 = call i32 @string_length(i8* %data49, i32 0)
  %new_length52 = add i32 %length50, %length51
  %new_length_nul53 = add i32 %new_length52, 1
  %mallocsize54 = mul i32 %new_length_nul53, ptrtoint (i8* getelementptr (i8
    , i8* null, i32 1) to i32)
  %new_string56 = tail call i8* @malloc(i32 %mallocsize54)
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string56, i8* %res40, i32 %
    length50, i1 true)
  %new_spot57 = getelementptr i8, i8* %new_string56, i32 %length50
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot57, i8* %data49, i32 %
    length51, i1 true)
  %string_term58 = getelementptr i8, i8* %new_string56, i32 %new_length52
  store i8 0, i8* %string_term58, align 1
  store i8* %new_string56, i8** %res, align 8
  %res59 = load i8*, i8** %res, align 8
  ret i8* %res59

while_body:                                       ; preds = %while
  %list_length4 = load i32, i32* %list_length, align 4
  %tmp = sub i32 %list_length4, 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
  %range_list = call %list_item** @range_function(i32 0, i32 %tmp, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist5 = load %list_item*, %list_item** %range_list, align 8
  %for_index6 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist5, i32 %
    for_index6)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i32*
  %data = load i32, i32* %cast_data_ptr, align 4
  store i32 %data, i32* %index, align 4
  %for_index7 = load i32, i32* %for_index, align 4
  %tmp8 = add i32 %for_index7, 1
  store i32 %tmp8, i32* %for_index, align 4
  %res9 = load i8*, i8** %res, align 8
  %text_list10 = load %list_item**, %list_item*** %text_list1, align 8
  %ilist11 = load %list_item*, %list_item** %text_list10, align 8
  %index12 = load i32, i32* %index, align 4
  %_result13 = call %list_item* @list_access(%list_item* %ilist11, i32 %
    index12)
  %data_ptr_ptr14 = getelementptr inbounds %list_item, %list_item* %
    _result13, i32 0, i32 0
  %data_ptr15 = load i8*, i8** %data_ptr_ptr14, align 8
  %cast_data_ptr16 = bitcast i8* %data_ptr15 to i8**
  %data17 = load i8*, i8** %cast_data_ptr16, align 8
  %length18 = call i32 @string_length(i8* %res9, i32 0)
```

```
  %length19 = call i32 @string_length(i8* %data17, i32 0)
  %new_length = add i32 %length18, %length19
  %new_length_nul = add i32 %new_length, 1
  %mallocsize = mul i32 %new_length_nul, ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32)
  %new_string = tail call i8* @malloc(i32 %mallocsize)
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %res9, i32 %
    length18, i1 true)
  %new_spot = getelementptr i8, i8* %new_string, i32 %length18
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot, i8* %data17, i32 %
    length19, i1 true)
  %string_term = getelementptr i8, i8* %new_string, i32 %new_length
  store i8 0, i8* %string_term, align 1
  %connector21 = load i8*, i8** %connector2, align 8
  %length22 = call i32 @string_length(i8* %new_string, i32 0)
  %length23 = call i32 @string_length(i8* %connector21, i32 0)
  %new_length24 = add i32 %length22, %length23
  %new_length_nul25 = add i32 %new_length24, 1
  %mallocsize26 = mul i32 %new_length_nul25, ptrtoint (i8* getelementptr (i8
    , i8* null, i32 1) to i32)
  %new_string28 = tail call i8* @malloc(i32 %mallocsize26)
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string28, i8* %new_string,
    i32 %length22, i1 true)
  %new_spot29 = getelementptr i8, i8* %new_string28, i32 %length22
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot29, i8* %connector21,
    i32 %length23, i1 true)
  %string_term30 = getelementptr i8, i8* %new_string28, i32 %new_length24
  store i8 0, i8* %string_term30, align 1
  store i8* %new_string28, i8** %res, align 8
  br label %while
}

define %list_item** @split(i8* %text, i8 %separator) {
entry:
  %right = alloca i32, align 4
  %left = alloca i32, align 4
  %text_length = alloca i32, align 4
  %result = alloca %list_item**, align 8
  %text1 = alloca i8*, align 8
  store i8* %text, i8** %text1, align 8
  %separator2 = alloca i8, align 1
  store i8 %separator, i8* %separator2, align 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %result, align 8
  %text3 = load i8*, i8** %text1, align 8
  %length = call i32 @string_length(i8* %text3, i32 0)
  store i32 %length, i32* %text_length, align 4
  store i32 0, i32* %left, align 4
  store i32 0, i32* %right, align 4
  br label %while

while:                                            ; preds = %merge7, %entry
  %right21 = load i32, i32* %right, align 4
  %text_length22 = load i32, i32* %text_length, align 4
  %tmp23 = icmp slt i32 %right21, %text_length22
  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
```

```
  %tmp17 = add i32 %right16, 1
  store i32 %tmp17, i32* %right, align 4
  %right18 = load i32, i32* %right, align 4
  store i32 %right18, i32* %left, align 4
  br label %merge7

else:                                             ; preds = %while_body
  %right19 = load i32, i32* %right, align 4
  %tmp20 = add i32 %right19, 1
  store i32 %tmp20, i32* %right, align 4
  br label %merge7
}

define i8* @string_reverse(i8* %text) {
entry:
  %index = alloca i32, align 4
  %string_length = alloca i32, align 4
  %res = alloca i8*, align 8
  %text1 = alloca i8*, align 8
  store i8* %text, i8** %text1, align 8
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.11, i32 0,
    i32 0), i8** %res, align 8
  %text2 = load i8*, i8** %text1, align 8
  %length = call i32 @string_length(i8* %text2, i32 0)
  store i32 %length, i32* %string_length, align 4
  %string_length3 = load i32, i32* %string_length, align 4
  %tmp = sub i32 %string_length3, 1
  store i32 %tmp, i32* %index, align 4
  br label %while

while:                                            ; preds = %while_body, %
    entry
  %index10 = load i32, i32* %index, align 4
  %tmp11 = icmp sge i32 %index10, 0
  br i1 %tmp11, label %while_body, label %merge

merge:                                            ; preds = %while
  %res12 = load i8*, i8** %res, align 8
  ret i8* %res12

while_body:                                       ; preds = %while
  %res4 = load i8*, i8** %res, align 8
  %text5 = load i8*, i8** %text1, align 8
  %index6 = load i32, i32* %index, align 4
  %get_char_ptr = getelementptr i8, i8* %text5, i32 %index6
  %get_char = load i8, i8* %get_char_ptr, align 1
  %length7 = call i32 @string_length(i8* %res4, i32 0)
  %new_length = add i32 %length7, 1
  %new_length_nul = add i32 %new_length, 1
  %mallocsize = mul i32 %new_length_nul, ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32)
  %new_string = tail call i8* @malloc(i32 %mallocsize)
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %res4, i32 %
    length7, i1 true)
  %new_spot = getelementptr i8, i8* %new_string, i32 %length7
  store i8 %get_char, i8* %new_spot, align 1
  %string_term = getelementptr i8, i8* %new_string, i32 %new_length
  store i8 0, i8* %string_term, align 1
  store i8* %new_string, i8** %res, align 8
```

```
  %index8 = load i32, i32* %index, align 4
  %tmp9 = sub i32 %index8, 1
  store i32 %tmp9, i32* %index, align 4
  br label %while
}

define i1 @startswith(i8* %text, i8 %s) {
entry:
  %text1 = alloca i8*, align 8
  store i8* %text, i8** %text1, align 8
  %s2 = alloca i8, align 1
  store i8 %s, i8* %s2, align 1
  %s3 = load i8, i8* %s2, align 1
  %text4 = load i8*, i8** %text1, align 8
  %get_char_ptr = getelementptr i8, i8* %text4, i32 0
  %get_char = load i8, i8* %get_char_ptr, align 1
  %tmp = icmp eq i8 %s3, %get_char
  ret i1 %tmp
}

define i1 @endswith(i8* %text, i8 %e) {
entry:
  %string_length = alloca i32, align 4
  %text1 = alloca i8*, align 8
  store i8* %text, i8** %text1, align 8
  %e2 = alloca i8, align 1
  store i8 %e, i8* %e2, align 1
  %text3 = load i8*, i8** %text1, align 8
  %length = call i32 @string_length(i8* %text3, i32 0)
  store i32 %length, i32* %string_length, align 4
  %e4 = load i8, i8* %e2, align 1
  %text5 = load i8*, i8** %text1, align 8
  %string_length6 = load i32, i32* %string_length, align 4
  %tmp = sub i32 %string_length6, 1
  %get_char_ptr = getelementptr i8, i8* %text5, i32 %tmp
  %get_char = load i8, i8* %get_char_ptr, align 1
  %tmp7 = icmp eq i8 %e4, %get_char
  ret i1 %tmp7
}

define %list_item** @string_to_list(i8* %text) {
entry:
  %c = alloca i8, align 1
  %for_index = alloca i32, align 4
  %result = alloca %list_item**, align 8
  %text1 = alloca i8*, align 8
  store i8* %text, i8** %text1, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %result, align 8
  store i32 0, i32* %for_index, align 4
  store i8 0, i8* %c, align 1
  br label %while

while:                                            ; preds = %while_body, %
    entry
  %for_index7 = load i32, i32* %for_index, align 4
```

```
  %text8 = load i8*, i8** %text1, align 8
  %length9 = call i32 @string_length(i8* %text8, i32 0)
  %tmp10 = icmp slt i32 %for_index7, %length9
  br i1 %tmp10, label %while_body, label %merge

merge:                                            ; preds = %while
  %result11 = load %list_item**, %list_item*** %result, align 8
  ret %list_item** %result11

while_body:                                       ; preds = %while
  %text2 = load i8*, i8** %text1, align 8
  %for_index3 = load i32, i32* %for_index, align 4
  %get_char_ptr = getelementptr i8, i8* %text2, i32 %for_index3
  %get_char = load i8, i8* %get_char_ptr, align 1
  store i8 %get_char, i8* %c, align 1
  %for_index4 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index4, 1
  store i32 %tmp, i32* %for_index, align 4
  %result5 = load %list_item**, %list_item*** %result, align 8
  %list_ptr = load %list_item*, %list_item** %result5, align 8
  %c6 = load i8, i8* %c, align 1
  %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_char(%list_item** %result5, i8 %
    c6, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %result, align 8
  br label %while
}

define i8 @lower(i8 %c) {
entry:
  %c_ref = alloca i8, align 1
  %for_index = alloca i32, align 4
  %index = alloca i32, align 4
  %c1 = alloca i8, align 1
  store i8 %c, i8* %c1, align 1
  store i32 -1, i32* %index, align 4
  store i32 0, i32* %for_index, align 4
  store i8 0, i8* %c_ref, align 1
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index24 = load i32, i32* %for_index, align 4
  %ASCII25 = load %list_item**, %list_item*** @ASCII, align 8
  %ilist26 = load %list_item*, %list_item** %ASCII25, align 8
  %length = call i32 @list_length(%list_item* %ilist26, i32 0)
  %tmp27 = icmp slt i32 %for_index24, %length
  br i1 %tmp27, label %while_body, label %merge

merge:                                            ; preds = %while
  %c28 = load i8, i8* %c1, align 1
  ret i8 %c28

while_body:                                       ; preds = %while
  %ASCII = load %list_item**, %list_item*** @ASCII, align 8
  %ilist = load %list_item*, %list_item** %ASCII, align 8
  %for_index2 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index2)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
```

```
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %data = load i8, i8* %data_ptr, align 1
  store i8 %data, i8* %c_ref, align 1
  %for_index3 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index3, 1
  store i32 %tmp, i32* %for_index, align 4
  %index4 = load i32, i32* %index, align 4
  %tmp5 = add i32 %index4, 1
  store i32 %tmp5, i32* %index, align 4
  %c6 = load i8, i8* %c1, align 1
  %c_ref7 = load i8, i8* %c_ref, align 1
  %tmp8 = icmp eq i8 %c6, %c_ref7
  br i1 %tmp8, label %then, label %else23

merge9:                                           ; preds = %else23, %
    merge12
  br label %while

then:                                             ; preds = %while_body
  %index10 = load i32, i32* %index, align 4
  %tmp11 = icmp slt i32 %index10, 26
  br i1 %tmp11, label %then13, label %else

merge12:                                          ; No predecessors!
  br label %merge9

then13:                                           ; preds = %then
  %c14 = load i8, i8* %c1, align 1
  ret i8 %c14

else:                                             ; preds = %then
  %ASCII15 = load %list_item**, %list_item*** @ASCII, align 8
  %ilist16 = load %list_item*, %list_item** %ASCII15, align 8
  %index17 = load i32, i32* %index, align 4
  %tmp18 = sub i32 %index17, 26
  %_result19 = call %list_item* @list_access(%list_item* %ilist16, i32 %
    tmp18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %data22 = load i8, i8* %data_ptr21, align 1
  ret i8 %data22

else23:                                           ; preds = %while_body
  br label %merge9
}

define i8 @upper(i8 %c) {
entry:
  %c_ref = alloca i8, align 1
  %for_index = alloca i32, align 4
  %index = alloca i32, align 4
  %c1 = alloca i8, align 1
  store i8 %c, i8* %c1, align 1
  store i32 -1, i32* %index, align 4
  store i32 0, i32* %for_index, align 4
  store i8 0, i8* %c_ref, align 1
  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
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %data22 = load i8, i8* %data_ptr21, align 1
  ret i8 %data22

else23:                                           ; preds = %while_body
  br label %merge9
}

define i1 @strcmp(i8* %str1, i8* %str2) {
entry:
  %c2 = alloca i8, align 1
  %c1 = alloca i8, align 1
  %i = alloca i32, align 4
  %str11 = alloca i8*, align 8
  store i8* %str1, i8** %str11, align 8
  %str22 = alloca i8*, align 8
  store i8* %str2, i8** %str22, align 8
  %str13 = load i8*, i8** %str11, align 8
  %length = call i32 @string_length(i8* %str13, i32 0)
  %str24 = load i8*, i8** %str22, align 8
  %length5 = call i32 @string_length(i8* %str24, i32 0)
  %tmp = icmp ne i32 %length, %length5
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else
  store i32 0, i32* %i, align 4
  br label %while

then:                                             ; preds = %entry
  ret i1 false

else:                                             ; preds = %entry
  br label %merge

while:                                            ; preds = %merge16, %merge
  %i21 = load i32, i32* %i, align 4
  %str122 = load i8*, i8** %str11, align 8
  %length23 = call i32 @string_length(i8* %str122, i32 0)
  %tmp24 = icmp slt i32 %i21, %length23
  br i1 %tmp24, label %while_body, label %merge6

merge6:                                           ; preds = %while
  ret i1 true

while_body:                                       ; preds = %while
  %str17 = load i8*, i8** %str11, align 8
  %i8 = load i32, i32* %i, align 4
  %get_char_ptr = getelementptr i8, i8* %str17, i32 %i8
  %get_char = load i8, i8* %get_char_ptr, align 1
  store i8 %get_char, i8* %c1, align 1
  %str29 = load i8*, i8** %str22, align 8
  %i10 = load i32, i32* %i, align 4
  %get_char_ptr11 = getelementptr i8, i8* %str29, i32 %i10
  %get_char12 = load i8, i8* %get_char_ptr11, align 1
  store i8 %get_char12, i8* %c2, align 1
  %c113 = load i8, i8* %c1, align 1
  %c214 = load i8, i8* %c2, align 1
  %tmp15 = icmp ne i8 %c113, %c214
```

```
  br i1 %tmp15, label %then17, label %else18

merge16:                                          ; preds = %else18
  %i19 = load i32, i32* %i, align 4
  %tmp20 = add i32 %i19, 1
  store i32 %tmp20, i32* %i, align 4
  br label %while

then17:                                           ; preds = %while_body
  ret i1 false

else18:                                           ; preds = %while_body
  br label %merge16
}

declare noalias i8* @malloc(i32)

define %list_item* @list_access(%list_item* %0, i32 %1) {
entry:
  %is_zero = icmp eq i32 %1, 0
  br i1 %is_zero, label %then, label %else

then:                                             ; preds = %entry
  ret %list_item* %0

else:                                             ; preds = %entry
  %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
  %next = load %list_item*, %list_item** %next_ptr, align 8
  %sub = sub i32 %1, 1
  %result = call %list_item* @list_access(%list_item* %next, i32 %sub)
  ret %list_item* %result
}

define i32 @list_length(%list_item* %0, i32 %1) {
entry:
  %ptr_is_null = icmp eq %list_item* %0, null
  br i1 %ptr_is_null, label %then, label %else

then:                                             ; preds = %entry
  ret i32 %1

else:                                             ; preds = %entry
  %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
  %next = load %list_item*, %list_item** %next_ptr, align 8
  %add = add i32 %1, 1
  %result = call i32 @list_length(%list_item* %next, i32 %add)
  ret i32 %result
}

define %list_item** @range_function(i32 %0, i32 %1, %list_item** %2, i32 %3)
    {
entry:
  %is_last = icmp eq i32 %0, %1
  br i1 %is_last, label %then, label %else

then:                                             ; preds = %entry
  ret %list_item** %2
```

```
3302
3303 else:                                             ; preds = %entry
3304   %head_ptr_ptr = call %list_item** @insert_int(%list_item** %2, i32 %0, i32
         %3)
3305   %next_s = add i32 1, %0
3306   %next_length = add i32 1, %3
3307   %4 = call %list_item** @range_function(i32 %next_s, i32 %1, %list_item** %
         head_ptr_ptr, i32 %next_length)
3308   ret %list_item** %4
3309 }
3310
3311 define %list_item** @insert_int(%list_item** %0, i32 %1, i32 %2) {
3312 entry:
3313   %list_ptr = load %list_item*, %list_item** %0, align 8
3314   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
         i1** null, i32 1) to i32))
3315   %new_list_ptr_ptr = bitcast i8* %malloccall to %list_item**
3316   store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3317   %last_node_ptr_ptr = call %list_item** @list_copy_int(%list_item* %
         list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3318   %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3319   %temp = alloca %list_item, align 8
3320   %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3321   store %list_item* %new_list_ptr, %list_item** %next, align 8
3322   %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3323   %data_node = bitcast i8* %malloccall1 to %list_item*
3324   %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
         , i32* null, i32 1) to i32))
3325   %data = bitcast i8* %malloccall2 to i32*
3326   store i32 %1, i32* %data, align 4
3327   %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
         i32 0
3328   %cast = bitcast i32* %data to i8*
3329   store i8* %cast, i8** %dat, align 8
3330   %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3331   %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
         1
3332   %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
         i32 1
3333   %temp4 = load %list_item*, %list_item** %test, align 8
3334   store %list_item* %temp4, %list_item** %dat3, align 8
3335   store %list_item* %data_node, %list_item** %test, align 8
3336   %temp5 = load %list_item*, %list_item** %next, align 8
3337   store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
3338   ret %list_item** %new_list_ptr_ptr
3339 }
3340
3341 define %list_item** @list_copy_int(%list_item* %0, i32 %1, %list_item** %2)
         {
3342 entry:
3343   %is_zero = icmp eq i32 %1, 0
3344   %ptr_is_null = icmp eq %list_item* %0, null
3345   %or_conds = or i1 %is_zero, %ptr_is_null
3346   br i1 %or_conds, label %then, label %else
3347
3348 then:                                             ; preds = %entry
3349   ret %list_item** %2
3350
```

```
3351 else:                                                ; preds = %entry
3352   %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3353   %new_struct_ptr = bitcast i8* %malloccall to %list_item*
3354   store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3355   %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
         , i32* null, i32 1) to i32))
3356   %ltyp = bitcast i8* %malloccall1 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   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
          i1** null, i32 1) to i32))
3378   %new_list_ptr_ptr = bitcast i8* %malloccall 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   %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3386   %data_node = bitcast i8* %malloccall1 to %list_item*
3387   %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
         *, i1** null, i32 1) to i32))
3388   %data = bitcast i8* %malloccall2 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
```

```
  %temp4 = load %list_item*, %list_item** %test, align 8
  store %list_item* %temp4, %list_item** %dat3, align 8
  store %list_item* %data_node, %list_item** %test, align 8
  %temp5 = load %list_item*, %list_item** %next, align 8
  store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
  ret %list_item** %new_list_ptr_ptr
}

define %list_item** @list_copy_string(%list_item* %0, i32 %1, %list_item**
    %2) {
entry:
  %is_zero = icmp eq i32 %1, 0
  %ptr_is_null = icmp eq %list_item* %0, null
  %or_conds = or i1 %is_zero, %ptr_is_null
  br i1 %or_conds, label %then, label %else

then:                                             ; preds = %entry
  ret %list_item** %2

else:                                             ; preds = %entry
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %new_struct_ptr = bitcast i8* %malloccall to %list_item*
  store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %ltyp = bitcast i8* %malloccall1 to i8**
  %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
     0, i32 0
  %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
  %cast_old_data_ptr = bitcast i8* %old_data_ptr to i8**
  %old_data = load i8*, i8** %cast_old_data_ptr, align 8
  store i8* %old_data, i8** %ltyp, align 8
  %data_ptr_cast = bitcast i8** %ltyp to i8*
  %store_new_data = getelementptr inbounds %list_item, %list_item* %
    new_struct_ptr, i32 0, i32 0
  store i8* %data_ptr_cast, i8** %store_new_data, align 8
  store %list_item* %new_struct_ptr, %list_item** %2, align 8
  %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
    i32 0, i32 1
  %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
  %next2 = load %list_item*, %list_item** %next_ptr, align 8
  %sub = sub i32 %1, 1
  %3 = call %list_item** @list_copy_string(%list_item* %next2, i32 %sub, %
    list_item** %next)
  ret %list_item** %3
}

define %list_item** @insert_char(%list_item** %0, i8 %1, i32 %2) {
entry:
  %list_ptr = load %list_item*, %list_item** %0, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %new_list_ptr_ptr = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
  %last_node_ptr_ptr = call %list_item** @list_copy_char(%list_item* %
    list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
  %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   %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3449   %data_node = bitcast i8* %malloccall1 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   %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3477   %new_struct_ptr = bitcast i8* %malloccall 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)
  ret %list_item** %3
}

define %list_item** @insert_bool(%list_item** %0, i1 %1, i32 %2) {
entry:
  %list_ptr = load %list_item*, %list_item** %0, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
  %new_list_ptr_ptr = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
  %last_node_ptr_ptr = call %list_item** @list_copy_bool(%list_item* %
    list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
  %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
  %temp = alloca %list_item, align 8
  %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
  store %list_item* %new_list_ptr, %list_item** %next, align 8
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %data_node = bitcast i8* %malloccall1 to %list_item*
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
    i1* null, i32 1) to i32))
  %data = bitcast i8* %malloccall2 to i1*
  store i1 %1, i1* %data, align 1
  %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 0
  %cast = bitcast i1* %data to i8*
  store i8* %cast, i8** %dat, align 8
  %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
  %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
    1
  %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 1
  %temp4 = load %list_item*, %list_item** %test, align 8
  store %list_item* %temp4, %list_item** %dat3, align 8
  store %list_item* %data_node, %list_item** %test, align 8
  %temp5 = load %list_item*, %list_item** %next, align 8
  store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
  ret %list_item** %new_list_ptr_ptr
}

define %list_item** @list_copy_bool(%list_item* %0, i32 %1, %list_item** %2)
    {
entry:
  %is_zero = icmp eq i32 %1, 0
  %ptr_is_null = icmp eq %list_item* %0, null
  %or_conds = or i1 %is_zero, %ptr_is_null
  br i1 %or_conds, label %then, label %else

then:                                             ; preds = %entry
  ret %list_item** %2

else:                                             ; preds = %entry
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %new_struct_ptr = bitcast i8* %malloccall to %list_item*
  store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
    i1* null, i32 1) to i32))
```

```
  %ltyp = bitcast i8* %malloccall1 to i1*
  %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32 0
  %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
  %cast_old_data_ptr = bitcast i8* %old_data_ptr to i1*
  %old_data = load i1, i1* %cast_old_data_ptr, align 1
  store i1 %old_data, i1* %ltyp, align 1
  %data_ptr_cast = bitcast i1* %ltyp to i8*
  %store_new_data = getelementptr inbounds %list_item, %list_item* %new_struct_ptr, i32 0, i32 0
  store i8* %data_ptr_cast, i8** %store_new_data, align 8
  store %list_item* %new_struct_ptr, %list_item** %2, align 8
  %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr, i32 0, i32 1
  %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32 1
  %next2 = load %list_item*, %list_item** %next_ptr, align 8
  %sub = sub i32 %1, 1
  %3 = call %list_item** @list_copy_bool(%list_item* %next2, i32 %sub, %list_item** %next)
  ret %list_item** %3
}

define %list_item** @insert_float(%list_item** %0, double %1, i32 %2) {
entry:
  %list_ptr = load %list_item*, %list_item** %0, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*, i1** null, i32 1) to i32))
  %new_list_ptr_ptr = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
  %last_node_ptr_ptr = call %list_item** @list_copy_float(%list_item* %list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
  %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
  %temp = alloca %list_item, align 8
  %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
  store %list_item* %new_list_ptr, %list_item** %next, align 8
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item* getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %data_node = bitcast i8* %malloccall1 to %list_item*
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (double, double* null, i32 1) to i32))
  %data = bitcast i8* %malloccall2 to double*
  store double %1, double* %data, align 8
  %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0, i32 0
  %cast = bitcast double* %data to i8*
  store i8* %cast, i8** %dat, align 8
  %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
  %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32 1
  %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0, i32 1
  %temp4 = load %list_item*, %list_item** %test, align 8
  store %list_item* %temp4, %list_item** %dat3, align 8
  store %list_item* %data_node, %list_item** %test, align 8
  %temp5 = load %list_item*, %list_item** %next, align 8
  store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
  ret %list_item** %new_list_ptr_ptr
}
```

```
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
```

```
define i32 @string_length(i8* %0, i32 %1) {
entry:
  %char = load i8, i8* %0, align 1
  %ptr_is_null = icmp eq i8 %char, 0
  br i1 %ptr_is_null, label %then, label %else

then:                                             ; preds = %entry
  ret i32 %1

else:                                             ; preds = %entry
  %next_ptr = getelementptr i8, i8* %0, i32 1
  %add = add i32 %1, 1
  %result = call i32 @string_length(i8* %next_ptr, i32 %add)
  ret i32 %result
}

define i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %2, i32 %3) {
entry:
  %charA_ptr = getelementptr i8, i8* %0, i32 %3
  %charA = load i8, i8* %charA_ptr, align 1
  %charB_ptr = getelementptr i8, i8* %1, i32 %3
  %charB = load i8, i8* %charB_ptr, align 1
  %not_same = icmp ne i8 %charA, %charB
  br i1 %not_same, label %then_not_same, label %else_same

then_not_same:                                    ; preds = %entry
  ret i1 false

else_same:                                        ; preds = %entry
  %last_char = icmp eq i32 %3, %2
  br i1 %last_char, label %then_end, label %else_not_end

then_end:                                         ; preds = %else_same
  ret i1 true

else_not_end:                                     ; preds = %else_same
  %next_index = add i32 1, %3
  %res = call i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %2, i32 %
    next_index)
  ret i1 %res
}

; Function Attrs: argmemonly nounwind willreturn
declare void @llvm.memcpy.p0i8.p0i8.i32(i8* noalias nocapture writeonly, i8*
    noalias nocapture readonly, i32, i1 immarg) #0

attributes #0 = { argmemonly nounwind willreturn }
```

9.3.2 maximum_subarray.tm

Below is the TEAM source code:

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

```
end

int maxSubArray(list nums):

    int total_sum = nums[0];
    int max_sum = nums[0];


    for num in nums[1:]:
        total_sum = max(total_sum + num, num);
        max_sum = max(max_sum, total_sum);
    end

    return max_sum;
end

print("%d\n", maxSubArray([-2,1,-3,4,-1,2,1,-5,4]));
// Output: 6
// Explanation: [4,-1,2,1] has the largest sum = 6.
```

The program is expected to have the following output:

```
6
```

Below is the LLVM generated by the compiler:

```
; ModuleID = 'TEAM'
source_filename = "TEAM"

%list_item = type <{ i8*, %list_item* }>

@string = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1

declare i32 @printf(i8*, ...)

declare double @pow(double, double)

declare i8* @fopen(i8*, i8*)

declare i32 @close(i8*)

declare i8* @readline(i8*)

declare i8* @write(i8*, i8*)

declare i1 @match(i8*, i8*)

declare i8* @find(i8*, i8*)

declare i8* @replace(i8*, i8*, i8*, i32)

declare i8* @replace_all(i8*, i8*, i8*)

declare %list_item** @find_all(i8*, i8*)

define i32 @main() {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall 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  %malloccall24 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
79  %list_item25 = bitcast i8* %malloccall24 to %list_item*
80  store %list_item zeroinitializer, %list_item* %list_item25, align 1
81  %malloccall26 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
      i32, i32* null, i32 1) to i32))
82  %copied27 = bitcast i8* %malloccall26 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  %malloccall31 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
90  %list_item32 = bitcast i8* %malloccall31 to %list_item*
91  store %list_item zeroinitializer, %list_item* %list_item32, align 1
92  %malloccall33 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
      i32, i32* null, i32 1) to i32))
93  %copied34 = bitcast i8* %malloccall33 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 %malloccall38 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
101 %list_item39 = bitcast i8* %malloccall38 to %list_item*
102 store %list_item zeroinitializer, %list_item* %list_item39, align 1
103 %malloccall40 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
      i32, i32* null, i32 1) to i32))
104 %copied41 = bitcast i8* %malloccall40 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 %malloccall45 = tail call i8* @malloc(i32 ptrtoint (%list_item*
      getelementptr (%list_item, %list_item* null, i32 1) to i32))
112 %list_item46 = bitcast i8* %malloccall45 to %list_item*
113 store %list_item zeroinitializer, %list_item* %list_item46, align 1
114 %malloccall47 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
      i32, i32* null, i32 1) to i32))
115 %copied48 = bitcast i8* %malloccall47 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
  store %list_item* %list_item39, %list_item** %next51, align 8
  %malloccall52 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item53 = bitcast i8* %malloccall52 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item53, align 1
  %malloccall54 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
  %copied55 = bitcast i8* %malloccall54 to i32*
  store i32 -2, i32* %copied55, align 4
  %cast_ptr56 = bitcast i32* %copied55 to i8*
  %data_ptr_container57 = getelementptr inbounds %list_item, %list_item* %
    list_item53, i32 0, i32 0
  store i8* %cast_ptr56, i8** %data_ptr_container57, align 8
  %next58 = getelementptr inbounds %list_item, %list_item* %list_item53, i32
     0, i32 1
  store %list_item* %list_item46, %list_item** %next58, align 8
  store %list_item* %list_item53, %list_item** %list, align 8
  %_result = call i32 @maxSubArray_int(%list_item** %list)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8
    ], [4 x i8]* @string, i32 0, i32 0), i32 %_result)
  ret i32 0
}

define i32 @maxSubArray_int(%list_item** %nums) {
entry:
  %num = alloca i32, align 4
  %for_index = alloca i32, align 4
  %max_sum = alloca i32, align 4
  %total_sum = alloca i32, align 4
  %nums1 = alloca %list_item**, align 8
  store %list_item** %nums, %list_item*** %nums1, align 8
  %nums2 = load %list_item**, %list_item*** %nums1, align 8
  %ilist = load %list_item*, %list_item** %nums2, align 8
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 0)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i32*
  %data = load i32, i32* %cast_data_ptr, align 4
  store i32 %data, i32* %total_sum, align 4
  %nums3 = load %list_item**, %list_item*** %nums1, align 8
  %ilist4 = load %list_item*, %list_item** %nums3, align 8
  %_result5 = call %list_item* @list_access(%list_item* %ilist4, i32 0)
  %data_ptr_ptr6 = getelementptr inbounds %list_item, %list_item* %_result5,
     i32 0, i32 0
  %data_ptr7 = load i8*, i8** %data_ptr_ptr6, align 8
  %cast_data_ptr8 = bitcast i8* %data_ptr7 to i32*
  %data9 = load i32, i32* %cast_data_ptr8, align 4
  store i32 %data9, i32* %max_sum, align 4
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %num, align 4
  br label %while

while:                                            ; preds = %while_body, %
    entry
  %for_index28 = load i32, i32* %for_index, align 4
  %nums29 = load %list_item**, %list_item*** %nums1, align 8
  %ilist30 = load %list_item*, %list_item** %nums29, align 8
```

```
  %start31 = call %list_item* @list_access(%list_item* %ilist30, i32 1)
  %malloccall32 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %new_list_ptr_ptr33 = bitcast i8* %malloccall32 to %list_item**
  %0 = call %list_item** @list_copy_int(%list_item* %start31, i32 -1, %
    list_item** %new_list_ptr_ptr33)
  %ilist34 = load %list_item*, %list_item** %new_list_ptr_ptr33, align 8
  %length = call i32 @list_length(%list_item* %ilist34, i32 0)
  %tmp35 = icmp slt i32 %for_index28, %length
  br i1 %tmp35, label %while_body, label %merge

merge:                                            ; preds = %while
  %max_sum36 = load i32, i32* %max_sum, align 4
  ret i32 %max_sum36

while_body:                                       ; preds = %while
  %nums10 = load %list_item**, %list_item*** %nums1, align 8
  %ilist11 = load %list_item*, %list_item** %nums10, align 8
  %start = call %list_item* @list_access(%list_item* %ilist11, i32 1)
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %new_list_ptr_ptr = bitcast i8* %malloccall to %list_item**
  %1 = call %list_item** @list_copy_int(%list_item* %start, i32 -1, %
    list_item** %new_list_ptr_ptr)
  %ilist12 = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
  %for_index13 = load i32, i32* %for_index, align 4
  %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
  %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
  %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
  %cast_data_ptr17 = bitcast i8* %data_ptr16 to i32*
  %data18 = load i32, i32* %cast_data_ptr17, align 4
  store i32 %data18, i32* %num, align 4
  %for_index19 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index19, 1
  store i32 %tmp, i32* %for_index, align 4
  %num20 = load i32, i32* %num, align 4
  %total_sum21 = load i32, i32* %total_sum, align 4
  %num22 = load i32, i32* %num, align 4
  %tmp23 = add i32 %total_sum21, %num22
  %_result24 = call i32 @max(i32 %tmp23, i32 %num20)
  store i32 %_result24, i32* %total_sum, align 4
  %total_sum25 = load i32, i32* %total_sum, align 4
  %max_sum26 = load i32, i32* %max_sum, align 4
  %_result27 = call i32 @max(i32 %max_sum26, i32 %total_sum25)
  store i32 %_result27, i32* %max_sum, align 4
  br label %while
}

define i32 @max(i32 %x, i32 %y) {
entry:
  %x1 = alloca i32, align 4
  store i32 %x, i32* %x1, align 4
  %y2 = alloca i32, align 4
  store i32 %y, i32* %y2, align 4
  %x3 = load i32, i32* %x1, align 4
  %y4 = load i32, i32* %y2, align 4
  %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 https://github.com/luluzheng1/TEAM into semant_expr
|Lulu Zheng	2021-03-30	exception changed to open
|gyawalisaurav	2021-03-30	Merge branch 'codegen_expr' of https://github.com/luluzheng1/TEAM 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 grammmar in parser
|nokada11	2021-04-17	Merge branch 'main' of https://github.com/luluzheng1/TEAM into main
|nokada11	2021-04-18	Merge branch 'main' of https://github.com/luluzheng1/TEAM into extende_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 github.com:luluzheng1/TEAM 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 https://github.com/luluzheng1/TEAM 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 https://github.com/luluzheng1/TEAM 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 https://github.com/luluzheng1/TEAM 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 https://github.com/luluzheng1/TEAM 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 github.com:luluzheng1/TEAM into main+Tim
|nokada11	2021-04-27	Fix print in codegen
|gyawalisaurav	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
|gyawalisaurav	2021-04-27	Fixed for loop for string
|Saurav Gyawali	2021-04-27	Merge pull request #54 from luluzheng1/declaration_fix
|gyawalisaurav	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
|gyawalisaurav	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 https://github.com/luluzheng1/TEAM 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 luluzheng1/extended_testsuite
|Lulu Zheng	2021-04-27	more changes to acommodate 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 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 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 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 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 https://github.com/luluzheng1/TEAM
|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 https://github.com/luluzheng1/TEAM
|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-compilescript
|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`

```
/* Author: Yingjie L., Lulu Z. */
bool contains(list l, list items):
    list res = [];
    for item in items:
        res = append(res, false);
        for ref in l:
            if ref == item:
                res[length(res) - 1] = true;
            end
        end
    end

    for r in res:
        if not r:
            return false;
        end
    end
    return true;
end

contains([1, 2], [1]);
contains(["hello", "wolrd"], ["hello"]);
contains(['h', 'e'], ['h']);
contains([true, false], [true]);
contains([1.1, 1.2], [1.1]);

list remove_int(list l, int elem, bool all):
    list retlist = [];
    int i = 0;
    int len = length(l);
    int remove_index = 0;

    if all:
        while i < len:
            if l[i] == elem:
                i += 1;
                continue;
            end
            retlist = append(retlist, l[i]);
            i += 1;
        end
    else:
        while i < len:
            if l[i] == elem:
                remove_index = i;
                break;
            end

            retlist = append(retlist, l[i]);

            i += 1;
        end
        /* can be optimized with list concat */
        if i != len:
            for index in remove_index+1..len:
                retlist = append(retlist, l[index]);
```

```
            end
        end
    end

    return retlist;
end


list remove_float(list l, float elem, bool all):
    list retlist = [];
    int i = 0;
    int len = length(l);
    int remove_index = 0;

    if all:
        while i < len:
            if l[i] == elem:
                i += 1;
                continue;
            end
            retlist = append(retlist, l[i]);
            i += 1;
        end
    else:
        while i < len:
            if l[i] == elem:
                remove_index = i;
                break;
            end

            retlist = append(retlist, l[i]);

            i += 1;
        end
        /* can be optimized with list concat */
        if i != len:
            for index in remove_index+1..len:
                retlist = append(retlist, l[index]);
            end
        end
    end

    return retlist;
end

list remove_bool(list l, bool elem, bool all):
    list retlist = [];
    int i = 0;
    int len = length(l);
    int remove_index = 0;

    if all:
        while i < len:
            if l[i] == elem:
                i += 1;
                continue;
            end
            retlist = append(retlist, l[i]);
            i += 1;
```

```
        end
    else:
        while i < len:
            if l[i] == elem:
                remove_index = i;
                break;
            end

            retlist = append(retlist, l[i]);

            i += 1;
        end
        /* can be optimized with list concat */
        if i != len:
            for index in remove_index+1..len:
                retlist = append(retlist, l[index]);
            end
        end
    end

    return retlist;
end


list remove_char(list l, char elem, bool all):
    list retlist = [];
    int i = 0;
    int len = length(l);
    int remove_index = 0;

    if all:
        while i < len:
            if l[i] == elem:
                i += 1;
                continue;
            end
            retlist = append(retlist, l[i]);
            i += 1;
        end
    else:
        while i < len:
            if l[i] == elem:
                remove_index = i;
                break;
            end

            retlist = append(retlist, l[i]);

            i += 1;
        end
        /* can be optimized with list concat */
        if i != len:
            for index in remove_index+1..len:
                retlist = append(retlist, l[index]);
            end
        end
    end

    return retlist;
```

```
end

list remove_string(list l, string elem, bool all):
    list retlist = [];
    int i = 0;
    int len = length(l);
    int remove_index = 0;

    if all:
        while i < len:
            if strcmp(l[i], elem):
                i += 1;
                continue;
            end
            retlist = append(retlist, l[i]);
            i += 1;
        end
    else:
        while i < len:
            if strcmp(l[i], elem):
                remove_index = i;
                break;
            end

            retlist = append(retlist, l[i]);

            i += 1;
        end
        /* can be optimized with list concat */
        if i != len:
            for index in remove_index+1..len:
                retlist = append(retlist, l[index]);
            end
        end
    end

    return retlist;
end

remove_int([1], 1, true);
remove_float([1.5], 1.5, false);
remove_bool([true], true, true);
remove_char(['a'], 'a', true);
remove_string(["hello"], "what", true);
```

11.2.2 standard_library/string.tm

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

```
                'U', 'V', 'W', 'X', 'Y',
                'Z'
            ];

list split(string text, char separator):
    list result = [];
    int text_length = length(text);
    int left = 0;
    int right = 0;
    while right < text_length:
        if text[right] == separator:
            result = append(result, text[left:right]);
            right = right + 1;
            left = right;
        else:
            right = right + 1;
        end
    end
    result = append(result, text[left:right]);
    return result;
end


string join(list text_list, string connector):
    string res = "";
    int list_length = length(text_list);
    for index in 0..(list_length - 1):
        res = (res + text_list[index] + connector);
    end
    res = res + text_list[list_length - 1];
    return res;
end

join(["hello"], "");

string string_reverse(string text):
    string res = "";
    int string_length = length(text);
    int index = string_length - 1;
    while index >= 0:
        res = res + text[index];
        index -= 1;
    end
    return res;
end

bool startswith(string text, char s):
    return s == text[0];
end

bool endswith(string text, char e):
    int string_length = length(text);
    return e == text[string_length - 1];
end

list string_to_list(string text):
    list result = [];
    for c in text:
        result = append(result, c);
```

```
    end
    return result;
end

char lower(char c):
    int index = -1;
    for c_ref in ASCII:
        index += 1;
        if c == c_ref:
            if index < 26:
                return c;
            else:
                return ASCII[index - 26];
            end
        end
    end
    return c;
end

char upper(char c):
    int index = -1;
    for c_ref in ASCII:
        index += 1;
        if c == c_ref:
            if index > 25:
                return c;
            else:
                return ASCII[index + 26];
            end
        end
    end
    return c;
end

bool strcmp(string str1, string str2):
    if length(str1) != length(str2):
        return false;
    end

    int i = 0;
    while i < length(str1):
        char c1 = str1[i];
        char c2 = str2[i];
        if c1 != c2:
            return false;
        end
        i += 1;
    end
    return true;
end
```

11.2.3 src/scanner.mll

```
(* Ocamllex scanner for TEAM *)
(* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)

{
  open Parser
  exception Scan_error of string
```

```
  let fail ch = raise (Scan_error (Char.escaped ch))
  let unescape s =
    Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
}

let digit = ['0' - '9']
let digits = digit+
let float = digits '.' digits
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let char = ''' (ascii | digit) '''
let escaped_char = '\\' ['\\' ''' '"' 'n' 'r' 't']
let string = '"' ( (ascii | escaped_char)* as s) '"'
rule token = parse
    [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
  | ';'      { SEMI }
  | "/*"     { comment lexbuf }    (* Blocky Comments *)
  | "//"     { slcomment lexbuf }  (* Single line Comments *)
  | '('      { LPAREN }
  | ')'      { RPAREN }
  | '['      { LSQUARE }
  | ']'      { RSQUARE }
  | ":"      { COLON }
  | ','      { COMMA }
  | '+'      { PLUS }
  | '-'      { MINUS }
  | '*'      { TIMES }
  | '/'      { DIVIDE }
  | '%'      { MOD }
  | '^'      { EXP }
  | "+="     { ADDASN }
  | "-="     { SUBASN }
  | "*="     { MULASN }
  | "/="     { DIVASN }
  | "%="     { MODASN }
  | '='      { ASSIGN }
  | ".."     { RANGE }
  | "=="     { EQ }
  | "!="     { NEQ }
  | '<'      { LT }
  | "<="     { LEQ }
  | ">"      { GT }
  | ">="     { GEQ }
  | "and"    { AND }
  | "or"     { OR }
  | "not"    { NOT }
  | "if"     { IF }
  | "elif"   { ELSEIF }
  | "else"   { ELSE }
  | "for"    { FOR }
  | "in"     { IN }
  | "while"  { WHILE }
  | "break"  { BREAK }
  | "continue" { CONTINUE }
  | "return" { RETURN }
  | "end"    { END }
  | "int"    { INT }
  | "float"  { FLOAT }
  | "bool"   { BOOL }
  | "string" { STRING }
```

```
  | "char"   { CHAR }
  | "void"   { VOID }
  | "file"   { FILE }
  | "true"   { BLIT(true) }
  | "false"  { BLIT(false) }
  | "list"  { LIST }
  | "->"     { ARROW }
  | digits as lxm { LITERAL(int_of_string lxm) }
  | float as lxm { FLIT(float_of_string lxm) }
  | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
  | char as lxm  { CLIT( String.get lxm 1 ) }
  | string    { SLIT(unescape s) }
  | eof { EOF }
  | _ as char { fail char }


and comment = parse
    "*/"   { token lexbuf }
  | _    { comment lexbuf }

and slcomment = parse
  '\n'   { token lexbuf }
  | eof  { token lexbuf }
  | _    { slcomment lexbuf }
```

11.2.4 src/parser.mly

```
/* Ocamlyacc parser for TEAM */
/* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. */
%{
open Ast
%}

%token LPAREN RPAREN LSQUARE RSQUARE COMMA ARROW COLON SEMI
%token PLUS MINUS TIMES DIVIDE MOD EXP
%token ADDASN SUBASN MULASN DIVASN MODASN ASSIGN NOT
%token EQ NEQ LT LEQ GT GEQ RANGE AND OR
%token IF ELSEIF ELSE FOR IN WHILE BREAK CONTINUE RETURN END
%token INT FLOAT BOOL STRING CHAR VOID
%token LIST FILE
%token <bool> BLIT
%token <int> LITERAL
%token <float> FLIT
%token <string> ID
%token <char> CLIT
%token <string> SLIT
%token EOF

%start program
%type <Ast.program> program

%left ARROW
%right ASSIGN ADDASN SUBASN MULASN DIVASN MODASN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%nonassoc RANGE
%left PLUS MINUS
```

```
%left TIMES DIVIDE MOD
%left EXP
%right NOT

%%

program:
  decls EOF { ( List.rev (fst $1), List.rev (snd $1)) }

decls:
  /* nothing */ { ([], []) }
  | decls fdecl { (($2 :: fst $1), snd $1) }
  | decls stmt  { (fst $1, ($2 :: snd $1)) }

fdecl:
  typ ID LPAREN formals_opt RPAREN COLON stmt_list END
  { {
    typ = $1;
    fname = $2;
    formals = $4;
    body = List.rev $7;
  } }
formals_opt:
  /* nothing */  { [] }
  | formals_list { List.rev $1 }

formals_list:
    typ ID { [($1, $2)] }
  | formals_list COMMA typ ID { ($3, $4) :: $1 }

typ:
    INT    { Int }
  | FLOAT  { Float }
  | BOOL   { Bool }
  | CHAR   { Char }
  | STRING { String }
  | VOID   { Void }
  | FILE   { File }
  | LIST   { List Unknown }
  | typ_list ARROW typ { Func($1, $3) }

typ_list_helper:
    typ { [$1] }
  | typ_list_helper COMMA typ { $3 :: $1 }

typ_list:
    LPAREN RPAREN { [] }
  | LPAREN typ_list_helper RPAREN { List.rev $2 }

vdecl:
    typ ID ASSIGN expr { Declaration($1, $2, $4) }
  | typ ID { Declaration($1, $2, Noexpr) }

stmt_list:
  /* nothing */ { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
  | 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)) }
```

```
  | expr SUBASN expr { Assign($1, Binop($1, Sub, $3)) }
  | expr MULASN expr { Assign($1, Binop($1, Mult, $3))  }
  | expr DIVASN expr { Assign($1, Binop($1, Div, $3))  }
  | expr MODASN expr { Assign($1, Binop($1, Mod, $3))  }

index:
    expr { Index $1 }
  | expr COLON expr { Slice($1, $3) }
  | COLON expr { Slice(IntLit 0, $2) }
  | expr COLON { Slice($1, End) }
  | COLON { Slice(IntLit 0, End) }

list_literal:
  /* nothing */ { [] }
  | expr        { [$1] }
  | list_literal COMMA expr { $3 :: $1 }

args_opt:
  /* nothing */ { [] }
  | args_list   { List.rev $1 }

args_list:
    expr { [$1] }
  | args_list COMMA expr { $3 :: $1 }
```

11.2.5 src/ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)
(* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
type op =
  | Add
  | Sub
  | Mult
  | Div
  | Mod
  | Equal
  | Neq
  | Less
  | Leq
  | Greater
  | Geq
  | And
  | Or
  | Exp
  | Range

type uop = Neg | Not

type expr =
  | IntLit of int
  | FloatLit of float
  | BoolLit of bool
  | CharLit of char
  | StringLit of string
  | ListLit of expr list
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | 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
```

```
let rec string_of_expr = function
  | IntLit l -> string_of_int l
  | FloatLit l -> string_of_float l
  | BoolLit true -> "true"
  | BoolLit false -> "false"
  | CharLit c -> String.make 1 c
  | StringLit s -> "\"" ^ s ^ "\""
  | ListLit l -> "[" ^ String.concat "," (List.map string_of_expr l) ^ "]"
  | SliceExpr (e, s) -> (
    match s with
    | Index i -> string_of_expr e ^ "[" ^ string_of_expr i ^ "]"
    | Slice (i, j) ->
        string_of_expr e ^ "[" ^ string_of_expr i ^ ":" ^ string_of_expr j ^
     "]"
    )
  | Id s -> s
  | Binop (e1, o, e2) -> (
    match o with
    | Range -> string_of_expr e1 ^ string_of_op o ^ string_of_expr e2
    | _ -> string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr
    e2 )
  | Unop (o, e) -> string_of_uop o ^ string_of_expr e
  | Assign (v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
  | Call (f, el) ->
      string_of_expr f ^ "("
      ^ String.concat ", " (List.map string_of_expr el)
      ^ ")"
  | End -> ""
  | Noexpr -> ""

let indent stmts_as_string =
  let rec take k xs =
    match k with
    | 0 -> []
    | k -> (
      match xs with [] -> failwith "take" | y :: ys -> y :: take (k - 1) ys
    )
  in
  let l = String.split_on_char '\n' stmts_as_string in
  let indent_stmt stmts_as_list =
    List.map (fun stmt_as_string -> "\t" ^ stmt_as_string) stmts_as_list
  in
  String.concat "\n" (indent_stmt (take (List.length l - 1) l)) ^ "\n"

let rec string_of_stmt = function
  | Block stmts -> String.concat "" (List.map string_of_stmt stmts)
  | Expr expr -> string_of_expr expr ^ ";\n"
  | Return expr -> "return " ^ string_of_expr expr ^ ";\n"
  | If (e, s1, s2) -> (
    match s2 with
    | Block [] ->
        "if " ^ string_of_expr e ^ ":\n" ^ indent (string_of_stmt s1) ^ "end
    \n"
    | _ ->
        "if " ^ string_of_expr e ^ ":\n"
        ^ indent (string_of_stmt s1)
        ^ "else:\n"
        ^ indent (string_of_stmt s2)
        ^ "end\n" )
```

```
  | For (s, e2, st) ->
      "for " ^ s ^ " in " ^ string_of_expr e2 ^ ":\n"
      ^ indent (string_of_stmt st)
      ^ "end\n"
  | While (e, s) ->
      "while " ^ string_of_expr e ^ ":\n" ^ indent (string_of_stmt s) ^ "end
    \n"
  | Declaration (t, id, e) -> (
    match e with
    | Noexpr -> string_of_typ t ^ " " ^ id ^ ";\n"
    | _ -> string_of_typ t ^ " " ^ id ^ " = " ^ string_of_expr e ^ ";\n" )
  | Break -> "break;\n"
  | Continue -> "continue;\n"

and string_of_typ = function
  | Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | Void -> "void"
  | Char -> "char"
  | String -> "string"
  | List t -> "list<" ^ string_of_typ t ^ ">"
  | Func (a, r) ->
      "("
      ^ String.concat "," (List.map string_of_typ a)
      ^ ")" ^ "->" ^ string_of_typ r
  | File -> "file"
  | Unknown -> "?"

let string_of_formals (formals : bind list) : string =
  String.concat ", "
    (List.map
       (fun (formal : bind) -> string_of_typ (fst formal) ^ " " ^ snd formal
    )
       formals )

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^ fdecl.fname ^ "("
  ^ string_of_formals fdecl.formals
  ^ "):\n"
  ^ indent (String.concat "" (List.map string_of_stmt fdecl.body))
  ^ "end\n"

let string_of_program (funcs, stmts) =
  String.concat "\n" (List.map string_of_fdecl funcs)
  ^ String.concat "" (List.map string_of_stmt stmts)
```

11.2.6 src/sast.ml

```
(* Semantically-checked Abstract Syntax Tree and functions for printing it
    *)
(* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
open Ast
module StringMap = Map.Make (String)

type symbol_table =
  { variables: typ StringMap.t
  ; (* Variables bound in current block *)
    functions: func_decl list
```

```
10          (* Functions that may need to be re-instantiated with concrete types
        *)
11    ; list_variables: symbol_table ref StringMap.t
12    ; parent: symbol_table option (* Enclosing scope *) }
13
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    ; rlist_variables: resolved_table ref StringMap.t
55    ; rparent: resolved_table option (* Enclosing scope *) }
56
57  type program = sfunc_decl list * sstmt list
58
59  (* Pretty-printing functions *)
60
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"
```

```
    | SBoolLit false -> "false"
    | SCharLit c -> String.make 1 c
    | SStringLit s -> "\"" ^ s ^ "\""
    | SListLit l -> "[" ^ String.concat "," (List.map string_of_sexpr l) ^ "
    ]"
    | SSliceExpr (e, s) -> (
      match s with
      | SIndex i -> string_of_sexpr e ^ "[" ^ string_of_sexpr i ^ "]"
      | SSlice (i, j) ->
          string_of_sexpr e ^ "[" ^ string_of_sexpr i ^ ":" ^
    string_of_sexpr j
          ^ "]" )
    | SId s -> s
    | SBinop (e1, o, e2) -> (
      match o with
      | Range -> string_of_sexpr e1 ^ string_of_op o ^ string_of_sexpr e2
      | _ ->
          string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr
    e2 )
    | SUnop (o, e) -> string_of_uop o ^ string_of_sexpr e
    | SAssign (v, e) -> string_of_sexpr v ^ " = " ^ string_of_sexpr e
    | SCall (f, el) ->
        string_of_sexpr f ^ "("
        ^ String.concat ", " (List.map string_of_sexpr el)
        ^ ")"
    | SEnd -> ""
    | SNoexpr -> "" )
  ^ ")"

let indent stmts_as_string =
  let rec take k xs =
    match k with
    | 0 -> []
    | k -> (
      match xs with [] -> failwith "take" | y :: ys -> y :: take (k - 1) ys
    )
  in
  let l = String.split_on_char '\n' stmts_as_string in
  let indent_stmt stmts_as_list =
    List.map (fun stmt_as_string -> "\t" ^ stmt_as_string) stmts_as_list
  in
  String.concat "\n" (indent_stmt (take (List.length l - 1) l)) ^ "\n"

let rec string_of_sstmt = function
  | SBlock stmts -> String.concat "" (List.map string_of_sstmt stmts)
  | SExpr expr -> string_of_sexpr expr ^ "\n"
  | SReturn expr -> "return " ^ string_of_sexpr expr ^ "\n"
  | SIf (e, s1, s2) -> (
    match s2 with
    | SBlock [] ->
        "if " ^ string_of_sexpr e ^ ":\n"
        ^ indent (string_of_sstmt s1)
        ^ "end\n"
    | _ ->
        "if " ^ string_of_sexpr e ^ ":\n"
        ^ indent (string_of_sstmt s1)
        ^ "else:\n"
        ^ indent (string_of_sstmt s2)
        ^ "end\n" )
```

```
  | SFor (s, e2, st) ->
      "for " ^ s ^ " in " ^ string_of_sexpr e2 ^ ":\n "
      ^ indent (string_of_sstmt st)
      ^ "end\n"
  | SWhile (e, s) ->
      "while " ^ string_of_sexpr e ^ ":\n"
      ^ indent (string_of_sstmt s)
      ^ "end\n"
  | SDeclaration (t, id, (tp, e)) -> (
    match e with
    | SNoexpr -> string_of_typ t ^ " " ^ id ^ "\n"
    | _ -> string_of_typ t ^ " " ^ id ^ " = " ^ string_of_sexpr (tp, e) ^ "\
    n" )
  | SBreak -> "break\n"
  | SContinue -> "continue\n"

let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^ fdecl.sfname ^ "("
  ^ String.concat ", " (List.map snd fdecl.sformals)
  ^ ")\n"
  ^ indent (String.concat "" (List.map string_of_sstmt fdecl.sbody))
  ^ "end\n"

let string_of_sprogram (funcs, stmts) =
  String.concat "" (List.map string_of_sfdecl funcs)
  ^ "\n"
  ^ String.concat "" (List.map string_of_sstmt stmts)
```

11.2.7 src/exceptions.ml

```
(* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
open Ast

exception NonUniformTypeContainer of typ * typ

exception UndefinedId of string

exception MismatchedTypes of typ * typ * expr

exception InvalidBinaryOperation of typ * op * typ * expr

exception InvalidUnaryOperation of typ * uop * expr

exception IllegalAssignment of typ * op option * typ * expr

exception IllegalDeclaration of typ * typ * stmt

exception NonListAccess of typ * typ * expr

exception InvalidIndex of typ * expr

exception TypeError of string

exception CannotRedefineBuiltIn of string

exception AlreadyDefined of string

exception VoidType of string

```

```
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',
     but \
98                got '%s'"
99               s1 s2 ) )
100  | UndefinedId n ->
101      raise
102        (TypeError
103           (Printf.sprintf "Error: variable '%s' was used before it was
     defined"
104              n ) )
105  | MismatchedTypes (t1, t2, e) ->
106      let s1 = string_of_typ t1
107      and s2 = string_of_typ t2
108      and s3 = string_of_expr e in
109      raise
110        (TypeError
111           (Printf.sprintf
112              "Type error: Expected value of type '%s', but got a value of \
113               type '%s' in '%s'"
114              s1 s2 s3 ) )
115  | InvalidBinaryOperation (t1, op, t2, e) ->
116      let s1 = string_of_typ t1
117      and s2 = string_of_op op
118      and s3 = string_of_typ t2
119      and s4 = string_of_expr e in
120      raise
121        (TypeError
122           (Printf.sprintf
123              "Type error: Illegal binary operator '%s' '%s' '%s' in '%s'"
     s1 s2
124              s3 s4 ) )
125  | InvalidUnaryOperation (t, op, e) ->
126      let s1 = string_of_typ t
127      and s2 = string_of_uop op
128      and s3 = string_of_expr e in
129      raise
130        (TypeError
131           (Printf.sprintf "Error: Illegal unary operator '%s' '%s' in '%s'"
      s2
132              s1 s3 ) )
133  | IllegalAssignment (t1, op, t2, e) ->
134      let s1 = string_of_typ t1
135      and s3 = string_of_typ t2
136      and s4 = string_of_expr e in
137      let s2 = match op with Some x -> string_of_op x | None -> "" in
138      raise
139        (TypeError
140           (Printf.sprintf "Error: Illegal assignment '%s' '%s'= '%s' in '%s
     '"
141              s1 s2 s3 s4 ) )
142  | 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
      s2
149               s3 ) )
150   | NonListAccess (t1, t2, e) ->
151       let s1 = string_of_typ t1
152       and s2 = string_of_typ t2
153       and s3 = string_of_expr e in
154       raise
155         (TypeError
156            (Printf.sprintf
157               "Type Error: Expected a lvalue of type List('%s'), but got \
158                lvalue of type '%s' in '%s'"
159               s1 s2 s3 ) )
160   | InvalidIndex (t1, e) ->
161       let s1 = string_of_typ t1 and s2 = string_of_expr e in
162       raise
163         (TypeError
164            (Printf.sprintf
165               "Type Error: Expected index of type Int, but got type '%s' in
      \
166                '%s'"
167               s1 s2 ) )
168   | CannotRedefineBuiltIn s ->
169       raise
170         (TypeError
171            (Printf.sprintf
172               "Error: Function '%s' may not be defined as it exists as a
      built \
173                in function"
174               s ) )
175   | AlreadyDefined s ->
176       raise
177         (TypeError
178            (Printf.sprintf
179               "Error: '%s' cannot be redefined in the current scope" s ) )
180   | VoidType n ->
181       raise
182         (TypeError
183            (Printf.sprintf "Type Error: variable '%s' cannot have type Void"
      n)
184         )
185   | Duplicate n ->
186       raise
187         (TypeError
188            (Printf.sprintf "Error: variable name '%s' has already defined" n
      ) )
189   | UndefinedFunction ->
190       raise
191         (TypeError
192            (Printf.sprintf "Error: function was called, but it is undefined"
      ) )
193   | WrongNumberOfArgs (exp, act, e) ->
194       let s1 = string_of_int exp
195       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
    '%s'"
200               s1 s2 s3 ) )
201   | PrintMissingArgs e ->
202       let s1 = string_of_expr e in
203       raise
204         (TypeError
205            (Printf.sprintf
206               "Error: expected more than 0 arguments but got 0 in '%s'" s1 )
     )
207   | PrintBadArgs e ->
208       raise
209         (TypeError
210            (Printf.sprintf "Error: expected either c, f or i but got '%c'" e
    ) )
211   | PrintWrongType e ->
212       let s1 = string_of_expr e in
213       raise (TypeError (Printf.sprintf "Error: expected string but got '%s'"
     s1))
214   | PrintWrongNumArgs (i1, i2) ->
215       let s1 = string_of_int i1 and s2 = string_of_int i2 in
216       raise
217         (TypeError
218            (Printf.sprintf "Error: expected %s addition arguments, but got %
    s"
219               s1 s2 ) )
220   | PrintTypeError (t1, t2) ->
221       let s1 = string_of_typ t1 and s2 = string_of_typ t2 in
222       raise
223         (TypeError
224            (Printf.sprintf
225               "Type Error: expected argument of type '%s' but got '%s'
    instead"
226               s1 s2 ) )
227   | IllegalArgument (t1, t2, e) ->
228       let s1 = string_of_typ t1
229       and s2 = string_of_typ t2
230       and s3 = string_of_expr e in
231       raise
232         (TypeError
233            (Printf.sprintf
234               "Type Error: Illegal argument found in '%s'. Expected argument
     \
235                of type '%s' but got '%s'"
236               s3 s1 s2 ) )
237   | IllegalSlice (e, t) ->
238       let s1 = string_of_expr e and s2 = string_of_typ t in
239       raise
240         (TypeError
241            (Printf.sprintf
242               "Type Error: Illegal Slice expression, expected '%s' to be \
243                either a list or array, but got a '%s'"
244               s1 s2 ) )
245   | WrongIndex (t, e) ->
246       let s1 = string_of_typ t and s2 = string_of_expr e in
247       raise
```

```
248         (TypeError
249            (Printf.sprintf
250               "Type Error: Expected index to be an int, but got '%s' in '%s'
    " s1
251               s2 ) )
252   | WrongSliceIndex (t1, t2, e1, e2) ->
253       let s1 = string_of_typ t1
254       and s2 = string_of_typ t2
255       and s3 = string_of_expr e1
256       and s4 = string_of_expr e2 in
257       raise
258         (TypeError
259            (Printf.sprintf
260               "Type Error: Expected left and right values of slice
    expression \
261                to be of type int, but got type '%s' and '%s' in '%s' and '%s
    '"
262               s1 s2 s3 s4 ) )
263   | ReturnNotLast ->
264       raise
265         (TypeError
266            (Printf.sprintf
267               "Error: There are unreacheable statements after return" ) )
268   | ReturnOutsideFunction ->
269       raise
270         (TypeError
271            (Printf.sprintf "Error: Return statement is outside of a function
    ")
272         )
273   | NoReturnInNonVoidFunction ->
274       raise
275         (TypeError
276            (Printf.sprintf
277               "Error: No return statement in function returning non-void" )
    )
278   | ReturnMismatchedTypes (t1, t2, s) ->
279       let s1 = string_of_typ t1
280       and s2 = string_of_typ t2
281       and s3 = string_of_stmt s in
282       raise
283         (TypeError
284            (Printf.sprintf
285               "Type Error: Expected value of type '%s', but got a value of \
286                type '%s' in '%s'"
287               s1 s2 s3 ) )
288   | InvalidFloatBinop ->
289       raise
290         (Failure
291            "Internal Error: Invalid operation on float. Semant should have \
292             rejected this" )
293   | InvalidIntBinop ->
294       raise
295         (Failure
296            "Internal Error: Invalid operation on int. Semant should have \
297             rejected this" )
298   | InvalidStringBinop ->
299       raise
300         (Failure
301            "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_typ t in
319       raise
320         (Failure
321            (Printf.sprintf
322               "Type Error: Expected first argument to append to be of type \
323                list, but got type '%s' instead"
324               s ) )
325   | LengthWrongArgument t ->
326       let s = string_of_typ t in
327       raise
328         (Failure
329            (Printf.sprintf
330               "Type Error: Expected argument to length to be of type list or
       \
331                string, but got type '%s' instead"
332               s ) )
333   | UnsupportedPrint t ->
334       let s = string_of_typ t in
335       raise
336         (Failure
337            (Printf.sprintf
338               "Type Error: Print does not support printing for type '%s'" s
      ) )
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         map
23     in
24     List.fold_left add_bind StringMap.empty
25       [ ("print", [(Unknown, "x")], Void)
26       ; ("open", [(String, "file_name"); (String, "mode")], File)
27       ; ("readline", [(File, "file_handle")], String)
28       ; ("write", [(File, "file_handle"); (String, "content")], Void)
29       ; ("close", [(File, "file_handle")], Void)
30       ; ("match", [(String, "target"); (String, "regex")], Bool)
31       ; ("find", [(String, "target"); (String, "regex")], String)
32       ; ( "replace"
33         , [ (String, "target")
34           ; (String, "regex")
35           ; (String, "replace")
36           ; (Int, "count") ]
37         , String )
38       ; ( "replaceall"
39         , [(String, "target"); (String, "regex"); (String, "replace")]
40         , String )
41       ; ("findall", [(String, "target"); (String, "regex")], List String)
42       ; ( "append"
43         , [(List Unknown, "input_list"); (Unknown, "element")]
44         , List Unknown )
45       ; ( "insert"
46         , [(List Unknown, "input_list"); (Unknown, "element"); (Int, "index"
     )]
47         , List Unknown )
48       ; ("length", [(Unknown, "input_list")], Int)
49       ; ("reverse", [(List Unknown, "input_list")], List Unknown) ]
```

```
  in
  (* fd.typ *)
  let add_func map fd =
    let n = fd.fname in
    (* Name of the function *)
    match fd with
    | _ when StringMap.mem n built_in_decls ->
        raise (E.CannotRedefineBuiltIn fd.fname)
    | _ when StringMap.mem n map -> raise (E.AlreadyDefined fd.fname)
    | _ -> StringMap.add n (func_ty fd) map
  in
  let function_decls = List.fold_left add_func built_in_decls functions in
  let variable_table =
    { variables= function_decls
    ; functions
    ; list_variables= StringMap.empty
    ; parent= None }
  in
  (* Create a reference to the global table. The scope will be passed
    through
     recurisve calls and be mutated when we need to add a new variable *)
  let global_scope = ref variable_table in
  let check_binds (to_check : bind list) =
    let name_compare (_, n1) (_, n2) = compare n1 n2 in
    let check_it checked binding =
      match binding with
      (* No void bindings *)
      | Void, name -> raise (E.VoidType name)
      | _, n1 -> (
        match checked with
        (* No duplicate bindings *)
        | (_, n2) :: _ when n1 = n2 -> raise (E.Duplicate n2)
        | _ -> binding :: checked )
    in
    let _ = List.fold_left check_it [] (List.sort name_compare to_check) in
    to_check
  in
  (* Finding a variable, beginning in a given scope and searching upwards *)
  let rec type_of_identifier (scope : symbol_table ref) name =
    try StringMap.find name !scope.variables
    with Not_found -> (
      match !scope.parent with
      | Some parent -> type_of_identifier (ref parent) name
      | _ -> raise (E.UndefinedId name) )
  in
  (* Add a variable to the given scope *)
  let add_var_to_scope (scope : symbol_table ref) id ty =
    try
      let _ = StringMap.find id !scope.variables in
      raise (E.Duplicate id)
    with Not_found ->
      scope :=
        { variables= StringMap.add id ty !scope.variables
        ; functions= !scope.functions
        ; list_variables= !scope.list_variables
        ; parent= !scope.parent }
  in
  let update_var (scope : symbol_table ref) id ty =
    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
        inferred *)
126   let get_list_scope id scope = StringMap.find_opt id !scope.list_variables
        in
127   let check_assign lvaluet rvaluet err =
128     if lvaluet = rvaluet then lvaluet
129     else
130       let ret =
131         match (lvaluet, rvaluet) with
132         | List Unknown, List ty -> List ty
133         | List ty, List Unknown -> List ty
134         | Void, List Unknown -> List Unknown
135         | ty, Unknown -> ty
136         | Unknown, ty -> ty
137         | _ -> raise err
138       in
139       ret
140   in
141   let rec innermost_ty ty =
142     match ty with List t -> innermost_ty t | nonlist_ty -> nonlist_ty
143   in
144   (* Return a semantically-checked expression with a type *)
145   let rec expr scope exp =
146     match exp with
147     | IntLit l -> (Int, SIntLit l)
148     | FloatLit l -> (Float, SFloatLit l)
149     | BoolLit l -> (Bool, SBoolLit l)
150     | CharLit l -> (Char, SCharLit l)
151     | StringLit l -> (String, SStringLit l)
152     | ListLit es -> (
153         let ts = List.map (fun x -> fst (expr scope x)) es in
154         match ts with
155         | [] -> (List Unknown, SListLit [])
156         | x :: _ ->
157             let ty = List.hd ts in
158             let check_type e =
159               let ty', e' = expr scope e in
160               if ty' = ty then (ty', e')
161               else raise (E.NonUniformTypeContainer (ty, ty'))
162             in
163             (List x, SListLit (List.map check_type es)) )
164     | Id s -> (type_of_identifier scope s, SId s)
```

```
    | Binop (e1, op, e2) as e ->
        let t1, e1' = expr scope e1 and t2, e2' = expr scope e2 in
        let same = t1 = t2 in
        let ty =
          match op with
          | (Add | Sub | Mult | Div | Mod) when same && t1 = Int -> Int
          | (Add | Sub | Mult | Div) when same && t1 = Float -> Float
          | (Add | Sub | Mult | Div) when t1 = Int && t2 = Float -> Float
          | (Add | Sub | Mult | Div) when t1 = Float && t2 = Int -> Float
          | Add when t1 = String && t2 = String -> String
          | Add when t1 = String && t2 = Char -> String
          | Add when t1 = Char && t2 = String -> String
          | Exp when same && t1 = Int -> Int
          | Exp when same && t1 = Float -> Float
          | Exp when t1 = Int && t2 = Float -> Float
          | Exp when t1 = Float && t2 = Int -> Float
          | (Equal | Neq) when t1 = Int && t2 = Float -> Bool
          | (Equal | Neq) when t1 = Float && t2 = Int -> Bool
          | (Equal | Neq) when same -> Bool
          | (Equal | Neq) when t1 = Unknown || t2 = Unknown -> Bool
          | (Less | Leq | Greater | Geq) when t1 = Int && t2 = Float -> Bool
          | (Less | Leq | Greater | Geq) when t1 = Float && t2 = Int -> Bool
          | (Less | Leq | Greater | Geq) when same && t1 = Int -> Bool
          | (Less | Leq | Greater | Geq) when same && t1 = Float -> Bool
          | (And | Or) when same && t1 = Bool -> Bool
          | Range when same && t1 = Int -> List Int
          | Add when same && match t1 with List _ -> true | _ -> false -> t1
          | _ when t1 = Unknown && t2 <> Unknown -> t2
          | _ when t1 <> Unknown && t2 = Unknown -> t1
          | _ when same && t1 = Unknown -> t1
          | _ -> raise (E.InvalidBinaryOperation (t1, op, t2, e))
        in
        (ty, SBinop ((t1, e1'), op, (t2, e2')))
    | Unop (op, e) as ex ->
        let t, e' = expr scope e in
        let ty =
          match op with
          | Neg when t = Int || t = Float -> t
          | Not when t = Bool -> Bool
          | _ -> raise (E.InvalidUnaryOperation (t, op, ex))
        in
        (ty, SUnop (op, (t, e')))
    | Assign (s, e) as ex ->
        let lt, s' = expr scope s in
        let rt, e' = expr scope e in
        let lrt = check_assign lt rt (E.IllegalAssignment (lt, None, rt, ex)
    ) in
        let s_name =
          match s with
          | Id n -> n
          | SliceExpr (Id n, _) -> n
          | _ -> raise (E.AssignNonVar ex)
        in
        let is_slice = match s with SliceExpr _ -> true | _ -> false in
        let is_list = match lt with List _ -> true | _ -> false in
        let non_slice =
          match (lt, rt) with
          | List _, List _ | Void, List _ ->
              let _ = update_var scope s_name lrt in
```

```
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,
    call))
247       else ()
248     in
249     let fty, fname' = expr scope fname in
250     let formals, ret_type =
251       match fty with
252       | Func (f, r) -> (f, r)
253       | _ -> raise E.UndefinedFunction
254     in
255     let _ = if fname <> Id "print" then check_length formals else () in
256     match fname with
257     | Id "print" ->
258         let et, _ = expr scope (hd args) in
259         let _ =
260           match et with String -> () | _ -> raise (E.UnsupportedPrint et
    )
261         in
262         ( Void
263         , SCall
264             ((Func ([et], Void), SId "print"), List.map (expr scope)
    args)
265         )
266     | Id "append" ->
267         let args' = List.map (expr scope) args in
268         let et1, args1' = hd args' in
269         let et2, args2' = hd (tl args') in
270         let inner_ty =
271           match et1 with List ty -> ty | _ -> raise (E.AppendNonList et1
    )
272         in
273         let is_list et = match et with List _ -> true | _ -> false in
274         let _ =
275           if
276             is_list et2
277             && 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            in
317            let is_list et = match et with List _ -> true | _ -> false in
318            let _ =
319              if
320                is_list et2
321                && innermost_ty et2 = Unknown
322                && innermost_ty et1 <> Unknown
323              then ()
324              else if innermost_ty et1 <> et2 && innermost_ty et1 != Unknown
325              then raise (E.MismatchedTypes (inner_ty, et2, call))
326              else ()
327            in
328            if innermost_ty et1 = Unknown then
329              ( List et2
330              , SCall
331                  ( (Func ([List et2; et2], List et2), SId "insert")
332                  , [(List et2, args1'); (et2, args2'); (et3, args3')] ) )
333            else if is_list et2 && innermost_ty et2 = Unknown then
334              ( 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       | Id "split" ->
354           let args' = List.map (expr scope) args in
355           let et1, _ = hd args' in
356           let _ =
357             match et1 with
358             | String -> ()
359             | _ -> raise (Failure "Mismatched types")
360           in
361           ( List String
362           , SCall ((Func ([String; Char], List String), SId "split"), args
    ')
363           )
364       | Id "string_to_list" ->
365           let args' = List.map (expr scope) args in
366           let et1, _ = hd args' in
367           let _ =
368             match et1 with
369             | String -> ()
370             | _ -> raise (Failure "Mismatched types")
371           in
372           ( List Char
373           , SCall ((Func ([String], List Char), SId "string_to_list"),
    args')
374           )
375       | _ ->
376           let check_call ft e =
377             let et, e' = expr scope e in
378             (check_assign ft et (E.IllegalArgument (ft, et, e)), e')
379           in
380           let args' = List.map2 check_call formals args in
381           let arg_types = List.map (fun e -> fst (expr scope e)) args in
382           (* get all list type arguments *)
383           let list_args =
384             List.filter
385               (fun x -> match x with List _ -> true | _ -> false)
386               arg_types
387           in
388           if length list_args > 0 then
389             let inner_types =
390               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_typ 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
```

```
                ( ret_type
                , SCall ((Func (arg_types, ret_type), SId new_fname), args')
    )
            else (ret_type, SCall ((fty, fname'), args')) )
    | SliceExpr (lexpr, slce) as slice ->
        let lt, lexpr' = expr scope lexpr in
        let check_slice_expr =
          match slce with
          | Index e ->
              let t, e' = expr scope e in
              let id_type =
                match lt with
                | List ty -> ty
                | String -> Char
                | _ -> raise (E.IllegalSlice (slice, lt))
              in
              if t = Int then
                (id_type, SSliceExpr ((lt, lexpr'), SIndex (t, e')))
              else raise (E.WrongIndex (t, e))
          | Slice (e1, e2) ->
              let t1, e1' = expr scope e1 and t2, e2' = expr scope e2 in
              let id_type =
                match lt with
                | List _ -> lt
                | String -> lt
                | _ -> raise (E.IllegalSlice (slice, lt))
              in
              if t1 = Int && t1 = t2 then
                ( id_type
                , SSliceExpr ((lt, lexpr'), SSlice ((t1, e1'), (t2, e2'))) )
              else raise (E.WrongSliceIndex (t1, t2, e1, e2))
        in
        check_slice_expr
    | End -> (Int, SEnd)
    | Noexpr -> (Void, SNoexpr)
  in
  let check_bool_expr scope e =
    let t', e' = expr scope e in
    if t' != Bool then raise (E.MismatchedTypes (t', Bool, e)) else (t', e')
  in
  (* check void type variable *)
  let check_void_type ty name =
    match ty with Void -> raise (E.VoidType name) | _ -> ty
  in
  let dummy = {typ= Int; fname= "toplevel"; formals= []; body= []} in
  (* Checks if there are any statements after return *)
  let rec check_return sl typ =
    match sl with
    | [] -> if typ != Void then raise E.NoReturnInNonVoidFunction else ()
    | [Return _] -> ()
    | Return _ :: _ -> raise E.ReturnNotLast
    | _ :: ss -> check_return ss typ
  in
  (* Return a semantically-checked statement containing exprs *)
  let rec check_stmt scope stmt loop fdecl =
    match stmt with
    | Expr e -> SExpr (expr scope e)
    | Block sl ->
        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     | "toplevel" -> 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")
```

```
    | Continue -> if loop > 0 then SContinue else raise (E.NotInLoop "Continue")
  in
  let check_functions func =
    let formals' = check_binds func.formals in
    let add_formal map (ty, name) = StringMap.add name ty map in
    let func_variable_table =
      { variables= List.fold_left add_formal StringMap.empty formals'
      ; functions= !global_scope.functions
      ; list_variables= !global_scope.list_variables
      ; parent= Some !global_scope }
    in
    let func_scope = ref func_variable_table in
    let _ = check_return func.body func.typ in
    let body' = check_stmt func_scope (Block func.body) 0 func in
    if func.typ = List Unknown then
      let updated_func =
        List.filter (fun f -> f.fname = func.fname) !global_scope.functions
      in
      let _ =
        if length updated_func = 0 then raise E.UndefinedFunction else ()
      in
      (* update function type to be type specific *)
      { styp= (hd updated_func).typ
      ; sfname= func.fname
      ; sformals= formals'
      ; sbody= [body'] }
    else {styp= func.typ; sfname= func.fname; sformals= formals'; sbody= [body']}
  in
  let check_stmts stmt = check_stmt global_scope stmt 0 dummy in
  let statements' =
    try List.map check_stmts statements with e -> E.handle_error e
  in
  let functions' =
    try List.map check_functions !global_scope.functions
    with e -> E.handle_error e
  in
  (functions', statements')
```

11.2.9 src/resolve.ml

```
(* Authors: Lulu Z. *)
open Ast
open Sast
module StringMap = Map.Make (String)
open List
module E = Exceptions

let resolve (functions, statements) =
  let variable_table : resolved_table =
    { rvariables= StringMap.empty
    ; rfunctions= functions
    ; rlist_variables= StringMap.empty
    ; rparent= None }
  in
  (* Create a reference to the global table. The scope will be passed through
     recurisve calls and be mutated when we need to add a new variable *)
```

```
let global_scope : resolved_table ref = ref variable_table in
(* Add a variable to the given scope *)
let add_var (scope : resolved_table ref) id ty =
  scope :=
    { rvariables= StringMap.add id ty !scope.rvariables
    ; rfunctions= !scope.rfunctions
    ; rlist_variables= !scope.rlist_variables
    ; rparent= !scope.rparent }
in
(* Finding a variable, beginning in a given scope and searching upwards *)
let rec type_of_identifier (scope : resolved_table ref) id =
  try StringMap.find id !scope.rvariables
  with Not_found -> (
    match !scope.rparent with
    | Some parent -> type_of_identifier (ref parent) id
    | _ -> raise (E.UndefinedId id) )
in
(* Add the scope of list variable with name id *)
let add_list_scope id (scope : resolved_table ref) =
  scope :=
    { rvariables= !scope.rvariables
    ; rfunctions= !scope.rfunctions
    ; rlist_variables= StringMap.add id scope !scope.rlist_variables
    ; rparent= !scope.rparent }
in
(* For finding a list outside of current scope that hasn't been type
  inferred *)
let get_list_scope id scope = StringMap.find_opt id !scope.rlist_variables
   in
let look_up_func (funcs : sfunc_decl list) fname =
  List.find_opt (fun f -> f.sfname = fname) funcs
in
let func_ty fd =
  let param_types = List.map (fun (a, _) -> a) fd.sformals in
  Func (param_types, fd.styp)
in
let rec expr scope ((t, e) : sexpr) =
  match e with
  | SIntLit i -> (t, SIntLit i)
  | SFloatLit f -> (t, SFloatLit f)
  | SBoolLit b -> (t, SBoolLit b)
  | SCharLit c -> (t, SCharLit c)
  | SStringLit s -> (t, SStringLit s)
  | SId n -> (
    try
      if (type_of_identifier scope n, SId n) = (List Unknown, SId n) then
        (t, SId n)
      else (type_of_identifier scope n, SId n)
    with _ -> (t, SId n) )
  | SListLit l -> (t, SListLit l)
  | SSliceExpr (lexpr, slce) ->
      let lt, lexpr' = expr scope lexpr in
      let check_slice_expr =
        match slce with
        | SIndex e ->
            let t, e' = expr scope e in
            let id_type =
              match lt with
              | List ty -> ty
```

```
74                | String -> Char
75                | _ -> raise E.IllegalSSlice
76              in
77              (id_type, SSliceExpr ((lt, lexpr'), SIndex (t, e')))
78          | SSlice _ -> (
79              (* look up new inferred type from the symbol table *)
80              let id_ty, id =
81                match lexpr 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 (lexpr, slce))
89              | List _ -> (id_ty, SSliceExpr ((id_ty, id), slce))
90              | _ -> (t, SSliceExpr (lexpr, 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             (* if new function hasn't been created yet *)
151             let new_func_created =
152               List.find_opt (fun f -> f.sfname = new_fname) functions
153             in
154             let _, ret_type =
155               match fty with
156               | Func (f, r) -> (f, r)
157               | _ -> raise E.UndefinedFunction
158             in
159             (* create new func with new types *)
160             if Option.is_none new_func_created then
161               let modified_func =
162                 { styp= ret_type
163                 ; sfname= new_fname
164                 ; sformals=
165                     List.combine resolved_arg_tys (List.map snd func.
    sformals)
166                 ; sbody= func.sbody }
167               in
168               (* remove old function from the function list *)
169               let _ =
170                 global_scope :=
171                   { rvariables= !global_scope.rvariables
172                   ; rfunctions=
173                       modified_func
174                       ::
175                       List.filter
176                         (fun f -> f.sfname != fname)
177                         !global_scope.rfunctions
178                   ; rlist_variables= !scope.rlist_variables
179                   ; rparent= !global_scope.rparent }
180               in
181               ( ret_type
182               , SCall
183                   ((Func (resolved_arg_tys, ret_type), SId new_fname),
    args')
184               )
185             else
186               ( ret_type
187               , SCall
```

```
                    ((Func (resolved_arg_tys, ret_type), SId new_fname),
  args')
              )
          else
            let ret =
              match func.styp with
              | List _ -> (func.styp, SCall ((func_ty func, SId fname),
  args'))
              | _ -> (t, SCall (f, args'))
            in
            ret
        else (t, SCall (f, args))
    | _, _ -> raise E.IllegalFname )
  | SEnd -> (t, SEnd)
  | SNoexpr -> (Void, SNoexpr)
in
let rec stmt scope st =
  match st with
  | SExpr e -> SExpr (expr scope e)
  | SBlock sl ->
      let new_scope : resolved_table =
        { rvariables= StringMap.empty
        ; rfunctions= !scope.rfunctions
        ; rlist_variables= !scope.rlist_variables
        ; rparent= Some !scope }
      in
      let new_scope_ref = ref new_scope in
      let rec stmt_list = function
        | SBlock sl :: ss -> stmt_list (sl @ ss)
        | s :: ss -> stmt new_scope_ref s :: stmt_list ss
        | [] -> []
      in
      SBlock (stmt_list sl)
  | SReturn e -> SReturn e
  | SIf (p, then_stmt, else_stmt) -> SIf (expr scope p, then_stmt,
  else_stmt)
  | SFor (s, e, sl) ->
      let t, e' = expr scope e in
      let list_name = match e' with SId s -> Some s | _ -> None in
      (* get resolved type if expr sl is a list *)
      let inner_ty =
        match t with
        | List ty -> ty
        | String -> Char
        | _ -> raise E.IllegalFor
      in
      let resolved_ty =
        if t = List Unknown && Option.is_some list_name then
          type_of_identifier scope (Option.get list_name)
        else t
      in
      let _ = add_var scope s inner_ty in
      let sexpr = SFor (s, (resolved_ty, e'), stmt scope sl) in
      let _ =
        scope :=
          { rvariables= StringMap.remove s !scope.rvariables
          ; rfunctions= !scope.rfunctions
          ; rlist_variables= !scope.rlist_variables
          ; 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         else if resolved_ty = Unknown then
261           let _ = add_var scope s ty in
262           SDeclaration (ty, s, (resolved_ty, e'))
263         else if ty = List Unknown && not is_generic_list then
264           let _ = add_var scope s resolved_ty in
265           SDeclaration (resolved_ty, s, (resolved_ty, e'))
266         else if resolved_ty = List Unknown then
267           let _ = add_list_scope s scope in
268           let _ = add_var scope s ty in
269           SDeclaration (ty, s, (resolved_ty, e'))
270         else SDeclaration (ty, s, (resolved_ty, e'))
271     | SBreak -> SBreak
272     | SContinue -> SContinue
273   in
274   let check_functions func =
275     let add_formal map (ty, name) = StringMap.add name ty map in
276     let func_variable_table =
277       { rvariables= List.fold_left add_formal StringMap.empty func.sformals
278       ; rfunctions= !global_scope.rfunctions
279       ; rlist_variables= !global_scope.rlist_variables
280       ; rparent= Some !global_scope }
281     in
282     let func_scope = ref func_variable_table in
283     let body' = stmt func_scope (SBlock func.sbody) in
284     if func.styp = List Unknown then
285       let updated_func =
286         List.filter (fun f -> f.sfname = func.sfname) !global_scope.
    rfunctions
287       in
288       let _ =
289         if length updated_func = 0 then raise E.UndefinedFunction else ()
290       in
291       (* func.typ should be updated to type specific *)
292       { styp= (hd updated_func).styp
293       ; sfname= func.sfname
294       ; sformals= func.sformals
295       ; sbody= [body'] }
296     else
297       { styp= func.styp
298       ; sfname= func.sfname
299       ; sformals= func.sformals
300       ; sbody= [body'] }
```

```
  in
  let statements' =
    try List.map (stmt global_scope) statements with e -> E.handle_error e
  in
  let functions' =
    try List.map check_functions !global_scope.rfunctions
    with e -> E.handle_error e
  in
  (functions', statements')
```

11.2.10 src/codegen.ml

```
(* Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
module L = Llvm
module A = Ast
open Sast
module E = Exceptions
module StringMap = Map.Make (String)
(* module Array *)

type var_table =
  {lvariables: L.llvalue StringMap.t; parent: var_table ref option}

let translate (functions, statements) =
  (* defining main function as the list of statements on the top level*)
  let main_func =
    {styp= A.Int; sfname= "main"; sformals= []; sbody= statements}
  in
  let functions = [main_func] @ functions in
  let context = L.global_context () in
  let i32_t = L.i32_type context
  and char_t = L.i8_type context
  and i8_t = L.i8_type context
  and void_t = L.void_type context
  and float_t = L.double_type context
  and i1_t = L.i1_type context
  and string_t = L.pointer_type (L.i8_type context)
  and file_t = L.pointer_type (L.i8_type context)
  and the_module = L.create_module context "TEAM" in
  let list_struct_type = L.named_struct_type context "list_item" in
  let list_struct_ptr = L.pointer_type list_struct_type in
  (* list is a linked list with a ptr to the data and the next ptr *)
  let _ =
    L.struct_set_body list_struct_type
      [|L.pointer_type i8_t; list_struct_ptr|]
      true
  in
  (* Convert TEAM types to LLVM types *)
  let rec ltype_of_typ = function
    | A.Int -> i32_t
    | A.String -> string_t
    | A.Bool -> i1_t
    | A.Float -> float_t
    | A.Void -> void_t
    | A.Char -> char_t
    | A.Unknown -> i32_t
    | A.Func (args_t, ret_t) -> func_ty args_t ret_t
    | A.List _ -> L.pointer_type list_struct_ptr
    | 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"
    builder
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                 in
203                 let mmcpy =
204                   L.declare_function "llvm.memcpy.p0i8.p0i8.i32" mmcpy_t
205                     the_module
206                 in
207                 (* Copying the slice to newly allocated space *)
208                 let _ =
209                   L.build_call mmcpy
210                     [|new_str; ptr; length; (L.const_int i1_t) 1|]
211                     "" builder
212                 in
213                 new_str )
214           | A.List _ -> (
215             match slice with
216             | 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                 (* Casting "void" pointer back to data pointer *)
230                 let type_casted =
231                   L.build_bitcast dat_ptr
232                     (L.pointer_type (ltype_of_typ t))
233                     "cast_data_ptr" builder
234                 in
235                 L.build_load type_casted "data" builder
236             | 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                 (* Checking for [i:] type of slice*)
243                 let j =
244                   match j with
245                   | _, SEnd -> L.const_int i32_t (-1)
246                   | _ -> L.build_sub (expr sc builder j) i "difference"
    builder
247                 in
248                 let lc_func = build_copy_function t in
249                 let new_list_ptr_ptr =
250                   L.build_malloc list_struct_ptr "new_list_ptr_ptr" builder
251                 in
252                 (* Copy the list slice into a new list *)
253                 let _ =
254                   L.build_call lc_func
255                     [|item_ptr; j; new_list_ptr_ptr|]
256                     "" builder
257                 in
258                 new_list_ptr_ptr )
259           | _ -> raise (Failure "Internal error: invalid slice") )
260       | SBinop (e1, op, e2) ->
261           let t1, _ = e1
262           and t2, _ = e2
263           and e1' = expr sc builder e1
264           and e2' = expr sc builder e2 in
265           (* Check for operand types *)
266           if t1 = A.Float && t2 = A.Int then
267             let cast_e2' = L.build_sitofp e2' float_t "cast" builder in
268             match op with
269             | A.Add -> L.build_fadd e1' cast_e2' "tmp" builder
270             | A.Sub -> L.build_fsub e1' cast_e2' "tmp" builder
271             | A.Mult -> L.build_fmul e1' cast_e2' "tmp" builder
272             | 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                 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 =
```

```
                  L.function_type void_t
                    [|L.pointer_type i8_t; L.pointer_type i8_t; i32_t; i1_t
|]
              in
              let mmcpy =
                L.declare_function "llvm.memcpy.p0i8.p0i8.i32" mmcpy_t
                  the_module
              in
              let _ =
                L.build_call mmcpy
                  [|new_str; e1'; length1; (L.const_int i1_t) 1|]
                  "" builder
              in
              let new_spot =
                L.build_gep new_str [|length1|] "new_spot" builder
              in
              let _ =
                L.build_call mmcpy
                  [|new_spot; e2'; length2; (L.const_int i1_t) 1|]
                  "" builder
              in
              let nul =
                L.build_gep new_str [|new_length|] "string_term" builder
              in
              let _ = L.build_store ((L.const_int i8_t) 0) nul builder in
              new_str
          | A.Equal ->
              (* Checking if strings are equal *)
              let strcmp_func = build_strcmp_function () in
              L.build_call strcmp_func [|e1'; e2'|] "strcmp_eq" builder
          | A.Neq ->
              let strcmp_func = build_strcmp_function () in
              let bool_val = L.build_call strcmp_func [|e1'; e2'|] "
strcmp_eq" builder in
              L.build_select bool_val (L.const_int i1_t 0) (L.const_int
i1_t 1) "strcmp_neq" builder
          | _ -> raise E.InvalidStringBinop
        else if t1 = A.String && t2 = A.Char then
          match op with
          | A.Add ->
            (* Adding character to a string *)
            let sl_func = build_string_length_function () in
            let length1 =
              L.build_call sl_func
                [|e1'; L.const_int i32_t 0|]
                "length" builder
            in
            let length2 =
              (L.const_int i32_t 1)
            in
            let new_length =
              L.build_add length1 length2 "new_length" builder
            in
            let new_length_w_null =
              L.build_add new_length (L.const_int i32_t 1) "new_length_nul
"
                builder
            in
            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                 builder
486             in
487             let new_str =
488               L.build_array_malloc i8_t new_length_w_null "new_string"
489                 builder
490             in
491             let mmcpy_t =
492               L.function_type void_t
493                 [|L.pointer_type i8_t; L.pointer_type i8_t; i32_t; i1_t|]
494             in
495             let mmcpy =
496               L.declare_function "llvm.memcpy.p0i8.p0i8.i32" mmcpy_t
497                 the_module
498             in
499             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
     in
518           let _ = L.build_store (L.const_null list_struct_ptr) new_list
    builder in
519           let slice = SSlice((A.Int, SIntLit 0), (A.Int, SIntLit 0)) in
520           let ilst = get_list_inner_typ t1 in
521           (* Copying the first list *)
522           let _ = build_asn_list sc builder ilst new_list slice (expr sc
    builder e2) in
523           (* Copying the second list *)
524           let _ = build_asn_list sc builder ilst new_list slice (expr sc
    builder e1) in
525           new_list
526         else (
527           print_endline (A.string_of_typ t1) ;
528           print_endline (A.string_of_typ t2) ;
529           raise (Failure "Not Yet Implemented") )
530     | SUnop (op, e) ->
531         (* Building unary operation code *)
532         let t, _ = e in
533         let e' = expr sc builder e in
534         ( match op with
535         | A.Neg when t = A.Float -> L.build_fneg
536         | A.Neg -> L.build_neg
537         | A.Not -> L.build_not )
538           e' "tmp" builder
539     | SAssign (le, re) ->
540         let re' = expr sc builder re in
541         let _ =
542           match le with
543           (* Updating regular variable assign *)
544           | _, SId s -> update_variable sc s re' builder
545           | _, SSliceExpr ((lst, ls), slc) -> (
546             match lst with
547             | A.List ilst ->
548                 (* Assigning list/element to a list slice/index *)
549                 let lis = expr sc builder (lst, ls) in
550                 build_asn_list sc builder ilst lis slc re'
551             | A.String -> raise (Failure "Cannot assign to string slice")
552             | _ -> raise (Failure "Internal Error") )
553           | _ -> raise (Failure "Internal Error")
554         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             "open" builder
573       | SCall ((_, SId "close"), [(A.File, file)]) ->
574           L.build_call close_func
575             [|expr sc builder (A.File, file)|]
576             "close" builder
577       | SCall ((_, SId "readline"), [(A.File, file)]) ->
578           L.build_call readline_func
579             [|expr sc builder (A.File, file)|]
580             "readline" builder
581       | SCall ((_, SId "write"), [(A.File, file); (A.String, st)]) ->
582           L.build_call write_func
583             [|expr sc builder (A.File, file); expr sc builder (A.String, st)
    |]
584             "write" builder
585       (* Defining reverse function for lists *)
586       | SCall ((_, SId "reverse"), [(A.List lt, lst)]) ->
587           let reverse_func = build_list_reverse_function () in
588           let list_ptr_ptr = expr sc builder (A.List lt, lst) in
589           let list_ptr = L.build_load list_ptr_ptr "list_ptr" builder in
590           let new_list_ptr_ptr =
591             L.build_malloc list_struct_ptr "new_list_ptr_ptr" builder
592           in
593           let _ =
594             L.build_store
595               (L.const_null list_struct_ptr)
596               new_list_ptr_ptr builder
597           in
598           let lc_func = build_copy_function (A.List lt) in
599           let _ =
600             L.build_call lc_func
601               [|list_ptr; L.const_int i32_t (-1); new_list_ptr_ptr|]
602               "last_node_ptr_ptr" builder
603           in
604           L.build_call reverse_func [|new_list_ptr_ptr|] "reversed_list"
    builder
605       | SCall ((_, SId "print"), args) ->
606           let eval_arg e =
607             let t, _ = e in
608             match t with
609             (* Adding the special case for boolean *)
610             | 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            "match" builder
632      | SCall ((_, SId "find"), [(A.String, st); (A.String, st2)]) ->
633          L.build_call find_func
634            [|expr sc builder (A.String, st); expr sc builder (A.String, st2
    )|]
635            "find" builder
636      | SCall
637          ( (_, SId "replace")
638          , [(A.String, st); (A.String, st2); (A.String, st3); (A.Int, i)] )
     ->
639          L.build_call replace_func
640            [| expr sc builder (A.String, st)
641             ; expr sc builder (A.String, st2)
642             ; expr sc builder (A.String, st3)
643             ; expr sc builder (A.Int, i) |]
644            "replace" builder
645      | SCall
646          ( (_, SId "replaceall")
647          , [(A.String, st); (A.String, st2); (A.String, st3)] ) ->
648          L.build_call replaceall_func
649            [| expr sc builder (A.String, st)
650             ; expr sc builder (A.String, st2)
651             ; expr sc builder (A.String, st3) |]
652            "replaceall" builder
653      | SCall ((_, SId "findall"), [(A.String, st); (A.String, st2)]) ->
654          L.build_call findall_func
655            [|expr sc builder (A.String, st); expr sc builder (A.String, st2
    )|]
656            "findall" builder
657      (* Appending to lists *)
658      | SCall ((_, SId "append"), [(lt, lst); e]) ->
659          let list_ptr_ptr = expr sc builder (lt, lst) in
660          let list_ptr = L.build_load list_ptr_ptr "list_ptr" builder in
661          let e' = expr sc builder e in
662          let ll_func = build_list_length_function () in
663          let length =
664            L.build_call ll_func
665              [|list_ptr; L.const_int i32_t 0|]
```

```
              "length" builder
          in
          (* Using insert function to insert at the end *)
          let insert_func = build_insert_function lt in
          L.build_call insert_func
            [|list_ptr_ptr; e'; length|]
            "list_ptr_ptr" builder
      (* Inserting to list at a certain index *)
      | SCall ((_, SId "insert"), [(lt, lst); e; i]) ->
          let list_ptr_ptr = expr sc builder (lt, lst) in
          let e' = expr sc builder e in
          let i' = expr sc builder i in
          let insert_func = build_insert_function lt in
          L.build_call insert_func [|list_ptr_ptr; e'; i'|] "list_ptr_ptr"
            builder
      (* Function call *)
      | SCall (f, args) ->
          (* Finding the function *)
          let fdef = expr sc builder f in
          (* Evaluating the function arguments *)
          let llarg = List.rev (List.map (expr sc builder) (List.rev args))
    in
          let ret_type =
            match f with
            | A.Func (_, rett), _ -> rett
            | _ -> raise (Failure "Internal Error")
          in
          let result = match ret_type with A.Void -> "" | _ -> "_result" in
          L.build_call fdef (Array.of_list llarg) result builder
      | SEnd -> raise (Failure "Not Yet Implemented")
      | SNoexpr -> L.const_null i32_t
    (* Function to generate code for assigning to list *)
    and build_asn_list sc builder ilst lis slc re' =
      match slc with
      (* In case of assigning to index *)
      | SIndex i ->
          let la_func = build_access_function () in
          let lis = L.build_load lis "ilist" builder in
          (* Accessing the struct at the given index *)
          let item_ptr =
            L.build_call la_func [|lis; expr sc builder i|] "result" builder
          in
          (* Accessing the data field in the struct *)
          let data_ptr_ptr =
            L.build_struct_gep item_ptr 0 "data_ptpt" builder
          in
          (* Allocating new space for copying the data *)
          let copy_data_ptr =
            L.build_malloc (ltype_of_typ ilst) "copy_ptr" builder
          in
          let _ = L.build_store re' copy_data_ptr builder in
          (* Casting the data pointer to "void" pointer *)
          let type_casted_copy =
            L.build_bitcast copy_data_ptr (L.pointer_type i8_t) "ccopy"
    builder
          in
          (* Storing the pointer to data *)
          let _ = L.build_store type_casted_copy data_ptr_ptr builder in
          ()
```

```
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 i1_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 i1_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 i1_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)
    in
934           (* Checking if the index is 0 which means list is done copying*)
935           let i_cond =
936             L.build_icmp L.Icmp.Eq (L.param lc_func 1) (L.const_int i32_t 0)
937               "is_zero" lc_builder
938           in
939           (* Checking if the list being copied is a null pointer *)
940           let n_cond =
941             L.build_is_null (L.param lc_func 0) "ptr_is_null" lc_builder
942           in
943           let bool_val = L.build_or i_cond n_cond "or_conds" lc_builder in
944           let then_bb = L.append_block context "then" lc_func in
945           (* Base case of recursion: Return the pointer to the next pointer
946              of the struct at the end *)
947           let _ =
948             L.build_ret (L.param lc_func 2) (L.builder_at_end context
    then_bb)
949           in
950           let else_bb = L.append_block context "else" lc_func in
951           let else_builder = L.builder_at_end context else_bb in
952           (* Allocating space for a new struct *)
953           let new_struct_ptr =
954             L.build_malloc list_struct_type "new_struct_ptr" else_builder
955           in
956           let _ =
957             L.build_store
958               (L.const_null list_struct_type)
959               new_struct_ptr else_builder
960           in
961           (* Allocating space for the data *)
962           let data_ptr = L.build_malloc (ltype_of_typ t) "ltyp" else_builder
     in
963           (* Getting the old data to be copied *)
964           let old_data_ptr_ptr =
965             L.build_struct_gep (L.param lc_func 0) 0 "old_data_ptr_ptr"
966               else_builder
967           in
968           let old_data_ptr =
969             L.build_load old_data_ptr_ptr "old_data_ptr" else_builder
970           in
971           (* Casting the "void" pointer back to data pointer *)
972           let old_data_ptr =
973             L.build_bitcast old_data_ptr
974               (L.pointer_type (ltype_of_typ t))
975               "cast_old_data_ptr" else_builder
976           in
977           let old_data = L.build_load old_data_ptr "old_data" else_builder
    in
978           (* Copy the old data into newly allocated space *)
```

```
        let _ = L.build_store old_data data_ptr else_builder in
        (* Casting the data pointer to "void" pointer *)
        let data_ptr_cast =
          L.build_bitcast data_ptr (L.pointer_type i8_t) "data_ptr_cast"
            else_builder
        in
        (* Storing pointer to copied data in the new struct *)
        let _ =
          L.build_store data_ptr_cast
            (L.build_struct_gep new_struct_ptr 0 "store_new_data"
  else_builder)
            else_builder
        in
        (* Storing the pointer to new struct in the previous struct's
           next struct pointer field *)
        let _ =
          L.build_store new_struct_ptr (L.param lc_func 2) else_builder
        in
        let ptr_ptr =
          L.build_struct_gep new_struct_ptr 1 "next" else_builder
        in
        (* Getting the pointer to next struct pointer field
           in current struct *)
        let next_ptr =
          L.build_struct_gep (L.param lc_func 0) 1 "next_ptr" else_builder
        in
        let next = L.build_load next_ptr "next" else_builder in
        (* Subtracting 1 from the index *)
        let sub =
          L.build_sub (L.param lc_func 1) (L.const_int i32_t 1) "sub"
            else_builder
        in
        (* Recursively calling the function for rest of the list *)
        let ret =
          L.build_call lc_func [|next; sub; ptr_ptr|] "" else_builder
        in
        let _ = L.build_ret ret else_builder in
        let _ = L.build_cond_br bool_val then_bb else_bb lc_builder in
        lc_func

  and build_string_length_function () =
    match L.lookup_function "string_length" the_module with
    | Some func -> func
    | None ->
        (* Function takes in the string and current length *)
        let sl_func_t = L.function_type i32_t [|string_t; i32_t|] in
        let sl_func =
          L.define_function "string_length" sl_func_t the_module
        in
        let sl_builder = L.builder_at_end context (L.entry_block sl_func)
  in
        (* Checking if the string is just a null character *)
        let bool_val =
          L.build_is_null
            (L.build_load (L.param sl_func 0) "char" sl_builder)
            "ptr_is_null" sl_builder
        in
        let then_bb = L.append_block context "then" sl_func in
        (* Returning the current length after finding a null character *)
```

```
        let _ =
          L.build_ret (L.param sl_func 1) (L.builder_at_end context
    then_bb)
        in
        let else_bb = L.append_block context "else" sl_func in
        let else_builder = L.builder_at_end context else_bb in
        (* Getting pointer to next character in the string *)
        let next =
          L.build_gep (L.param sl_func 0)
            [|L.const_int i32_t 1|]
            "next_ptr" else_builder
        in
        (* Adding 1 to the current length *)
        let add =
          L.build_add (L.param sl_func 1) (L.const_int i32_t 1) "add"
            else_builder
        in
        (* Recursively calling length function with rest of the string*)
        let ret = L.build_call sl_func [|next; add|] "result" else_builder
     in
        let _ = L.build_ret ret else_builder in
        let _ = L.build_cond_br bool_val then_bb else_bb sl_builder in
        sl_func

    (* list_reverse_helper is used by list_reverse *)
    and build_list_reverse_helper_function () =
      match L.lookup_function "list_reverse_helper" the_module with
      | Some func -> func
      | None ->
        let reverse_helper_func_t =
          L.function_type
            (L.pointer_type list_struct_ptr)
            [|L.pointer_type list_struct_ptr; L.pointer_type
    list_struct_ptr|]
        in
        let reverse_helper_func =
          L.define_function "list_reverse_helper" reverse_helper_func_t
            the_module
        in
        let reverse_helper_builder =
          L.builder_at_end context (L.entry_block reverse_helper_func)
        in

        (* get arguments *)
        let prev_node_ptr_ptr = L.param reverse_helper_func 0 in
        let curr_node_ptr_ptr = L.param reverse_helper_func 1 in

        (* get pointer to the previous node *)
        let prev_node_ptr =
          L.build_load prev_node_ptr_ptr "prev_node_ptr"
            reverse_helper_builder
        in

        (* get pointer to the current node *)
        let curr_node_ptr =
          L.build_load curr_node_ptr_ptr "curr_node_ptr"
            reverse_helper_builder
        in

```

```
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          let bool_val =
1112            L.build_is_null next_node_ptr "ptr_is_null"
    reverse_helper_builder
1113          in
1114          let _ =
1115            L.build_store prev_node_ptr next_node_ptr_ptr
    reverse_helper_builder
1116          in
1117          let then_bb = L.append_block context "then" reverse_helper_func in
1118          let _ =
1119            L.build_ret curr_node_ptr_ptr (L.builder_at_end context then_bb)
1120          in
1121          let else_bb = L.append_block context "else" reverse_helper_func in
1122          let else_builder = L.builder_at_end context else_bb in
1123
1124          (* recursively call reverse_helper_func to reverse the rest of the
     list *)
1125          let ret =
1126            L.build_call reverse_helper_func
1127              [|curr_node_ptr_ptr; temp_ptr_ptr|]
1128              "result" else_builder
1129          in
1130          let _ = L.build_ret ret else_builder in
1131          let _ =
1132            L.build_cond_br bool_val then_bb else_bb reverse_helper_builder
1133          in
1134          reverse_helper_func
1135
1136    (* building reverse function *)
1137    and build_list_reverse_function () =
1138      match L.lookup_function "list_reverse" the_module with
1139      | Some func -> func
1140      | None ->
1141          let reverse_func_t =
1142            L.function_type
1143              (L.pointer_type list_struct_ptr)
1144              [|L.pointer_type list_struct_ptr|]
1145          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_typ typ in
1229      (* insert function has to be defined for different types *)
1230      let func_name = "insert_" ^ A.string_of_typ t in
1231      match L.lookup_function func_name the_module with
1232      | Some func -> func
1233      | None ->
1234          let ltype = ltype_of_typ 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
      in
1252
1253          (* malloc space for a new list *)
1254          let new_list_ptr_ptr =
1255            L.build_malloc list_struct_ptr "new_list_ptr_ptr" insert_builder
1256          in
1257          let _ =
1258            L.build_store
1259              (L.const_null list_struct_ptr)
```

```
            new_list_ptr_ptr insert_builder
        in

        (* get the length of the function *)
        let lc_func = build_copy_function typ in

        (* copy the list to a new one so that the old list is not mutated
    *)
        let _ =
          L.build_call lc_func
            [|list_ptr; L.const_int i32_t (-1); new_list_ptr_ptr|]
            "last_node_ptr_ptr" insert_builder
        in
        let new_list_ptr =
          L.build_load new_list_ptr_ptr "new_list_ptr" insert_builder
        in
        let la_func = build_access_function () in
        let temp = L.build_alloca list_struct_type "temp" insert_builder
    in
        let next = L.build_struct_gep temp 1 "next" insert_builder in
        let _ = L.build_store new_list_ptr next insert_builder in

        (* malloc space for the actual data *)
        let dat_struct =
          L.build_malloc list_struct_type "data_node" insert_builder
        in

        (* malloc space for the pointer to the data *)
        let dat_ptr = L.build_malloc ltype "data" insert_builder in
        let _ = L.build_store e' dat_ptr insert_builder in

        (* store the data pointer to the node struct *)
        let dat_ptr_ptr =
          L.build_struct_gep dat_struct 0 "dat" insert_builder
        in
        let type_casted =
          L.build_bitcast dat_ptr (L.pointer_type i8_t) "cast"
    insert_builder
        in
        let _ = L.build_store type_casted dat_ptr_ptr insert_builder in

        (* get the node specified by the index *)
        let item_ptr =
          L.build_call la_func [|temp; i'|] "result" insert_builder
        in
        let cur_next = L.build_struct_gep item_ptr 1 "test" insert_builder
     in

        (* keep a record of what next currently pointer to *)
        let _ =
          L.build_store
            (L.build_load cur_next "temp" insert_builder)
            (L.build_struct_gep dat_struct 1 "dat" insert_builder)
            insert_builder
        in

        (* insertion happends here *)
        let _ = L.build_store dat_struct cur_next insert_builder in

```

```
        (* connect old list to the new one *)
        let _ =
          L.build_store
            (L.build_load next "temp" insert_builder)
            new_list_ptr_ptr insert_builder
        in
        let _ = L.build_ret new_list_ptr_ptr insert_builder in
        insert_func
  (* Building function to get length of list *)
  and build_list_length_function () =
    match L.lookup_function "list_length" the_module with
    | Some func -> func
    | None ->
        (* Functiont takes a list and current length *)
        let ll_func_t = L.function_type i32_t [|list_struct_ptr; i32_t|]
  in
        let ll_func = L.define_function "list_length" ll_func_t the_module
   in
        let ll_builder = L.builder_at_end context (L.entry_block ll_func)
  in
        (* Checking if a list is null *)
        let bool_val =
          L.build_is_null (L.param ll_func 0) "ptr_is_null" ll_builder
        in
        let then_bb = L.append_block context "then" ll_func in
        (* Return the current length if list is null *)
        let _ =
          L.build_ret (L.param ll_func 1) (L.builder_at_end context
  then_bb)
        in
        let else_bb = L.append_block context "else" ll_func in
        let else_builder = L.builder_at_end context else_bb in
        (* Get the next struct in the list *)
        let next_ptr =
          L.build_struct_gep (L.param ll_func 0) 1 "next_ptr" else_builder
        in
        let next = L.build_load next_ptr "next" else_builder in
        (* Add 1 to the current length *)
        let add =
          L.build_add (L.param ll_func 1) (L.const_int i32_t 1) "add"
            else_builder
        in
        (* Recursively call the function with the rest of the list *)
        let ret = L.build_call ll_func [|next; add|] "result" else_builder
   in
        let _ = L.build_ret ret else_builder in
        let _ = L.build_cond_br bool_val then_bb else_bb ll_builder in
        ll_func
  (* Function to access a node at certain index *)
  and build_access_function () =
    match L.lookup_function "list_access" the_module with
    | Some func -> func
    | None ->
        (* Function takes in a list and index *)
        let la_func_t =
          L.function_type list_struct_ptr [|list_struct_ptr; i32_t|]
        in
        let la_func = L.define_function "list_access" la_func_t the_module
   in
```

```
1368          let la_builder = L.builder_at_end context (L.entry_block la_func)
    in
1369          (* Check if we are at the end of the list by comparing with null
    *)
1370          let bool_val =
1371            L.build_icmp L.Icmp.Eq (L.param la_func 1) (L.const_int i32_t 0)
1372              "is_zero" la_builder
1373          in
1374          let then_bb = L.append_block context "then" la_func in
1375          (* Return the current struct if index is 0 *)
1376          let _ =
1377            L.build_ret (L.param la_func 0) (L.builder_at_end context
    then_bb)
1378          in
1379          let else_bb = L.append_block context "else" la_func in
1380          let else_builder = L.builder_at_end context else_bb in
1381          (* Get the next node in the list *)
1382          let next_ptr =
1383            L.build_struct_gep (L.param la_func 0) 1 "next_ptr" else_builder
1384          in
1385          let next = L.build_load next_ptr "next" else_builder in
1386          (* Subtract 1 from the index *)
1387          let sub =
1388            L.build_sub (L.param la_func 1) (L.const_int i32_t 1) "sub"
1389              else_builder
1390          in
1391          (* Recursively call with the rest of the list *)
1392          let ret = L.build_call la_func [|next; sub|] "result" else_builder
     in
1393          let _ = L.build_ret ret else_builder in
1394          let _ = L.build_cond_br bool_val then_bb else_bb la_builder in
1395          la_func
1396
1397    (* Function to build a list *)
1398    and build_list list_typ lis (scope : var_table ref) builder =
1399      let typ = get_list_inner_typ list_typ in
1400      let ltyp = ltype_of_typ typ in
1401      (* Function to build an individual struct *)
1402      let build_link prev data =
1403        (* Allocate a single struct *)
1404        let entry_ptr = L.build_malloc list_struct_type "list_item" builder
    in
1405        (* Initialize the allocated struct with null *)
1406        let _ =
1407          L.build_store (L.const_null list_struct_type) entry_ptr builder
1408        in
1409        (* Allocate space for the data *)
1410        let data_ptr = L.build_malloc ltyp "copied" builder in
1411        (* Store data in the allocated space *)
1412        let _ = L.build_store (expr scope builder data) data_ptr builder in
1413        (* Cast data pointer to "void" pointer *)
1414        let typcast_ptr =
1415          L.build_bitcast data_ptr (L.pointer_type i8_t) "cast_ptr" builder
1416        in
1417        (* Get the pointer to data pointer in the struct *)
1418        let data_ptr_container =
1419          L.build_struct_gep entry_ptr 0 "data_ptr_container" builder
1420        in
1421        (* 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}
     in
1438          List.fold_left (fun b s -> build_stmt new_scope b s loop) builder
    sl
1439      | SExpr e ->
1440          (* Build instruction for expression *)
1441          let _ = expr sc builder e in
1442          builder
1443      | SReturn e ->
1444          (* Generate a return statement *)
1445          let _ =
1446            match fdecl.styp with
1447            | A.Void -> L.build_ret_void builder
1448            | _ -> L.build_ret (expr sc builder e) builder
1449          in
1450          builder
1451      | SIf (predicate, then_stmt, else_stmt) ->
1452          (* Same implementation as microc *)
1453          let bool_val = expr sc builder predicate in
1454          let merge_bb = L.append_block context "merge" the_function in
1455          let branch_instr = L.build_br merge_bb in
1456          let then_bb = L.append_block context "then" the_function in
1457          let then_builder =
1458            build_stmt sc (L.builder_at_end context then_bb) then_stmt loop
1459          in
1460          let () = add_terminal then_builder branch_instr in
1461          let else_bb = L.append_block context "else" the_function in
1462          let else_builder =
1463            build_stmt sc (L.builder_at_end context else_bb) else_stmt loop
1464          in
1465          let () = add_terminal else_builder branch_instr in
1466          let _ = L.build_cond_br bool_val then_bb else_bb builder in
1467          L.builder_at_end context merge_bb
1468      | SFor (s, (t, e), sl) ->
1469          (* An equivalent for loop code written using while loop and
1470             and other statemtns *)
1471          let equivalent =
1472            match t with
1473            | A.List s_ty ->
1474                (* Get length of the list *)
1475                let len_call =
1476                  ( A.Int
1477                  , SCall
1478                      ((A.Func ([A.List A.Int], A.Int), SId "length"), [(t,
```

```
    e)])
                  )
                in
                (* Create a new variable for index *)
                let index_expr = (A.Int, SId "for_index") in
                let while_cond =
                  (A.Bool, SBinop (index_expr, A.Less, len_call))
                in
                (* Create a new variable to store the current list entry *)
                let element = (s_ty, SId s) in
                SBlock
                  (* Declare the index variable *)
                  [ SDeclaration (A.Int, "for_index", (A.Int, SIntLit 0))
                  ; SDeclaration (s_ty, s, (A.Void, SNoexpr))
                  ; SWhile
                      ( while_cond
                      , SBlock
                          [ SExpr
                              ( s_ty
                              , SAssign
                                  ( element
                                  , ( s_ty
                                    , SSliceExpr ((t, e), SIndex index_expr)
     )
                                  ) )
                          ; SExpr
                              ( A.Int
                              , SAssign
                                  ( index_expr
                                  , ( A.Int
                                    , SBinop
                                        (index_expr, A.Add, (A.Int, SIntLit
    1))
                                  ) ) )
                          ; sl ] ) ]
            (* Using a similar  strategy to for loop for lists*)
            | A.String ->
                let len_call =
                  ( A.Int
                  , SCall ((A.Func ([A.String], A.Int), SId "length"), [(t,
    e)])
                  )
                in
                let index_expr = (A.Int, SId "for_index") in
                let while_cond =
                  (A.Bool, SBinop (index_expr, A.Less, len_call))
                in
                let element = (A.Char, SId s) in
                SBlock
                  [ SDeclaration (A.Int, "for_index", (A.Int, SIntLit 0))
                  ; SDeclaration (A.Char, s, (A.Void, SNoexpr))
                  ; SWhile
                      ( while_cond
                      , SBlock
                          [ SExpr
                              ( A.Char
                              , SAssign
                                  ( element
                                  , ( A.Char
```

```
                                    , SSliceExpr ((t, e), SIndex index_expr)
    )
                                ) )
                          ; SExpr
                              ( A.Int
                              , SAssign
                                  ( index_expr
                                  , ( A.Int
                                    , SBinop
                                        (index_expr, A.Add, (A.Int, SIntLit
    1))
                                    ) ) )
                          ; sl ] ) ]
          | _ -> raise (Failure "internal error")
        in
        build_stmt sc builder equivalent loop
    | SDeclaration (t, n, e) ->
        let e =
          match e with
          (* Handling case for declaration without initial value *)
          | A.Void, SNoexpr -> (
            match t with
            | A.List _ ->
                let ptr_ptr =
                  L.build_malloc list_struct_ptr "ptr_ptr" builder
                in
                let _ =
                  L.build_store (L.const_null list_struct_ptr) ptr_ptr
    builder
                in
                ptr_ptr
            | A.String -> L.build_global_stringptr "" "string" builder
            | _ -> L.const_null (ltype_of_typ t) )
          | asd -> expr sc builder asd
        in
        let _ =
          match fdecl.sfname with
          (* When Variable is declared on the top level *)
          | "main" ->
              (* Allocate as global for variables on top level *)
              let global =
                L.define_global n (L.const_null (ltype_of_typ t))
    the_module
              in
              let _ = L.build_store e global builder in
              add_variable_to_scope sc n global
          (* When the variable is declared inside a function *)
          | _ ->
              (* Allocate space for variable on the top of function*)
              let init_pos = L.instr_begin (L.entry_block the_function) in
              let new_builder = L.builder_at context init_pos in
              let local = L.build_alloca (ltype_of_typ t) n new_builder in
              let _ = L.build_store e local builder in
              add_variable_to_scope sc n local
        in
        builder
    | SWhile (predicate, body) ->
        (* Similar strategy as microc *)
        let pred_bb = L.append_block context "while" the_function in
```

```
          let _ = L.build_br pred_bb builder in
          let merge_bb = L.append_block context "merge" the_function in
          let body_bb = L.append_block context "while_body" the_function in
          (* Pass the builder for break and continue statements to refer *)
          let while_builder =
            build_stmt sc
              (L.builder_at_end context body_bb)
              body
              ((pred_bb, merge_bb) :: loop)
          in
          let () = add_terminal while_builder (L.build_br pred_bb) in
          let pred_builder = L.builder_at_end context pred_bb in
          let bool_val = expr sc pred_builder predicate in
          let _ = L.build_cond_br bool_val body_bb merge_bb pred_builder in
          L.builder_at_end context merge_bb
      (* Add instruction to builders passed from a loop *)
      | SBreak ->
          let () = add_terminal builder (L.build_br (snd (List.hd loop))) in
          builder
      | SContinue ->
          let () = add_terminal builder (L.build_br (fst (List.hd loop))) in
          builder
    in
    let builder =
      List.fold_left (fun b s -> build_stmt scope b s []) builder fdecl.
    sbody
    in
    (* Function to add terminals to a basic block *)
    add_terminal builder
      ( match fdecl.styp with
      | A.Void -> L.build_ret_void
      | A.Float -> L.build_ret (L.const_float float_t 0.0)
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0) )
  in
  (* Call buil_function_body on all functions *)
  let functions' =
    try List.iter (build_function_body globals) functions
    with e -> E.handle_error e
  in
  functions' ; the_module
```

11.2.11 src/team.ml

```
(* team.ml: scan & parse & sematically analyze the input,
 * check the resulting AST and generate SAST from it,
 * generate LLVM IR, and dump the module
   Authors: Naoki O., Yingjie L., Lulu Z., Saurav G. *)
type action = Ast | Sast | Resolve | LLVM_IR

let make_err err = raise (Failure err)

(* Print the scanner error with the line and character number information *)
let scan_error lexbuf ch =
  let start_p = Lexing.lexeme_start_p lexbuf in
  let end_p = Lexing.lexeme_end_p lexbuf in
  let line = string_of_int start_p.Lexing.pos_lnum in
  let ch_start =
    string_of_int (start_p.Lexing.pos_cnum - start_p.Lexing.pos_bol)
  in
```

```
  let ch_end = string_of_int (end_p.Lexing.pos_cnum - end_p.Lexing.pos_bol)
    in
  make_err
    ( "Illegal character at line " ^ line ^ ", characters " ^ ch_start ^ "-"
    ^ ch_end ^ ": " ^ "\"" ^ ch ^ "\"" )

(* Print the parser error with the line and character number information *)
let parse_error lexbuf =
  let start_p = Lexing.lexeme_start_p lexbuf in
  let end_p = Lexing.lexeme_end_p lexbuf in
  let line = string_of_int start_p.Lexing.pos_lnum in
  let ch_start =
    string_of_int (start_p.Lexing.pos_cnum - start_p.Lexing.pos_bol)
  in
  let ch_end = string_of_int (end_p.Lexing.pos_cnum - end_p.Lexing.pos_bol)
    in
  let lexeme = Lexing.lexeme lexbuf in
  make_err
    ( "Syntax error at line " ^ line ^ ", characters " ^ ch_start ^ "-" ^
    ch_end
    ^ ": " ^ lexeme )

let parse lexbuf =
  try Parser.program Scanner.token lexbuf with
  | Scanner.Scan_error ch -> scan_error lexbuf ch
  | Parsing.Parse_error -> parse_error lexbuf

let () =
  let action = ref LLVM_IR in
  let set_action a () = action := a in
  let set_channel channel filename =
    let file =
      try open_in filename
      with Sys_error s ->
        let _ = print_endline s in
        exit 1
    in
    channel := file
  in
  let speclist =
    [ ("-a", Arg.Unit (set_action Ast), "Print the AST")
    ; ("-s", Arg.Unit (set_action Resolve), "Print the Resolved SAST")
    ; ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR") ]
  in
  let usage_msg = "usage: ./team.native [-a|-s|-l] [file.tm]" in
  (* get buffers/channel for list and string standard library *)
  let string_channel = ref stdin in
  let list_channel = ref stdin in
  let channel = ref stdin in
  set_channel string_channel "standard_library/string.tm" ;
  set_channel list_channel "standard_library/list.tm" ;
  (* parse program and standard library *)
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg
    ;
  let lexbuf = Lexing.from_channel !channel
  and string_lexbuf = Lexing.from_channel !string_channel
  and list_lexbuf = Lexing.from_channel !list_channel in
  let ast = parse lexbuf in
  let string_ast = parse string_lexbuf in
```

```
  let list_ast = parse list_lexbuf in
  (* prepend standard library to program *)
  let ast =
    ( fst string_ast @ fst list_ast @ fst ast
    , snd string_ast @ snd list_ast @ snd ast )
  in
  let sast = Semant.check ast in
  let resolved_sast = Resolve.resolve sast in
  match !action with
  | Ast -> print_string (Ast.string_of_program ast)
  | Sast -> print_string (Sast.string_of_sprogram sast)
  | Resolve -> print_string (Sast.string_of_sprogram resolved_sast)
  | LLVM_IR ->
      let m = Codegen.translate resolved_sast in
      Llvm_analysis.assert_valid_module m ;
      print_string (Llvm.string_of_llmodule m)
```

11.2.12 scripts/compile.sh

```
# Authors: Naoki O., Yingjie L., Lulu Z., Saurav G.
#!/bin/bash
export PATH=$PATH:/usr/local/opt/llvm/bin
# needed for compilation
CC="gcc"
LIBS="-g -Wall -lpcreposix -lpcre2-8"
# file is command line argument, filename is file without extension
file=$1
filename=$(echo "$file" | cut -f 1 -d ".")

func=$2

create() {
    # generate code, need to pass in file since it could be modified
    ./team.native -l "$1" > "$filename".ll
    llc "$2".ll
    eval "$CC $LIBS -o $filename.exe $filename.s regex.o fileio.o"
    # echo "$filename.exe created"
}

# SCRIPT BEGINS HERE
if [[ $# -eq 0 ]]
then
    echo "usage: ./compile.sh <file.tm> [func]"
    exit 0
fi

if [ ! -e $file ]
then
    echo "file $file not found"
    exit 0
fi

if [[ "$func" == "clean" ]]
then
    # echo "cleaning: $filename.s $filename.ll $filename.exe"
    rm "$filename".s "$filename".ll "$filename".exe
    rm -rf "$filename".exe.dSYM
    exit 0
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>
```

```c
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <stdlib.h>
#define PCRE2_CODE_UNIT_WIDTH 8
#include <pcre2.h>

/* struct representation of a list node */
typedef struct list_item_
{
    void *dat;
    struct list_item_ *next;
} list_item;

list_item **append(char *s, list_item **head_ref)
{
    list_item *node = malloc(sizeof(list_item));
    char **s_ptr = malloc(sizeof(char *));
    *s_ptr = s;
    node->dat = s_ptr;
    node->next = NULL;
    list_item *last = *head_ref;

    if (*head_ref == NULL)
    {
        *head_ref = node;
        return head_ref;
    }
    while (last->next != NULL)
    {
        last = last->next;
    }
    last->next = node;

    return head_ref;
}

/* prints the content in the list, for debugging purposes */
// void print_list(list_item **head_ref)
// {
//     list_item *temp = *head_ref;

//     if ((*head_ref)->dat == NULL && (*head_ref)->next == NULL)
//     {
//         return;
//     }

//     while (temp != NULL)
//     {
//         printf("%s\n", (*(char **)temp->dat));
//         if (temp->next == NULL)
//         {
//             break;
//         }

//         temp = temp->next;
//     }
// }

```

```c
int length(list_item **head_ref)
{
    list_item *temp = *head_ref;
    int count = 0;
    if ((*head_ref)->dat == NULL && (*head_ref)->next == NULL)
    {
        return count;
    }

    while (temp != NULL)
    {
        count += 1;
        if (temp->next == NULL)
        {
            break;
        }

        temp = temp->next;
    }
    return count;
}
/*
 * Match string against the extended regular expression in
 * pattern, treating errors as no match.
 *
 * Return 1 for match, 0 for no match.
 */

int match(const char *target, char *pattern)
{
    int status;
    regex_t re;

    if (regcomp(&re, pattern, REG_EXTENDED | REG_NOSUB) != 0)
    {
        return (0); /* Report error. */
    }
    /* re = precompiled pattern
     * string = pattern to be match in regex
     * NULL = information regarding location of the matches
     * 0 = to specify the change in matching behaviour */

    status = regexec(&re, target, (size_t)0, NULL, 0);
    regfree(&re);
    if (status != 0)
    {
        return (0); /* Report error. */
    }
    return (1);
}

void substr(char *str, char *sub, int start, int len)
{
    memcpy(sub, &str[start], len);
    sub[len] = '\0';
}

/* Takes a target string and a regex and returns the
 * first substring of the string that matches
```

```
 * the regular expression.
 *
 * If no match is found, return the empty string.
 */

char *find(char *target, char *regex)
{
    pcre2_code *re;
    PCRE2_SPTR subject = (const unsigned char *)target;
    PCRE2_SPTR pattern = (const unsigned char *)regex;
    int errornumber;
    int rc;

    PCRE2_SIZE erroroffset;
    PCRE2_SIZE *ovector;
    PCRE2_SIZE subject_length;

    pcre2_match_data *match_data;

    subject_length = (PCRE2_SIZE)strlen((char *)subject);

    /* compile the regular expression pattern, and handle
       any errors that are detected. */

    re = pcre2_compile(
        pattern,               /* the pattern */
        PCRE2_ZERO_TERMINATED, /* indicates pattern is zero-terminated */
        0,                     /* default options */
        &errornumber,          /* for error number */
        &erroroffset,          /* for error offset */
        NULL);                 /* use default compile context */

    /* Compilation failed: print the error message and exit. */

    if (re == NULL)
    {
        PCRE2_UCHAR buffer[256];
        pcre2_get_error_message(errornumber, buffer, sizeof(buffer));
        printf("PCRE2 compilation failed at offset %d: %s\n", (int)
    erroroffset,
                buffer);
        return "";
    }

    match_data = pcre2_match_data_create_from_pattern(re, NULL);

    /* Now run the match. */

    rc = pcre2_match(
        re,             /* the compiled pattern */
        subject,        /* the subject string */
        subject_length, /* the length of the subject */
        0,              /* start at offset 0 in the subject */
        0,              /* default options */
        match_data,     /* block for storing the result */
        NULL);          /* use default match context */

    /* Matching failed: handle error cases */

```

```
    if (rc < 0)
    {
        switch (rc)
        {
        case PCRE2_ERROR_NOMATCH:
            /* No Match */
            break;
            /* Handle other special cases if you like */
        default:
            /* Some other match error */
            break;
        }
        pcre2_match_data_free(match_data); /* Release memory used for the match */
        pcre2_code_free(re);               /*   data and the compiled pattern. */
        return "";
    }

    /* Match succeeded. Get a pointer to the output vector,
     * where string offsets are stored. */

    ovector = pcre2_get_ovector_pointer(match_data);
    // printf("Match succeeded at offset %d\n", (int)ovector[0]);

    /* The output vector wasn't big enough.
     * This should not happen, because we used
     * pcre2_match_data_create_from_pattern() above. */

    if (rc == 0)
        printf("ovector was not big enough for all the captured substrings\n");

    /* Show substrings stored in the output vector by number.  */

    PCRE2_SPTR substring_start = subject + ovector[0];
    PCRE2_SIZE substring_length = ovector[1] - ovector[0];
    char *sub = (char *)malloc(sizeof(char) * substring_length);
    substr((char *)substring_start, sub, 0, (int)substring_length);
    return sub;
}

list_item **find_all(char *target, char *regex)
{
    pcre2_code *re;
    PCRE2_SPTR name_table;
    PCRE2_SPTR subject = (const unsigned char *)target;
    PCRE2_SPTR pattern = (const unsigned char *)regex;
    int crlf_is_newline;
    int errornumber;
    int i;
    int rc;
    int utf8;

    uint32_t option_bits;
    uint32_t namecount;
    uint32_t name_entry_size;
    uint32_t newline;

```

```
    PCRE2_SIZE erroroffset;
    PCRE2_SIZE *ovector;
    PCRE2_SIZE subject_length;

    pcre2_match_data *match_data;

    subject_length = (PCRE2_SIZE)strlen((char *)subject);
    list_item **ret = malloc(sizeof(list_item *));
    *ret = NULL;
    /* compile the regular expression pattern, and handle
       any errors that are detected. */

    re = pcre2_compile(
        pattern,               /* the pattern */
        PCRE2_ZERO_TERMINATED, /* indicates pattern is zero-terminated */
        0,                     /* default options */
        &errornumber,          /* for error number */
        &erroroffset,          /* for error offset */
        NULL);                 /* use default compile context */

    /* Compilation failed: print the error message and exit. */

    if (re == NULL)
    {
        PCRE2_UCHAR buffer[256];
        pcre2_get_error_message(errornumber, buffer, sizeof(buffer));
        printf("PCRE2 compilation failed at offset %d: %s\n", (int)
    erroroffset,
                buffer);
        return ret;
    }

    match_data = pcre2_match_data_create_from_pattern(re, NULL);

    /* Now run the match. */

    rc = pcre2_match(
        re,             /* the compiled pattern */
        subject,        /* the subject string */
        subject_length, /* the length of the subject */
        0,              /* start at offset 0 in the subject */
        0,              /* default options */
        match_data,     /* block for storing the result */
        NULL);          /* use default match context */

    /* Matching failed: handle error cases */

    if (rc < 0)
    {
        list_item *list = malloc(sizeof(list_item));
        switch (rc)
        {
        case PCRE2_ERROR_NOMATCH:
            /* no match found, return empty list */
            list->dat = NULL;
            list->next = NULL;
            *ret = list;
            break;
        /*
```

```
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_ovector_pointer(match_data);
308     // printf("Match succeeded at offset %d\n", (int)ovector[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("ovector 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,
    the name,
353   and the substring itself. In the 8-bit library the number is held in two
354   bytes, most significant first. */
355 
356         tabptr = name_table;
357         for (i = 0; i < namecount; i++)
358         {
359             int n = (tabptr[0] << 8) | tabptr[1];
360             printf("(%d) %*s: %.*s\n", n, name_entry_size - 3, tabptr + 2,
361                    (int)(ovector[2 * n + 1] - ovector[2 * n]), subject +
    ovector[2 * n]);
362             tabptr += name_entry_size;
363         }
364     }
365 
366     /* Before running the loop, check for UTF-8 and whether CRLF is a valid
    newline
367 sequence. First, find the options with which the regex was compiled and
    extract
368 the UTF state. */
369 
370     (void)pcre2_pattern_info(re, PCRE2_INFO_ALLOPTIONS, &option_bits);
371     utf8 = (option_bits & PCRE2_UTF) != 0;
372 
373     /* Now find the newline convention and see whether CRLF is a valid
    newline
374 sequence. */
375 
376     (void)pcre2_pattern_info(re, PCRE2_INFO_NEWLINE, &newline);
377     crlf_is_newline = newline == PCRE2_NEWLINE_ANY ||
378                       newline == PCRE2_NEWLINE_CRLF ||
379                       newline == PCRE2_NEWLINE_ANYCRLF;
380 
381     /* Loop for second and subsequent matches */
382 
383     for (;;)
384     {
385         uint32_t options = 0;                 /* Normally no options */
386         PCRE2_SIZE start_offset = ovector[1]; /* Start at end of previous
    match */
387 
388         /* If the previous match was for an empty string, we are finished if
     we are
389   at the end of the subject. Otherwise, arrange to run another match at the
390   same point to see if a non-empty match can be found. */
391 
392         if (ovector[0] == ovector[1])
393         {
394             if (ovector[0] == subject_length)
395                 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                 start_offset = startchar + 1; /* Advance by one character.
    */
406                 if (utf8)                     /* If UTF-8, it may be more
    */
407                 {                             /*   than one code unit.
    */
408                     for (; start_offset < subject_length; start_offset++)
409                         if ((subject[start_offset] & 0xc0) != 0x80)
410                             break;
411                 }
412             }
413         }
414
415         /* Run the next matching operation */
416
417         rc = pcre2_match(
418             re,             /* the compiled pattern */
419             subject,        /* the subject string */
420             subject_length, /* the length of the subject */
421             start_offset,   /* starting offset in the subject */
422             options,        /* options */
423             match_data,     /* block for storing the result */
424             NULL);          /* use default match context */
425
426         if (rc == PCRE2_ERROR_NOMATCH)
427         {
428             if (options == 0)
429             {
430                 break;
431             }
432             /* All matches found */
433             ovector[1] = start_offset + 1;           /* Advance one code
    unit */
434             if (crlf_is_newline &&                   /* If CRLF is a newline
     & */
435                 start_offset < subject_length - 1 && /* we are at CRLF, */
436                 subject[start_offset] == '\r' &&
437                 subject[start_offset + 1] == '\n')
438                 ovector[1] += 1;                     /* Advance by one more.
    */
439             else if (utf8)                           /* Otherwise, ensure we
    */
440             {                                        /* advance a whole UTF-8
     */
441                 while (ovector[1] < subject_length) /* character. */
442                 {
443                     if ((subject[ovector[1]] & 0xc0) != 0x80)
444                         break;
445                     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         /* The match succeeded, but the output vector wasn't big enough.
    This
466   should not happen. */
467 
468         if (rc == 0)
469             printf("ovector was not big enough for all the captured
    substrings\n");
470 
471         /* We must guard against patterns such as /(?=.\K)/ that use \K in
    an
472   assertion to set the start of a match later than its end. In this
473   demonstration program, we just detect this case and give up. */
474 
475         if (ovector[0] > ovector[1])
476         {
477             printf("\\K was used in an assertion to set the match start
    after its end.\n"
478                    "From end to start the match was: %.*s\n",
479                    (int)(ovector[0] - ovector[1]),
480                    (char *)(subject + ovector[1]));
481             printf("Run abandoned\n");
482             pcre2_match_data_free(match_data);
483             pcre2_code_free(re);
484             return ret;
485         }
486 
487         /* As before, show substrings stored in the output vector by number,
     and then
488   also any named substrings. */
489 
490         PCRE2_SPTR substring_start = subject + ovector[0];
491         size_t substring_length = ovector[1] - ovector[0];
492         char *sub = (char *)malloc(sizeof(char) * (substring_length + 1));
493         substr((char *)substring_start, sub, 0, (int)substring_length);
494         ret = append(sub, ret);
495 
496         if (namecount == 0)
497         {
498         }
```

```
        else
        {
            PCRE2_SPTR tabptr = name_table;
            for (i = 0; i < namecount; i++)
            {
                int n = (tabptr[0] << 8) | tabptr[1];
                printf("(%d) %*s: %.*s\n", n, name_entry_size - 3, tabptr +
    2,
                       (int)(ovector[2 * n + 1] - ovector[2 * n]), subject +
     ovector[2 * n]);
                tabptr += name_entry_size;
            }
        }
    } /* End of loop to find second and subsequent matches */

    pcre2_match_data_free(match_data);
    pcre2_code_free(re);
    return ret;
}

char *str_replace(char *orig, char *rep, char *with)
{
    char *result;  // the return string
    char *ins;     // the next insert point
    char *tmp;     // varies
    int len_rep;   // length of rep (the string to remove)
    int len_with;  // length of with (the string to replace rep with)
    int len_front; // distance between rep and end of last rep
    int count;     // number of replacements

    // sanity checks and initialization
    if (!orig || !rep)
        return NULL;
    len_rep = strlen(rep);
    if (len_rep == 0)
        return NULL; // empty rep causes infinite loop during count
    if (!with)
        with = "";
    len_with = strlen(with);

    // count the number of replacements needed
    ins = orig;
    for (count = 0; (tmp = strstr(ins, rep)); ++count)
    {
        ins = tmp + len_rep;
    }

    tmp = result = malloc(strlen(orig) + (len_with - len_rep) * count + 1);

    if (!result)
        return orig;

    // only replace the first one
    ins = strstr(orig, rep);
    len_front = ins - orig;
    tmp = strncpy(tmp, orig, len_front) + len_front;
    tmp = strcpy(tmp, with) + len_with;
    orig += len_front + len_rep; // move to next "end of rep"

```

```
    strcpy(tmp, orig);
    return result;
}
// You must free the result if result is non-NULL.
char *str_replace_all(char *orig, char *rep, char *with)
{
    char *result;  // the return string
    char *ins;     // the next insert point
    char *tmp;     // varies
    int len_rep;   // length of rep (the string to remove)
    int len_with;  // length of with (the string to replace rep with)
    int len_front; // distance between rep and end of last rep
    int count;     // number of replacements

    // sanity checks and initialization
    if (!orig || !rep)
        return NULL;
    len_rep = strlen(rep);
    if (len_rep == 0)
        return NULL; // empty rep causes infinite loop during count
    if (!with)
        with = "";
    len_with = strlen(with);

    // count the number of replacements needed
    ins = orig;
    for (count = 0; (tmp = strstr(ins, rep)); ++count)
    {
        ins = tmp + len_rep;
    }

    tmp = result = malloc(strlen(orig) + (len_with - len_rep) * count + 1);

    if (!result)
        return orig;

    // first time through the loop, all the variable are set correctly
    // from here on,
    //    tmp points to the end of the result string
    //    ins points to the next occurrence of rep in orig
    //    orig points to the remainder of orig after "end of rep"
    while (count--)
    {
        ins = strstr(orig, rep);
        len_front = ins - orig;
        tmp = strncpy(tmp, orig, len_front) + len_front;
        tmp = strcpy(tmp, with) + len_with;
        orig += len_front + len_rep; // move to next "end of rep"
    }
    strcpy(tmp, orig);
    return result;
}

char *replace(char *target, char *regex, char *replc, int count)
{
    char *result = malloc(sizeof(char) * strlen(target));
    strcpy(result, target);
    while (count--)
    {
```

```
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);
```

```
    s = find("There goes the bat", "[b-chm-pP]at|ot");
    assert(strcmp(s, "bat") == 0);
    s = find("a", "\\w");
    assert(strcmp(s, "a") == 0);
    s = find("2+2", "\\d+[\\+-x\\*]\\d+");
    assert(strcmp(s, "2+2") == 0);
    s = find("x+y", "\\w+[\\+-x\\*]\\w+");
    assert(strcmp(s, "x+y") == 0);
    s = find("?+?", "\\W+[\\+-x\\*]\\W+");
    assert(strcmp(s, "?+?") == 0);
    // Begins with dog
    s = find("dog collar", "^dog");
    assert(strcmp(s, "dog") == 0);
    // No match
    s = find("This is a dog", "^dog");
    assert(strcmp(s, "") == 0);
    // Ends with dog
    s = find("hot dog", "dog$");
    assert(strcmp(s, "dog") == 0);
    // Just dog
    s = find("dog", "^dog$");
    assert(strcmp(s, "dog") == 0);
    // Time in 24 hr format
    s = find("12:03", "^([01]?[0-9]|2[0-3]):[0-5][0-9]$");
    assert(strcmp(s, "12:03") == 0);
    // No match
    s = find("z", "z{3,6}");
    assert(strcmp(s, "") == 0);
}
void test_match()
{
    int i = match("ggle google", "go*gle");
    assert(i == 1);
    i = match("goooogle", "go*gle");
    assert(i == 1);
    i = match("grey", "gr(a|e)y");
    assert(i == 1);
    i = match("There goes the bat", "[b-chm-pP]at|ot");
    assert(i == 1);
    i = match("a", "\\w");
    assert(i == 1);
    i = match("2+2", "\\d+[\\+-x\\*]\\d+");
    assert(i == 1);
    i = match("x+y", "\\w+[\\+-x\\*]\\w+");
    assert(i == 1);
    i = match("?+?", "\\W+[\\+-x\\*]\\W+");
    assert(i == 1);
    // Begins with dog
    i = match("dog collar", "^dog");
    assert(i == 1);
    // No match
    i = match("This is a dog", "^dog");
    assert(i == 0);
    // Ends with dog
    i = match("hot dog", "dog$");
    assert(i == 1);
    // Just dog
    i = match("dog", "^dog$");
    assert(i == 1);
```

```
    // Time in 24 hr format
    i = match("12:03", "^([01]?[0-9]|2[0-3]):[0-5][0-9]$");
    assert(i == 1);
    // No match
    i = match("z", "z{3,6}");
    assert(i == 0);
}
void test_replace()
{
    char *s = replace("ggle google", "go*gle", "replaced", 1);
    assert(strcmp(s, "replaced google") == 0);
    s = replace("ggle google", "go*gle", "replaced", 5);
    assert(strcmp(s, "replaced replaced") == 0);
    s = replace("ggle google", "go*gle", "replaced", 0);
    assert(strcmp(s, "ggle google") == 0);
    s = replace("google google google", "goo", "foo", 3);
    assert(strcmp(s, "foogle foogle foogle") == 0);
}
int main()
{
    test_replace_all();
    test_find();
    test_match();
    test_replace();
    test_find_all();
    return 0;
}
#endif
```

11.2.15 README & Makefile

```
# TEAM

Note: This file is best viewed in a Markdown reader.

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.

## Compilation

To compile TEAM, do

      make

## Environment Setup

Team uses PCRE2 (Perl Compatible Regular Expressions) library to support
    regular expressions. To setup dependencies, do:

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

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:

      cd pcre2-10.36 && make clean
```

```
      cd .. && make clean && make

For more information on the PCRE2 API, please visit:

https://www.pcre.org/current/doc/html/pcre2api.html

## Testing

To run all tests, do

      python scripts/runtests.py -m all

To run tests in a specific directory, do

      python scripts/runtests.py -m <mode>

| Mode    | Description                                                                                  |
| ------- | --------------------------------------------------------------------------------------------- |
| ast     | run the tests in `ast_tests/` and validate the pretty printed ast against a gold standard. Tests that pass the validation are marked with OK! and those that don't are marked with FAILED!                        |
| sast    | run the tests in `sast_tests/` and validates the pretty printed sast against a gold standard. Tests that pass the validation are marked with OK! and those that don't are marked with FAILED!                 |
| codegen | run the tests in `codegen_tests/`, compile the resulting LLVM code, execute the resulting file, and validate the output against a gold standard. If the validation fails, a diff of the two files will be printed to standard output. |

Files used for testing are located in `<mode>_tests/`.
The generated outputs are located in `<mode>_log/` and
the expected outputs (gold standard) are located in `<mode>_ref/`.

The default mode is ast if none was provided.

To execute a single TEAM file (file.tm), do

      python scripts/runtests.py -t file.tm -r file.log -m <mode>

- -t specifies the file to be executed

- -r specifies the file that is the gold standard

- -m when running a single test, the mode can not be `all`.

## Extended Testsuite

To run the extended testsuite, do

      python scripts/runtests.py -m extended

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 mistmatch 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  -lpcreposix -lpcre2-8 -o regex -DBUILD_TEST ./c_library/regex.c
15
16 .PHONY : clean
```

```
clean :
  ocamlbuild -clean
  rm *.o
  rm regex
  rm fileio
  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.

```
; ModuleID = 'TEAM'
source_filename = "TEAM"

%list_item = type <{ i8*, %list_item* }>

@ASCII = global %list_item** null
@string = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@string.1 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
@string.2 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
@string.3 = private unnamed_addr constant [6 x i8] c"wolrd\00", align 1
@string.4 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
@string.5 = private unnamed_addr constant [5 x i8] c"what\00", align 1
@string.6 = private unnamed_addr constant [6 x i8] c"hello\00", align 1
@string.7 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@string.8 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@string.9 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1
@string.10 = private unnamed_addr constant [1 x i8] zeroinitializer, align 1

declare i32 @printf(i8*, ...)

declare double @pow(double, double)

declare i8* @fopen(i8*, i8*)

declare i32 @close(i8*)

declare i8* @readline(i8*)

declare i8* @write(i8*, i8*)

declare i1 @match(i8*, i8*)

declare i8* @find(i8*, i8*)

declare i8* @replace(i8*, i8*, i8*, i32)

declare i8* @replace_all(i8*, i8*, i8*)

declare %list_item** @find_all(i8*, i8*)

define i32 @main() {
entry:
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
```

```
%malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item = bitcast i8* %malloccall1 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item, align 1
%copied = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
  null, i32 1) to i32))
store i8 90, i8* %copied, align 1
%data_ptr_container = getelementptr inbounds %list_item, %list_item* %
  list_item, i32 0, i32 0
store i8* %copied, i8** %data_ptr_container, align 8
%next = getelementptr inbounds %list_item, %list_item* %list_item, i32 0,
  i32 1
store %list_item* null, %list_item** %next, align 8
%malloccall3 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item4 = bitcast i8* %malloccall3 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item4, align 1
%copied6 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
  null, i32 1) to i32))
store i8 89, i8* %copied6, align 1
%data_ptr_container7 = getelementptr inbounds %list_item, %list_item* %
  list_item4, i32 0, i32 0
store i8* %copied6, i8** %data_ptr_container7, align 8
%next8 = getelementptr inbounds %list_item, %list_item* %list_item4, i32
  0, i32 1
store %list_item* %list_item, %list_item** %next8, align 8
%malloccall9 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item10 = bitcast i8* %malloccall9 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item10, align 1
%copied12 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
   null, i32 1) to i32))
store i8 88, i8* %copied12, align 1
%data_ptr_container13 = getelementptr inbounds %list_item, %list_item* %
  list_item10, i32 0, i32 0
store i8* %copied12, i8** %data_ptr_container13, align 8
%next14 = getelementptr inbounds %list_item, %list_item* %list_item10, i32
   0, i32 1
store %list_item* %list_item4, %list_item** %next14, align 8
%malloccall15 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item16 = bitcast i8* %malloccall15 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item16, align 1
%copied18 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
   null, i32 1) to i32))
store i8 87, i8* %copied18, align 1
%data_ptr_container19 = getelementptr inbounds %list_item, %list_item* %
  list_item16, i32 0, i32 0
store i8* %copied18, i8** %data_ptr_container19, align 8
%next20 = getelementptr inbounds %list_item, %list_item* %list_item16, i32
   0, i32 1
store %list_item* %list_item10, %list_item** %next20, align 8
%malloccall21 = tail call i8* @malloc(i32 ptrtoint (%list_item*
  getelementptr (%list_item, %list_item* null, i32 1) to i32))
%list_item22 = bitcast i8* %malloccall21 to %list_item*
store %list_item zeroinitializer, %list_item* %list_item22, align 1
%copied24 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
   null, i32 1) to i32))
store i8 86, i8* %copied24, align 1
```

```
  %data_ptr_container25 = getelementptr inbounds %list_item, %list_item* %
    list_item22, i32 0, i32 0
  store i8* %copied24, i8** %data_ptr_container25, align 8
  %next26 = getelementptr inbounds %list_item, %list_item* %list_item22, i32
     0, i32 1
  store %list_item* %list_item16, %list_item** %next26, align 8
  %malloccall27 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item28 = bitcast i8* %malloccall27 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item28, align 1
  %copied30 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 85, i8* %copied30, align 1
  %data_ptr_container31 = getelementptr inbounds %list_item, %list_item* %
    list_item28, i32 0, i32 0
  store i8* %copied30, i8** %data_ptr_container31, align 8
  %next32 = getelementptr inbounds %list_item, %list_item* %list_item28, i32
     0, i32 1
  store %list_item* %list_item22, %list_item** %next32, align 8
  %malloccall33 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item34 = bitcast i8* %malloccall33 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item34, align 1
  %copied36 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 84, i8* %copied36, align 1
  %data_ptr_container37 = getelementptr inbounds %list_item, %list_item* %
    list_item34, i32 0, i32 0
  store i8* %copied36, i8** %data_ptr_container37, align 8
  %next38 = getelementptr inbounds %list_item, %list_item* %list_item34, i32
     0, i32 1
  store %list_item* %list_item28, %list_item** %next38, align 8
  %malloccall39 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item40 = bitcast i8* %malloccall39 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item40, align 1
  %copied42 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 83, i8* %copied42, align 1
  %data_ptr_container43 = getelementptr inbounds %list_item, %list_item* %
    list_item40, i32 0, i32 0
  store i8* %copied42, i8** %data_ptr_container43, align 8
  %next44 = getelementptr inbounds %list_item, %list_item* %list_item40, i32
     0, i32 1
  store %list_item* %list_item34, %list_item** %next44, align 8
  %malloccall45 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item46 = bitcast i8* %malloccall45 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item46, align 1
  %copied48 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 82, i8* %copied48, align 1
  %data_ptr_container49 = getelementptr inbounds %list_item, %list_item* %
    list_item46, i32 0, i32 0
  store i8* %copied48, i8** %data_ptr_container49, align 8
  %next50 = getelementptr inbounds %list_item, %list_item* %list_item46, i32
     0, i32 1
  store %list_item* %list_item40, %list_item** %next50, align 8
```

```
  %malloccall51 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item52 = bitcast i8* %malloccall51 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item52, align 1
  %copied54 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 81, i8* %copied54, align 1
  %data_ptr_container55 = getelementptr inbounds %list_item, %list_item* %
    list_item52, i32 0, i32 0
  store i8* %copied54, i8** %data_ptr_container55, align 8
  %next56 = getelementptr inbounds %list_item, %list_item* %list_item52, i32
     0, i32 1
  store %list_item* %list_item46, %list_item** %next56, align 8
  %malloccall57 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item58 = bitcast i8* %malloccall57 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item58, align 1
  %copied60 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 80, i8* %copied60, align 1
  %data_ptr_container61 = getelementptr inbounds %list_item, %list_item* %
    list_item58, i32 0, i32 0
  store i8* %copied60, i8** %data_ptr_container61, align 8
  %next62 = getelementptr inbounds %list_item, %list_item* %list_item58, i32
     0, i32 1
  store %list_item* %list_item52, %list_item** %next62, align 8
  %malloccall63 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item64 = bitcast i8* %malloccall63 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item64, align 1
  %copied66 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 79, i8* %copied66, align 1
  %data_ptr_container67 = getelementptr inbounds %list_item, %list_item* %
    list_item64, i32 0, i32 0
  store i8* %copied66, i8** %data_ptr_container67, align 8
  %next68 = getelementptr inbounds %list_item, %list_item* %list_item64, i32
     0, i32 1
  store %list_item* %list_item58, %list_item** %next68, align 8
  %malloccall69 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item70 = bitcast i8* %malloccall69 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item70, align 1
  %copied72 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 78, i8* %copied72, align 1
  %data_ptr_container73 = getelementptr inbounds %list_item, %list_item* %
    list_item70, i32 0, i32 0
  store i8* %copied72, i8** %data_ptr_container73, align 8
  %next74 = getelementptr inbounds %list_item, %list_item* %list_item70, i32
     0, i32 1
  store %list_item* %list_item64, %list_item** %next74, align 8
  %malloccall75 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item76 = bitcast i8* %malloccall75 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item76, align 1
  %copied78 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 77, i8* %copied78, align 1
```

```
  %data_ptr_container79 = getelementptr inbounds %list_item, %list_item* %
    list_item76, i32 0, i32 0
  store i8* %copied78, i8** %data_ptr_container79, align 8
  %next80 = getelementptr inbounds %list_item, %list_item* %list_item76, i32
     0, i32 1
  store %list_item* %list_item70, %list_item** %next80, align 8
  %malloccall81 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item82 = bitcast i8* %malloccall81 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item82, align 1
  %copied84 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 76, i8* %copied84, align 1
  %data_ptr_container85 = getelementptr inbounds %list_item, %list_item* %
    list_item82, i32 0, i32 0
  store i8* %copied84, i8** %data_ptr_container85, align 8
  %next86 = getelementptr inbounds %list_item, %list_item* %list_item82, i32
     0, i32 1
  store %list_item* %list_item76, %list_item** %next86, align 8
  %malloccall87 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item88 = bitcast i8* %malloccall87 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item88, align 1
  %copied90 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 75, i8* %copied90, align 1
  %data_ptr_container91 = getelementptr inbounds %list_item, %list_item* %
    list_item88, i32 0, i32 0
  store i8* %copied90, i8** %data_ptr_container91, align 8
  %next92 = getelementptr inbounds %list_item, %list_item* %list_item88, i32
     0, i32 1
  store %list_item* %list_item82, %list_item** %next92, align 8
  %malloccall93 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item94 = bitcast i8* %malloccall93 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item94, align 1
  %copied96 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
     null, i32 1) to i32))
  store i8 74, i8* %copied96, align 1
  %data_ptr_container97 = getelementptr inbounds %list_item, %list_item* %
    list_item94, i32 0, i32 0
  store i8* %copied96, i8** %data_ptr_container97, align 8
  %next98 = getelementptr inbounds %list_item, %list_item* %list_item94, i32
     0, i32 1
  store %list_item* %list_item88, %list_item** %next98, align 8
  %malloccall99 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item100 = bitcast i8* %malloccall99 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item100, align 1
  %copied102 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 73, i8* %copied102, align 1
  %data_ptr_container103 = getelementptr inbounds %list_item, %list_item* %
    list_item100, i32 0, i32 0
  store i8* %copied102, i8** %data_ptr_container103, align 8
  %next104 = getelementptr inbounds %list_item, %list_item* %list_item100,
    i32 0, i32 1
  store %list_item* %list_item94, %list_item** %next104, align 8
```

```
  %malloccall105 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item106 = bitcast i8* %malloccall105 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item106, align 1
  %copied108 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 72, i8* %copied108, align 1
  %data_ptr_container109 = getelementptr inbounds %list_item, %list_item* %
    list_item106, i32 0, i32 0
  store i8* %copied108, i8** %data_ptr_container109, align 8
  %next110 = getelementptr inbounds %list_item, %list_item* %list_item106,
    i32 0, i32 1
  store %list_item* %list_item100, %list_item** %next110, align 8
  %malloccall111 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item112 = bitcast i8* %malloccall111 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item112, align 1
  %copied114 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 71, i8* %copied114, align 1
  %data_ptr_container115 = getelementptr inbounds %list_item, %list_item* %
    list_item112, i32 0, i32 0
  store i8* %copied114, i8** %data_ptr_container115, align 8
  %next116 = getelementptr inbounds %list_item, %list_item* %list_item112,
    i32 0, i32 1
  store %list_item* %list_item106, %list_item** %next116, align 8
  %malloccall117 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item118 = bitcast i8* %malloccall117 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item118, align 1
  %copied120 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 70, i8* %copied120, align 1
  %data_ptr_container121 = getelementptr inbounds %list_item, %list_item* %
    list_item118, i32 0, i32 0
  store i8* %copied120, i8** %data_ptr_container121, align 8
  %next122 = getelementptr inbounds %list_item, %list_item* %list_item118,
    i32 0, i32 1
  store %list_item* %list_item112, %list_item** %next122, align 8
  %malloccall123 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item124 = bitcast i8* %malloccall123 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item124, align 1
  %copied126 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 69, i8* %copied126, align 1
  %data_ptr_container127 = getelementptr inbounds %list_item, %list_item* %
    list_item124, i32 0, i32 0
  store i8* %copied126, i8** %data_ptr_container127, align 8
  %next128 = getelementptr inbounds %list_item, %list_item* %list_item124,
    i32 0, i32 1
  store %list_item* %list_item118, %list_item** %next128, align 8
  %malloccall129 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item130 = bitcast i8* %malloccall129 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item130, align 1
  %copied132 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  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   %malloccall135 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
253   %list_item136 = bitcast i8* %malloccall135 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   %malloccall141 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
262   %list_item142 = bitcast i8* %malloccall141 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   %malloccall147 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
271   %list_item148 = bitcast i8* %malloccall147 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   %malloccall153 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
280   %list_item154 = bitcast i8* %malloccall153 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
```

```
  %malloccall159 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item160 = bitcast i8* %malloccall159 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item160, align 1
  %copied162 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 121, i8* %copied162, align 1
  %data_ptr_container163 = getelementptr inbounds %list_item, %list_item* %
    list_item160, i32 0, i32 0
  store i8* %copied162, i8** %data_ptr_container163, align 8
  %next164 = getelementptr inbounds %list_item, %list_item* %list_item160,
    i32 0, i32 1
  store %list_item* %list_item154, %list_item** %next164, align 8
  %malloccall165 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item166 = bitcast i8* %malloccall165 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item166, align 1
  %copied168 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 120, i8* %copied168, align 1
  %data_ptr_container169 = getelementptr inbounds %list_item, %list_item* %
    list_item166, i32 0, i32 0
  store i8* %copied168, i8** %data_ptr_container169, align 8
  %next170 = getelementptr inbounds %list_item, %list_item* %list_item166,
    i32 0, i32 1
  store %list_item* %list_item160, %list_item** %next170, align 8
  %malloccall171 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item172 = bitcast i8* %malloccall171 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item172, align 1
  %copied174 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 119, i8* %copied174, align 1
  %data_ptr_container175 = getelementptr inbounds %list_item, %list_item* %
    list_item172, i32 0, i32 0
  store i8* %copied174, i8** %data_ptr_container175, align 8
  %next176 = getelementptr inbounds %list_item, %list_item* %list_item172,
    i32 0, i32 1
  store %list_item* %list_item166, %list_item** %next176, align 8
  %malloccall177 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item178 = bitcast i8* %malloccall177 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item178, align 1
  %copied180 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 118, i8* %copied180, align 1
  %data_ptr_container181 = getelementptr inbounds %list_item, %list_item* %
    list_item178, i32 0, i32 0
  store i8* %copied180, i8** %data_ptr_container181, align 8
  %next182 = getelementptr inbounds %list_item, %list_item* %list_item178,
    i32 0, i32 1
  store %list_item* %list_item172, %list_item** %next182, align 8
  %malloccall183 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item184 = bitcast i8* %malloccall183 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item184, align 1
  %copied186 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 117, i8* %copied186, align 1
```

```
  %data_ptr_container187 = getelementptr inbounds %list_item, %list_item* %
    list_item184, i32 0, i32 0
  store i8* %copied186, i8** %data_ptr_container187, align 8
  %next188 = getelementptr inbounds %list_item, %list_item* %list_item184,
    i32 0, i32 1
  store %list_item* %list_item178, %list_item** %next188, align 8
  %malloccall189 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item190 = bitcast i8* %malloccall189 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item190, align 1
  %copied192 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 116, i8* %copied192, align 1
  %data_ptr_container193 = getelementptr inbounds %list_item, %list_item* %
    list_item190, i32 0, i32 0
  store i8* %copied192, i8** %data_ptr_container193, align 8
  %next194 = getelementptr inbounds %list_item, %list_item* %list_item190,
    i32 0, i32 1
  store %list_item* %list_item184, %list_item** %next194, align 8
  %malloccall195 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item196 = bitcast i8* %malloccall195 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item196, align 1
  %copied198 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 115, i8* %copied198, align 1
  %data_ptr_container199 = getelementptr inbounds %list_item, %list_item* %
    list_item196, i32 0, i32 0
  store i8* %copied198, i8** %data_ptr_container199, align 8
  %next200 = getelementptr inbounds %list_item, %list_item* %list_item196,
    i32 0, i32 1
  store %list_item* %list_item190, %list_item** %next200, align 8
  %malloccall201 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item202 = bitcast i8* %malloccall201 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item202, align 1
  %copied204 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 114, i8* %copied204, align 1
  %data_ptr_container205 = getelementptr inbounds %list_item, %list_item* %
    list_item202, i32 0, i32 0
  store i8* %copied204, i8** %data_ptr_container205, align 8
  %next206 = getelementptr inbounds %list_item, %list_item* %list_item202,
    i32 0, i32 1
  store %list_item* %list_item196, %list_item** %next206, align 8
  %malloccall207 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item208 = bitcast i8* %malloccall207 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item208, align 1
  %copied210 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 113, i8* %copied210, align 1
  %data_ptr_container211 = getelementptr inbounds %list_item, %list_item* %
    list_item208, i32 0, i32 0
  store i8* %copied210, i8** %data_ptr_container211, align 8
  %next212 = getelementptr inbounds %list_item, %list_item* %list_item208,
    i32 0, i32 1
  store %list_item* %list_item202, %list_item** %next212, align 8
```

```
  %malloccall213 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item214 = bitcast i8* %malloccall213 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item214, align 1
  %copied216 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 112, i8* %copied216, align 1
  %data_ptr_container217 = getelementptr inbounds %list_item, %list_item* %
    list_item214, i32 0, i32 0
  store i8* %copied216, i8** %data_ptr_container217, align 8
  %next218 = getelementptr inbounds %list_item, %list_item* %list_item214,
    i32 0, i32 1
  store %list_item* %list_item208, %list_item** %next218, align 8
  %malloccall219 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item220 = bitcast i8* %malloccall219 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item220, align 1
  %copied222 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 111, i8* %copied222, align 1
  %data_ptr_container223 = getelementptr inbounds %list_item, %list_item* %
    list_item220, i32 0, i32 0
  store i8* %copied222, i8** %data_ptr_container223, align 8
  %next224 = getelementptr inbounds %list_item, %list_item* %list_item220,
    i32 0, i32 1
  store %list_item* %list_item214, %list_item** %next224, align 8
  %malloccall225 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item226 = bitcast i8* %malloccall225 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item226, align 1
  %copied228 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 110, i8* %copied228, align 1
  %data_ptr_container229 = getelementptr inbounds %list_item, %list_item* %
    list_item226, i32 0, i32 0
  store i8* %copied228, i8** %data_ptr_container229, align 8
  %next230 = getelementptr inbounds %list_item, %list_item* %list_item226,
    i32 0, i32 1
  store %list_item* %list_item220, %list_item** %next230, align 8
  %malloccall231 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item232 = bitcast i8* %malloccall231 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item232, align 1
  %copied234 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 109, i8* %copied234, align 1
  %data_ptr_container235 = getelementptr inbounds %list_item, %list_item* %
    list_item232, i32 0, i32 0
  store i8* %copied234, i8** %data_ptr_container235, align 8
  %next236 = getelementptr inbounds %list_item, %list_item* %list_item232,
    i32 0, i32 1
  store %list_item* %list_item226, %list_item** %next236, align 8
  %malloccall237 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item238 = bitcast i8* %malloccall237 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item238, align 1
  %copied240 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  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   %malloccall243 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
415   %list_item244 = bitcast i8* %malloccall243 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   %malloccall249 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
424   %list_item250 = bitcast i8* %malloccall249 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   %malloccall255 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
433   %list_item256 = bitcast i8* %malloccall255 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   %malloccall261 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
442   %list_item262 = bitcast i8* %malloccall261 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
```

```
  %malloccall267 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item268 = bitcast i8* %malloccall267 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item268, align 1
  %copied270 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 103, i8* %copied270, align 1
  %data_ptr_container271 = getelementptr inbounds %list_item, %list_item* %
    list_item268, i32 0, i32 0
  store i8* %copied270, i8** %data_ptr_container271, align 8
  %next272 = getelementptr inbounds %list_item, %list_item* %list_item268,
    i32 0, i32 1
  store %list_item* %list_item262, %list_item** %next272, align 8
  %malloccall273 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item274 = bitcast i8* %malloccall273 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item274, align 1
  %copied276 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 102, i8* %copied276, align 1
  %data_ptr_container277 = getelementptr inbounds %list_item, %list_item* %
    list_item274, i32 0, i32 0
  store i8* %copied276, i8** %data_ptr_container277, align 8
  %next278 = getelementptr inbounds %list_item, %list_item* %list_item274,
    i32 0, i32 1
  store %list_item* %list_item268, %list_item** %next278, align 8
  %malloccall279 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item280 = bitcast i8* %malloccall279 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item280, align 1
  %copied282 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 101, i8* %copied282, align 1
  %data_ptr_container283 = getelementptr inbounds %list_item, %list_item* %
    list_item280, i32 0, i32 0
  store i8* %copied282, i8** %data_ptr_container283, align 8
  %next284 = getelementptr inbounds %list_item, %list_item* %list_item280,
    i32 0, i32 1
  store %list_item* %list_item274, %list_item** %next284, align 8
  %malloccall285 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item286 = bitcast i8* %malloccall285 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item286, align 1
  %copied288 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  store i8 100, i8* %copied288, align 1
  %data_ptr_container289 = getelementptr inbounds %list_item, %list_item* %
    list_item286, i32 0, i32 0
  store i8* %copied288, i8** %data_ptr_container289, align 8
  %next290 = getelementptr inbounds %list_item, %list_item* %list_item286,
    i32 0, i32 1
  store %list_item* %list_item280, %list_item** %next290, align 8
  %malloccall291 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item292 = bitcast i8* %malloccall291 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item292, align 1
  %copied294 = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32))
  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   %malloccall297 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
496   %list_item298 = bitcast i8* %malloccall297 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   %malloccall303 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
505   %list_item304 = bitcast i8* %malloccall303 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   %malloccall309 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
        i1*, i1** null, i32 1) to i32))
516   %list310 = bitcast i8* %malloccall309 to %list_item**
517   %malloccall311 = tail call i8* @malloc(i32 ptrtoint (%list_item*
        getelementptr (%list_item, %list_item* null, i32 1) to i32))
518   %list_item312 = bitcast i8* %malloccall311 to %list_item*
519   store %list_item zeroinitializer, %list_item* %list_item312, align 1
520   %malloccall313 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
        i1*, i1** null, i32 1) to i32))
521   %copied314 = bitcast i8* %malloccall313 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   %malloccall317 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
        i1*, i1** null, i32 1) to i32))
531   %list318 = bitcast i8* %malloccall317 to %list_item**
```

```
  %malloccall319 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item320 = bitcast i8* %malloccall319 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item320, align 1
  %malloccall321 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
  %copied322 = bitcast i8* %malloccall321 to i32*
  store i32 1, i32* %copied322, align 4
  %cast_ptr323 = bitcast i32* %copied322 to i8*
  %data_ptr_container324 = getelementptr inbounds %list_item, %list_item* %
    list_item320, i32 0, i32 0
  store i8* %cast_ptr323, i8** %data_ptr_container324, align 8
  %next325 = getelementptr inbounds %list_item, %list_item* %list_item320,
    i32 0, i32 1
  store %list_item* null, %list_item** %next325, align 8
  store %list_item* %list_item320, %list_item** %list318, align 8
  %malloccall326 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list327 = bitcast i8* %malloccall326 to %list_item**
  %malloccall328 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item329 = bitcast i8* %malloccall328 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item329, align 1
  %malloccall330 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
  %copied331 = bitcast i8* %malloccall330 to i32*
  store i32 2, i32* %copied331, align 4
  %cast_ptr332 = bitcast i32* %copied331 to i8*
  %data_ptr_container333 = getelementptr inbounds %list_item, %list_item* %
    list_item329, i32 0, i32 0
  store i8* %cast_ptr332, i8** %data_ptr_container333, align 8
  %next334 = getelementptr inbounds %list_item, %list_item* %list_item329,
    i32 0, i32 1
  store %list_item* null, %list_item** %next334, align 8
  %malloccall335 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item336 = bitcast i8* %malloccall335 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item336, align 1
  %malloccall337 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (
    i32, i32* null, i32 1) to i32))
  %copied338 = bitcast i8* %malloccall337 to i32*
  store i32 1, i32* %copied338, align 4
  %cast_ptr339 = bitcast i32* %copied338 to i8*
  %data_ptr_container340 = getelementptr inbounds %list_item, %list_item* %
    list_item336, i32 0, i32 0
  store i8* %cast_ptr339, i8** %data_ptr_container340, align 8
  %next341 = getelementptr inbounds %list_item, %list_item* %list_item336,
    i32 0, i32 1
  store %list_item* %list_item329, %list_item** %next341, align 8
  store %list_item* %list_item336, %list_item** %list327, align 8
  %_result342 = call i1 @contains_int_int(%list_item** %list327, %list_item
    ** %list318)
  %malloccall343 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list344 = bitcast i8* %malloccall343 to %list_item**
  %malloccall345 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item346 = bitcast i8* %malloccall345 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item346, align 1
```

```
  %malloccall347 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %copied348 = bitcast i8* %malloccall347 to i8**
  store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.2, i32 0,
    i32 0), i8** %copied348, align 8
  %cast_ptr349 = bitcast i8** %copied348 to i8*
  %data_ptr_container350 = getelementptr inbounds %list_item, %list_item* %
    list_item346, i32 0, i32 0
  store i8* %cast_ptr349, i8** %data_ptr_container350, align 8
  %next351 = getelementptr inbounds %list_item, %list_item* %list_item346,
    i32 0, i32 1
  store %list_item* null, %list_item** %next351, align 8
  store %list_item* %list_item346, %list_item** %list344, align 8
  %malloccall352 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list353 = bitcast i8* %malloccall352 to %list_item**
  %malloccall354 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item355 = bitcast i8* %malloccall354 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item355, align 1
  %malloccall356 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %copied357 = bitcast i8* %malloccall356 to i8**
  store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.3, i32 0,
    i32 0), i8** %copied357, align 8
  %cast_ptr358 = bitcast i8** %copied357 to i8*
  %data_ptr_container359 = getelementptr inbounds %list_item, %list_item* %
    list_item355, i32 0, i32 0
  store i8* %cast_ptr358, i8** %data_ptr_container359, align 8
  %next360 = getelementptr inbounds %list_item, %list_item* %list_item355,
    i32 0, i32 1
  store %list_item* null, %list_item** %next360, align 8
  %malloccall361 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item362 = bitcast i8* %malloccall361 to %list_item*
  store %list_item zeroinitializer, %list_item* %list_item362, align 1
  %malloccall363 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %copied364 = bitcast i8* %malloccall363 to i8**
  store i8* getelementptr inbounds ([6 x i8], [6 x i8]* @string.4, i32 0,
    i32 0), i8** %copied364, align 8
  %cast_ptr365 = bitcast i8** %copied364 to i8*
  %data_ptr_container366 = getelementptr inbounds %list_item, %list_item* %
    list_item362, i32 0, i32 0
  store i8* %cast_ptr365, i8** %data_ptr_container366, align 8
  %next367 = getelementptr inbounds %list_item, %list_item* %list_item362,
    i32 0, i32 1
  store %list_item* %list_item355, %list_item** %next367, align 8
  store %list_item* %list_item362, %list_item** %list353, align 8
  %_result368 = call i1 @contains_string_string(%list_item** %list353, %
    list_item** %list344)
  %malloccall369 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %list370 = bitcast i8* %malloccall369 to %list_item**
  %malloccall371 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %list_item372 = bitcast i8* %malloccall371 to %list_item*
  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, %
      then13, %then
835   %i27 = load i32, i32* %i, align 4
836   %len28 = load i32, i32* %len, align 4
837   %tmp29 = icmp slt i32 %i27, %len28
838   br i1 %tmp29, label %while_body, label %merge6
839
840 merge6:                                           ; preds = %while
841   br label %merge
842
843 while_body:                                       ; preds = %while
844   %elem7 = load i8*, i8** %elem2, align 8
845   %l8 = load %list_item**, %list_item*** %l1, align 8
846   %ilist9 = load %list_item*, %list_item** %l8, align 8
847   %i10 = load i32, i32* %i, align 4
848   %_result = call %list_item* @list_access(%list_item* %ilist9, i32 %i10)
849   %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
        i32 0, i32 0
850   %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
851   %cast_data_ptr = bitcast i8* %data_ptr to i8**
852   %data = load i8*, i8** %cast_data_ptr, align 8
853   %_result11 = call i1 @strcmp(i8* %data, i8* %elem7)
854   br i1 %_result11, label %then13, label %else
855
856 merge12:                                          ; preds = %else
857   %retlist15 = load %list_item**, %list_item*** %retlist, align 8
858   %list_ptr = load %list_item*, %list_item** %retlist15, align 8
859   %l16 = load %list_item**, %list_item*** %l1, align 8
860   %ilist17 = load %list_item*, %list_item** %l16, align 8
861   %i18 = load i32, i32* %i, align 4
862   %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
863   %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
        _result19, i32 0, i32 0
864   %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
865   %cast_data_ptr22 = bitcast i8* %data_ptr21 to i8**
866   %data23 = load i8*, i8** %cast_data_ptr22, align 8
867   %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
868   %list_ptr_ptr = call %list_item** @insert_string(%list_item** %retlist15,
        i8* %data23, i32 %length24)
869   store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
870   %i25 = load i32, i32* %i, align 4
871   %tmp26 = add i32 %i25, 1
872   store i32 %tmp26, i32* %i, align 4
873   br label %while
874
875 then13:                                           ; preds = %while_body
876   %i14 = load i32, i32* %i, align 4
877   %tmp = add i32 %i14, 1
878   store i32 %tmp, i32* %i, align 4
879   br label %while
880
881 else:                                             ; preds = %while_body
882   br label %merge12
883
884 else30:                                           ; preds = %entry
```

```
885   br label %while31
886
887 while31:                                          ; preds = %merge44, %
        else30
888   %i62 = load i32, i32* %i, align 4
889   %len63 = load i32, i32* %len, align 4
890   %tmp64 = icmp slt i32 %i62, %len63
891   br i1 %tmp64, label %while_body33, label %merge32
892
893 merge32:                                          ; preds = %while31, %
        then45
894   %i65 = load i32, i32* %i, align 4
895   %len66 = load i32, i32* %len, align 4
896   %tmp67 = icmp ne i32 %i65, %len66
897   br i1 %tmp67, label %then69, label %else108
898
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* %
        _result38, i32 0, i32 0
906   %data_ptr40 = load i8*, i8** %data_ptr_ptr39, align 8
907   %cast_data_ptr41 = bitcast i8* %data_ptr40 to i8**
908   %data42 = load i8*, i8** %cast_data_ptr41, align 8
909   %_result43 = call i1 @strcmp(i8* %data42, i8* %elem34)
910   br i1 %_result43, label %then45, label %else47
911
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* %
        _result53, i32 0, i32 0
920   %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
921   %cast_data_ptr56 = bitcast i8* %data_ptr55 to i8**
922   %data57 = load i8*, i8** %cast_data_ptr56, align 8
923   %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
924   %list_ptr_ptr59 = call %list_item** @insert_string(%list_item** %retlist48
        , i8* %data57, i32 %length58)
925   store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
926   %i60 = load i32, i32* %i, align 4
927   %tmp61 = add i32 %i60, 1
928   store i32 %tmp61, i32* %i, align 4
929   br label %while31
930
931 then45:                                           ; preds = %while_body33
932   %i46 = load i32, i32* %i, align 4
933   store i32 %i46, i32* %remove_index, align 4
934   br label %merge32
935
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   %ilist105 = load %list_item*, %list_item** %range_list104, align 8
957   %length106 = call i32 @list_length(%list_item* %ilist105, 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   %malloccall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
        *, i1** null, i32 1) to i32))
969   %head_ptr_ptr = bitcast i8* %malloccall76 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   %ilist77 = 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* %ilist77, 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   %ilist89 = 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* %ilist89, i32 %
        index90)
```

```
  %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
  %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
  %cast_data_ptr94 = bitcast i8* %data_ptr93 to i8**
  %data95 = load i8*, i8** %cast_data_ptr94, align 8
  %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
  %list_ptr_ptr97 = call %list_item** @insert_string(%list_item** %retlist86
    , i8* %data95, i32 %length96)
  store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
  br label %while70

else108:                                          ; preds = %merge32
  br label %merge68
}

define %list_item** @remove_char_char_char_bool(%list_item** %l, i8 %elem,
    i1 %all) {
entry:
  %index = alloca i32, align 4
  %for_index = alloca i32, align 4
  %remove_index = alloca i32, align 4
  %len = alloca i32, align 4
  %i = alloca i32, align 4
  %retlist = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %elem2 = alloca i8, align 1
  store i8 %elem, i8* %elem2, align 1
  %all3 = alloca i1, align 1
  store i1 %all, i1* %all3, align 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %retlist, align 8
  store i32 0, i32* %i, align 4
  %l4 = load %list_item**, %list_item*** %l1, align 8
  %ilist = load %list_item*, %list_item** %l4, align 8
  %length = call i32 @list_length(%list_item* %ilist, i32 0)
  store i32 %length, i32* %len, align 4
  store i32 0, i32* %remove_index, align 4
  %all5 = load i1, i1* %all3, align 1
  br i1 %all5, label %then, label %else29

merge:                                            ; preds = %merge65, %
    merge6
  %retlist104 = load %list_item**, %list_item*** %retlist, align 8
  ret %list_item** %retlist104

then:                                             ; preds = %entry
  br label %while

while:                                            ; preds = %merge11, %
    then12, %then
  %i26 = load i32, i32* %i, align 4
  %len27 = load i32, i32* %len, align 4
  %tmp28 = icmp slt i32 %i26, %len27
  br i1 %tmp28, label %while_body, label %merge6

```

```
merge6:                                           ; preds = %while
  br label %merge

while_body:                                       ; preds = %while
  %l7 = load %list_item**, %list_item*** %l1, align 8
  %ilist8 = load %list_item*, %list_item** %l7, align 8
  %i9 = load i32, i32* %i, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %data = load i8, i8* %data_ptr, align 1
  %elem10 = load i8, i8* %elem2, align 1
  %tmp = icmp eq i8 %data, %elem10
  br i1 %tmp, label %then12, label %else

merge11:                                          ; preds = %else
  %retlist15 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr = load %list_item*, %list_item** %retlist15, align 8
  %l16 = load %list_item**, %list_item*** %l1, align 8
  %ilist17 = load %list_item*, %list_item** %l16, align 8
  %i18 = load i32, i32* %i, align 4
  %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %data22 = load i8, i8* %data_ptr21, align 1
  %length23 = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_char(%list_item** %retlist15, i8
     %data22, i32 %length23)
  store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
  %i24 = load i32, i32* %i, align 4
  %tmp25 = add i32 %i24, 1
  store i32 %tmp25, i32* %i, align 4
  br label %while

then12:                                           ; preds = %while_body
  %i13 = load i32, i32* %i, align 4
  %tmp14 = add i32 %i13, 1
  store i32 %tmp14, i32* %i, align 4
  br label %while

else:                                             ; preds = %while_body
  br label %merge11

else29:                                           ; preds = %entry
  br label %while30

while30:                                          ; preds = %merge42, %
    else29
  %i59 = load i32, i32* %i, align 4
  %len60 = load i32, i32* %len, align 4
  %tmp61 = icmp slt i32 %i59, %len60
  br i1 %tmp61, label %while_body32, label %merge31

merge31:                                          ; preds = %while30, %
    then43
  %i62 = load i32, i32* %i, align 4
  %len63 = load i32, i32* %len, align 4
```

```
  %tmp64 = icmp ne i32 %i62, %len63
  br i1 %tmp64, label %then66, label %else103

while_body32:                                     ; preds = %while30
  %l33 = load %list_item**, %list_item*** %l1, align 8
  %ilist34 = load %list_item*, %list_item** %l33, align 8
  %i35 = load i32, i32* %i, align 4
  %_result36 = call %list_item* @list_access(%list_item* %ilist34, i32 %i35)
  %data_ptr_ptr37 = getelementptr inbounds %list_item, %list_item* %
    _result36, i32 0, i32 0
  %data_ptr38 = load i8*, i8** %data_ptr_ptr37, align 8
  %data39 = load i8, i8* %data_ptr38, align 1
  %elem40 = load i8, i8* %elem2, align 1
  %tmp41 = icmp eq i8 %data39, %elem40
  br i1 %tmp41, label %then43, label %else45

merge42:                                          ; preds = %else45
  %retlist46 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr47 = load %list_item*, %list_item** %retlist46, align 8
  %l48 = load %list_item**, %list_item*** %l1, align 8
  %ilist49 = load %list_item*, %list_item** %l48, align 8
  %i50 = load i32, i32* %i, align 4
  %_result51 = call %list_item* @list_access(%list_item* %ilist49, i32 %i50)
  %data_ptr_ptr52 = getelementptr inbounds %list_item, %list_item* %
    _result51, i32 0, i32 0
  %data_ptr53 = load i8*, i8** %data_ptr_ptr52, align 8
  %data54 = load i8, i8* %data_ptr53, align 1
  %length55 = call i32 @list_length(%list_item* %list_ptr47, i32 0)
  %list_ptr_ptr56 = call %list_item** @insert_char(%list_item** %retlist46,
    i8 %data54, i32 %length55)
  store %list_item** %list_ptr_ptr56, %list_item*** %retlist, align 8
  %i57 = load i32, i32* %i, align 4
  %tmp58 = add i32 %i57, 1
  store i32 %tmp58, i32* %i, align 4
  br label %while30

then43:                                           ; preds = %while_body32
  %i44 = load i32, i32* %i, align 4
  store i32 %i44, i32* %remove_index, align 4
  br label %merge31

else45:                                           ; preds = %while_body32
  br label %merge42

merge65:                                          ; preds = %else103, %
    merge68
  br label %merge

then66:                                           ; preds = %merge31
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while67

while67:                                          ; preds = %while_body69, %
    then66
  %for_index93 = load i32, i32* %for_index, align 4
  %remove_index94 = load i32, i32* %remove_index, align 4
  %tmp95 = add i32 %remove_index94, 1
  %len96 = load i32, i32* %len, align 4
```

```
  %malloccall97 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr98 = bitcast i8* %malloccall97 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr98, align 8
  %range_list99 = call %list_item** @range_function(i32 %tmp95, i32 %len96,
    %list_item** %head_ptr_ptr98, i32 0)
  %ilist100 = load %list_item*, %list_item** %range_list99, align 8
  %length101 = call i32 @list_length(%list_item* %ilist100, i32 0)
  %tmp102 = icmp slt i32 %for_index93, %length101
  br i1 %tmp102, label %while_body69, label %merge68

merge68:                                          ; preds = %while67
  br label %merge65

while_body69:                                     ; preds = %while67
  %remove_index70 = load i32, i32* %remove_index, align 4
  %tmp71 = add i32 %remove_index70, 1
  %len72 = load i32, i32* %len, align 4
  %malloccall73 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall73 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
  %range_list = call %list_item** @range_function(i32 %tmp71, i32 %len72, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist74 = load %list_item*, %list_item** %range_list, align 8
  %for_index75 = load i32, i32* %for_index, align 4
  %_result76 = call %list_item* @list_access(%list_item* %ilist74, i32 %
    for_index75)
  %data_ptr_ptr77 = getelementptr inbounds %list_item, %list_item* %
    _result76, i32 0, i32 0
  %data_ptr78 = load i8*, i8** %data_ptr_ptr77, align 8
  %cast_data_ptr = bitcast i8* %data_ptr78 to i32*
  %data79 = load i32, i32* %cast_data_ptr, align 4
  store i32 %data79, i32* %index, align 4
  %for_index80 = load i32, i32* %for_index, align 4
  %tmp81 = add i32 %for_index80, 1
  store i32 %tmp81, i32* %for_index, align 4
  %retlist82 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr83 = load %list_item*, %list_item** %retlist82, align 8
  %l84 = load %list_item**, %list_item*** %l1, align 8
  %ilist85 = load %list_item*, %list_item** %l84, align 8
  %index86 = load i32, i32* %index, align 4
  %_result87 = call %list_item* @list_access(%list_item* %ilist85, i32 %
    index86)
  %data_ptr_ptr88 = getelementptr inbounds %list_item, %list_item* %
    _result87, i32 0, i32 0
  %data_ptr89 = load i8*, i8** %data_ptr_ptr88, align 8
  %data90 = load i8, i8* %data_ptr89, align 1
  %length91 = call i32 @list_length(%list_item* %list_ptr83, i32 0)
  %list_ptr_ptr92 = call %list_item** @insert_char(%list_item** %retlist82,
    i8 %data90, i32 %length91)
  store %list_item** %list_ptr_ptr92, %list_item*** %retlist, align 8
  br label %while67

else103:                                          ; preds = %merge31
  br label %merge65
}
```

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

```
  br i1 %tmp, label %then12, label %else

merge11:                                          ; preds = %else
  %retlist15 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr = load %list_item*, %list_item** %retlist15, align 8
  %l16 = load %list_item**, %list_item*** %l1, align 8
  %ilist17 = load %list_item*, %list_item** %l16, align 8
  %i18 = load i32, i32* %i, align 4
  %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %cast_data_ptr22 = bitcast i8* %data_ptr21 to i1*
  %data23 = load i1, i1* %cast_data_ptr22, align 1
  %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %retlist15, i1
     %data23, i32 %length24)
  store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
  %i25 = load i32, i32* %i, align 4
  %tmp26 = add i32 %i25, 1
  store i32 %tmp26, i32* %i, align 4
  br label %while

then12:                                           ; preds = %while_body
  %i13 = load i32, i32* %i, align 4
  %tmp14 = add i32 %i13, 1
  store i32 %tmp14, i32* %i, align 4
  br label %while

else:                                             ; preds = %while_body
  br label %merge11

else30:                                           ; preds = %entry
  br label %while31

while31:                                          ; preds = %merge44, %
    else30
  %i62 = load i32, i32* %i, align 4
  %len63 = load i32, i32* %len, align 4
  %tmp64 = icmp slt i32 %i62, %len63
  br i1 %tmp64, label %while_body33, label %merge32

merge32:                                          ; preds = %while31, %
    then45
  %i65 = load i32, i32* %i, align 4
  %len66 = load i32, i32* %len, align 4
  %tmp67 = icmp ne i32 %i65, %len66
  br i1 %tmp67, label %then69, label %else108

while_body33:                                     ; preds = %while31
  %l34 = load %list_item**, %list_item*** %l1, align 8
  %ilist35 = load %list_item*, %list_item** %l34, align 8
  %i36 = load i32, i32* %i, align 4
  %_result37 = call %list_item* @list_access(%list_item* %ilist35, i32 %i36)
  %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
    _result37, i32 0, i32 0
  %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
  %cast_data_ptr40 = bitcast i8* %data_ptr39 to i1*
  %data41 = load i1, i1* %cast_data_ptr40, align 1
```

```
  %elem42 = load i1, i1* %elem2, align 1
  %tmp43 = icmp eq i1 %data41, %elem42
  br i1 %tmp43, label %then45, label %else47

merge44:                                          ; preds = %else47
  %retlist48 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
  %l50 = load %list_item**, %list_item*** %l1, align 8
  %ilist51 = load %list_item*, %list_item** %l50, align 8
  %i52 = load i32, i32* %i, align 4
  %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
  %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
    _result53, i32 0, i32 0
  %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
  %cast_data_ptr56 = bitcast i8* %data_ptr55 to i1*
  %data57 = load i1, i1* %cast_data_ptr56, align 1
  %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
  %list_ptr_ptr59 = call %list_item** @insert_bool(%list_item** %retlist48,
    i1 %data57, i32 %length58)
  store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
  %i60 = load i32, i32* %i, align 4
  %tmp61 = add i32 %i60, 1
  store i32 %tmp61, i32* %i, align 4
  br label %while31

then45:                                           ; preds = %while_body33
  %i46 = load i32, i32* %i, align 4
  store i32 %i46, i32* %remove_index, align 4
  br label %merge32

else47:                                           ; preds = %while_body33
  br label %merge44

merge68:                                          ; preds = %else108, %
    merge71
  br label %merge

then69:                                           ; preds = %merge32
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while70

while70:                                          ; preds = %while_body72, %
    then69
  %for_index98 = load i32, i32* %for_index, align 4
  %remove_index99 = load i32, i32* %remove_index, align 4
  %tmp100 = add i32 %remove_index99, 1
  %len101 = load i32, i32* %len, align 4
  %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %head_ptr_ptr103 = bitcast i8* %malloccall102 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr103, align 8
  %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
  %ilist105 = load %list_item*, %list_item** %range_list104, align 8
  %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
  %tmp107 = icmp slt i32 %for_index98, %length106
  br i1 %tmp107, label %while_body72, label %merge71

```

```
merge71:                                          ; preds = %while70
  br label %merge68

while_body72:                                     ; preds = %while70
  %remove_index73 = load i32, i32* %remove_index, align 4
  %tmp74 = add i32 %remove_index73, 1
  %len75 = load i32, i32* %len, align 4
  %malloccall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall76 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
  %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist77 = load %list_item*, %list_item** %range_list, align 8
  %for_index78 = load i32, i32* %for_index, align 4
  %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
    for_index78)
  %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
  %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
  %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
  %data83 = load i32, i32* %cast_data_ptr82, align 4
  store i32 %data83, i32* %index, align 4
  %for_index84 = load i32, i32* %for_index, align 4
  %tmp85 = add i32 %for_index84, 1
  store i32 %tmp85, i32* %for_index, align 4
  %retlist86 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
  %l88 = load %list_item**, %list_item*** %l1, align 8
  %ilist89 = load %list_item*, %list_item** %l88, align 8
  %index90 = load i32, i32* %index, align 4
  %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
  %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
  %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
  %cast_data_ptr94 = bitcast i8* %data_ptr93 to i1*
  %data95 = load i1, i1* %cast_data_ptr94, align 1
  %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
  %list_ptr_ptr97 = call %list_item** @insert_bool(%list_item** %retlist86,
    i1 %data95, i32 %length96)
  store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
  br label %while70

else108:                                          ; preds = %merge32
  br label %merge68
}

define %list_item** @remove_float_float_float_bool(%list_item** %l, double %
    elem, i1 %all) {
entry:
  %index = alloca i32, align 4
  %for_index = alloca i32, align 4
  %remove_index = alloca i32, align 4
  %len = alloca i32, align 4
  %i = alloca i32, align 4
  %retlist = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
```

```
  %elem2 = alloca double, align 8
  store double %elem, double* %elem2, align 8
  %all3 = alloca i1, align 1
  store i1 %all, i1* %all3, align 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %retlist, align 8
  store i32 0, i32* %i, align 4
  %l4 = load %list_item**, %list_item*** %l1, align 8
  %ilist = load %list_item*, %list_item** %l4, align 8
  %length = call i32 @list_length(%list_item* %ilist, i32 0)
  store i32 %length, i32* %len, align 4
  store i32 0, i32* %remove_index, align 4
  %all5 = load i1, i1* %all3, align 1
  br i1 %all5, label %then, label %else30

merge:                                            ; preds = %merge68, %
    merge6
  %retlist109 = load %list_item**, %list_item*** %retlist, align 8
  ret %list_item** %retlist109

then:                                             ; preds = %entry
  br label %while

while:                                            ; preds = %merge11, %
    then12, %then
  %i27 = load i32, i32* %i, align 4
  %len28 = load i32, i32* %len, align 4
  %tmp29 = icmp slt i32 %i27, %len28
  br i1 %tmp29, label %while_body, label %merge6

merge6:                                           ; preds = %while
  br label %merge

while_body:                                       ; preds = %while
  %l7 = load %list_item**, %list_item*** %l1, align 8
  %ilist8 = load %list_item*, %list_item** %l7, align 8
  %i9 = load i32, i32* %i, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to double*
  %data = load double, double* %cast_data_ptr, align 8
  %elem10 = load double, double* %elem2, align 8
  %tmp = fcmp oeq double %data, %elem10
  br i1 %tmp, label %then12, label %else

merge11:                                          ; preds = %else
  %retlist15 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr = load %list_item*, %list_item** %retlist15, align 8
  %l16 = load %list_item**, %list_item*** %l1, align 8
  %ilist17 = load %list_item*, %list_item** %l16, align 8
  %i18 = load i32, i32* %i, align 4
  %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
```

```
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %cast_data_ptr22 = bitcast i8* %data_ptr21 to double*
  %data23 = load double, double* %cast_data_ptr22, align 8
  %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_float(%list_item** %retlist15,
    double %data23, i32 %length24)
  store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
  %i25 = load i32, i32* %i, align 4
  %tmp26 = add i32 %i25, 1
  store i32 %tmp26, i32* %i, align 4
  br label %while

then12:                                           ; preds = %while_body
  %i13 = load i32, i32* %i, align 4
  %tmp14 = add i32 %i13, 1
  store i32 %tmp14, i32* %i, align 4
  br label %while

else:                                             ; preds = %while_body
  br label %merge11

else30:                                           ; preds = %entry
  br label %while31

while31:                                          ; preds = %merge44, %
    else30
  %i62 = load i32, i32* %i, align 4
  %len63 = load i32, i32* %len, align 4
  %tmp64 = icmp slt i32 %i62, %len63
  br i1 %tmp64, label %while_body33, label %merge32

merge32:                                          ; preds = %while31, %
    then45
  %i65 = load i32, i32* %i, align 4
  %len66 = load i32, i32* %len, align 4
  %tmp67 = icmp ne i32 %i65, %len66
  br i1 %tmp67, label %then69, label %else108

while_body33:                                     ; preds = %while31
  %l34 = load %list_item**, %list_item*** %l1, align 8
  %ilist35 = load %list_item*, %list_item** %l34, align 8
  %i36 = load i32, i32* %i, align 4
  %_result37 = call %list_item* @list_access(%list_item* %ilist35, i32 %i36)
  %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
    _result37, i32 0, i32 0
  %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
  %cast_data_ptr40 = bitcast i8* %data_ptr39 to double*
  %data41 = load double, double* %cast_data_ptr40, align 8
  %elem42 = load double, double* %elem2, align 8
  %tmp43 = fcmp oeq double %data41, %elem42
  br i1 %tmp43, label %then45, label %else47

merge44:                                          ; preds = %else47
  %retlist48 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
  %l50 = load %list_item**, %list_item*** %l1, align 8
  %ilist51 = load %list_item*, %list_item** %l50, align 8
  %i52 = load i32, i32* %i, align 4
  %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
```

```
  %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
    _result53, i32 0, i32 0
  %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
  %cast_data_ptr56 = bitcast i8* %data_ptr55 to double*
  %data57 = load double, double* %cast_data_ptr56, align 8
  %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
  %list_ptr_ptr59 = call %list_item** @insert_float(%list_item** %retlist48,
     double %data57, i32 %length58)
  store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
  %i60 = load i32, i32* %i, align 4
  %tmp61 = add i32 %i60, 1
  store i32 %tmp61, i32* %i, align 4
  br label %while31

then45:                                           ; preds = %while_body33
  %i46 = load i32, i32* %i, align 4
  store i32 %i46, i32* %remove_index, align 4
  br label %merge32

else47:                                           ; preds = %while_body33
  br label %merge44

merge68:                                          ; preds = %else108, %
    merge71
  br label %merge

then69:                                           ; preds = %merge32
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while70

while70:                                          ; preds = %while_body72, %
    then69
  %for_index98 = load i32, i32* %for_index, align 4
  %remove_index99 = load i32, i32* %remove_index, align 4
  %tmp100 = add i32 %remove_index99, 1
  %len101 = load i32, i32* %len, align 4
  %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %head_ptr_ptr103 = bitcast i8* %malloccall102 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr103, align 8
  %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
  %ilist105 = load %list_item*, %list_item** %range_list104, align 8
  %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
  %tmp107 = icmp slt i32 %for_index98, %length106
  br i1 %tmp107, label %while_body72, label %merge71

merge71:                                          ; preds = %while70
  br label %merge68

while_body72:                                     ; preds = %while70
  %remove_index73 = load i32, i32* %remove_index, align 4
  %tmp74 = add i32 %remove_index73, 1
  %len75 = load i32, i32* %len, align 4
  %malloccall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall76 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
```

```
  %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist77 = load %list_item*, %list_item** %range_list, align 8
  %for_index78 = load i32, i32* %for_index, align 4
  %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
    for_index78)
  %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
  %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
  %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
  %data83 = load i32, i32* %cast_data_ptr82, align 4
  store i32 %data83, i32* %index, align 4
  %for_index84 = load i32, i32* %for_index, align 4
  %tmp85 = add i32 %for_index84, 1
  store i32 %tmp85, i32* %for_index, align 4
  %retlist86 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
  %l88 = load %list_item**, %list_item*** %l1, align 8
  %ilist89 = load %list_item*, %list_item** %l88, align 8
  %index90 = load i32, i32* %index, align 4
  %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
  %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
  %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
  %cast_data_ptr94 = bitcast i8* %data_ptr93 to double*
  %data95 = load double, double* %cast_data_ptr94, align 8
  %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
  %list_ptr_ptr97 = call %list_item** @insert_float(%list_item** %retlist86,
     double %data95, i32 %length96)
  store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
  br label %while70

else108:                                          ; preds = %merge32
  br label %merge68
}

define %list_item** @remove_int_int_int_bool(%list_item** %l, i32 %elem, i1
    %all) {
entry:
  %index = alloca i32, align 4
  %for_index = alloca i32, align 4
  %remove_index = alloca i32, align 4
  %len = alloca i32, align 4
  %i = alloca i32, align 4
  %retlist = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %elem2 = alloca i32, align 4
  store i32 %elem, i32* %elem2, align 4
  %all3 = alloca i1, align 1
  store i1 %all, i1* %all3, align 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %retlist, align 8
  store i32 0, i32* %i, align 4
  %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
1635   br label %while
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
1644   br label %merge
1645 
1646 while_body:                                       ; preds = %while
1647   %l7 = load %list_item**, %list_item*** %l1, align 8
1648   %ilist8 = load %list_item*, %list_item** %l7, align 8
1649   %i9 = load i32, i32* %i, align 4
1650   %_result = call %list_item* @list_access(%list_item* %ilist8, i32 %i9)
1651   %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
       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
1660   %retlist15 = load %list_item**, %list_item*** %retlist, align 8
1661   %list_ptr = load %list_item*, %list_item** %retlist15, align 8
1662   %l16 = load %list_item**, %list_item*** %l1, align 8
1663   %ilist17 = load %list_item*, %list_item** %l16, align 8
1664   %i18 = load i32, i32* %i, align 4
1665   %_result19 = call %list_item* @list_access(%list_item* %ilist17, i32 %i18)
1666   %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
       _result19, i32 0, i32 0
1667   %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
1668   %cast_data_ptr22 = bitcast i8* %data_ptr21 to i32*
1669   %data23 = load i32, i32* %cast_data_ptr22, align 4
1670   %length24 = call i32 @list_length(%list_item* %list_ptr, i32 0)
1671   %list_ptr_ptr = call %list_item** @insert_int(%list_item** %retlist15, i32
        %data23, i32 %length24)
1672   store %list_item** %list_ptr_ptr, %list_item*** %retlist, align 8
1673   %i25 = load i32, i32* %i, align 4
1674   %tmp26 = add i32 %i25, 1
1675   store i32 %tmp26, i32* %i, align 4
1676   br label %while
```

```
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, %
       else30
1691   %i62 = load i32, i32* %i, align 4
1692   %len63 = load i32, i32* %len, align 4
1693   %tmp64 = icmp slt i32 %i62, %len63
1694   br i1 %tmp64, label %while_body33, label %merge32
1695
1696 merge32:                                          ; preds = %while31, %
       then45
1697   %i65 = load i32, i32* %i, align 4
1698   %len66 = load i32, i32* %len, align 4
1699   %tmp67 = icmp ne i32 %i65, %len66
1700   br i1 %tmp67, label %then69, label %else108
1701
1702 while_body33:                                     ; preds = %while31
1703   %l34 = load %list_item**, %list_item*** %l1, align 8
1704   %ilist35 = load %list_item*, %list_item** %l34, align 8
1705   %i36 = load i32, i32* %i, align 4
1706   %_result37 = call %list_item* @list_access(%list_item* %ilist35, i32 %i36)
1707   %data_ptr_ptr38 = getelementptr inbounds %list_item, %list_item* %
       _result37, i32 0, i32 0
1708   %data_ptr39 = load i8*, i8** %data_ptr_ptr38, align 8
1709   %cast_data_ptr40 = bitcast i8* %data_ptr39 to i32*
1710   %data41 = load i32, i32* %cast_data_ptr40, align 4
1711   %elem42 = load i32, i32* %elem2, align 4
1712   %tmp43 = icmp eq i32 %data41, %elem42
1713   br i1 %tmp43, label %then45, label %else47
1714
1715 merge44:                                          ; preds = %else47
1716   %retlist48 = load %list_item**, %list_item*** %retlist, align 8
1717   %list_ptr49 = load %list_item*, %list_item** %retlist48, align 8
1718   %l50 = load %list_item**, %list_item*** %l1, align 8
1719   %ilist51 = load %list_item*, %list_item** %l50, align 8
1720   %i52 = load i32, i32* %i, align 4
1721   %_result53 = call %list_item* @list_access(%list_item* %ilist51, i32 %i52)
1722   %data_ptr_ptr54 = getelementptr inbounds %list_item, %list_item* %
       _result53, i32 0, i32 0
1723   %data_ptr55 = load i8*, i8** %data_ptr_ptr54, align 8
1724   %cast_data_ptr56 = bitcast i8* %data_ptr55 to i32*
1725   %data57 = load i32, i32* %cast_data_ptr56, align 4
1726   %length58 = call i32 @list_length(%list_item* %list_ptr49, i32 0)
1727   %list_ptr_ptr59 = call %list_item** @insert_int(%list_item** %retlist48,
       i32 %data57, i32 %length58)
1728   store %list_item** %list_ptr_ptr59, %list_item*** %retlist, align 8
1729   %i60 = load i32, i32* %i, align 4
1730   %tmp61 = add i32 %i60, 1
```

```
  store i32 %tmp61, i32* %i, align 4
  br label %while31

then45:                                           ; preds = %while_body33
  %i46 = load i32, i32* %i, align 4
  store i32 %i46, i32* %remove_index, align 4
  br label %merge32

else47:                                           ; preds = %while_body33
  br label %merge44

merge68:                                          ; preds = %else108, %
    merge71
  br label %merge

then69:                                           ; preds = %merge32
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  br label %while70

while70:                                          ; preds = %while_body72, %
    then69
  %for_index98 = load i32, i32* %for_index, align 4
  %remove_index99 = load i32, i32* %remove_index, align 4
  %tmp100 = add i32 %remove_index99, 1
  %len101 = load i32, i32* %len, align 4
  %malloccall102 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (
    i1*, i1** null, i32 1) to i32))
  %head_ptr_ptr103 = bitcast i8* %malloccall102 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr103, align 8
  %range_list104 = call %list_item** @range_function(i32 %tmp100, i32 %
    len101, %list_item** %head_ptr_ptr103, i32 0)
  %ilist105 = load %list_item*, %list_item** %range_list104, align 8
  %length106 = call i32 @list_length(%list_item* %ilist105, i32 0)
  %tmp107 = icmp slt i32 %for_index98, %length106
  br i1 %tmp107, label %while_body72, label %merge71

merge71:                                          ; preds = %while70
  br label %merge68

while_body72:                                     ; preds = %while70
  %remove_index73 = load i32, i32* %remove_index, align 4
  %tmp74 = add i32 %remove_index73, 1
  %len75 = load i32, i32* %len, align 4
  %malloccall76 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %head_ptr_ptr = bitcast i8* %malloccall76 to %list_item**
  store %list_item* null, %list_item** %head_ptr_ptr, align 8
  %range_list = call %list_item** @range_function(i32 %tmp74, i32 %len75, %
    list_item** %head_ptr_ptr, i32 0)
  %ilist77 = load %list_item*, %list_item** %range_list, align 8
  %for_index78 = load i32, i32* %for_index, align 4
  %_result79 = call %list_item* @list_access(%list_item* %ilist77, i32 %
    for_index78)
  %data_ptr_ptr80 = getelementptr inbounds %list_item, %list_item* %
    _result79, i32 0, i32 0
  %data_ptr81 = load i8*, i8** %data_ptr_ptr80, align 8
  %cast_data_ptr82 = bitcast i8* %data_ptr81 to i32*
  %data83 = load i32, i32* %cast_data_ptr82, align 4
```

```
  store i32 %data83, i32* %index, align 4
  %for_index84 = load i32, i32* %for_index, align 4
  %tmp85 = add i32 %for_index84, 1
  store i32 %tmp85, i32* %for_index, align 4
  %retlist86 = load %list_item**, %list_item*** %retlist, align 8
  %list_ptr87 = load %list_item*, %list_item** %retlist86, align 8
  %l88 = load %list_item**, %list_item*** %l1, align 8
  %ilist89 = load %list_item*, %list_item** %l88, align 8
  %index90 = load i32, i32* %index, align 4
  %_result91 = call %list_item* @list_access(%list_item* %ilist89, i32 %
    index90)
  %data_ptr_ptr92 = getelementptr inbounds %list_item, %list_item* %
    _result91, i32 0, i32 0
  %data_ptr93 = load i8*, i8** %data_ptr_ptr92, align 8
  %cast_data_ptr94 = bitcast i8* %data_ptr93 to i32*
  %data95 = load i32, i32* %cast_data_ptr94, align 4
  %length96 = call i32 @list_length(%list_item* %list_ptr87, i32 0)
  %list_ptr_ptr97 = call %list_item** @insert_int(%list_item** %retlist86,
    i32 %data95, i32 %length96)
  store %list_item** %list_ptr_ptr97, %list_item*** %retlist, align 8
  br label %while70

else108:                                          ; preds = %merge32
  br label %merge68
}

define i1 @contains_float_float(%list_item** %l, %list_item** %items) {
entry:
  %r = alloca i1, align 1
  %for_index42 = alloca i32, align 4
  %ref = alloca double, align 8
  %for_index7 = alloca i32, align 4
  %item = alloca double, align 8
  %for_index = alloca i32, align 4
  %res = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %items2 = alloca %list_item**, align 8
  store %list_item** %items, %list_item*** %items2, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %res, align 8
  store i32 0, i32* %for_index, align 4
  store double 0.000000e+00, double* %item, align 8
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index37 = load i32, i32* %for_index, align 4
  %items38 = load %list_item**, %list_item*** %items2, align 8
  %ilist39 = load %list_item*, %list_item** %items38, align 8
  %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
  %tmp41 = icmp slt i32 %for_index37, %length40
  br i1 %tmp41, label %while_body, label %merge

merge:                                            ; preds = %while
  store i32 0, i32* %for_index42, align 4
  store i1 false, i1* %r, align 1
```

```
  br label %while43

while_body:                                       ; preds = %while
  %items3 = load %list_item**, %list_item*** %items2, align 8
  %ilist = load %list_item*, %list_item** %items3, align 8
  %for_index4 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to double*
  %data = load double, double* %cast_data_ptr, align 8
  store double %data, double* %item, align 8
  %for_index5 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index5, 1
  store i32 %tmp, i32* %for_index, align 4
  %res6 = load %list_item**, %list_item*** %res, align 8
  %list_ptr = load %list_item*, %list_item** %res6, align 8
  %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
  store i32 0, i32* %for_index7, align 4
  store double 0.000000e+00, double* %ref, align 8
  br label %while8

while8:                                           ; preds = %merge24, %
    while_body
  %for_index32 = load i32, i32* %for_index7, align 4
  %l33 = load %list_item**, %list_item*** %l1, align 8
  %ilist34 = load %list_item*, %list_item** %l33, align 8
  %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
  %tmp36 = icmp slt i32 %for_index32, %length35
  br i1 %tmp36, label %while_body10, label %merge9

merge9:                                           ; preds = %while8
  br label %while

while_body10:                                     ; preds = %while8
  %l11 = load %list_item**, %list_item*** %l1, align 8
  %ilist12 = load %list_item*, %list_item** %l11, align 8
  %for_index13 = load i32, i32* %for_index7, align 4
  %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
  %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
  %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
  %cast_data_ptr17 = bitcast i8* %data_ptr16 to double*
  %data18 = load double, double* %cast_data_ptr17, align 8
  store double %data18, double* %ref, align 8
  %for_index19 = load i32, i32* %for_index7, align 4
  %tmp20 = add i32 %for_index19, 1
  store i32 %tmp20, i32* %for_index7, align 4
  %ref21 = load double, double* %ref, align 8
  %item22 = load double, double* %item, align 8
  %tmp23 = fcmp oeq double %ref21, %item22
  br i1 %tmp23, label %then, label %else

```

```
merge24:                                          ; preds = %else, %then
  br label %while8

then:                                             ; preds = %while_body10
  %res25 = load %list_item**, %list_item*** %res, align 8
  %ilist26 = load %list_item*, %list_item** %res25, align 8
  %res27 = load %list_item**, %list_item*** %res, align 8
  %ilist28 = load %list_item*, %list_item** %res27, align 8
  %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
  %tmp30 = sub i32 %length29, 1
  %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
  %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
  %malloccall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
     i1* null, i32 1) to i32))
  %copy_ptr = bitcast i8* %malloccall31 to i1*
  store i1 true, i1* %copy_ptr, align 1
  %ccopy = bitcast i1* %copy_ptr to i8*
  store i8* %ccopy, i8** %data_ptpt, align 8
  br label %merge24

else:                                             ; preds = %while_body10
  br label %merge24

while43:                                          ; preds = %merge58, %merge
  %for_index61 = load i32, i32* %for_index42, align 4
  %res62 = load %list_item**, %list_item*** %res, align 8
  %ilist63 = load %list_item*, %list_item** %res62, align 8
  %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
  %tmp65 = icmp slt i32 %for_index61, %length64
  br i1 %tmp65, label %while_body45, label %merge44

merge44:                                          ; preds = %while43
  ret i1 true

while_body45:                                     ; preds = %while43
  %res46 = load %list_item**, %list_item*** %res, align 8
  %ilist47 = load %list_item*, %list_item** %res46, align 8
  %for_index48 = load i32, i32* %for_index42, align 4
  %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
    for_index48)
  %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
    _result49, i32 0, i32 0
  %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
  %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
  %data53 = load i1, i1* %cast_data_ptr52, align 1
  store i1 %data53, i1* %r, align 1
  %for_index54 = load i32, i32* %for_index42, align 4
  %tmp55 = add i32 %for_index54, 1
  store i32 %tmp55, i32* %for_index42, align 4
  %r56 = load i1, i1* %r, align 1
  %tmp57 = xor i1 %r56, true
  br i1 %tmp57, label %then59, label %else60

merge58:                                          ; preds = %else60
  br label %while43

then59:                                           ; preds = %while_body45
  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   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))
1964   %list = bitcast i8* %malloccall to %list_item**
1965   store %list_item* null, %list_item** %list, align 8
1966   store %list_item** %list, %list_item*** %res, align 8
1967   store i32 0, i32* %for_index, align 4
1968   store i1 false, i1* %item, align 1
1969   br label %while
1970
1971 while:                                            ; preds = %merge9, %entry
1972   %for_index37 = load i32, i32* %for_index, align 4
1973   %items38 = load %list_item**, %list_item*** %items2, align 8
1974   %ilist39 = load %list_item*, %list_item** %items38, align 8
1975   %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
1976   %tmp41 = icmp slt i32 %for_index37, %length40
1977   br i1 %tmp41, label %while_body, label %merge
1978
1979 merge:                                            ; preds = %while
1980   store i32 0, i32* %for_index42, align 4
1981   store i1 false, i1* %r, align 1
1982   br label %while43
1983
1984 while_body:                                       ; preds = %while
1985   %items3 = load %list_item**, %list_item*** %items2, align 8
1986   %ilist = load %list_item*, %list_item** %items3, align 8
1987   %for_index4 = load i32, i32* %for_index, align 4
1988   %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
        for_index4)
1989   %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
        i32 0, i32 0
1990   %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
1991   %cast_data_ptr = bitcast i8* %data_ptr to i1*
1992   %data = load i1, i1* %cast_data_ptr, align 1
1993   store i1 %data, i1* %item, align 1
1994   %for_index5 = load i32, i32* %for_index, align 4
1995   %tmp = add i32 %for_index5, 1
1996   store i32 %tmp, i32* %for_index, align 4
1997   %res6 = load %list_item**, %list_item*** %res, align 8
1998   %list_ptr = load %list_item*, %list_item** %res6, align 8
1999   %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
```

```
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
  store i32 0, i32* %for_index7, align 4
  store i1 false, i1* %ref, align 1
  br label %while8

while8:                                           ; preds = %merge24, %
    while_body
  %for_index32 = load i32, i32* %for_index7, align 4
  %l33 = load %list_item**, %list_item*** %l1, align 8
  %ilist34 = load %list_item*, %list_item** %l33, align 8
  %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
  %tmp36 = icmp slt i32 %for_index32, %length35
  br i1 %tmp36, label %while_body10, label %merge9

merge9:                                           ; preds = %while8
  br label %while

while_body10:                                     ; preds = %while8
  %l11 = load %list_item**, %list_item*** %l1, align 8
  %ilist12 = load %list_item*, %list_item** %l11, align 8
  %for_index13 = load i32, i32* %for_index7, align 4
  %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
  %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
  %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
  %cast_data_ptr17 = bitcast i8* %data_ptr16 to i1*
  %data18 = load i1, i1* %cast_data_ptr17, align 1
  store i1 %data18, i1* %ref, align 1
  %for_index19 = load i32, i32* %for_index7, align 4
  %tmp20 = add i32 %for_index19, 1
  store i32 %tmp20, i32* %for_index7, align 4
  %ref21 = load i1, i1* %ref, align 1
  %item22 = load i1, i1* %item, align 1
  %tmp23 = icmp eq i1 %ref21, %item22
  br i1 %tmp23, label %then, label %else

merge24:                                          ; preds = %else, %then
  br label %while8

then:                                             ; preds = %while_body10
  %res25 = load %list_item**, %list_item*** %res, align 8
  %ilist26 = load %list_item*, %list_item** %res25, align 8
  %res27 = load %list_item**, %list_item*** %res, align 8
  %ilist28 = load %list_item*, %list_item** %res27, align 8
  %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
  %tmp30 = sub i32 %length29, 1
  %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
  %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
  %malloccall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
     i1* null, i32 1) to i32))
  %copy_ptr = bitcast i8* %malloccall31 to i1*
  store i1 true, i1* %copy_ptr, align 1
  %ccopy = bitcast i1* %copy_ptr to i8*
  store i8* %ccopy, i8** %data_ptpt, align 8
  br label %merge24
```

```
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   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))
```

```
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %res, align 8
  store i32 0, i32* %for_index, align 4
  store i8 0, i8* %item, align 1
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index36 = load i32, i32* %for_index, align 4
  %items37 = load %list_item**, %list_item*** %items2, align 8
  %ilist38 = load %list_item*, %list_item** %items37, align 8
  %length39 = call i32 @list_length(%list_item* %ilist38, i32 0)
  %tmp40 = icmp slt i32 %for_index36, %length39
  br i1 %tmp40, label %while_body, label %merge

merge:                                            ; preds = %while
  store i32 0, i32* %for_index41, align 4
  store i1 false, i1* %r, align 1
  br label %while42

while_body:                                       ; preds = %while
  %items3 = load %list_item**, %list_item*** %items2, align 8
  %ilist = load %list_item*, %list_item** %items3, align 8
  %for_index4 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %data = load i8, i8* %data_ptr, align 1
  store i8 %data, i8* %item, align 1
  %for_index5 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index5, 1
  store i32 %tmp, i32* %for_index, align 4
  %res6 = load %list_item**, %list_item*** %res, align 8
  %list_ptr = load %list_item*, %list_item** %res6, align 8
  %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
  store i32 0, i32* %for_index7, align 4
  store i8 0, i8* %ref, align 1
  br label %while8

while8:                                           ; preds = %merge23, %
    while_body
  %for_index31 = load i32, i32* %for_index7, align 4
  %l32 = load %list_item**, %list_item*** %l1, align 8
  %ilist33 = load %list_item*, %list_item** %l32, align 8
  %length34 = call i32 @list_length(%list_item* %ilist33, i32 0)
  %tmp35 = icmp slt i32 %for_index31, %length34
  br i1 %tmp35, label %while_body10, label %merge9

merge9:                                           ; preds = %while8
  br label %while

while_body10:                                     ; preds = %while8
  %l11 = load %list_item**, %list_item*** %l1, align 8
  %ilist12 = load %list_item*, %list_item** %l11, align 8
```

```
  %for_index13 = load i32, i32* %for_index7, align 4
  %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
  %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
  %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
  %data17 = load i8, i8* %data_ptr16, align 1
  store i8 %data17, i8* %ref, align 1
  %for_index18 = load i32, i32* %for_index7, align 4
  %tmp19 = add i32 %for_index18, 1
  store i32 %tmp19, i32* %for_index7, align 4
  %ref20 = load i8, i8* %ref, align 1
  %item21 = load i8, i8* %item, align 1
  %tmp22 = icmp eq i8 %ref20, %item21
  br i1 %tmp22, label %then, label %else

merge23:                                          ; preds = %else, %then
  br label %while8

then:                                             ; preds = %while_body10
  %res24 = load %list_item**, %list_item*** %res, align 8
  %ilist25 = load %list_item*, %list_item** %res24, align 8
  %res26 = load %list_item**, %list_item*** %res, align 8
  %ilist27 = load %list_item*, %list_item** %res26, align 8
  %length28 = call i32 @list_length(%list_item* %ilist27, i32 0)
  %tmp29 = sub i32 %length28, 1
  %result = call %list_item* @list_access(%list_item* %ilist25, i32 %tmp29)
  %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
  %malloccall30 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
     i1* null, i32 1) to i32))
  %copy_ptr = bitcast i8* %malloccall30 to i1*
  store i1 true, i1* %copy_ptr, align 1
  %ccopy = bitcast i1* %copy_ptr to i8*
  store i8* %ccopy, i8** %data_ptpt, align 8
  br label %merge23

else:                                             ; preds = %while_body10
  br label %merge23

while42:                                          ; preds = %merge56, %merge
  %for_index59 = load i32, i32* %for_index41, align 4
  %res60 = load %list_item**, %list_item*** %res, align 8
  %ilist61 = load %list_item*, %list_item** %res60, align 8
  %length62 = call i32 @list_length(%list_item* %ilist61, i32 0)
  %tmp63 = icmp slt i32 %for_index59, %length62
  br i1 %tmp63, label %while_body44, label %merge43

merge43:                                          ; preds = %while42
  ret i1 true

while_body44:                                     ; preds = %while42
  %res45 = load %list_item**, %list_item*** %res, align 8
  %ilist46 = load %list_item*, %list_item** %res45, align 8
  %for_index47 = load i32, i32* %for_index41, align 4
  %_result48 = call %list_item* @list_access(%list_item* %ilist46, i32 %
    for_index47)
  %data_ptr_ptr49 = getelementptr inbounds %list_item, %list_item* %
    _result48, i32 0, i32 0
```

```
  %data_ptr50 = load i8*, i8** %data_ptr_ptr49, align 8
  %cast_data_ptr = bitcast i8* %data_ptr50 to i1*
  %data51 = load i1, i1* %cast_data_ptr, align 1
  store i1 %data51, i1* %r, align 1
  %for_index52 = load i32, i32* %for_index41, align 4
  %tmp53 = add i32 %for_index52, 1
  store i32 %tmp53, i32* %for_index41, align 4
  %r54 = load i1, i1* %r, align 1
  %tmp55 = xor i1 %r54, true
  br i1 %tmp55, label %then57, label %else58

merge56:                                          ; preds = %else58
  br label %while42

then57:                                           ; preds = %while_body44
  ret i1 false

else58:                                           ; preds = %while_body44
  br label %merge56
}

define i1 @contains_string_string(%list_item** %l, %list_item** %items) {
entry:
  %r = alloca i1, align 1
  %for_index41 = alloca i32, align 4
  %ref = alloca i8*, align 8
  %for_index7 = alloca i32, align 4
  %item = alloca i8*, align 8
  %for_index = alloca i32, align 4
  %res = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %items2 = alloca %list_item**, align 8
  store %list_item** %items, %list_item*** %items2, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %res, align 8
  store i32 0, i32* %for_index, align 4
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.7, i32 0,
    i32 0), i8** %item, align 8
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index36 = load i32, i32* %for_index, align 4
  %items37 = load %list_item**, %list_item*** %items2, align 8
  %ilist38 = load %list_item*, %list_item** %items37, align 8
  %length39 = call i32 @list_length(%list_item* %ilist38, i32 0)
  %tmp40 = icmp slt i32 %for_index36, %length39
  br i1 %tmp40, label %while_body, label %merge

merge:                                            ; preds = %while
  store i32 0, i32* %for_index41, align 4
  store i1 false, i1* %r, align 1
  br label %while42

while_body:                                       ; preds = %while
  %items3 = load %list_item**, %list_item*** %items2, align 8
```

```
  %ilist = load %list_item*, %list_item** %items3, align 8
  %for_index4 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i8**
  %data = load i8*, i8** %cast_data_ptr, align 8
  store i8* %data, i8** %item, align 8
  %for_index5 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index5, 1
  store i32 %tmp, i32* %for_index, align 4
  %res6 = load %list_item**, %list_item*** %res, align 8
  %list_ptr = load %list_item*, %list_item** %res6, align 8
  %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
  store i32 0, i32* %for_index7, align 4
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.8, i32 0,
    i32 0), i8** %ref, align 8
  br label %while8

while8:                                           ; preds = %merge23, %
    while_body
  %for_index31 = load i32, i32* %for_index7, align 4
  %l32 = load %list_item**, %list_item*** %l1, align 8
  %ilist33 = load %list_item*, %list_item** %l32, align 8
  %length34 = call i32 @list_length(%list_item* %ilist33, i32 0)
  %tmp35 = icmp slt i32 %for_index31, %length34
  br i1 %tmp35, label %while_body10, label %merge9

merge9:                                           ; preds = %while8
  br label %while

while_body10:                                     ; preds = %while8
  %l11 = load %list_item**, %list_item*** %l1, align 8
  %ilist12 = load %list_item*, %list_item** %l11, align 8
  %for_index13 = load i32, i32* %for_index7, align 4
  %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
  %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
  %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
  %cast_data_ptr17 = bitcast i8* %data_ptr16 to i8**
  %data18 = load i8*, i8** %cast_data_ptr17, align 8
  store i8* %data18, i8** %ref, align 8
  %for_index19 = load i32, i32* %for_index7, align 4
  %tmp20 = add i32 %for_index19, 1
  store i32 %tmp20, i32* %for_index7, align 4
  %ref21 = load i8*, i8** %ref, align 8
  %item22 = load i8*, i8** %item, align 8
  %strcmp_eq = call i1 @strcmp_function(i8* %ref21, i8* %item22)
  br i1 %strcmp_eq, label %then, label %else

merge23:                                          ; preds = %else, %then
  br label %while8

```

```
then:                                             ; preds = %while_body10
  %res24 = load %list_item**, %list_item*** %res, align 8
  %ilist25 = load %list_item*, %list_item** %res24, align 8
  %res26 = load %list_item**, %list_item*** %res, align 8
  %ilist27 = load %list_item*, %list_item** %res26, align 8
  %length28 = call i32 @list_length(%list_item* %ilist27, i32 0)
  %tmp29 = sub i32 %length28, 1
  %result = call %list_item* @list_access(%list_item* %ilist25, i32 %tmp29)
  %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
  %malloccall30 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
     i1* null, i32 1) to i32))
  %copy_ptr = bitcast i8* %malloccall30 to i1*
  store i1 true, i1* %copy_ptr, align 1
  %ccopy = bitcast i1* %copy_ptr to i8*
  store i8* %ccopy, i8** %data_ptpt, align 8
  br label %merge23

else:                                             ; preds = %while_body10
  br label %merge23

while42:                                          ; preds = %merge57, %merge
  %for_index60 = load i32, i32* %for_index41, align 4
  %res61 = load %list_item**, %list_item*** %res, align 8
  %ilist62 = load %list_item*, %list_item** %res61, align 8
  %length63 = call i32 @list_length(%list_item* %ilist62, i32 0)
  %tmp64 = icmp slt i32 %for_index60, %length63
  br i1 %tmp64, label %while_body44, label %merge43

merge43:                                          ; preds = %while42
  ret i1 true

while_body44:                                     ; preds = %while42
  %res45 = load %list_item**, %list_item*** %res, align 8
  %ilist46 = load %list_item*, %list_item** %res45, align 8
  %for_index47 = load i32, i32* %for_index41, align 4
  %_result48 = call %list_item* @list_access(%list_item* %ilist46, i32 %
    for_index47)
  %data_ptr_ptr49 = getelementptr inbounds %list_item, %list_item* %
    _result48, i32 0, i32 0
  %data_ptr50 = load i8*, i8** %data_ptr_ptr49, align 8
  %cast_data_ptr51 = bitcast i8* %data_ptr50 to i1*
  %data52 = load i1, i1* %cast_data_ptr51, align 1
  store i1 %data52, i1* %r, align 1
  %for_index53 = load i32, i32* %for_index41, align 4
  %tmp54 = add i32 %for_index53, 1
  store i32 %tmp54, i32* %for_index41, align 4
  %r55 = load i1, i1* %r, align 1
  %tmp56 = xor i1 %r55, true
  br i1 %tmp56, label %then58, label %else59

merge57:                                          ; preds = %else59
  br label %while42

then58:                                           ; preds = %while_body44
  ret i1 false

else59:                                           ; preds = %while_body44
  br label %merge57
```

```
}

define i1 @contains_int_int(%list_item** %l, %list_item** %items) {
entry:
  %r = alloca i1, align 1
  %for_index42 = alloca i32, align 4
  %ref = alloca i32, align 4
  %for_index7 = alloca i32, align 4
  %item = alloca i32, align 4
  %for_index = alloca i32, align 4
  %res = alloca %list_item**, align 8
  %l1 = alloca %list_item**, align 8
  store %list_item** %l, %list_item*** %l1, align 8
  %items2 = alloca %list_item**, align 8
  store %list_item** %items, %list_item*** %items2, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %res, align 8
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %item, align 4
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index37 = load i32, i32* %for_index, align 4
  %items38 = load %list_item**, %list_item*** %items2, align 8
  %ilist39 = load %list_item*, %list_item** %items38, align 8
  %length40 = call i32 @list_length(%list_item* %ilist39, i32 0)
  %tmp41 = icmp slt i32 %for_index37, %length40
  br i1 %tmp41, label %while_body, label %merge

merge:                                            ; preds = %while
  store i32 0, i32* %for_index42, align 4
  store i1 false, i1* %r, align 1
  br label %while43

while_body:                                       ; preds = %while
  %items3 = load %list_item**, %list_item*** %items2, align 8
  %ilist = load %list_item*, %list_item** %items3, align 8
  %for_index4 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index4)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i32*
  %data = load i32, i32* %cast_data_ptr, align 4
  store i32 %data, i32* %item, align 4
  %for_index5 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index5, 1
  store i32 %tmp, i32* %for_index, align 4
  %res6 = load %list_item**, %list_item*** %res, align 8
  %list_ptr = load %list_item*, %list_item** %res6, align 8
  %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_bool(%list_item** %res6, i1
    false, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %res, align 8
  store i32 0, i32* %for_index7, align 4
```

```
  store i32 0, i32* %ref, align 4
  br label %while8

while8:                                           ; preds = %merge24, %
    while_body
  %for_index32 = load i32, i32* %for_index7, align 4
  %l33 = load %list_item**, %list_item*** %l1, align 8
  %ilist34 = load %list_item*, %list_item** %l33, align 8
  %length35 = call i32 @list_length(%list_item* %ilist34, i32 0)
  %tmp36 = icmp slt i32 %for_index32, %length35
  br i1 %tmp36, label %while_body10, label %merge9

merge9:                                           ; preds = %while8
  br label %while

while_body10:                                     ; preds = %while8
  %l11 = load %list_item**, %list_item*** %l1, align 8
  %ilist12 = load %list_item*, %list_item** %l11, align 8
  %for_index13 = load i32, i32* %for_index7, align 4
  %_result14 = call %list_item* @list_access(%list_item* %ilist12, i32 %
    for_index13)
  %data_ptr_ptr15 = getelementptr inbounds %list_item, %list_item* %
    _result14, i32 0, i32 0
  %data_ptr16 = load i8*, i8** %data_ptr_ptr15, align 8
  %cast_data_ptr17 = bitcast i8* %data_ptr16 to i32*
  %data18 = load i32, i32* %cast_data_ptr17, align 4
  store i32 %data18, i32* %ref, align 4
  %for_index19 = load i32, i32* %for_index7, align 4
  %tmp20 = add i32 %for_index19, 1
  store i32 %tmp20, i32* %for_index7, align 4
  %ref21 = load i32, i32* %ref, align 4
  %item22 = load i32, i32* %item, align 4
  %tmp23 = icmp eq i32 %ref21, %item22
  br i1 %tmp23, label %then, label %else

merge24:                                          ; preds = %else, %then
  br label %while8

then:                                             ; preds = %while_body10
  %res25 = load %list_item**, %list_item*** %res, align 8
  %ilist26 = load %list_item*, %list_item** %res25, align 8
  %res27 = load %list_item**, %list_item*** %res, align 8
  %ilist28 = load %list_item*, %list_item** %res27, align 8
  %length29 = call i32 @list_length(%list_item* %ilist28, i32 0)
  %tmp30 = sub i32 %length29, 1
  %result = call %list_item* @list_access(%list_item* %ilist26, i32 %tmp30)
  %data_ptpt = getelementptr inbounds %list_item, %list_item* %result, i32
    0, i32 0
  %malloccall31 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
     i1* null, i32 1) to i32))
  %copy_ptr = bitcast i8* %malloccall31 to i1*
  store i1 true, i1* %copy_ptr, align 1
  %ccopy = bitcast i1* %copy_ptr to i8*
  store i8* %ccopy, i8** %data_ptpt, align 8
  br label %merge24

else:                                             ; preds = %while_body10
  br label %merge24
```

```
while43:                                          ; preds = %merge58, %merge
  %for_index61 = load i32, i32* %for_index42, align 4
  %res62 = load %list_item**, %list_item*** %res, align 8
  %ilist63 = load %list_item*, %list_item** %res62, align 8
  %length64 = call i32 @list_length(%list_item* %ilist63, i32 0)
  %tmp65 = icmp slt i32 %for_index61, %length64
  br i1 %tmp65, label %while_body45, label %merge44

merge44:                                          ; preds = %while43
  ret i1 true

while_body45:                                     ; preds = %while43
  %res46 = load %list_item**, %list_item*** %res, align 8
  %ilist47 = load %list_item*, %list_item** %res46, align 8
  %for_index48 = load i32, i32* %for_index42, align 4
  %_result49 = call %list_item* @list_access(%list_item* %ilist47, i32 %
    for_index48)
  %data_ptr_ptr50 = getelementptr inbounds %list_item, %list_item* %
    _result49, i32 0, i32 0
  %data_ptr51 = load i8*, i8** %data_ptr_ptr50, align 8
  %cast_data_ptr52 = bitcast i8* %data_ptr51 to i1*
  %data53 = load i1, i1* %cast_data_ptr52, align 1
  store i1 %data53, i1* %r, align 1
  %for_index54 = load i32, i32* %for_index42, align 4
  %tmp55 = add i32 %for_index54, 1
  store i32 %tmp55, i32* %for_index42, align 4
  %r56 = load i1, i1* %r, align 1
  %tmp57 = xor i1 %r56, true
  br i1 %tmp57, label %then59, label %else60

merge58:                                          ; preds = %else60
  br label %while43

then59:                                           ; preds = %while_body45
  ret i1 false

else60:                                           ; preds = %while_body45
  br label %merge58
}

define i8* @join_string_string(%list_item** %text_list, i8* %connector) {
entry:
  %index = alloca i32, align 4
  %for_index = alloca i32, align 4
  %list_length = alloca i32, align 4
  %res = alloca i8*, align 8
  %text_list1 = alloca %list_item**, align 8
  store %list_item** %text_list, %list_item*** %text_list1, align 8
  %connector2 = alloca i8*, align 8
  store i8* %connector, i8** %connector2, align 8
  store i8* getelementptr inbounds ([1 x i8], [1 x i8]* @string.9, i32 0,
    i32 0), i8** %res, align 8
  %text_list3 = load %list_item**, %list_item*** %text_list1, align 8
  %ilist = load %list_item*, %list_item** %text_list3, align 8
  %length = call i32 @list_length(%list_item* %ilist, i32 0)
  store i32 %length, i32* %list_length, align 4
  store i32 0, i32* %for_index, align 4
  store i32 0, i32* %index, align 4
  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   %malloccall34 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
       *, i1** null, i32 1) to i32))
2552   %head_ptr_ptr35 = bitcast i8* %malloccall34 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   %ilist37 = load %list_item*, %list_item** %range_list36, align 8
2556   %length38 = call i32 @list_length(%list_item* %ilist37, 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   %ilist42 = 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* %ilist42, 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   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
        i1** null, i32 1) to i32))
2590   %head_ptr_ptr = bitcast i8* %malloccall 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   %ilist5 = load %list_item*, %list_item** %range_list, align 8
2594   %for_index6 = load i32, i32* %for_index, align 4
```

```
  %_result = call %list_item* @list_access(%list_item* %ilist5, i32 %
    for_index6)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %cast_data_ptr = bitcast i8* %data_ptr to i32*
  %data = load i32, i32* %cast_data_ptr, align 4
  store i32 %data, i32* %index, align 4
  %for_index7 = load i32, i32* %for_index, align 4
  %tmp8 = add i32 %for_index7, 1
  store i32 %tmp8, i32* %for_index, align 4
  %res9 = load i8*, i8** %res, align 8
  %text_list10 = load %list_item**, %list_item*** %text_list1, align 8
  %ilist11 = load %list_item*, %list_item** %text_list10, align 8
  %index12 = load i32, i32* %index, align 4
  %_result13 = call %list_item* @list_access(%list_item* %ilist11, i32 %
    index12)
  %data_ptr_ptr14 = getelementptr inbounds %list_item, %list_item* %
    _result13, i32 0, i32 0
  %data_ptr15 = load i8*, i8** %data_ptr_ptr14, align 8
  %cast_data_ptr16 = bitcast i8* %data_ptr15 to i8**
  %data17 = load i8*, i8** %cast_data_ptr16, align 8
  %length18 = call i32 @string_length(i8* %res9, i32 0)
  %length19 = call i32 @string_length(i8* %data17, i32 0)
  %new_length = add i32 %length18, %length19
  %new_length_nul = add i32 %new_length, 1
  %mallocsize = mul i32 %new_length_nul, ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32)
  %new_string = tail call i8* @malloc(i32 %mallocsize)
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %res9, i32 %
    length18, i1 true)
  %new_spot = getelementptr i8, i8* %new_string, i32 %length18
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot, i8* %data17, i32 %
    length19, i1 true)
  %string_term = getelementptr i8, i8* %new_string, i32 %new_length
  store i8 0, i8* %string_term, align 1
  %connector21 = load i8*, i8** %connector2, align 8
  %length22 = call i32 @string_length(i8* %new_string, i32 0)
  %length23 = call i32 @string_length(i8* %connector21, i32 0)
  %new_length24 = add i32 %length22, %length23
  %new_length_nul25 = add i32 %new_length24, 1
  %mallocsize26 = mul i32 %new_length_nul25, ptrtoint (i8* getelementptr (i8
    , i8* null, i32 1) to i32)
  %new_string28 = tail call i8* @malloc(i32 %mallocsize26)
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string28, i8* %new_string,
    i32 %length22, i1 true)
  %new_spot29 = getelementptr i8, i8* %new_string28, i32 %length22
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_spot29, i8* %connector21,
    i32 %length23, i1 true)
  %string_term30 = getelementptr i8, i8* %new_string28, i32 %new_length24
  store i8 0, i8* %string_term30, align 1
  store i8* %new_string28, i8** %res, align 8
  br label %while
}

define %list_item** @split(i8* %text, i8 %separator) {
entry:
  %right = alloca i32, align 4
  %left = alloca i32, align 4
```

```
  %text_length = alloca i32, align 4
  %result = alloca %list_item**, align 8
  %text1 = alloca i8*, align 8
  store i8* %text, i8** %text1, align 8
  %separator2 = alloca i8, align 1
  store i8 %separator, i8* %separator2, align 1
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %result, align 8
  %text3 = load i8*, i8** %text1, align 8
  %length = call i32 @string_length(i8* %text3, i32 0)
  store i32 %length, i32* %text_length, align 4
  store i32 0, i32* %left, align 4
  store i32 0, i32* %right, align 4
  br label %while

while:                                            ; preds = %merge7, %entry
  %right21 = load i32, i32* %right, align 4
  %text_length22 = load i32, i32* %text_length, align 4
  %tmp23 = icmp slt i32 %right21, %text_length22
  br i1 %tmp23, label %while_body, label %merge

merge:                                            ; preds = %while
  %result24 = load %list_item**, %list_item*** %result, align 8
  %list_ptr25 = load %list_item*, %list_item** %result24, align 8
  %text26 = load i8*, i8** %text1, align 8
  %left27 = load i32, i32* %left, align 4
  %get_char_ptr28 = getelementptr i8, i8* %text26, i32 %left27
  %left29 = load i32, i32* %left, align 4
  %right30 = load i32, i32* %right, align 4
  %subb31 = sub i32 %right30, %left29
  %length_w_nul32 = add i32 %subb31, 1
  %mallocsize33 = mul i32 %length_w_nul32, ptrtoint (i8* getelementptr (i8,
    i8* null, i32 1) to i32)
  %new_string35 = tail call i8* @malloc(i32 %mallocsize33)
  %string_term36 = getelementptr i8, i8* %new_string35, i32 %subb31
  store i8 0, i8* %string_term36, align 1
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string35, i8* %
    get_char_ptr28, i32 %subb31, i1 true)
  %length37 = call i32 @list_length(%list_item* %list_ptr25, i32 0)
  %list_ptr_ptr38 = call %list_item** @insert_string(%list_item** %result24,
     i8* %new_string35, i32 %length37)
  store %list_item** %list_ptr_ptr38, %list_item*** %result, align 8
  %result39 = load %list_item**, %list_item*** %result, align 8
  ret %list_item** %result39

while_body:                                       ; preds = %while
  %text4 = load i8*, i8** %text1, align 8
  %right5 = load i32, i32* %right, align 4
  %get_char_ptr = getelementptr i8, i8* %text4, i32 %right5
  %get_char = load i8, i8* %get_char_ptr, align 1
  %separator6 = load i8, i8* %separator2, align 1
  %tmp = icmp eq i8 %get_char, %separator6
  br i1 %tmp, label %then, label %else

merge7:                                           ; preds = %else, %then
  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
```

```
merge:                                            ; preds = %while
  %res12 = load i8*, i8** %res, align 8
  ret i8* %res12

while_body:                                       ; preds = %while
  %res4 = load i8*, i8** %res, align 8
  %text5 = load i8*, i8** %text1, align 8
  %index6 = load i32, i32* %index, align 4
  %get_char_ptr = getelementptr i8, i8* %text5, i32 %index6
  %get_char = load i8, i8* %get_char_ptr, align 1
  %length7 = call i32 @string_length(i8* %res4, i32 0)
  %new_length = add i32 %length7, 1
  %new_length_nul = add i32 %new_length, 1
  %mallocsize = mul i32 %new_length_nul, ptrtoint (i8* getelementptr (i8, i8
    * null, i32 1) to i32)
  %new_string = tail call i8* @malloc(i32 %mallocsize)
  call void @llvm.memcpy.p0i8.p0i8.i32(i8* %new_string, i8* %res4, i32 %
    length7, i1 true)
  %new_spot = getelementptr i8, i8* %new_string, i32 %length7
  store i8 %get_char, i8* %new_spot, align 1
  %string_term = getelementptr i8, i8* %new_string, i32 %new_length
  store i8 0, i8* %string_term, align 1
  store i8* %new_string, i8** %res, align 8
  %index8 = load i32, i32* %index, align 4
  %tmp9 = sub i32 %index8, 1
  store i32 %tmp9, i32* %index, align 4
  br label %while
}

define i1 @startswith(i8* %text, i8 %s) {
entry:
  %text1 = alloca i8*, align 8
  store i8* %text, i8** %text1, align 8
  %s2 = alloca i8, align 1
  store i8 %s, i8* %s2, align 1
  %s3 = load i8, i8* %s2, align 1
  %text4 = load i8*, i8** %text1, align 8
  %get_char_ptr = getelementptr i8, i8* %text4, i32 0
  %get_char = load i8, i8* %get_char_ptr, align 1
  %tmp = icmp eq i8 %s3, %get_char
  ret i1 %tmp
}

define i1 @endswith(i8* %text, i8 %e) {
entry:
  %string_length = alloca i32, align 4
  %text1 = alloca i8*, align 8
  store i8* %text, i8** %text1, align 8
  %e2 = alloca i8, align 1
  store i8 %e, i8* %e2, align 1
  %text3 = load i8*, i8** %text1, align 8
  %length = call i32 @string_length(i8* %text3, i32 0)
  store i32 %length, i32* %string_length, align 4
  %e4 = load i8, i8* %e2, align 1
  %text5 = load i8*, i8** %text1, align 8
  %string_length6 = load i32, i32* %string_length, align 4
  %tmp = sub i32 %string_length6, 1
  %get_char_ptr = getelementptr i8, i8* %text5, i32 %tmp
  %get_char = load i8, i8* %get_char_ptr, align 1
```

```
  %tmp7 = icmp eq i8 %e4, %get_char
  ret i1 %tmp7
}

define %list_item** @string_to_list(i8* %text) {
entry:
  %c = alloca i8, align 1
  %for_index = alloca i32, align 4
  %result = alloca %list_item**, align 8
  %text1 = alloca i8*, align 8
  store i8* %text, i8** %text1, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %list = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %list, align 8
  store %list_item** %list, %list_item*** %result, align 8
  store i32 0, i32* %for_index, align 4
  store i8 0, i8* %c, align 1
  br label %while

while:                                            ; preds = %while_body, %
    entry
  %for_index7 = load i32, i32* %for_index, align 4
  %text8 = load i8*, i8** %text1, align 8
  %length9 = call i32 @string_length(i8* %text8, i32 0)
  %tmp10 = icmp slt i32 %for_index7, %length9
  br i1 %tmp10, label %while_body, label %merge

merge:                                            ; preds = %while
  %result11 = load %list_item**, %list_item*** %result, align 8
  ret %list_item** %result11

while_body:                                       ; preds = %while
  %text2 = load i8*, i8** %text1, align 8
  %for_index3 = load i32, i32* %for_index, align 4
  %get_char_ptr = getelementptr i8, i8* %text2, i32 %for_index3
  %get_char = load i8, i8* %get_char_ptr, align 1
  store i8 %get_char, i8* %c, align 1
  %for_index4 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index4, 1
  store i32 %tmp, i32* %for_index, align 4
  %result5 = load %list_item**, %list_item*** %result, align 8
  %list_ptr = load %list_item*, %list_item** %result5, align 8
  %c6 = load i8, i8* %c, align 1
  %length = call i32 @list_length(%list_item* %list_ptr, i32 0)
  %list_ptr_ptr = call %list_item** @insert_char(%list_item** %result5, i8 %
    c6, i32 %length)
  store %list_item** %list_ptr_ptr, %list_item*** %result, align 8
  br label %while
}

define i8 @lower(i8 %c) {
entry:
  %c_ref = alloca i8, align 1
  %for_index = alloca i32, align 4
  %index = alloca i32, align 4
  %c1 = alloca i8, align 1
  store i8 %c, i8* %c1, align 1
  store i32 -1, i32* %index, align 4
```

```
  store i32 0, i32* %for_index, align 4
  store i8 0, i8* %c_ref, align 1
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index24 = load i32, i32* %for_index, align 4
  %ASCII25 = load %list_item**, %list_item*** @ASCII, align 8
  %ilist26 = load %list_item*, %list_item** %ASCII25, align 8
  %length = call i32 @list_length(%list_item* %ilist26, i32 0)
  %tmp27 = icmp slt i32 %for_index24, %length
  br i1 %tmp27, label %while_body, label %merge

merge:                                            ; preds = %while
  %c28 = load i8, i8* %c1, align 1
  ret i8 %c28

while_body:                                       ; preds = %while
  %ASCII = load %list_item**, %list_item*** @ASCII, align 8
  %ilist = load %list_item*, %list_item** %ASCII, align 8
  %for_index2 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index2)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %data = load i8, i8* %data_ptr, align 1
  store i8 %data, i8* %c_ref, align 1
  %for_index3 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index3, 1
  store i32 %tmp, i32* %for_index, align 4
  %index4 = load i32, i32* %index, align 4
  %tmp5 = add i32 %index4, 1
  store i32 %tmp5, i32* %index, align 4
  %c6 = load i8, i8* %c1, align 1
  %c_ref7 = load i8, i8* %c_ref, align 1
  %tmp8 = icmp eq i8 %c6, %c_ref7
  br i1 %tmp8, label %then, label %else23

merge9:                                           ; preds = %else23, %
    merge12
  br label %while

then:                                             ; preds = %while_body
  %index10 = load i32, i32* %index, align 4
  %tmp11 = icmp slt i32 %index10, 26
  br i1 %tmp11, label %then13, label %else

merge12:                                          ; No predecessors!
  br label %merge9

then13:                                           ; preds = %then
  %c14 = load i8, i8* %c1, align 1
  ret i8 %c14

else:                                             ; preds = %then
  %ASCII15 = load %list_item**, %list_item*** @ASCII, align 8
  %ilist16 = load %list_item*, %list_item** %ASCII15, align 8
  %index17 = load i32, i32* %index, align 4
  %tmp18 = sub i32 %index17, 26
```

```
  %_result19 = call %list_item* @list_access(%list_item* %ilist16, i32 %
    tmp18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %data22 = load i8, i8* %data_ptr21, align 1
  ret i8 %data22

else23:                                           ; preds = %while_body
  br label %merge9
}

define i8 @upper(i8 %c) {
entry:
  %c_ref = alloca i8, align 1
  %for_index = alloca i32, align 4
  %index = alloca i32, align 4
  %c1 = alloca i8, align 1
  store i8 %c, i8* %c1, align 1
  store i32 -1, i32* %index, align 4
  store i32 0, i32* %for_index, align 4
  store i8 0, i8* %c_ref, align 1
  br label %while

while:                                            ; preds = %merge9, %entry
  %for_index24 = load i32, i32* %for_index, align 4
  %ASCII25 = load %list_item**, %list_item*** @ASCII, align 8
  %ilist26 = load %list_item*, %list_item** %ASCII25, align 8
  %length = call i32 @list_length(%list_item* %ilist26, i32 0)
  %tmp27 = icmp slt i32 %for_index24, %length
  br i1 %tmp27, label %while_body, label %merge

merge:                                            ; preds = %while
  %c28 = load i8, i8* %c1, align 1
  ret i8 %c28

while_body:                                       ; preds = %while
  %ASCII = load %list_item**, %list_item*** @ASCII, align 8
  %ilist = load %list_item*, %list_item** %ASCII, align 8
  %for_index2 = load i32, i32* %for_index, align 4
  %_result = call %list_item* @list_access(%list_item* %ilist, i32 %
    for_index2)
  %data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %_result,
    i32 0, i32 0
  %data_ptr = load i8*, i8** %data_ptr_ptr, align 8
  %data = load i8, i8* %data_ptr, align 1
  store i8 %data, i8* %c_ref, align 1
  %for_index3 = load i32, i32* %for_index, align 4
  %tmp = add i32 %for_index3, 1
  store i32 %tmp, i32* %for_index, align 4
  %index4 = load i32, i32* %index, align 4
  %tmp5 = add i32 %index4, 1
  store i32 %tmp5, i32* %index, align 4
  %c6 = load i8, i8* %c1, align 1
  %c_ref7 = load i8, i8* %c_ref, align 1
  %tmp8 = icmp eq i8 %c6, %c_ref7
  br i1 %tmp8, label %then, label %else23

```

```
merge9:                                           ; preds = %else23, %
    merge12
  br label %while

then:                                             ; preds = %while_body
  %index10 = load i32, i32* %index, align 4
  %tmp11 = icmp sgt i32 %index10, 25
  br i1 %tmp11, label %then13, label %else

merge12:                                          ; No predecessors!
  br label %merge9

then13:                                           ; preds = %then
  %c14 = load i8, i8* %c1, align 1
  ret i8 %c14

else:                                             ; preds = %then
  %ASCII15 = load %list_item**, %list_item*** @ASCII, align 8
  %ilist16 = load %list_item*, %list_item** %ASCII15, align 8
  %index17 = load i32, i32* %index, align 4
  %tmp18 = add i32 %index17, 26
  %_result19 = call %list_item* @list_access(%list_item* %ilist16, i32 %
    tmp18)
  %data_ptr_ptr20 = getelementptr inbounds %list_item, %list_item* %
    _result19, i32 0, i32 0
  %data_ptr21 = load i8*, i8** %data_ptr_ptr20, align 8
  %data22 = load i8, i8* %data_ptr21, align 1
  ret i8 %data22

else23:                                           ; preds = %while_body
  br label %merge9
}

define i1 @strcmp(i8* %str1, i8* %str2) {
entry:
  %c2 = alloca i8, align 1
  %c1 = alloca i8, align 1
  %i = alloca i32, align 4
  %str11 = alloca i8*, align 8
  store i8* %str1, i8** %str11, align 8
  %str22 = alloca i8*, align 8
  store i8* %str2, i8** %str22, align 8
  %str13 = load i8*, i8** %str11, align 8
  %length = call i32 @string_length(i8* %str13, i32 0)
  %str24 = load i8*, i8** %str22, align 8
  %length5 = call i32 @string_length(i8* %str24, i32 0)
  %tmp = icmp ne i32 %length, %length5
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else
  store i32 0, i32* %i, align 4
  br label %while

then:                                             ; preds = %entry
  ret i1 false

else:                                             ; preds = %entry
  br label %merge

```

```
while:                                            ; preds = %merge16, %merge
  %i21 = load i32, i32* %i, align 4
  %str122 = load i8*, i8** %str11, align 8
  %length23 = call i32 @string_length(i8* %str122, i32 0)
  %tmp24 = icmp slt i32 %i21, %length23
  br i1 %tmp24, label %while_body, label %merge6

merge6:                                           ; preds = %while
  ret i1 true

while_body:                                       ; preds = %while
  %str17 = load i8*, i8** %str11, align 8
  %i8 = load i32, i32* %i, align 4
  %get_char_ptr = getelementptr i8, i8* %str17, i32 %i8
  %get_char = load i8, i8* %get_char_ptr, align 1
  store i8 %get_char, i8* %c1, align 1
  %str29 = load i8*, i8** %str22, align 8
  %i10 = load i32, i32* %i, align 4
  %get_char_ptr11 = getelementptr i8, i8* %str29, i32 %i10
  %get_char12 = load i8, i8* %get_char_ptr11, align 1
  store i8 %get_char12, i8* %c2, align 1
  %c113 = load i8, i8* %c1, align 1
  %c214 = load i8, i8* %c2, align 1
  %tmp15 = icmp ne i8 %c113, %c214
  br i1 %tmp15, label %then17, label %else18

merge16:                                          ; preds = %else18
  %i19 = load i32, i32* %i, align 4
  %tmp20 = add i32 %i19, 1
  store i32 %tmp20, i32* %i, align 4
  br label %while

then17:                                           ; preds = %while_body
  ret i1 false

else18:                                           ; preds = %while_body
  br label %merge16
}

declare noalias i8* @malloc(i32)

define i32 @list_length(%list_item* %0, i32 %1) {
entry:
  %ptr_is_null = icmp eq %list_item* %0, null
  br i1 %ptr_is_null, label %then, label %else

then:                                             ; preds = %entry
  ret i32 %1

else:                                             ; preds = %entry
  %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
  %next = load %list_item*, %list_item** %next_ptr, align 8
  %add = add i32 %1, 1
  %result = call i32 @list_length(%list_item* %next, i32 %add)
  ret i32 %result
}

define %list_item* @list_access(%list_item* %0, i32 %1) {
```

```
entry:
  %is_zero = icmp eq i32 %1, 0
  br i1 %is_zero, label %then, label %else

then:                                             ; preds = %entry
  ret %list_item* %0

else:                                             ; preds = %entry
  %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
  %next = load %list_item*, %list_item** %next_ptr, align 8
  %sub = sub i32 %1, 1
  %result = call %list_item* @list_access(%list_item* %next, i32 %sub)
  ret %list_item* %result
}

define %list_item** @insert_string(%list_item** %0, i8* %1, i32 %2) {
entry:
  %list_ptr = load %list_item*, %list_item** %0, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
    i1** null, i32 1) to i32))
  %new_list_ptr_ptr = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
  %last_node_ptr_ptr = call %list_item** @list_copy_string(%list_item* %
    list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
  %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
  %temp = alloca %list_item, align 8
  %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
  store %list_item* %new_list_ptr, %list_item** %next, align 8
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %data_node = bitcast i8* %malloccall1 to %list_item*
  %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %data = bitcast i8* %malloccall2 to i8**
  store i8* %1, i8** %data, align 8
  %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 0
  %cast = bitcast i8** %data to i8*
  store i8* %cast, i8** %dat, align 8
  %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
  %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
    1
  %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
    i32 1
  %temp4 = load %list_item*, %list_item** %test, align 8
  store %list_item* %temp4, %list_item** %dat3, align 8
  store %list_item* %data_node, %list_item** %test, align 8
  %temp5 = load %list_item*, %list_item** %next, align 8
  store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
  ret %list_item** %new_list_ptr_ptr
}

define %list_item** @list_copy_string(%list_item* %0, i32 %1, %list_item**
    %2) {
entry:
  %is_zero = icmp eq i32 %1, 0
  %ptr_is_null = icmp eq %list_item* %0, null
  %or_conds = or i1 %is_zero, %ptr_is_null
```

```
  br i1 %or_conds, label %then, label %else

then:                                             ; preds = %entry
  ret %list_item** %2

else:                                             ; preds = %entry
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %new_struct_ptr = bitcast i8* %malloccall to %list_item*
  store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1
    *, i1** null, i32 1) to i32))
  %ltyp = bitcast i8* %malloccall1 to i8**
  %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
     0, i32 0
  %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
  %cast_old_data_ptr = bitcast i8* %old_data_ptr to i8**
  %old_data = load i8*, i8** %cast_old_data_ptr, align 8
  store i8* %old_data, i8** %ltyp, align 8
  %data_ptr_cast = bitcast i8** %ltyp to i8*
  %store_new_data = getelementptr inbounds %list_item, %list_item* %
    new_struct_ptr, i32 0, i32 0
  store i8* %data_ptr_cast, i8** %store_new_data, align 8
  store %list_item* %new_struct_ptr, %list_item** %2, align 8
  %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
    i32 0, i32 1
  %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
  %next2 = load %list_item*, %list_item** %next_ptr, align 8
  %sub = sub i32 %1, 1
  %3 = call %list_item** @list_copy_string(%list_item* %next2, i32 %sub, %
    list_item** %next)
  ret %list_item** %3
}

define %list_item** @range_function(i32 %0, i32 %1, %list_item** %2, i32 %3)
     {
entry:
  %is_last = icmp eq i32 %0, %1
  br i1 %is_last, label %then, label %else

then:                                             ; preds = %entry
  ret %list_item** %2

else:                                             ; preds = %entry
  %head_ptr_ptr = call %list_item** @insert_int(%list_item** %2, i32 %0, i32
     %3)
  %next_s = add i32 1, %0
  %next_length = add i32 1, %3
  %4 = call %list_item** @range_function(i32 %next_s, i32 %1, %list_item** %
    head_ptr_ptr, i32 %next_length)
  ret %list_item** %4
}

define %list_item** @insert_int(%list_item** %0, i32 %1, i32 %2) {
entry:
  %list_ptr = load %list_item*, %list_item** %0, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
```

```
3188   %new_list_ptr_ptr = bitcast i8* %malloccall 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   %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3196   %data_node = bitcast i8* %malloccall1 to %list_item*
3197   %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
         , i32* null, i32 1) to i32))
3198   %data = bitcast i8* %malloccall2 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:                                             ; preds = %entry
3222   ret %list_item** %2
3223
3224 else:                                             ; preds = %entry
3225   %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3226   %new_struct_ptr = bitcast i8* %malloccall to %list_item*
3227   store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3228   %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i32* getelementptr (i32
         , i32* null, i32 1) to i32))
3229   %ltyp = bitcast i8* %malloccall1 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:                                             ; preds = %entry
3286   %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3287   %new_struct_ptr = bitcast i8* %malloccall to %list_item*
3288   store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3289   %ltyp = tail call i8* @malloc(i32 ptrtoint (i8* getelementptr (i8, i8*
         null, i32 1) to i32))
3290   %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
          0, i32 0
3291   %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
3292   %old_data = load i8, i8* %old_data_ptr, align 1
3293   store i8 %old_data, i8* %ltyp, align 1
3294   %store_new_data = getelementptr inbounds %list_item, %list_item* %
         new_struct_ptr, i32 0, i32 0
3295   store i8* %ltyp, i8** %store_new_data, align 8
3296   store %list_item* %new_struct_ptr, %list_item** %2, align 8
3297   %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
         i32 0, i32 1
3298   %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
         1
3299   %next2 = load %list_item*, %list_item** %next_ptr, align 8
3300   %sub = sub i32 %1, 1
3301   %3 = call %list_item** @list_copy_char(%list_item* %next2, i32 %sub, %
         list_item** %next)
3302   ret %list_item** %3
3303 }
3304
3305 define %list_item** @insert_bool(%list_item** %0, i1 %1, i32 %2) {
3306 entry:
3307   %list_ptr = load %list_item*, %list_item** %0, align 8
3308   %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
          i1** null, i32 1) to i32))
3309   %new_list_ptr_ptr = bitcast i8* %malloccall to %list_item**
3310   store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
3311   %last_node_ptr_ptr = call %list_item** @list_copy_bool(%list_item* %
         list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
3312   %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
3313   %temp = alloca %list_item, align 8
3314   %next = getelementptr inbounds %list_item, %list_item* %temp, i32 0, i32 1
3315   store %list_item* %new_list_ptr, %list_item** %next, align 8
3316   %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3317   %data_node = bitcast i8* %malloccall1 to %list_item*
3318   %malloccall2 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
         i1* null, i32 1) to i32))
3319   %data = bitcast i8* %malloccall2 to i1*
3320   store i1 %1, i1* %data, align 1
3321   %dat = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
         i32 0
3322   %cast = bitcast i1* %data to i8*
3323   store i8* %cast, i8** %dat, align 8
3324   %result = call %list_item* @list_access(%list_item* %temp, i32 %2)
3325   %test = getelementptr inbounds %list_item, %list_item* %result, i32 0, i32
          1
3326   %dat3 = getelementptr inbounds %list_item, %list_item* %data_node, i32 0,
         i32 1
3327   %temp4 = load %list_item*, %list_item** %test, align 8
```

```
  store %list_item* %temp4, %list_item** %dat3, align 8
  store %list_item* %data_node, %list_item** %test, align 8
  %temp5 = load %list_item*, %list_item** %next, align 8
  store %list_item* %temp5, %list_item** %new_list_ptr_ptr, align 8
  ret %list_item** %new_list_ptr_ptr
}

define %list_item** @list_copy_bool(%list_item* %0, i32 %1, %list_item** %2)
    {
entry:
  %is_zero = icmp eq i32 %1, 0
  %ptr_is_null = icmp eq %list_item* %0, null
  %or_conds = or i1 %is_zero, %ptr_is_null
  br i1 %or_conds, label %then, label %else

then:                                             ; preds = %entry
  ret %list_item** %2

else:                                             ; preds = %entry
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
    getelementptr (%list_item, %list_item* null, i32 1) to i32))
  %new_struct_ptr = bitcast i8* %malloccall to %list_item*
  store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
  %malloccall1 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1,
    i1* null, i32 1) to i32))
  %ltyp = bitcast i8* %malloccall1 to i1*
  %old_data_ptr_ptr = getelementptr inbounds %list_item, %list_item* %0, i32
     0, i32 0
  %old_data_ptr = load i8*, i8** %old_data_ptr_ptr, align 8
  %cast_old_data_ptr = bitcast i8* %old_data_ptr to i1*
  %old_data = load i1, i1* %cast_old_data_ptr, align 1
  store i1 %old_data, i1* %ltyp, align 1
  %data_ptr_cast = bitcast i1* %ltyp to i8*
  %store_new_data = getelementptr inbounds %list_item, %list_item* %
    new_struct_ptr, i32 0, i32 0
  store i8* %data_ptr_cast, i8** %store_new_data, align 8
  store %list_item* %new_struct_ptr, %list_item** %2, align 8
  %next = getelementptr inbounds %list_item, %list_item* %new_struct_ptr,
    i32 0, i32 1
  %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32
    1
  %next2 = load %list_item*, %list_item** %next_ptr, align 8
  %sub = sub i32 %1, 1
  %3 = call %list_item** @list_copy_bool(%list_item* %next2, i32 %sub, %
    list_item** %next)
  ret %list_item** %3
}

define %list_item** @insert_float(%list_item** %0, double %1, i32 %2) {
entry:
  %list_ptr = load %list_item*, %list_item** %0, align 8
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1** getelementptr (i1*,
     i1** null, i32 1) to i32))
  %new_list_ptr_ptr = bitcast i8* %malloccall to %list_item**
  store %list_item* null, %list_item** %new_list_ptr_ptr, align 8
  %last_node_ptr_ptr = call %list_item** @list_copy_float(%list_item* %
    list_ptr, i32 -1, %list_item** %new_list_ptr_ptr)
  %new_list_ptr = load %list_item*, %list_item** %new_list_ptr_ptr, align 8
  %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   %malloccall1 = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3380   %data_node = bitcast i8* %malloccall1 to %list_item*
3381   %malloccall2 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (
         double, double* null, i32 1) to i32))
3382   %data = bitcast i8* %malloccall2 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   %malloccall = tail call i8* @malloc(i32 ptrtoint (%list_item*
         getelementptr (%list_item, %list_item* null, i32 1) to i32))
3410   %new_struct_ptr = bitcast i8* %malloccall to %list_item*
3411   store %list_item zeroinitializer, %list_item* %new_struct_ptr, align 1
3412   %malloccall1 = tail call i8* @malloc(i32 ptrtoint (double* getelementptr (
         double, double* null, i32 1) to i32))
3413   %ltyp = bitcast i8* %malloccall1 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
```

```
  %next_ptr = getelementptr inbounds %list_item, %list_item* %0, i32 0, i32 1
  %next2 = load %list_item*, %list_item** %next_ptr, align 8
  %sub = sub i32 %1, 1
  %3 = call %list_item** @list_copy_float(%list_item* %next2, i32 %sub, %list_item** %next)
  ret %list_item** %3
}

define i1 @strcmp_function(i8* %0, i8* %1) {
entry:
  %length = call i32 @string_length(i8* %0, i32 0)
  %length1 = call i32 @string_length(i8* %1, i32 0)
  %same_length = icmp ne i32 %length, %length1
  br i1 %same_length, label %then, label %else

then:                                             ; preds = %entry
  ret i1 false

else:                                             ; preds = %entry
  %last_index = sub i32 %length, 1
  %res = call i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %last_index, i32 0)
  ret i1 %res
}

define i32 @string_length(i8* %0, i32 %1) {
entry:
  %char = load i8, i8* %0, align 1
  %ptr_is_null = icmp eq i8 %char, 0
  br i1 %ptr_is_null, label %then, label %else

then:                                             ; preds = %entry
  ret i32 %1

else:                                             ; preds = %entry
  %next_ptr = getelementptr i8, i8* %0, i32 1
  %add = add i32 %1, 1
  %result = call i32 @string_length(i8* %next_ptr, i32 %add)
  ret i32 %result
}

define i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %2, i32 %3) {
entry:
  %charA_ptr = getelementptr i8, i8* %0, i32 %3
  %charA = load i8, i8* %charA_ptr, align 1
  %charB_ptr = getelementptr i8, i8* %1, i32 %3
  %charB = load i8, i8* %charB_ptr, align 1
  %not_same = icmp ne i8 %charA, %charB
  br i1 %not_same, label %then_not_same, label %else_same

then_not_same:                                    ; preds = %entry
  ret i1 false

else_same:                                        ; preds = %entry
  %last_char = icmp eq i32 %3, %2
  br i1 %last_char, label %then_end, label %else_not_end

then_end:                                         ; preds = %else_same
```

```
  ret i1 true

else_not_end:                                     ; preds = %else_same
  %next_index = add i32 1, %3
  %res = call i1 @strcmp_helper_function(i8* %0, i8* %1, i32 %2, i32 %
    next_index)
  ret i1 %res
}

; Function Attrs: argmemonly nounwind willreturn
declare void @llvm.memcpy.p0i8.p0i8.i32(i8* noalias nocapture writeonly, i8*
    noalias nocapture readonly, i32, i1 immarg) #0

attributes #0 = { argmemonly nounwind willreturn }
```