FunC
Project Report

Fan Gao
fg2432@columbia.edu

May 17, 2019

Contents

1 Introduction

1.1 Overview

FunC is a C-like programming language that aims to fuse imperative and functional programming. The reason for creating this language is that we all like C! C is an imperative programming language that's easy to learn and compile. And most of the state-of-the-art software have been written in C. Despite the popularity of languages such as Java, Python or Go, C is not going anywhere anytime soon. Another popular paradigm is functional programming. In functional programming, functional transformations are implemented as pure functions, which are self-contained and stateless. These functions can then be composed into a chain of operations. These characteristics bring a number of advantages that C can benefit from. However, functions in C are second-class citizens as one can only write higher order functions through function pointers. Imperative programming and functional programming should not be mutually exclusive, which is already reflected in many new releases of programming languages that were originally designed to primarily support imperative programming, e.g., C++, C#, Java. FunC's support of language features such as first class functions, higher order functions, and anonymous functions will aid functional programming in C. The addition of higher order methods such as map, reduce and Unix-like pipe operator is FunC's syntactic sugar for writing compact and elegant code. Programs can be written in FunC in a way such that they share advantageous properties of functional programs without compromising the ability to write imperative code.

1.2 Motivation and Use Cases

The motivation for creating FunC is to leverage the benefits of functional programming in a traditional imperative language. This includes:

1. **Increased readability and maintainability:** Each function is designed to accomplish a specific task given its arguments without relying on any external state.

2. **Better code structure:** It encourages more functions rather than "bigger" functions, which leads to more code reuse.

3. **Easier code refactoring:** Because large and complicated transformation are decomposed into a chain of small and self-contained functions, it's easier to change the behavior of a component or rearrange the order of executions.

4. **Easier testing:** Pure functions can more easily be tested in isolation.

The aim for FunC is to do functional programming in C. This includes:

- **First-class functions**
  First-class functions open the way for functional programming. They make it easy to pass around units of behavior and work with functions like with any other values.

- **Higher-order functions**
  Not only can functions accept behaviors through arguments but they can also return behaviors.

- **Anonymous functions**
  These self-contained inline functions are quick and easy to write to replace local operations that don't need to be named and added to the namespace.

- **Map and reduce**
  Using these functions, users can avoid writing for loops and managing the states.

- **Pipe**
  Pipe allows users to decompose a long series of successive data transformations into a functional pipeline. Functions in this pipeline form a chain of operations. In addition to better readability, another benefit is that once a pipe takes inputs, it performs computation within the pipeline without interfering the environment.

2 Language Tutorial

2.1 Installation (macOS)

1. Install ocaml:

   ```
   $ brew install ocaml
   ```

2. Install opam:

   ```
   $ brew install opam
   ```

3. Set up opam environment:

   ```
   $ opam init
   $ eval ‘opam env‘
   ```

4. Install LLVM 7.0.1.:

   ```
   $ brew install llvm@7
   ```

5. Install the ocaml llvm library:

   ```
   $ opam install llvm.7.0.0
   ```

6. Add llvm to path:

   ```
   $ export PATH=$PATH:/usr/local/opt/llvm/bin
   ```

7. Make sure you have permission to execute ./testall.sh:

   ```
   $ sudo chmod u+x ./testall.sh
   ```

2.2 Compilation

- Compile FunC:

  $ ocamlbuild -use-ocamlfind func.native

- Run tests:

  $ ./testall.sh

- Compile FunC and run tests:

  $ make

- Clean up the directory, remove build, output and log files:

  $ make clean

2.3 Hello world

1. Compile FunC:

   $ make

2. Use FunC to compile the tests/helloworld.fc file:

   $ ./func.native -c tests/test-helloworld.fc > helloworld.ll

3. Run helloworld.ll with lli:

   $ lli helloworld.ll

2.4 Key language features

```
int twice(func<int(int)> f, int x)
{
    return f(f(x));
}

func<void()> greet(bool formal)
{
    func<void()> greet_formal;
    func<void()> greet_casual;

    greet_formal = () => void { prints("How are you?"); };
    greet_casual = () => void { prints("What's up?"); };

    if (formal)
    {
        return greet_formal;
    }

    return greet_casual;
}

int main()
{
    /* Declarations */
    int r1;
    int r2;
    int r3;
    int len;
    int[5] numbers;
    int[5] squared_numbers;

    /* First-class functions */
    func<int(int)> square;
    func<int(int, int)> sum;
    func<void()> greet_casual;

    /* Array */
    len = 5;
    numbers = [1, 2, 3, 4, 5];

    /* Anonymous functions */
    square = (int a) => int { return a * a; };
    sum = (int a, int b) => int { return a + b; };

    /* Higher-order functions */
    r1 = twice(square, 2);
    print(r1); /* 16 */
    greet_casual = greet(false);
    greet_casual(); /* What's up? */

    /* Map & reduce */
    squared_numbers = map(square, len, numbers);
    r2 = reduce(sum, len, squared_numbers);
    print(r2); /* 55 */

    /* Piping */
    r3 = numbers |> map(square, len) |> reduce(sum, len) |;
    print(r3); /* 55 */

    return 0;
}
```

3 Language Reference Manual

FunC is stylistically and syntactically similar to C, with additional elements to support functional features.

3.1 Lexical Conventions

3.1.1 Tokens

Tokens include identifiers, keywords, constants, operators and other separators. Spaces (`' '`), tabs (`'\t'`), carriage returns (`'\r'`) and newlines (`'\n'`) are ignored by the compiler except as they separate tokens.

3.1.2 Comments

Comments are enclosed by /* and */. Comments can span multiple lines and do not nest.

```
/* comment goes here */

/*
  more comments
  can go here
*/
```

3.1.3 Identifiers

An identifier is a sequence of ASCII letters, digits and underscores ('_'). An identifier must begin with a letter and cannot be a reserved keyword. Identifiers are case sensitive.

```
/* valid identifiers */
num
NUM
My_Num

/* invalid identifiers */
2num
_num
```

3.1.4 Keywords

The following identifiers are reserved keywords and may not be used otherwise:

bool	else	false	float	for
func	if	int	map	new
print	printf	prints	reduce	return
true	void	while		

3.1.5 Literal Constants

3.1.5.1 Integer Constants

An integer constant consists of a sequence of decimal digits, e.g., 0, 168.

3.1.5.2 Floating Constants

Floating-point constants consists of an integer part, a decimal point, a fraction part, an e or an E, and an optionally signed integer exponent. The integer and fraction parts both consist a sequence of digits. Either the integer part or the fraction part may be missing (not both); either the decimal point or the e/E and the exponent may be missing (not both). You can represent floating point literals either in decimal form or exponential form. Here are some examples of floating constant: 1.2, 1., .2, .3E+3, 0.5e-15, 1e5.

3.1.5.3 String Constants

A string constant contains a sequence of characters. String constants are enclosed in double quotes "". Here is an example of string constant: "Hello, world!"

3.2 Types

3.2.1 Basic Types

There are a few fundamental types:

- **int:** The int type is limited to the range $[-2^{30}, 2^{30} - 1]$. It represents signed integer values. Integer arithmetic is taken modulo 2^{31}; that is, an integer operation that overflows does not raise an error, but the result simply wraps around.
- **float:** IEEE 64-bit floating point type.
- **bool:** The bool type is composed of two values - **true** and **false**.
- **void:** The void type specifies an empty set of values. It is used as the type returned by functions that generate no value.

3.2.2 Arrays

Arrays are allocated on the heap and inherently a pointer to the memory address on the heap. It means that if an array is created outside the scope of a function and is passed in as an argument, any changes to this array will be reflected in the original array.

Variable-length arrays are supported. You can pass an array around without specifying the size of it.

```
void square(int[] arr, int length)
{
    int i;
    for (i = 0; i < length; i = i + 1)
        arr[i] = arr[i] * arr[i];
}

int[] square2(int[] arr, int length)
{
    int i;
    new_arr = new int[length];

    for (i = 0; i < length; i = i + 1)
        new_arr[i] = arr[i] * arr[i];

    return new_arr;
}

int main()
{
    int[3] arr;
    int[3] new_arr;

    arr = [1, 2, 3];
    /* In-place */
    square(arr, 3); /* arr = [1, 4, 9] */

    /* Return a new array */
    new_arr = square2(arr, 3); /* arr = [1, 4, 9] */
    /* new_arr = [1, 16, 81] */

    return 0;
}
```

3.2.3 Functions

Functions are first-class citizens in FunC. The func type represents a function with any arbitrary arguments and return type. The return type and a list of argument types must be specified in the form of func<return type(comma delimited list of argument types)>. For example, a function that takes in 2 integer arguments and returns an integer value has type func<int(int, int)>.

3.3 Expressions

The precedence of expression operators follows the order of the subsections from the highest to the lowest. Operators have the same precedence within each subsection.

3.3.1 Primary Expressions

Primary expression are identifiers, constants, or expression in parentheses. An identifier has to be declared first. Its type is specified by its declaration. A parenthesized expression is equivalent to the unadorned expression.

```
/* invalid expression: a needs to be declared first. */
a * 10

int a;
a = 10; /* valid expression */
(((a) * 2) == 20) /* true and equivalent to a * 2 == 20 */
```

3.3.2 Postfix Expressions

The operators in postfix expressions are left-associative.

3.3.2.1 Array References

Square brackets following an expression denote a subscripted array reference. Subscript i must be of integer type.

```
array[i]
```

3.3.2.2 Function Calls

Parentheses following an expression denote a function call. They may contain a comma-delimited list of argument expressions.

```
wait()
sum(int a, int b)
```

3.3.3 Unary Operators

Unary operators in expressions are right-associative.

3.3.3.1 Unary Minus Operator

The operand of the unary - operator must have arithmetic type, and the result is the negative of its operand. Negative zero is still zero.

3.3.3.2 Logical Negation Operator

The operand of the ! operator must have Boolean type. The result is true if the value of its operand equal to false, and false otherwise. The type of the result is bool.

```
!(x < 10) /* equivalent to x >= 10 */
!true     /* false */
```

3.3.4 Multiplicative Operators

The multiplicative operators *, / and % are left-associative. The operands must have arithmetic type.

The * operator denotes multiplication. The / operator yields the quotient. The % operator returns the remainder of an integer division. % is undefined for floating numbers.

```
2 * 3 /* 6 */
6 / 2 /* 3 */
6 % 2 /* 0 */
```

3.3.5 Additive Operators

The additive operators + and - are left-associtive. The operands must have arithmetic type.

The + operator denotes the addition of the operands. The - operator means subtracting the right operand from the left operand.

```
2 + 3 /* 5 */
2 - 3 /* -1 */
```

3.3.6 Relational Operators

The relational operators are left-associative. For example, $a < b < c$ is parsed as $(a < b) < c$, and $a < b$ evaluates to either true or false.

The operators < (less than), > (greater than), <= (less than or equal to) and >= (greater than or equal to) all return true if the specified relation is true and false otherwise.

```
2 < 3  /* true */
2 > 3  /* false */
2 <= 3 /* true */
2 >= 3 /* false */
```

3.3.7 Equality Operators

The == (equal to) and the != (not equal to) operators are analogous to the relational operators except for their lower precedence.

```
2 < 3 == 2 > 3   /* false */
2 < 3 == 2 <= 3  /* true */
2 < 3 != 2 > 3   /* true */
2 < 3 != 2 <= 3  /* false */
```

3.3.8 Logical And Operator

The && operator groups left-to-right and returns true if both of its operands are true, false otherwise.

```
2 < 3 && 2 > 3   /* false */
2 < 3 && 2 <= 3  /* true */
```

3.3.9 Logical Or Operator

The || operator groups left-to-right and returns true if either of its operands are true, false otherwise.

```
2 < 3 || 2 > 3   /* true */
2 > 3 || 2 >= 3  /* false */
```

3.3.10 New Operator

The new operator is right associative. It allocates space for an array on the heap based on the given element type and length of the array.

```
int[] a;
a = new int[3];
```

3.3.11 Lambda Operator

The => operator is right associative. It takes input arguments (if any) on the left side of the lambda operator, and return type and statement block on the right side.

```
(int a, int b) => int { return a + b; }
```

3.3.12 Pipe & Bar Operator

The | > and | operators group left-to-right and concatenates a sequence of function calls. The pipe operator takes the output from its left operand and feeds it as input to its right operand. The output of the left operand can be of any type, but the type must match that of the last argument of the right operand. The bar operator is simply an indicator of the end of the piping.

```
f1(arg1, arg2) |> f2(arg1) |
```

3.3.13 Assignment Expressions

The = operator is right-associative. The value of the right expression replaces that of the identifier on the left. Both operands must be of the same type.

```
int a = 1;
```

3.3.14 Comma Operator

A pair of expressions separated by a comma is left associative, and the value of the left expression is discarded. The type and value of the result are the type and value of the right operand. Commas can also be used to separate arguments in a list of function arguments or elements in an array initializer.

3.3.15 Constant Expressions

The value of a constant expression is determined during compilation. Constant expressions can be composed of integer, float or Boolean value.

Constant expressions are restricted to a few places: in the subscript declarator as the description of the array bound or in array initializers.

3.4 Declarations

Declarations specify the interpretation given to each identifier. They also reserves storage associated with the identifier.

3.4.1 Type Specifiers

At most one type-specifier may be given in a declaration. The supported type-specifiers are:

- int
- bool
- float
- void
- func

3.4.2 Declarators

A declarator has the form **type identifier**. It yields an object of the specified type.

```
int a;
float b;
bool c;
```

3.4.3 Array Declaration

Array is a sequence of elements of the same type separated by commas and enclosed by brackets. Array declaration must follow the form of **type**[**size**] **identifier**, where **size** is optional and can be determined during assignment. This allows a function to accept or return an array of any size. Element-by-element assignment is invalid when size is not given in the declaration. You must use either an initializer or the new operator to reserve space for the array. An initializer assigns values to the array directly.

```
int[3] a;
int[] b;
a = [1, 2, 3];
b = new int[3];
```

3.5 Statements

Statements are executed in sequence for their effect. They do not have values and are terminated by semicolons.

3.5.1 Expression Statement

Expression statements are commonly used for assignments or function calls. A null statement will be used if the expression is missing.

```
int area;
area = height * width;
random();
```

3.5.2 Block Statement

A block statement is a group of statements enclosed by {} braces. The body of a function definition, for example, is a block statement.

```
{
    statement1
    statement2
    ...
}
```

3.5.3 Selection Statements

Selection statements can take two flows of control:

- if (expression) statement1
- if (expression) statement1 else statement2

In both forms, the expression must have Boolean type. If it evaluates to true, statement1 is executed. In the second case, statement2 is executed when the expression evaluates to false.

```
bool pass(int score)
{
    if (score >= 60)
        return true;
    else
        return false;
}
```

3.5.4 Iteration Statements

Iteration statements allows substatements to be executed repeatedly as long as the conditional expression evaluates to true. Two iteration statements are supported:

- while (conditional_expression) statement
- for (expression; conditional_expression; expression) statement

```
int cnt;
int i;
cnt = 0;

while (cnt < 10)
    cnt = cnt + 1;

for (i = 0; i < 20; i = i + 1)
    cnt = cnt + 1;
```

3.6 Functions

3.6.1 Function Declaration

There are two ways to declare functions:

type identifier (arg1, arg2, ...): The traditional C-style function definition is supported. Nested functions are not allowed in this type of declaration.

```
int sum(int a, int b) {
    return a + b;
}
```

func<return type(type1 arg1, type2 arg2, ...)> identifier: In this case, the types of input arguments and the return type of the function must be specified in the declaration. Nested functions are allowed in this type of declaration.

```
func(int<int, int>) sum;
sum = (int a, int b) => int { return a + b; }
```

3.6.2 Arguments and Return Types

The argument list can be empty, or a comma delimited list of types and identifiers. The argument and return types of subsequent assignments must match those specified in the function type.

```
func(int<int, int>) sum;
sum = (int a, int b)   => int { ... }   /* Valid   */
sum = (float a, int b) => int { ... }   /* Invalid */
sum = (int a, int b)   => float { ... } /* Invalid */
sum = (int a)          => int { ... }   /* Invalid */
```

A function may return an arithmetic type, a Boolean, an array, a function, or void.

3.6.3 Higher Order Functions

A function can be passed into another function as an argument. A function can also return a function.

```
int cube(func<int(int)> square, int n) {
    return square(n) * n;
}

int main()
{
    func<int(int)> square;
    square = (int a) => int { return a * a; };
    print(cube(square, 2)); /* 8 */

    return 0;
}
```

3.6.4 Anonymous Functions

As shown above, anonymous functions can be declared using the lambda operator. An anonymous function is just a function without a name. Typically, it's an inline function that performs a local operation that doesn't need to be named and added to the namespace, although it can span over multiple lines as the body is a block statement.

The arguments of the anonymous function is defined within () on the left side of the operator, and the return type and body of the function are defined on the right hand side.

3.6.5 Function Piping

Functions can be chained using the | > and | operators. Return value of one function is passed to the next function and is inserted as the last argument to the latter function. The chain executes functions from left to right and returns the output of the last function.

```
int[] numbers;
int len;
func<int(int)> square;
func<int(int, int)> sum;
int s;

numbers = [1, 2, 3, 4, 5];
len = 5;
square = (int x) => int { return x * x; };
sum = (int x, int y) => int { return x + y; };

s = numbers |> map(square, len) |> reduce(sum, len) |; /* 55 */
```

3.7 Scope

The lexical scope of an object or function identifier begins at the end of its declarator and persists to the end of the {} enclosing block. The scope of a parameter of a function definition begins at the start of the block defining the function, and persists through the function.

If an identifier is declared in a block, any declaration of the identifier outside the block is suspended until the end of the block.

3.8 Memory Management

The FunC language is ”pass-by-value”. Local variables are allocated on the stack and destroyed at the end of their scope. Global variables and functions are global variables (neither stack or heap). Arrays are allocated on the heap using malloc behind the scene. Garbage collection is out of the scope of this project.

3.9 Standard Library Functions

3.9.1 print

print() in general sends formatted output to the console. There are a few flavors: print is for integers, printf is for floating numbers, and prints is for strings.

```
print(10);                /* 10 */
printf(3.5);              /* 3.5 */
prints("Hello, world!"); /* Hello, world! */
```

3.9.2 map

map() takes in a function and an array and applies the given function to each item of the given array. It returns a new array of the size of the input array.

```
int[] numbers;
int len;
func<int(int)> square;
int[] squares;

numbers = [1, 2, 3, 4, 5];
len = 5;
square = (int x) => int { return x * x; };
squares = map(square, len, numbers); /* {1, 4, 9, 16, 25} */
```

3.9.3 reduce

reduce() takes in a function and an array and applies a rolling computation to sequential pairs of values in an array. It returns a single value.

```
int[] numbers;
int len;
func<int(int, int)> sum;
int s;

numbers = [1, 2, 3, 4, 5];
len = 5;
sum = (int x, int y) => int { return x + y; };
s = reduce(sum, len, numbers);
print(s); /* 15 */
```

4 Project Plan

4.1 Planning Process

Since I worked on this project by myself, the planning process was fairly easy. The scope of the project was determined at the beginning of the semester after a discussion with Justin Wong. After that, it was just learning LLVM and materializing all the features outlined in the language proposal. I set a number of milestones to keep track of my progress. In addition, I had biweekly meetings with my project TA, Ryan Bernstein. Ryan was extremely helpful, especially in providing different implementation strategies for some of the features.

4.2 Specification Process

The initial specification of the language and features were born when the first draft of the language reference manual was written. The spec heavily relied on the language reference manual of C because the goal of the project was to add additional features to C. Overall, the spec served as a guideline for the project, but was also updated progressively to reflect the reality as I gained better understanding of some of the problems during implementation. For example, variable-length array wasn't planned initially but became absolutely necessary in order to implement map.

4.3 Development Process

Development was broken down to features. For each feature, development followed the stages of the compiler architecture. First of all, unit tests were written so the behavior of a feature was clearly defined. Then each feature was just a incremental change to Scanner, AST, Parser, SAST, Semantic checker, and Code Generator. Passing all the tests signified the completion of a feature.

4.4 Programming Style Guide

The programming style of FunC follows the following guideline: OCaml Programming Guidelines.

4.5 Project Timeline

Date	Milestone
January 31, 2019	Project consultation with TA
February 13, 2019	Language proposal complete
March 3, 2019	Language reference manual complete
March 26, 2019	Basic code structure complete
March 31, 2019	Add small features such as modulo
April 8, 2019	Hello, world!
April 15, 2019	Support a data structure
April 25, 2019	Support first-class and anonymous functions
April 30, 2019	Support higher-order functions and closure
May 5, 2019	Support function piping
May 10, 2019	Support map and reduce
May 15, 2019	Project presentation
May 20, 2019	Project final report complete

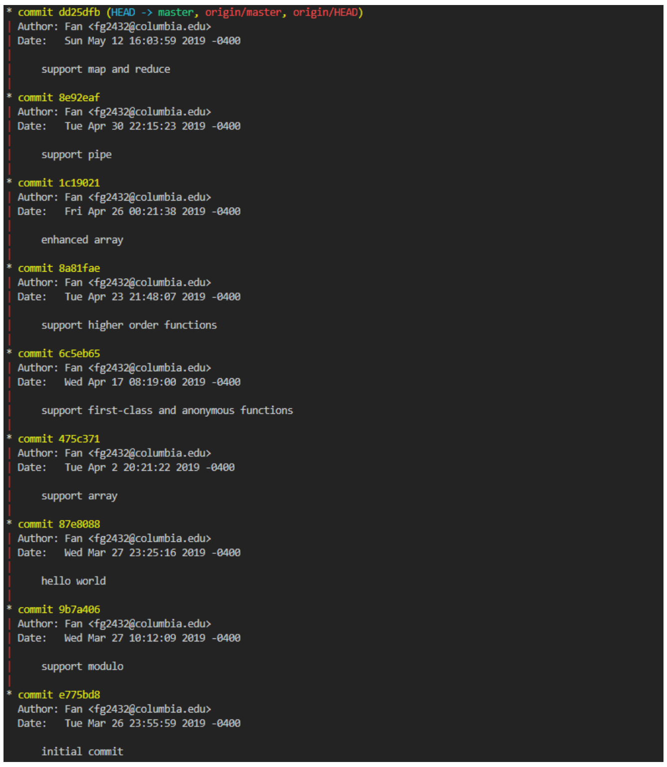
Table 1: Project Timeline

4.6 Software Development Environment

- **Programming language**
  - OCaml - v4.07.1: for majority of the coding.
  - ocamllex and ocamlyacc: for compiling the scanner and parser front-end
  - LLVM - 7.0.0: for code generation and back-end
  - C: for a small amount of C code
  - Shell script: for file manipulation and testing
- **Tools**
  - OS: macOS Mojave Version 10.14.4
  - IDE: Visual Studio Code, emacs
  - Package manager: opam

- Source control: git
- Build automation: make

4.7 Project Log



```
* commit dd25dfb (HEAD -> master, origin/master, origin/HEAD)
| Author: Fan <fg2432@columbia.edu>
| Date:   Sun May 12 16:03:59 2019 -0400
|
|     support map and reduce
|
* commit 8e92eaf
| Author: Fan <fg2432@columbia.edu>
| Date:   Tue Apr 30 22:15:23 2019 -0400
|
|     support pipe
|
* commit 1c19021
| Author: Fan <fg2432@columbia.edu>
| Date:   Fri Apr 26 00:21:38 2019 -0400
|
|     enhanced array
|
* commit 8a81fae
| Author: Fan <fg2432@columbia.edu>
| Date:   Tue Apr 23 21:48:07 2019 -0400
|
|     support higher order functions
|
* commit 6c5eb65
| Author: Fan <fg2432@columbia.edu>
| Date:   Wed Apr 17 08:19:00 2019 -0400
|
|     support first-class and anonymous functions
|
* commit 475c371
| Author: Fan <fg2432@columbia.edu>
| Date:   Tue Apr 2 20:21:22 2019 -0400
|
|     support array
|
* commit 87e8088
| Author: Fan <fg2432@columbia.edu>
| Date:   Wed Mar 27 23:25:16 2019 -0400
|
|     hello world
|
* commit 9b7a406
| Author: Fan <fg2432@columbia.edu>
| Date:   Wed Mar 27 10:12:09 2019 -0400
|
|     support modulo
|
* commit e775bd8
  Author: Fan <fg2432@columbia.edu>
  Date:   Tue Mar 26 23:55:59 2019 -0400

      initial commit
```

Figure 1: Project Log

5 Architectural Design

The architecture of FunC consists of five modules and two interfaces. The modules are FunC, Scanner, Parser, Semantic Checker, and Code Generator. The interfaces are AST and SAST.

Overall, FunC follows a traditional compiler architecture design:

1. **FunC (func.ml)** is the organizer that glues all the components together.
2. **Scanner (scanner.mll)** takes as input a FunC source file and generate tokens for identifiers, operators, keywords and constant literals.
3. **AST (ast.ml)** is an abstract syntax tree in which the grammar of FunC is defined.
4. **Parser (parser.mly)** generates an abstract syntax tree out of the given tokens based on FunC's grammar.

5. **SAST (sast.ml)** is a semantically checked abstract syntax tree. It carries additional type information.

6. **Semantic Checker (semant.ml)** recursively traverses the AST and converts it into a SAST. This is where expressions and statements are matched against their types in the symbol table, verified within lexical scope, and added to the SAST.

7. **Code Generator (codegen.ml)** traverses the SAST to generate LLVM intermediate representation (IR) code in post-order fashion. The LLVM OCaml library is used throughout this phase. You can find the documentation at https://llvm.moe/ocaml/Llvm.html

8. After code generation, the control is handed over to LLVM and gcc compilers and linker. Local C libraries will be compiled and linked in this phase. Eventually, the output is an executable called func.native - this is the FunC compiler.

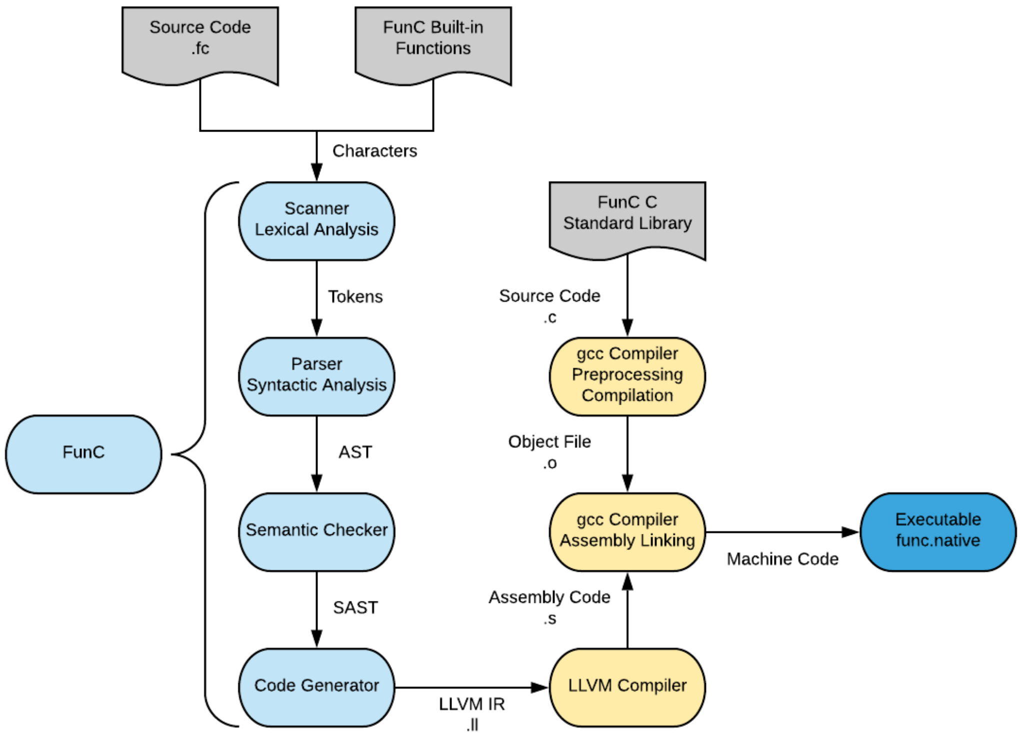


Figure 2: Architecture of FunC

6 Test Plan

Tests were written before the development started to make sure expected behaviors are well defined and met. Subsequent revisions were made only when it was absolutely necessary or to complement the original tests. There are two cases and two categories of tests:

- Failing case:

  These tests cover the unwarranted behaviors and error message. The compiler should provide adequate error messages to help a user debugging code.

- Passing case:

  These tests cover the expected behaviors and output. They serve as sanity checks on parsing and LLVM code generation.

- Unit testing:

  Each feature has one or more dedicated tests to make sure it's fully functional.

- Integration testing

  The goal of these end-to-end tests is to make sure multiple features integrate well with each other.

Tests were committed along with each feature check-in. Test coverage for the project is 100%.

6.1 Source Code and Target Language Programs

6.1.1 gcd

Listing 1: gcd.fc

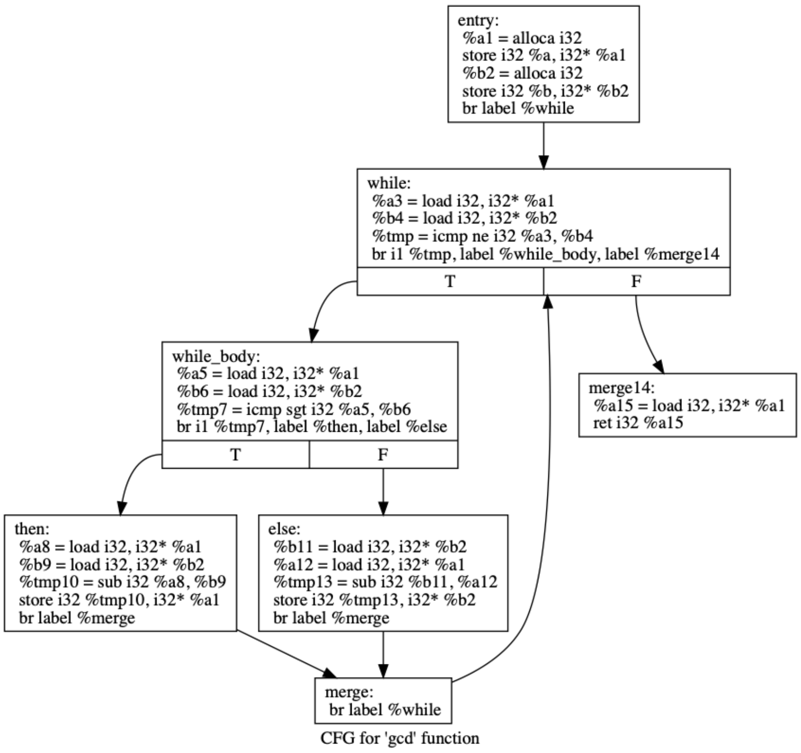
```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

int main()
{
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
  return 0;
}
```

Listing 2: gcd.ll

```
; ModuleID = 'FunC'
source_filename = "FunC"
```

```
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@gcd_ptr = global i32 (i32, i32)* null
@main_ptr = global i32 ()* null
@map_ptr = global i32* (i32 (i32)*, i32, i32*)* null
@reduce_ptr = global i32 (i32 (i32, i32)*, i32, i32*)* null

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

declare i32 @printbig(i32)

define i32* @map(i32 (i32)* %f, i32 %length, i32* %arr) {
entry:
  %f1 = alloca i32 (i32)*
  store i32 (i32)* %f, i32 (i32)** %f1
  %length2 = alloca i32
  store i32 %length, i32* %length2
  %arr3 = alloca i32*
  store i32* %arr, i32** %arr3
  %narr = alloca i32*
  %i = alloca i32
  %length4 = load i32, i32* %length2
  %mallocsize = mul i32 %length4, ptrtoint (i32* getelementptr
   ↪  (i32, i32* null, i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %arrptr = bitcast i8* %malloccall to i32*
  store i32* %arrptr, i32** %narr
  store i32 0, i32* %i
  br label %while

while:                                                 ; preds =
 ↪  %while_body, %entry
  %i5 = load i32, i32* %i
  %length6 = load i32, i32* %length2
  %tmp = icmp slt i32 %i5, %length6
  br i1 %tmp, label %while_body, label %merge

while_body:                                            ; preds =
 ↪  %while
  %tmp7 = load i32 (i32)*, i32 (i32)** %f1
  %arr8 = load i32*, i32** %arr3
  %i9 = load i32, i32* %i
  %tmp10 = getelementptr i32, i32* %arr8, i32 %i9
  %tmp11 = load i32, i32* %tmp10
  %0 = call i32 %tmp7(i32 %tmp11)
  %narr12 = load i32*, i32** %narr
  %i13 = load i32, i32* %i
  %narr14 = getelementptr i32, i32* %narr12, i32 %i13
  store i32 %0, i32* %narr14
  %i15 = load i32, i32* %i
  %tmp16 = add i32 %i15, 1
  store i32 %tmp16, i32* %i
  br label %while

merge:                                            ; preds =
↪  %while
  %narr17 = load i32*, i32** %narr
  ret i32* %narr17
}

define i32 @reduce(i32 (i32, i32)* %f, i32 %length, i32* %arr) {
entry:
  %f1 = alloca i32 (i32, i32)*
  store i32 (i32, i32)* %f, i32 (i32, i32)** %f1
  %length2 = alloca i32
  store i32 %length, i32* %length2
  %arr3 = alloca i32*
  store i32* %arr, i32** %arr3
  %i = alloca i32
  %result = alloca i32
  %length4 = load i32, i32* %length2
  %tmp = icmp sle i32 %length4, 0
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else
  %arr5 = load i32*, i32** %arr3
  %tmp6 = getelementptr i32, i32* %arr5, i32 0
  %tmp7 = load i32, i32* %tmp6
  store i32 %tmp7, i32* %result
  store i32 1, i32* %i
  br label %while

then:                                             ; preds =
↪  %entry
  ret i32 0

else:                                             ; preds =
↪  %entry
  br label %merge

while:                                            ; preds =
↪  %while_body, %merge
  %i8 = load i32, i32* %i
  %length9 = load i32, i32* %length2
  %tmp10 = icmp slt i32 %i8, %length9
  br i1 %tmp10, label %while_body, label %merge19
```

```
while_body:                                        ; preds =
↪  %while
  %tmp11 = load i32 (i32, i32)*, i32 (i32, i32)** %f1
  %arr12 = load i32*, i32** %arr3
  %i13 = load i32, i32* %i
  %tmp14 = getelementptr i32, i32* %arr12, i32 %i13
  %tmp15 = load i32, i32* %tmp14
  %result16 = load i32, i32* %result
  %0 = call i32 %tmp11(i32 %result16, i32 %tmp15)
  store i32 %0, i32* %result
  %i17 = load i32, i32* %i
  %tmp18 = add i32 %i17, 1
  store i32 %tmp18, i32* %i
  br label %while

merge19:                                           ; preds =
↪  %while
  %result20 = load i32, i32* %result
  ret i32 %result20
}

define i32 @main() {
entry:
  store i32 (i32, i32)* @gcd, i32 (i32, i32)** @gcd_ptr
  store i32 ()* @main, i32 ()** @main_ptr
  store i32* (i32 (i32)*, i32, i32*)* @map, i32* (i32 (i32)*,
  ↪  i32, i32*)** @map_ptr
  store i32 (i32 (i32, i32)*, i32, i32*)* @reduce, i32 (i32 (i32,
  ↪  i32)*, i32, i32*)** @reduce_ptr
  %tmp = load i32 (i32, i32)*, i32 (i32, i32)** @gcd_ptr
  %0 = call i32 %tmp(i32 2, i32 14)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr
  ↪  inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %0)
  %tmp1 = load i32 (i32, i32)*, i32 (i32, i32)** @gcd_ptr
  %1 = call i32 %tmp1(i32 3, i32 15)
  %printf2 = call i32 (i8*, ...) @printf(i8* getelementptr
  ↪  inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %1)
  %tmp3 = load i32 (i32, i32)*, i32 (i32, i32)** @gcd_ptr
  %2 = call i32 %tmp3(i32 99, i32 121)
  %printf4 = call i32 (i8*, ...) @printf(i8* getelementptr
  ↪  inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32 %2)
  ret i32 0
}

define i32 @gcd(i32 %a, i32 %b) {
entry:
  %a1 = alloca i32
  store i32 %a, i32* %a1
  %b2 = alloca i32
  store i32 %b, i32* %b2
  br label %while

while:                                              ; preds =
↪  %merge, %entry
  %a3 = load i32, i32* %a1
  %b4 = load i32, i32* %b2
  %tmp = icmp ne i32 %a3, %b4
  br i1 %tmp, label %while_body, label %merge14

while_body:                                         ; preds =
↪  %while
  %a5 = load i32, i32* %a1
  %b6 = load i32, i32* %b2
  %tmp7 = icmp sgt i32 %a5, %b6
  br i1 %tmp7, label %then, label %else

merge:                                              ; preds =
↪  %else, %then
  br label %while

then:                                               ; preds =
↪  %while_body
  %a8 = load i32, i32* %a1
  %b9 = load i32, i32* %b2
  %tmp10 = sub i32 %a8, %b9
  store i32 %tmp10, i32* %a1
  br label %merge

else:                                               ; preds =
↪  %while_body
  %b11 = load i32, i32* %b2
  %a12 = load i32, i32* %a1
  %tmp13 = sub i32 %b11, %a12
  store i32 %tmp13, i32* %b2
  br label %merge

merge14:                                            ; preds =
↪  %while
  %a15 = load i32, i32* %a1
  ret i32 %a15
}

declare noalias i8* @malloc(i32)
```


```
entry:
 %a1 = alloca i32
 store i32 %a, i32* %a1
 %b2 = alloca i32
 store i32 %b, i32* %b2
 br label %while
```
```
while:
 %a3 = load i32, i32* %a1
 %b4 = load i32, i32* %b2
 %tmp = icmp ne i32 %a3, %b4
 br i1 %tmp, label %while_body, label %merge14
```

```
while_body:
 %a5 = load i32, i32* %a1
 %b6 = load i32, i32* %b2
 %tmp7 = icmp sgt i32 %a5, %b6
 br i1 %tmp7, label %then, label %else
```

```
merge14:
 %a15 = load i32, i32* %a1
 ret i32 %a15
```
```
then:
 %a8 = load i32, i32* %a1
 %b9 = load i32, i32* %b2
 %tmp10 = sub i32 %a8, %b9
 store i32 %tmp10, i32* %a1
 br label %merge
```
```
else:
 %b11 = load i32, i32* %b2
 %a12 = load i32, i32* %a1
 %tmp13 = sub i32 %b11, %a12
 store i32 %tmp13, i32* %b2
 br label %merge
```
```
merge:
 br label %while
```

Figure 3: Control Flow Graph for GCD

6.1.2 demo

Listing 3: demo.fc

```
int twice(func<int(int)> f, int x)
{
    return f(f(x));
}

func<void()> greet(bool formal)
{
    func<void()> greet_formal;
    func<void()> greet_casual;

    greet_formal = () => void { prints("How are you?"); };
    greet_casual = () => void { prints("What's up?"); };

    if (formal)
    {
        return greet_formal;
    }

    return greet_casual;
}

int main()
{
    /* Declarations */
    int r1;
    int r2;
    int r3;
    int len;
    int[5] numbers;
    int[5] squared_numbers;

    /* First-class functions */
    func<int(int)> square;
    func<int(int, int)> sum;
    func<void()> greet_casual;

    /* Array */
    len = 5;
    numbers = [1, 2, 3, 4, 5];

    /* Anonymous functions */
    square = (int a) => int { return a * a; };
    sum = (int a, int b) => int { return a + b; };

    /* Higher-order functions */
    r1 = twice(square, 2);
    print(r1); /* 16 */
    greet_casual = greet(false);
    greet_casual(); /* What's up? */

    /* Map & reduce */
    squared_numbers = map(square, len, numbers);
    r2 = reduce(sum, len, squared_numbers);
    print(r2); /* 55 */

    /* Piping */
    r3 = numbers |> map(square, len) |> reduce(sum, len) |;
    print(r3); /* 55 */

    return 0;
}
```

Listing 4: demo.ll

```
; ModuleID = 'FunC'
source_filename = "FunC"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
```

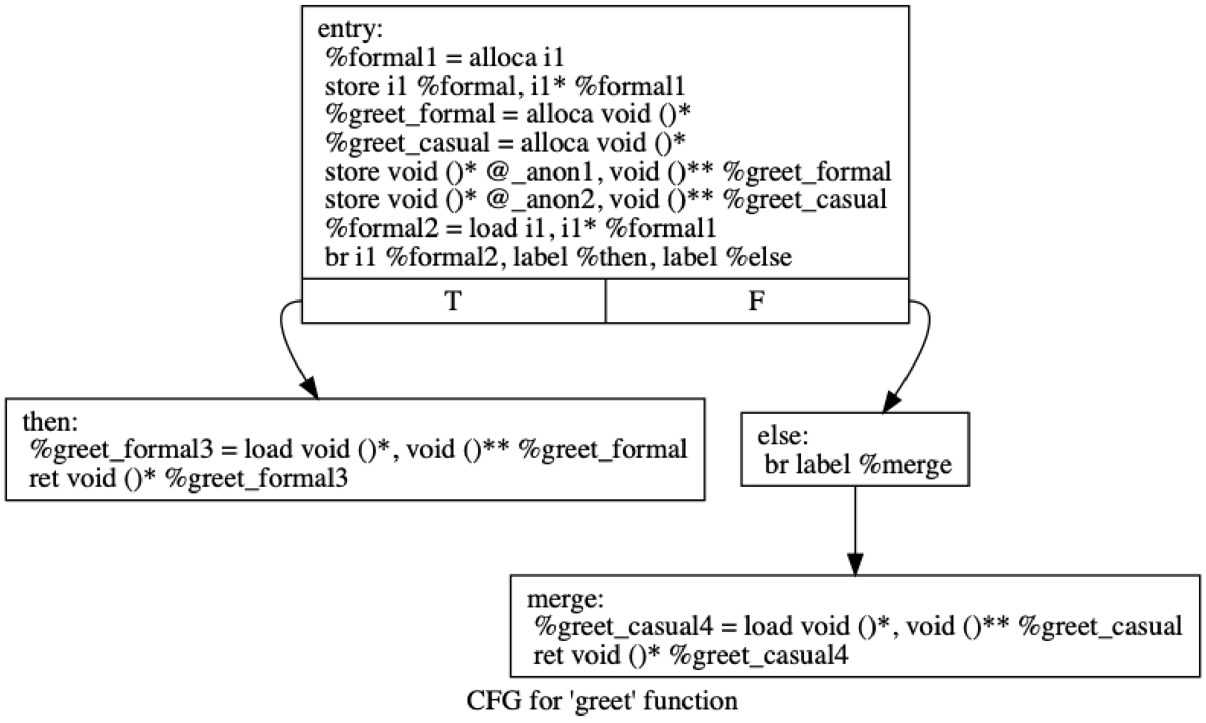
```
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00"
@greet_ptr = global void ()* (i1)* null
@main_ptr = global i32 ()* null
@map_ptr = global i32* (i32 (i32)*, i32, i32*)* null
@reduce_ptr = global i32 (i32 (i32, i32)*, i32, i32*)* null
@twice_ptr = global i32 (i32 (i32)*, i32)* null
@tmp = private unnamed_addr constant [13 x i8] c"How are you?\00"
@tmp.3 = private unnamed_addr constant [11 x i8] c"What's up?\00"

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

declare i32 @printbig(i32)

define i32* @map(i32 (i32)* %f, i32 %length, i32* %arr) {
entry:
  %f1 = alloca i32 (i32)*
  store i32 (i32)* %f, i32 (i32)** %f1
  %length2 = alloca i32
  store i32 %length, i32* %length2
  %arr3 = alloca i32*
  store i32* %arr, i32** %arr3
  %narr = alloca i32*
  %i = alloca i32
  %length4 = load i32, i32* %length2
  %mallocsize = mul i32 %length4, ptrtoint (i32* getelementptr
   ↪  (i32, i32* null, i32 1) to i32)
  %malloccall = tail call i8* @malloc(i32 %mallocsize)
  %arrptr = bitcast i8* %malloccall to i32*
  store i32* %arrptr, i32** %narr
  store i32 0, i32* %i
  br label %while

while:                                                ; preds =
 ↪  %while_body, %entry
  %i5 = load i32, i32* %i
  %length6 = load i32, i32* %length2
  %tmp = icmp slt i32 %i5, %length6
  br i1 %tmp, label %while_body, label %merge

while_body:                                           ; preds =
 ↪  %while
  %tmp7 = load i32 (i32)*, i32 (i32)** %f1
  %arr8 = load i32*, i32** %arr3
  %i9 = load i32, i32* %i
  %tmp10 = getelementptr i32, i32* %arr8, i32 %i9
  %tmp11 = load i32, i32* %tmp10
  %0 = call i32 %tmp7(i32 %tmp11)
  %narr12 = load i32*, i32** %narr
  %i13 = load i32, i32* %i
  %narr14 = getelementptr i32, i32* %narr12, i32 %i13
  store i32 %0, i32* %narr14
  %i15 = load i32, i32* %i
  %tmp16 = add i32 %i15, 1
  store i32 %tmp16, i32* %i
  br label %while

merge:                                            ; preds =
↪  %while
  %narr17 = load i32*, i32** %narr
  ret i32* %narr17
}

define i32 @reduce(i32 (i32, i32)* %f, i32 %length, i32* %arr) {
entry:
  %f1 = alloca i32 (i32, i32)*
  store i32 (i32, i32)* %f, i32 (i32, i32)** %f1
  %length2 = alloca i32
  store i32 %length, i32* %length2
  %arr3 = alloca i32*
  store i32* %arr, i32** %arr3
  %i = alloca i32
  %result = alloca i32
  %length4 = load i32, i32* %length2
  %tmp = icmp sle i32 %length4, 0
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else
  %arr5 = load i32*, i32** %arr3
  %tmp6 = getelementptr i32, i32* %arr5, i32 0
  %tmp7 = load i32, i32* %tmp6
  store i32 %tmp7, i32* %result
  store i32 1, i32* %i
  br label %while

then:                                             ; preds =
↪  %entry
  ret i32 0

else:                                             ; preds =
↪  %entry
  br label %merge

while:                                            ; preds =
↪  %while_body, %merge
  %i8 = load i32, i32* %i
  %length9 = load i32, i32* %length2
  %tmp10 = icmp slt i32 %i8, %length9
  br i1 %tmp10, label %while_body, label %merge19
```

```
while_body:                                       ; preds =
 ↪  %while
  %tmp11 = load i32 (i32, i32)*, i32 (i32, i32)** %f1
  %arr12 = load i32*, i32** %arr3
  %i13 = load i32, i32* %i
  %tmp14 = getelementptr i32, i32* %arr12, i32 %i13
  %tmp15 = load i32, i32* %tmp14
  %result16 = load i32, i32* %result
  %0 = call i32 %tmp11(i32 %result16, i32 %tmp15)
  store i32 %0, i32* %result
  %i17 = load i32, i32* %i
  %tmp18 = add i32 %i17, 1
  store i32 %tmp18, i32* %i
  br label %while

merge19:                                          ; preds =
 ↪  %while
  %result20 = load i32, i32* %result
  ret i32 %result20
}

define i32 @main() {
entry:
  store void ()* (i1)* @greet, void ()* (i1)** @greet_ptr
  store i32 ()* @main, i32 ()** @main_ptr
  store i32* (i32 (i32)*, i32, i32*)* @map, i32* (i32 (i32)*,
   ↪  i32, i32*)** @map_ptr
  store i32 (i32 (i32, i32)*, i32, i32*)* @reduce, i32 (i32 (i32,
   ↪  i32)*, i32, i32*)** @reduce_ptr
  store i32 (i32 (i32)*, i32)* @twice, i32 (i32 (i32)*, i32)**
   ↪  @twice_ptr
  %r1 = alloca i32
  %r2 = alloca i32
  %r3 = alloca i32
  %len = alloca i32
  %numbers = alloca i32*
  %squared_numbers = alloca i32*
  %square = alloca i32 (i32)*
  %sum = alloca i32 (i32, i32)*
  %greet_casual = alloca void ()*
  store i32 5, i32* %len
  %malloccall = tail call i8* @malloc(i32 mul (i32 ptrtoint (i32*
   ↪  getelementptr (i32, i32* null, i32 1) to i32), i32 5))
  %arrptr = bitcast i8* %malloccall to i32*
  %tmp = getelementptr i32, i32* %arrptr, i32 0
  store i32 1, i32* %tmp
  %tmp1 = getelementptr i32, i32* %arrptr, i32 1
  store i32 2, i32* %tmp1
  %tmp2 = getelementptr i32, i32* %arrptr, i32 2
  store i32 3, i32* %tmp2
```

```
%tmp3 = getelementptr i32, i32* %arrptr, i32 3
store i32 4, i32* %tmp3
%tmp4 = getelementptr i32, i32* %arrptr, i32 4
store i32 5, i32* %tmp4
store i32* %arrptr, i32** %numbers
store i32 (i32)* @_anon3, i32 (i32)** %square
store i32 (i32, i32)* @_anon4, i32 (i32, i32)** %sum
%tmp5 = load i32 (i32 (i32)*, i32)*, i32 (i32 (i32)*, i32)**
↪  @twice_ptr
%square6 = load i32 (i32)*, i32 (i32)** %square
%0 = call i32 %tmp5(i32 (i32)* %square6, i32 2)
store i32 %0, i32* %r1
%r17 = load i32, i32* %r1
%printf = call i32 (i8*, ...) @printf(i8* getelementptr
↪  inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
↪  %r17)
%tmp8 = load void ()* (i1)*, void ()* (i1)** @greet_ptr
%1 = call void ()* %tmp8(i1 false)
store void ()* %1, void ()** %greet_casual
%tmp9 = load void ()*, void ()** %greet_casual
call void %tmp9()
%tmp10 = load i32* (i32 (i32)*, i32, i32*)*, i32* (i32 (i32)*,
↪  i32, i32*)** @map_ptr
%numbers11 = load i32*, i32** %numbers
%len12 = load i32, i32* %len
%square13 = load i32 (i32)*, i32 (i32)** %square
%2 = call i32* %tmp10(i32 (i32)* %square13, i32 %len12, i32*
↪  %numbers11)
store i32* %2, i32** %squared_numbers
%tmp14 = load i32 (i32 (i32, i32)*, i32, i32*)*, i32 (i32 (i32,
↪  i32)*, i32, i32*)** @reduce_ptr
%squared_numbers15 = load i32*, i32** %squared_numbers
%len16 = load i32, i32* %len
%sum17 = load i32 (i32, i32)*, i32 (i32, i32)** %sum
%3 = call i32 %tmp14(i32 (i32, i32)* %sum17, i32 %len16, i32*
↪  %squared_numbers15)
store i32 %3, i32* %r2
%r218 = load i32, i32* %r2
%printf19 = call i32 (i8*, ...) @printf(i8* getelementptr
↪  inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
↪  %r218)
%tmp20 = load i32 (i32 (i32, i32)*, i32, i32*)*, i32 (i32 (i32,
↪  i32)*, i32, i32*)** @reduce_ptr
%tmp21 = load i32* (i32 (i32)*, i32, i32*)*, i32* (i32 (i32)*,
↪  i32, i32*)** @map_ptr
%numbers22 = load i32*, i32** %numbers
%len23 = load i32, i32* %len
%square24 = load i32 (i32)*, i32 (i32)** %square
%4 = call i32* %tmp21(i32 (i32)* %square24, i32 %len23, i32*
↪  %numbers22)
```

```
  %len25 = load i32, i32* %len
  %sum26 = load i32 (i32, i32)*, i32 (i32, i32)** %sum
  %5 = call i32 %tmp20(i32 (i32, i32)* %sum26, i32 %len25, i32*
   ↪  %4)
  store i32 %5, i32* %r3
  %r327 = load i32, i32* %r3
  %printf28 = call i32 (i8*, ...) @printf(i8* getelementptr
   ↪  inbounds ([4 x i8], [4 x i8]* @fmt, i32 0, i32 0), i32
   ↪  %r327)
  ret i32 0
}

define void ()* @greet(i1 %formal) {
entry:
  %formal1 = alloca i1
  store i1 %formal, i1* %formal1
  %greet_formal = alloca void ()*
  %greet_casual = alloca void ()*
  store void ()* @_anon1, void ()** %greet_formal
  store void ()* @_anon2, void ()** %greet_casual
  %formal2 = load i1, i1* %formal1
  br i1 %formal2, label %then, label %else

merge:                                            ; preds = %else
  %greet_casual4 = load void ()*, void ()** %greet_casual
  ret void ()* %greet_casual4

then:                                             ; preds =
 ↪  %entry
  %greet_formal3 = load void ()*, void ()** %greet_formal
  ret void ()* %greet_formal3

else:                                             ; preds =
 ↪  %entry
  br label %merge
}

define i32 @twice(i32 (i32)* %f, i32 %x) {
entry:
  %f1 = alloca i32 (i32)*
  store i32 (i32)* %f, i32 (i32)** %f1
  %x2 = alloca i32
  store i32 %x, i32* %x2
  %tmp = load i32 (i32)*, i32 (i32)** %f1
  %tmp3 = load i32 (i32)*, i32 (i32)** %f1
  %x4 = load i32, i32* %x2
  %0 = call i32 %tmp3(i32 %x4)
  %1 = call i32 %tmp(i32 %0)
  ret i32 %1
}
```

```
define void @_anon1() {
entry:
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr
   ↪  inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8*
   ↪  getelementptr inbounds ([13 x i8], [13 x i8]* @tmp, i32 0,
   ↪  i32 0))
  ret void
}

define void @_anon2() {
entry:
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr
   ↪  inbounds ([4 x i8], [4 x i8]* @fmt.2, i32 0, i32 0), i8*
   ↪  getelementptr inbounds ([11 x i8], [11 x i8]* @tmp.3, i32
   ↪  0, i32 0))
  ret void
}

declare noalias i8* @malloc(i32)

define i32 @_anon3(i32 %a) {
entry:
  %a1 = alloca i32
  store i32 %a, i32* %a1
  %a2 = load i32, i32* %a1
  %a3 = load i32, i32* %a1
  %tmp = mul i32 %a2, %a3
  ret i32 %tmp
}

define i32 @_anon4(i32 %a, i32 %b) {
entry:
  %a1 = alloca i32
  store i32 %a, i32* %a1
  %b2 = alloca i32
  store i32 %b, i32* %b2
  %a3 = load i32, i32* %a1
  %b4 = load i32, i32* %b2
  %tmp = add i32 %a3, %b4
  ret i32 %tmp
}
```



```
entry:
 %formal1 = alloca i1
 store i1 %formal, i1* %formal1
 %greet_formal = alloca void ()*
 %greet_casual = alloca void ()*
 store void ()* @_anon1, void ()** %greet_formal
 store void ()* @_anon2, void ()** %greet_casual
 %formal2 = load i1, i1* %formal1
 br i1 %formal2, label %then, label %else
```

```
then:
 %greet_formal3 = load void ()*, void ()** %greet_formal
 ret void ()* %greet_formal3
```

```
else:
 br label %merge
```

```
merge:
 %greet_casual4 = load void ()*, void ()** %greet_casual
 ret void ()* %greet_casual4
```

Figure 4: Control Flow Graph for Greet in Demo

6.2 Test Scripts

Listing 5: Test Scripts

```
fail-anon1.fc:
int main()
{
  func<int(float)> tmp;
  tmp = (float x) => int { return x; }; /* Error: invalid return
  ↪   type */
  print(tmp(1.0));
  return 0;
}

fail-anon1.err:
Fatal error: exception Failure("return gives float expected int
↪   in x")


fail-array1.fc:
int foo()
{
  int[] a;
  a = [1, 2, 3];
  return a[2];
}

int main()
{
  int x;
  x = a[2]; /* Fail: array is out of scope */
}
fail-array1.err:
Fatal error: exception Failure("undeclared identifier a")


fail-assign1.fc:
int main()
{
  int i;
  bool b;

  i = 42;
  i = 10;
  b = true;
  b = false;
  i = false; /* Fail: assigning a bool to an integer */
}

fail-assign1.err:
Fatal error: exception Failure("illegal assignment int = bool in
 ↪  i = false")


fail-assign2.fc:
int main()
{
  int i;
  bool b;

  b = 48; /* Fail: assigning an integer to a bool */
}

fail-assign2.err:
Fatal error: exception Failure("illegal assignment bool = int in
 ↪  b = 48")


fail-assign3.fc:
void myvoid()
{
  return;
}
```

```
int main()
{
  int i;

  i = myvoid(); /* Fail: assigning a void to an integer */
}
```

```
fail-assign3.err:
Fatal error: exception Failure("illegal assignment int = void in
↪  i = myvoid()")
```

```
fail-dead1.fc:
int main()
{
  int i;

  i = 15;
  return i;
  i = 32; /* Error: code after a return */
}
```

```
fail-dead1.err:
Fatal error: exception Failure("nothing may follow a return")
```

```
fail-dead2.fc:
int main()
{
  int i;

  {
    i = 15;
    return i;
  }
  i = 32; /* Error: code after a return */
}
```

```
fail-dead2.err:
Fatal error: exception Failure("nothing may follow a return")
```

```
fail-expr1.fc:
int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + int */
}

int main()
{
  return 0;
}
```

```
fail-expr1.err:
Fatal error: exception Failure("illegal binary operator bool +
 ↪  int in d + a")
```

```
fail-expr2.fc:
int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; /* Error: bool + int */
}

int main()
{
  return 0;
}
```

```
fail-expr2.err:
Fatal error: exception Failure("illegal binary operator bool +
 ↪  int in b + a")
```

```
fail-expr3.fc:
int a;
float b;

void foo(int c, float d)
{
  int d;
  float e;
  b + a; /* Error: float + int */
}
```

```
int main()
{
  return 0;
}

fail-expr3.err:
Fatal error: exception Failure("illegal binary operator float +
↪  int in b + a")


fail-float1.fc:
int main()
{
  -3.5 && 1; /* Float with AND? */
  return 0;
}

fail-float1.err:
Fatal error: exception Failure("illegal binary operator float &&
↪  int in -3.5 && 1")


fail-float2.fc:
int main()
{
  -3.5 && 2.5; /* Float with AND? */
  return 0;
}

fail-float2.err:
Fatal error: exception Failure("illegal binary operator float &&
↪  float in -3.5 && 2.5")


fail-for1.fc:
int main()
{
  int i;
  for ( ; true ; ) {} /* OK: Forever */

  for (i = 0 ; i < 10 ; i = i + 1) {
    if (i == 3) return 42;
  }

  for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}

fail-for1.err:
```

```
Fatal error: exception Failure("undeclared identifier j")


fail-for2.fc:
int main()
{
  int i;

  for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */

  return 0;
}

fail-for2.err:
Fatal error: exception Failure("undeclared identifier j")


fail-for3.fc:
int main()
{
  int i;

  for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean
   ↪  */

  return 0;
}

fail-for3.err:
Fatal error: exception Failure("expected Boolean expression in
 ↪  i")


fail-for4.fc:
int main()
{
  int i;

  for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */

  return 0;
}

fail-for4.err:
Fatal error: exception Failure("undeclared identifier j")


fail-for5.fc:
int main()
{
  int i;

  for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
  }

  return 0;
}

fail-for5.err:
Fatal error: exception Failure("undeclared identifier foo")


fail-fstcls1.fc:
int main()
{
  func<int(int, int)> fst;
  fst = (int x, float y) => int { return x; }; /* Error: argument
   ↪  type mismatch */
  print(fst(1, 2.0));
  return 0;
}

fail-fstcls1.err:
Fatal error: exception Failure("illegal argument found float
 ↪  expected int in 2.0")


fail-fstcls2.fc:
int main()
{
  func<int(int, int)> tmp;
  tmp = (int x, int y) => float { return 1.0; }; /* Error: return
   ↪  type mismatch */
  print(tmp(1, 2));
  return 0;
}

fail-fstcls2.err:
Fatal error: exception Failure("illegal assignment func <int(int,
 ↪  int)> = func <float(int, int)> in tmp = float _anon1(x, y)
{
return 1.0;
}
")


fail-fstcls3.fc:
int main()
{
  func<int(int, int)> tmp;
  tmp = (int x) => int { return x; }; /* Error: args mismatch */
  print(tmp(1));
  return 0;
}

fail-fstcls3.err:
Fatal error: exception Failure("expecting 2 arguments in tmp(1)")


fail-func1.fc:
int foo() {}

int bar() {}

int baz() {}

void bar() {} /* Error: duplicate function bar */

int main()
{
  return 0;
}

fail-func1.err:
Fatal error: exception Failure("duplicate function bar")


fail-func2.fc:
int foo(int a, bool b, int c) { }

void bar(int a, bool b, int a) {} /* Error: duplicate formal a in
↪  bar */

int main()
{
  return 0;
}

fail-func2.err:
Fatal error: exception Failure("duplicate formal a")


fail-func3.fc:
int foo(int a, bool b, int c) { }

void bar(int a, void b, int c) {} /* Error: illegal void formal b
↪  */

int main()
```

```
{
  return 0;
}

fail-func3.err:
Fatal error: exception Failure("illegal void formal b")


fail-func4.fc:
int foo() {}

void bar() {}

int print() {} /* Should not be able to define print */

void baz() {}

int main()
{
  return 0;
}

fail-func4.err:
Fatal error: exception Failure("function print may not be
 ↪  defined")


fail-func5.fc:
int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */
  bool c;

  return 0;
}

int main()
{
  return 0;
}

fail-func5.err:
Fatal error: exception Failure("illegal void local b")


fail-func6.fc:
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}
```

```
fail-func6.err:
Fatal error: exception Failure("expecting 2 arguments in
↪  foo(42)")
```

```
fail-func7.fc:
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}
```

```
fail-func7.err:
Fatal error: exception Failure("expecting 2 arguments in foo(42,
↪  true, false)")
```

```
fail-func8.fc:
void foo(int a, bool b)
{
}

void bar()
{
}

int main()
{
  foo(42, true);
  foo(42, bar()); /* int and void, not int and bool */
}
```

```
fail-func8.err:
Fatal error: exception Failure("illegal argument found void
↪  expected bool in bar()")
```

```
fail-func9.fc:
```

```
void foo(int a, bool b)
{
}

int main()
{
  foo(42, true);
  foo(42, 42); /* Fail: int, not bool */
}
```

```
fail-func9.err:
Fatal error: exception Failure("illegal argument found int
↪  expected bool in 42")
```

```
fail-global1.fc:
int c;
bool b;
void a; /* global variables should not be void */


int main()
{
  return 0;
}
```

```
fail-global1.err:
Fatal error: exception Failure("illegal void global a")
```

```
fail-global2.fc:
int b;
bool c;
int a;
int b; /* Duplicate global variable */

int main()
{
  return 0;
}
```

```
fail-global2.err:
Fatal error: exception Failure("duplicate global b")
```

```
fail-if1.fc:
int main()
{
  if (true) {}
  if (false) {} else {}
  if (42) {} /* Error: non-bool predicate */
}

fail-if1.err:
Fatal error: exception Failure("expected Boolean expression in
 ↪  42")


fail-if2.fc:
int main()
{
  if (true) {
    foo; /* Error: undeclared variable */
  }
}

fail-if2.err:
Fatal error: exception Failure("undeclared identifier foo")


fail-if3.fc:
int main()
{
  if (true) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}

fail-if3.err:
Fatal error: exception Failure("undeclared identifier bar")


fail-map1.fc:
int map(int a, int b) /* Error: map is a reserved keyword */
{
  return a + b;
}

int main()
{
  print(map(3, 5));
  return 0;
}
fail-map1.err:
Fatal error: exception Failure("function map may not be defined")


fail-map2.fc:
```

```
int main()
{
  int[] arr;
  int f;

  arr = [76, 2, 0];
  f = 5;
  map(f, 3, arr); /* Error: first argument must be a function */

  return 0;
}
fail-map2.err:
Fatal error: exception Failure("illegal argument found int
 ↪  expected func <int(int)> in f")


fail-map3.fc:
int add_one(int a)
{
  return a + 1;
}

int main()
{
  int arr;

  map(add_one, 3, arr); /* Error: last argument must be an array
   ↪  */

  return 0;
}
fail-map3.err:
Fatal error: exception Failure("illegal argument found int
 ↪  expected int[] in arr")


fail-mod1.fc:
float fmod(float x, float y)
{
    return x % y;
}

int main()
{
  printf(fmod(36.5, 5.0)); /* mod not supported for floating
   ↪  point */
  return 0;
}
fail-mod1.err:
Fatal error: exception Failure("mod is not supported for float")
```

```
fail-nomain.fc:

fail-nomain.err:
Fatal error: exception Failure("unrecognized function main")


fail-pipe1.fc:
int sum(int a, int b)
{
  return a + b;
}

int main()
{
  int result;
  result = 5 |> sum(10) |> 5+4 |; /* Error: all items, except the
   ↪  first item, need to be a function call */

  return 0;
}

fail-pipe1.err:
Fatal error: exception Failure("expression needs to be a function
 ↪  call")


fail-pipe2.fc:
int sum(int a, int b)
{
  return a + b;
}

int main()
{
  int result;
  int a;
  result = (a = 5) |> sum(10) |; /* Error: assignment is invalid
   ↪  as a first expression for pipe */

  return 0;
}

fail-pipe2.err:
Fatal error: exception Failure("invalid first expression for
 ↪  pipe")

fail-print.fc:
/* Should be illegal to redefine */
void print() {}
```

```
fail-print.err:
Fatal error: exception Failure("function print may not be
 ↪  defined")
```

```
fail-printb.fc:
/* Should be illegal to redefine */
void printb() {}

fail-printb.err:
Fatal error: exception Failure("function printb may not be
 ↪  defined")
```

```
fail-printbig.fc:
/* Should be illegal to redefine */
void printbig() {}

fail-printbig.err:
Fatal error: exception Failure("function printbig may not be
 ↪  defined")
```

```
fail-reduce1.fc:
int reduce(int a, int b) /* Error: reduce is a reserved keyword
 ↪  */
{
  return a + b;
}

int main()
{
  print(reduce(3, 5));
  return 0;
}
fail-reduce1.err:
Fatal error: exception Failure("function reduce may not be
 ↪  defined")
```

```
fail-reduce2.fc:
int main()
{
  int[] arr;
  int f;

  arr = [76, 2, 0];
  f = 5;
  reduce(f, 3, arr); /* Error: first argument must be a function
   ↪  */

  return 0;
}
fail-reduce2.err:
Fatal error: exception Failure("illegal argument found int
 ↪  expected func <int(int, int)> in f")


fail-reduce3.fc:
int add(int a, int b)
{
  return a + b;
}

int main()
{
  int arr;

  reduce(add, 3, arr); /* Error: last argument must be an array
   ↪  */

  return 0;
}
fail-reduce3.err:
Fatal error: exception Failure("illegal argument found int
 ↪  expected int[] in arr")


fail-return1.fc:
int main()
{
  return true; /* Should return int */
}

fail-return1.err:
Fatal error: exception Failure("return gives bool expected int in
 ↪  true")


fail-return2.fc:
void foo()
{
  if (true) return 42; /* Should return void */
  else return;
}

int main()
{
  return 42;
}

fail-return2.err:
Fatal error: exception Failure("return gives int expected void in
 ↪  42")


fail-while1.fc:
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (42) { /* Should be boolean */
    i = i + 1;
  }

}

fail-while1.err:
Fatal error: exception Failure("expected Boolean expression in
 ↪  42")


fail-while2.fc:
int main()
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (true) {
    foo(); /* foo undefined */
  }

}

fail-while2.err:
Fatal error: exception Failure("undeclared identifier foo")


test-add1.fc:
int add(int x, int y)
{
  return x + y;
}

int main()
{
  print( add(17, 25) );
  return 0;
}
```

```
test-add1.out:
42
```

```
test-anon1.fc:
int main()
{
  func<int(int, int)> subtract;

  subtract = (int x, int y) => int { return x - y; };
  print(subtract(3, 2));

  return 0;
}
```

```
test-anon1.out:
1
```

```
test-arith1.fc:
int main()
{
  print(39 + 3);
  return 0;
}
```

```
test-arith1.out:
42
```

```
test-arith2.fc:
int main()
{
  print(1 + 2 * 3 + 4);
  return 0;
}
```

```
test-arith2.out:
11
```

test-arith3.fc:
```
int foo(int a)
{
  return a;
}

int main()
{
  int a;
  a = 42;
  a = a + 5;
  print(a);
  return 0;
}
```

test-arith3.out:
```
47
```

test-array1.fc:
```
int main()
{
  int[] a;
  a = [1, 2, 3];

  print(a[0]);
  print(a[1]);
  print(a[2]);

  return 0;
}
```

test-array1.out:
```
1
2
3
```

test-array2.fc:
```
int main()
{
  float[] a;
  a = [1.1, 2.2, 3.3];

  printf(a[0]);
  printf(a[1]);
  printf(a[2]);

  return 0;
}
```

```
test-array2.out:
1.1
2.2
3.3


test-array3.fc:
int main()
{
  int[] a;
  int x;
  int y;
  int z;
  a = [1, 2, 3];
  x = a[0];
  y = 5 + a[1];
  z = y + a[1] + a[2];

  print(x);
  print(y);
  print(z);

  return 0;
}
test-array3.out:
1
7
12


test-array4.fc:
int foo()
{
  int[] a;
  int i;

  a = new int[3];

  for (i = 0; i < 3; i = i + 1) {
    a[i] = i + 1;
  }

  return a[2];
}

int main()
{
  int x;
  x = foo();
  print(x);
  return 0;
}
test-array4.out:
3


test-array5.fc:
void square(int[] arr, int length)
{
  int i;

  for(i = 0; i < length; i = i + 1)
  {
    arr[i] = arr[i] * arr[i];
  }
}

int main()
{
  int[] arr;
  int i;

  arr = [1, 2, 3];

  square(arr, 3);

  for(i = 0; i < 3; i = i + 1)
  {
    print(arr[i]);
  }

  return 0;
}
test-array5.out:
1
4
9


test-array6.fc:
int[] square(int[] arr, int length)
{
  int[] narr;
  int i;

  narr = new int[length];

  for(i = 0; i < length; i = i + 1)
  {
    narr[i] = arr[i] * arr[i];
  }

  return narr;
}

int main()
{
  int[3] arr;
  int[3] narr;
  int i;

  arr = [1, 2, 3];
  narr = square(arr, 3);

  for(i = 0; i < 3; i = i + 1)
  {
    print(narr[i]);
  }

  return 0;
}
test-array6.out:
1
4
9


test-fib.fc:
int fib(int x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

int main()
{
  print(fib(0));
  print(fib(1));
  print(fib(2));
  print(fib(3));
  print(fib(4));
  print(fib(5));
  return 0;
}

test-fib.out:
1
1
```

```
2
3
5
8
```

```
test-float1.fc:
int main()
{
  float a;
  a = 3.14159267;
  printf(a);
  return 0;
}

test-float1.out:
3.14159
```

```
test-float2.fc:
int main()
{
  float a;
  float b;
  float c;
  a = 3.14159267;
  b = -2.71828;
  c = a + b;
  printf(c);
  return 0;
}

test-float2.out:
0.423313
```

```
test-float3.fc:
void testfloat(float a, float b)
{
  printf(a + b);
  printf(a - b);
  printf(a * b);
  printf(a / b);
  printb(a == b);
  printb(a == a);
  printb(a != b);
  printb(a != a);
  printb(a > b);
  printb(a >= b);
  printb(a < b);
  printb(a <= b);
}

int main()
{
  float c;
  float d;

  c = 42.0;
  d = 3.14159;

  testfloat(c, d);

  testfloat(d, d);

  return 0;
}

test-float3.out:
45.1416
38.8584
131.947
13.369
0
1
1
0
1
1
0
0
6.28318
0
9.86959
1
1
1
0
0
0
1
0
1


test-for1.fc:
int main()
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
  }
  print(42);
  return 0;
}

test-for1.out:
0
1
2
3
4
42


test-for2.fc:
int main()
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    print(i);
    i = i + 1;
  }
  print(42);
  return 0;
}

test-for2.out:
0
1
2
3
4
42


test-fstcls1.fc:
int main()
{
  func<int(int)> square;
  func<float(int, float)> snd;
  int a;

  square = (int x) => int { return x * x; };
  print(square(2));

  snd = (int x, float y) => float { return y; };
  printf(snd(1, 2.5));
  return 0;
}

test-fstcls1.out:
4
2.5


test-func1.fc:
int add(int a, int b)
{
  return a + b;
}

int main()
{
  int a;
  a = add(39, 3);
  print(a);
  return 0;
}

test-func1.out:
42


test-func2.fc:
/* Bug noticed by Pin-Chin Huang */

int fun(int x, int y)
{
  return 0;
}

int main()
{
  int i;
  i = 1;

  fun(i = 2, i = i+1);

  print(i);
  return 0;
}


test-func2.out:
2
```

```
test-func3.fc:
void printem(int a, int b, int c, int d)
{
  print(a);
  print(b);
  print(c);
  print(d);
}

int main()
{
  printem(42,17,192,8);
  return 0;
}

test-func3.out:
42
17
192
8


test-func4.fc:
int add(int a, int b)
{
  int c;
  c = a + b;
  return c;
}

int main()
{
  int d;
  d = add(52, 10);
  print(d);
  return 0;
}

test-func4.out:
62


test-func5.fc:
int foo(int a)
{
  return a;
}

int main()
{
  return 0;
}

test-func5.out:


test-func6.fc:
void foo() {}

int bar(int a, bool b, int c) { return a + c; }

int main()
{
  print(bar(17, false, 25));
  return 0;
}

test-func6.out:
42


test-func7.fc:
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print(a);
  return 0;
}

test-func7.out:
115


test-func8.fc:
void foo(int a)
{
  print(a + 3);
}

int main()
{
  foo(40);
  return 0;
```

```
}

test-func8.out:
43


test-func9.fc:
void foo(int a)
{
  print(a + 3);
  return;
}

int main()
{
  foo(40);
  return 0;
}

test-func9.out:
43


test-gcd.fc:
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

int main()
{
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));
  return 0;
}

test-gcd.out:
2
3
11


test-gcd2.fc:
int gcd(int a, int b) {
  while (a != b)
    if (a > b) a = a - b;
    else b = b - a;
  return a;
}

int main()
{
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
  return 0;
}
```

```
test-gcd2.out:
7
4
11
```

```
test-global1.fc:
int a;
int b;

void printa()
{
  print(a);
}

void printbb()
{
  print(b);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

int main()
{
  a = 42;
  b = 21;
  printa();
  printbb();
  incab();
  printa();
  printbb();
  return 0;
}
```

```
test-global1.out:
42
21
43
22


test-global2.fc:
bool i;

int main()
{
  int i; /* Should hide the global i */

  i = 42;
  print(i + i);
  return 0;
}

test-global2.out:
84


test-global3.fc:
int i;
bool b;
int j;

int main()
{
  i = 42;
  j = 10;
  print(i + j);
  return 0;
}

test-global3.out:
52


test-hello.fc:
int main()
{
  print(42);
  print(71);
  print(1);
  return 0;
}

test-hello.out:
```

```
42
71
1


test-helloworld.fc:
int main()
{
  prints("Hello World!");
  return 0;
}

test-helloworld.out:
Hello World!


test-higherorder1.fc:
int cubic(func<int(int)> square, int x)
{
  return square(x) * x;
}

int main()
{
  func<int(int)> square;

  square = (int x) => int { return x * x; };
  print(cubic(square, 2));

  return 0;
}

test-higherorder1.out:
8


test-higherorder2.fc:
func<void()> greet(bool formal)
{
  func<void()> greet_formal;
  func<void()> greet_casual;

  greet_formal = () => void { prints("How are you?"); };
  greet_casual = () => void { prints("What's up?"); };

  if (formal)
  {
    return greet_formal;
  }
  return greet_casual;
}

int main()
{
  bool formal;
  func<void()> greet_casual;

  formal = false;
  greet_casual = greet(formal);
  greet_casual();

  return 0;
}
```

```
test-higherorder2.out:
What's up?
```

```
test-if1.fc:
int main()
{
  if (true) print(42);
  print(17);
  return 0;
}
```

```
test-if1.out:
42
17
```

```
test-if2.fc:
int main()
{
  if (true) print(42); else print(8);
  print(17);
  return 0;
}
```

```
test-if2.out:
42
17
```

```
test-if3.fc:
int main()
{
  if (false) print(42);
  print(17);
  return 0;
}

test-if3.out:
17


test-if4.fc:
int main()
{
  if (false) print(42); else print(8);
  print(17);
  return 0;
}

test-if4.out:
8
17


test-if5.fc:
int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}

test-if5.out:
42
17


test-if6.fc:
int cond(bool b)
{
  int x;
  x = 10;
  if (b)
```

```
    if (x == 10)
      x = 42;
  else
    x = 17;
  return x;
}

int main()
{
 print(cond(true));
 print(cond(false));
 return 0;
}

test-if6.out:
42
10


test-local1.fc:
void foo(bool i)
{
  int i; /* Should hide the formal i */

  i = 42;
  print(i + i);
}

int main()
{
  foo(true);
  return 0;
}

test-local1.out:
84


test-local2.fc:
int foo(int a, bool b)
{
  int c;
  bool d;

  c = a;

  return c + 10;
}

int main() {
 print(foo(37, false));
 return 0;
}

test-local2.out:
47


test-map1.fc:
int add_two(int a)
{
  return a + 2;
}

int main()
{
  int[] a;
  int[] result;
  int i;

  a = [4, 7, 10];

  result = map(add_two, 3, a);

  for (i = 0; i < 3; i = i + 1)
  {
    print(result[i]);
  }

  return 0;
}
test-map1.out:
6
9
12


test-mod1.fc:
int mod(int x, int y)
{
  return x % y;
}

int main()
{
  print(mod(9, 5));
  return 0;
}

test-mod1.out:
```

```
4
```

```
test-ops1.fc:
int main()
{
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  printb(1 == 2);
  printb(1 == 1);
  print(99);
  printb(1 != 2);
  printb(1 != 1);
  print(99);
  printb(1 < 2);
  printb(2 < 1);
  print(99);
  printb(1 <= 2);
  printb(1 <= 1);
  printb(2 <= 1);
  print(99);
  printb(1 > 2);
  printb(2 > 1);
  print(99);
  printb(1 >= 2);
  printb(1 >= 1);
  printb(2 >= 1);
  return 0;
}
```

```
test-ops1.out:
3
-1
2
50
99
0
1
99
1
0
99
1
0
99
1
1
```

```
0
99
0
1
99
0
1
1
```

test-ops2.fc:

```
int main()
{
  printb(true);
  printb(false);
  printb(true && true);
  printb(true && false);
  printb(false && true);
  printb(false && false);
  printb(true || true);
  printb(true || false);
  printb(false || true);
  printb(false || false);
  printb(!false);
  printb(!true);
  print(-10);
  print(--42);
}
```

test-ops2.out:

```
1
0
1
0
0
0
1
1
1
0
1
0
-10
42
```

test-pipe1.fc:

```
int sum(int a, int b)
{
  return a + b;
```

```
}

int main()
{
  int result;
  result = 5 |> sum(10) |> sum(20) |> sum(30) |;
  print(result);
  return 0;
}

test-pipe1.out:
65


test-pipe2.fc:
int sum(int a, int b, int c)
{
  return a + b + c;
}

int main()
{
  int result;
  result = 5 |> sum(10, 6) |;
  print(result);
  return 0;
}

test-pipe2.out:
21


test-pipe3.fc:
int[] add_one(int length, int[] arr)
{
  int[] narr;
  int i;

  narr = new int[length];

  for(i = 0; i < length; i = i + 1)
  {
    narr[i] = arr[i] + 1;
  }

  return narr;
}

int main()
{
  int[3] result;
  int i;

  result = [1, 2, 3] |> add_one(3) |> add_one(3) |;

  for(i = 0; i < 3; i = i + 1)
  {
    print(result[i]);
  }

  return 0;
}

test-pipe3.out:
3
4
5


test-pipe4.fc:
int square(int a)
{
  return a * a;
}

int main()
{
  int result;
  result = square(3) |> square() |;
  print(result);
  return 0;
}

test-pipe4.out:
81


test-pipe5.fc:
int square(int a)
{
  return a * a;
}

int sum(int a, int b)
{
  return a + b;
}

int main()
{
  int[] numbers;
  int result;

  numbers = [1, 2, 3, 4, 5];
  result = numbers |> map(square, 5) |> reduce(sum, 5) |;
  print(result);

  return 0;
}
test-pipe5.out:
55


test-printbig.fc:
/*
 * Test for linking external C functions to LLVM-generated code
 *
 * printbig is defined as an external function, much like printf
 * The C compiler generates printbig.o
 * The LLVM compiler, llc, translates the .ll to an assembly .s
↪  file
 * The C compiler assembles the .s file and links the .o file to
↪  generate
 * an executable
 */

int main()
{
  printbig(72); /* H */
  printbig(69); /* E */
  printbig(76); /* L */
  printbig(76); /* L */
  printbig(79); /* O */
  printbig(32); /*   */
  printbig(87); /* W */
  printbig(79); /* O */
  printbig(82); /* R */
  printbig(76); /* L */
  printbig(68); /* D */
  return 0;
}

test-printbig.out:
  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
        XX
        XX
        XX
  XXXXXXXXXXXXXX
  XXXXXXXXXXXXXX
```

```
XXXXXXXXXXXXXX
XXXXXXXXXXXXXX
XX    XX    XX
XX    XX    XX
XX    XX    XX
XX          XX


XXXXXXXXXXXXXX
XXXXXXXXXXXXXX
XX
XX
XX
XX


XXXXXXXXXXXXXX
XXXXXXXXXXXXXX
XX
XX
XX
XX

  XXXXXXXXXX
XXXXXXXXXXXXXX
XX          XX
XX          XX
XX          XX
XXXXXXXXXXXXXX
  XXXXXXXXXX









    XXXXXXXXXX
XXXXXXXXXXXXXX
  XXXXXX
    XXXXXX
  XXXXXX
XXXXXXXXXXXXXX
    XXXXXXXXXX

  XXXXXXXXXX
XXXXXXXXXXXXXX
XX          XX
XX          XX
XX          XX
XXXXXXXXXXXXXX
  XXXXXXXXXX

XXXXXXXXXXXXXX
XXXXXXXXXXXXXX
    XX      XX
  XXXX      XX
XXXXXXXX    XX
XXXX  XXXXXXXX
XX    XXXXXX


XXXXXXXXXXXXXX
XXXXXXXXXXXXXX
XX
XX
XX
XX

XXXXXXXXXXXXXX
XXXXXXXXXXXXXX
XX          XX
XX          XX
XXXX      XXXX
  XXXXXXXXXX
    XXXXXX
```

```
test-reduce1.fc:
int sum(int a, int b)
{
  return a + b;
}

int main()
{
  int[] arr;
  int result;

  arr = [4, 7, 10];

  result = reduce(sum, 3, arr);
  print(result);

  return 0;
}
test-reduce1.out:
21


test-var1.fc:
int main()
{
  int a;
  a = 42;
  print(a);
  return 0;
}

test-var1.out:
42


test-var2.fc:
int a;

void foo(int c)
{
  a = c + 42;
}

int main()
{
  foo(73);
  print(a);
  return 0;
}

test-var2.out:
115


test-while1.fc:
int main()
{
  int i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  return 0;
}
```

```
test-while1.out:
5
4
3
2
1
42


test-while2.fc:
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

int main()
{
  print(foo(7));
  return 0;
}

test-while2.out:
14
```

Listing 6: Test Log

```
/usr/local/opt/llvm/bin/lli

###### Testing test-add1
./func.native tests/test-add1.fc > test-add1.ll
llc -relocation-model=pic test-add1.ll > test-add1.s
cc -o test-add1.exe test-add1.s printbig.o
./test-add1.exe
diff -b test-add1.out tests/test-add1.out > test-add1.diff
###### SUCCESS

###### Testing test-anon1
./func.native tests/test-anon1.fc > test-anon1.ll
llc -relocation-model=pic test-anon1.ll > test-anon1.s
cc -o test-anon1.exe test-anon1.s printbig.o
./test-anon1.exe
diff -b test-anon1.out tests/test-anon1.out > test-anon1.diff
###### SUCCESS
```

```
###### Testing test-arith1
./func.native tests/test-arith1.fc > test-arith1.ll
llc -relocation-model=pic test-arith1.ll > test-arith1.s
cc -o test-arith1.exe test-arith1.s printbig.o
./test-arith1.exe
diff -b test-arith1.out tests/test-arith1.out > test-arith1.diff
###### SUCCESS

###### Testing test-arith2
./func.native tests/test-arith2.fc > test-arith2.ll
llc -relocation-model=pic test-arith2.ll > test-arith2.s
cc -o test-arith2.exe test-arith2.s printbig.o
./test-arith2.exe
diff -b test-arith2.out tests/test-arith2.out > test-arith2.diff
###### SUCCESS

###### Testing test-arith3
./func.native tests/test-arith3.fc > test-arith3.ll
llc -relocation-model=pic test-arith3.ll > test-arith3.s
cc -o test-arith3.exe test-arith3.s printbig.o
./test-arith3.exe
diff -b test-arith3.out tests/test-arith3.out > test-arith3.diff
###### SUCCESS

###### Testing test-array1
./func.native tests/test-array1.fc > test-array1.ll
llc -relocation-model=pic test-array1.ll > test-array1.s
cc -o test-array1.exe test-array1.s printbig.o
./test-array1.exe
diff -b test-array1.out tests/test-array1.out > test-array1.diff
###### SUCCESS

###### Testing test-array2
./func.native tests/test-array2.fc > test-array2.ll
llc -relocation-model=pic test-array2.ll > test-array2.s
cc -o test-array2.exe test-array2.s printbig.o
./test-array2.exe
diff -b test-array2.out tests/test-array2.out > test-array2.diff
###### SUCCESS

###### Testing test-array3
./func.native tests/test-array3.fc > test-array3.ll
llc -relocation-model=pic test-array3.ll > test-array3.s
cc -o test-array3.exe test-array3.s printbig.o
./test-array3.exe
diff -b test-array3.out tests/test-array3.out > test-array3.diff
###### SUCCESS

###### Testing test-array4
```

```
./func.native tests/test-array4.fc > test-array4.ll
llc -relocation-model=pic test-array4.ll > test-array4.s
cc -o test-array4.exe test-array4.s printbig.o
./test-array4.exe
diff -b test-array4.out tests/test-array4.out > test-array4.diff
###### SUCCESS

###### Testing test-array5
./func.native tests/test-array5.fc > test-array5.ll
llc -relocation-model=pic test-array5.ll > test-array5.s
cc -o test-array5.exe test-array5.s printbig.o
./test-array5.exe
diff -b test-array5.out tests/test-array5.out > test-array5.diff
###### SUCCESS

###### Testing test-array6
./func.native tests/test-array6.fc > test-array6.ll
llc -relocation-model=pic test-array6.ll > test-array6.s
cc -o test-array6.exe test-array6.s printbig.o
./test-array6.exe
diff -b test-array6.out tests/test-array6.out > test-array6.diff
###### SUCCESS

###### Testing test-fib
./func.native tests/test-fib.fc > test-fib.ll
llc -relocation-model=pic test-fib.ll > test-fib.s
cc -o test-fib.exe test-fib.s printbig.o
./test-fib.exe
diff -b test-fib.out tests/test-fib.out > test-fib.diff
###### SUCCESS

###### Testing test-float1
./func.native tests/test-float1.fc > test-float1.ll
llc -relocation-model=pic test-float1.ll > test-float1.s
cc -o test-float1.exe test-float1.s printbig.o
./test-float1.exe
diff -b test-float1.out tests/test-float1.out > test-float1.diff
###### SUCCESS

###### Testing test-float2
./func.native tests/test-float2.fc > test-float2.ll
llc -relocation-model=pic test-float2.ll > test-float2.s
cc -o test-float2.exe test-float2.s printbig.o
./test-float2.exe
diff -b test-float2.out tests/test-float2.out > test-float2.diff
###### SUCCESS

###### Testing test-float3
./func.native tests/test-float3.fc > test-float3.ll
llc -relocation-model=pic test-float3.ll > test-float3.s
```

```
cc -o test-float3.exe test-float3.s printbig.o
./test-float3.exe
diff -b test-float3.out tests/test-float3.out > test-float3.diff
###### SUCCESS

###### Testing test-for1
./func.native tests/test-for1.fc > test-for1.ll
llc -relocation-model=pic test-for1.ll > test-for1.s
cc -o test-for1.exe test-for1.s printbig.o
./test-for1.exe
diff -b test-for1.out tests/test-for1.out > test-for1.diff
###### SUCCESS

###### Testing test-for2
./func.native tests/test-for2.fc > test-for2.ll
llc -relocation-model=pic test-for2.ll > test-for2.s
cc -o test-for2.exe test-for2.s printbig.o
./test-for2.exe
diff -b test-for2.out tests/test-for2.out > test-for2.diff
###### SUCCESS

###### Testing test-fstcls1
./func.native tests/test-fstcls1.fc > test-fstcls1.ll
llc -relocation-model=pic test-fstcls1.ll > test-fstcls1.s
cc -o test-fstcls1.exe test-fstcls1.s printbig.o
./test-fstcls1.exe
diff -b test-fstcls1.out tests/test-fstcls1.out >
↪  test-fstcls1.diff
###### SUCCESS

###### Testing test-func1
./func.native tests/test-func1.fc > test-func1.ll
llc -relocation-model=pic test-func1.ll > test-func1.s
cc -o test-func1.exe test-func1.s printbig.o
./test-func1.exe
diff -b test-func1.out tests/test-func1.out > test-func1.diff
###### SUCCESS

###### Testing test-func2
./func.native tests/test-func2.fc > test-func2.ll
llc -relocation-model=pic test-func2.ll > test-func2.s
cc -o test-func2.exe test-func2.s printbig.o
./test-func2.exe
diff -b test-func2.out tests/test-func2.out > test-func2.diff
###### SUCCESS

###### Testing test-func3
./func.native tests/test-func3.fc > test-func3.ll
llc -relocation-model=pic test-func3.ll > test-func3.s
cc -o test-func3.exe test-func3.s printbig.o
```

```
./test-func3.exe
diff -b test-func3.out tests/test-func3.out > test-func3.diff
###### SUCCESS

###### Testing test-func4
./func.native tests/test-func4.fc > test-func4.ll
llc -relocation-model=pic test-func4.ll > test-func4.s
cc -o test-func4.exe test-func4.s printbig.o
./test-func4.exe
diff -b test-func4.out tests/test-func4.out > test-func4.diff
###### SUCCESS

###### Testing test-func5
./func.native tests/test-func5.fc > test-func5.ll
llc -relocation-model=pic test-func5.ll > test-func5.s
cc -o test-func5.exe test-func5.s printbig.o
./test-func5.exe
diff -b test-func5.out tests/test-func5.out > test-func5.diff
###### SUCCESS

###### Testing test-func6
./func.native tests/test-func6.fc > test-func6.ll
llc -relocation-model=pic test-func6.ll > test-func6.s
cc -o test-func6.exe test-func6.s printbig.o
./test-func6.exe
diff -b test-func6.out tests/test-func6.out > test-func6.diff
###### SUCCESS

###### Testing test-func7
./func.native tests/test-func7.fc > test-func7.ll
llc -relocation-model=pic test-func7.ll > test-func7.s
cc -o test-func7.exe test-func7.s printbig.o
./test-func7.exe
diff -b test-func7.out tests/test-func7.out > test-func7.diff
###### SUCCESS

###### Testing test-func8
./func.native tests/test-func8.fc > test-func8.ll
llc -relocation-model=pic test-func8.ll > test-func8.s
cc -o test-func8.exe test-func8.s printbig.o
./test-func8.exe
diff -b test-func8.out tests/test-func8.out > test-func8.diff
###### SUCCESS

###### Testing test-func9
./func.native tests/test-func9.fc > test-func9.ll
llc -relocation-model=pic test-func9.ll > test-func9.s
cc -o test-func9.exe test-func9.s printbig.o
./test-func9.exe
diff -b test-func9.out tests/test-func9.out > test-func9.diff
```

```
###### SUCCESS

###### Testing test-gcd
./func.native tests/test-gcd.fc > test-gcd.ll
llc -relocation-model=pic test-gcd.ll > test-gcd.s
cc -o test-gcd.exe test-gcd.s printbig.o
./test-gcd.exe
diff -b test-gcd.out tests/test-gcd.out > test-gcd.diff
###### SUCCESS

###### Testing test-gcd2
./func.native tests/test-gcd2.fc > test-gcd2.ll
llc -relocation-model=pic test-gcd2.ll > test-gcd2.s
cc -o test-gcd2.exe test-gcd2.s printbig.o
./test-gcd2.exe
diff -b test-gcd2.out tests/test-gcd2.out > test-gcd2.diff
###### SUCCESS

###### Testing test-global1
./func.native tests/test-global1.fc > test-global1.ll
llc -relocation-model=pic test-global1.ll > test-global1.s
cc -o test-global1.exe test-global1.s printbig.o
./test-global1.exe
diff -b test-global1.out tests/test-global1.out >
 ↪  test-global1.diff
###### SUCCESS

###### Testing test-global2
./func.native tests/test-global2.fc > test-global2.ll
llc -relocation-model=pic test-global2.ll > test-global2.s
cc -o test-global2.exe test-global2.s printbig.o
./test-global2.exe
diff -b test-global2.out tests/test-global2.out >
 ↪  test-global2.diff
###### SUCCESS

###### Testing test-global3
./func.native tests/test-global3.fc > test-global3.ll
llc -relocation-model=pic test-global3.ll > test-global3.s
cc -o test-global3.exe test-global3.s printbig.o
./test-global3.exe
diff -b test-global3.out tests/test-global3.out >
 ↪  test-global3.diff
###### SUCCESS

###### Testing test-hello
./func.native tests/test-hello.fc > test-hello.ll
llc -relocation-model=pic test-hello.ll > test-hello.s
cc -o test-hello.exe test-hello.s printbig.o
./test-hello.exe
```

```
diff -b test-hello.out tests/test-hello.out > test-hello.diff
###### SUCCESS

###### Testing test-helloworld
./func.native tests/test-helloworld.fc > test-helloworld.ll
llc -relocation-model=pic test-helloworld.ll > test-helloworld.s
cc -o test-helloworld.exe test-helloworld.s printbig.o
./test-helloworld.exe
diff -b test-helloworld.out tests/test-helloworld.out >
 ↪  test-helloworld.diff
###### SUCCESS

###### Testing test-higherorder1
./func.native tests/test-higherorder1.fc > test-higherorder1.ll
llc -relocation-model=pic test-higherorder1.ll >
 ↪  test-higherorder1.s
cc -o test-higherorder1.exe test-higherorder1.s printbig.o
./test-higherorder1.exe
diff -b test-higherorder1.out tests/test-higherorder1.out >
 ↪  test-higherorder1.diff
###### SUCCESS

###### Testing test-higherorder2
./func.native tests/test-higherorder2.fc > test-higherorder2.ll
llc -relocation-model=pic test-higherorder2.ll >
 ↪  test-higherorder2.s
cc -o test-higherorder2.exe test-higherorder2.s printbig.o
./test-higherorder2.exe
diff -b test-higherorder2.out tests/test-higherorder2.out >
 ↪  test-higherorder2.diff
###### SUCCESS

###### Testing test-if1
./func.native tests/test-if1.fc > test-if1.ll
llc -relocation-model=pic test-if1.ll > test-if1.s
cc -o test-if1.exe test-if1.s printbig.o
./test-if1.exe
diff -b test-if1.out tests/test-if1.out > test-if1.diff
###### SUCCESS

###### Testing test-if2
./func.native tests/test-if2.fc > test-if2.ll
llc -relocation-model=pic test-if2.ll > test-if2.s
cc -o test-if2.exe test-if2.s printbig.o
./test-if2.exe
diff -b test-if2.out tests/test-if2.out > test-if2.diff
###### SUCCESS

###### Testing test-if3
./func.native tests/test-if3.fc > test-if3.ll
```

```
llc -relocation-model=pic test-if3.ll > test-if3.s
cc -o test-if3.exe test-if3.s printbig.o
./test-if3.exe
diff -b test-if3.out tests/test-if3.out > test-if3.diff
###### SUCCESS

###### Testing test-if4
./func.native tests/test-if4.fc > test-if4.ll
llc -relocation-model=pic test-if4.ll > test-if4.s
cc -o test-if4.exe test-if4.s printbig.o
./test-if4.exe
diff -b test-if4.out tests/test-if4.out > test-if4.diff
###### SUCCESS

###### Testing test-if5
./func.native tests/test-if5.fc > test-if5.ll
llc -relocation-model=pic test-if5.ll > test-if5.s
cc -o test-if5.exe test-if5.s printbig.o
./test-if5.exe
diff -b test-if5.out tests/test-if5.out > test-if5.diff
###### SUCCESS

###### Testing test-if6
./func.native tests/test-if6.fc > test-if6.ll
llc -relocation-model=pic test-if6.ll > test-if6.s
cc -o test-if6.exe test-if6.s printbig.o
./test-if6.exe
diff -b test-if6.out tests/test-if6.out > test-if6.diff
###### SUCCESS

###### Testing test-local1
./func.native tests/test-local1.fc > test-local1.ll
llc -relocation-model=pic test-local1.ll > test-local1.s
cc -o test-local1.exe test-local1.s printbig.o
./test-local1.exe
diff -b test-local1.out tests/test-local1.out > test-local1.diff
###### SUCCESS

###### Testing test-local2
./func.native tests/test-local2.fc > test-local2.ll
llc -relocation-model=pic test-local2.ll > test-local2.s
cc -o test-local2.exe test-local2.s printbig.o
./test-local2.exe
diff -b test-local2.out tests/test-local2.out > test-local2.diff
###### SUCCESS

###### Testing test-map1
./func.native tests/test-map1.fc > test-map1.ll
llc -relocation-model=pic test-map1.ll > test-map1.s
cc -o test-map1.exe test-map1.s printbig.o
```

```
./test-map1.exe
diff -b test-map1.out tests/test-map1.out > test-map1.diff
###### SUCCESS

###### Testing test-mod1
./func.native tests/test-mod1.fc > test-mod1.ll
llc -relocation-model=pic test-mod1.ll > test-mod1.s
cc -o test-mod1.exe test-mod1.s printbig.o
./test-mod1.exe
diff -b test-mod1.out tests/test-mod1.out > test-mod1.diff
###### SUCCESS

###### Testing test-ops1
./func.native tests/test-ops1.fc > test-ops1.ll
llc -relocation-model=pic test-ops1.ll > test-ops1.s
cc -o test-ops1.exe test-ops1.s printbig.o
./test-ops1.exe
diff -b test-ops1.out tests/test-ops1.out > test-ops1.diff
###### SUCCESS

###### Testing test-ops2
./func.native tests/test-ops2.fc > test-ops2.ll
llc -relocation-model=pic test-ops2.ll > test-ops2.s
cc -o test-ops2.exe test-ops2.s printbig.o
./test-ops2.exe
diff -b test-ops2.out tests/test-ops2.out > test-ops2.diff
###### SUCCESS

###### Testing test-pipe1
./func.native tests/test-pipe1.fc > test-pipe1.ll
llc -relocation-model=pic test-pipe1.ll > test-pipe1.s
cc -o test-pipe1.exe test-pipe1.s printbig.o
./test-pipe1.exe
diff -b test-pipe1.out tests/test-pipe1.out > test-pipe1.diff
###### SUCCESS

###### Testing test-pipe2
./func.native tests/test-pipe2.fc > test-pipe2.ll
llc -relocation-model=pic test-pipe2.ll > test-pipe2.s
cc -o test-pipe2.exe test-pipe2.s printbig.o
./test-pipe2.exe
diff -b test-pipe2.out tests/test-pipe2.out > test-pipe2.diff
###### SUCCESS

###### Testing test-pipe3
./func.native tests/test-pipe3.fc > test-pipe3.ll
llc -relocation-model=pic test-pipe3.ll > test-pipe3.s
cc -o test-pipe3.exe test-pipe3.s printbig.o
./test-pipe3.exe
diff -b test-pipe3.out tests/test-pipe3.out > test-pipe3.diff
```

```
###### SUCCESS

###### Testing test-pipe4
./func.native tests/test-pipe4.fc > test-pipe4.ll
llc -relocation-model=pic test-pipe4.ll > test-pipe4.s
cc -o test-pipe4.exe test-pipe4.s printbig.o
./test-pipe4.exe
diff -b test-pipe4.out tests/test-pipe4.out > test-pipe4.diff
###### SUCCESS

###### Testing test-pipe5
./func.native tests/test-pipe5.fc > test-pipe5.ll
llc -relocation-model=pic test-pipe5.ll > test-pipe5.s
cc -o test-pipe5.exe test-pipe5.s printbig.o
./test-pipe5.exe
diff -b test-pipe5.out tests/test-pipe5.out > test-pipe5.diff
###### SUCCESS

###### Testing test-printbig
./func.native tests/test-printbig.fc > test-printbig.ll
llc -relocation-model=pic test-printbig.ll > test-printbig.s
cc -o test-printbig.exe test-printbig.s printbig.o
./test-printbig.exe
diff -b test-printbig.out tests/test-printbig.out >
 ↪  test-printbig.diff
###### SUCCESS

###### Testing test-reduce1
./func.native tests/test-reduce1.fc > test-reduce1.ll
llc -relocation-model=pic test-reduce1.ll > test-reduce1.s
cc -o test-reduce1.exe test-reduce1.s printbig.o
./test-reduce1.exe
diff -b test-reduce1.out tests/test-reduce1.out >
 ↪  test-reduce1.diff
###### SUCCESS

###### Testing test-var1
./func.native tests/test-var1.fc > test-var1.ll
llc -relocation-model=pic test-var1.ll > test-var1.s
cc -o test-var1.exe test-var1.s printbig.o
./test-var1.exe
diff -b test-var1.out tests/test-var1.out > test-var1.diff
###### SUCCESS

###### Testing test-var2
./func.native tests/test-var2.fc > test-var2.ll
llc -relocation-model=pic test-var2.ll > test-var2.s
cc -o test-var2.exe test-var2.s printbig.o
./test-var2.exe
diff -b test-var2.out tests/test-var2.out > test-var2.diff
```

```
###### SUCCESS

###### Testing test-while1
./func.native tests/test-while1.fc > test-while1.ll
llc -relocation-model=pic test-while1.ll > test-while1.s
cc -o test-while1.exe test-while1.s printbig.o
./test-while1.exe
diff -b test-while1.out tests/test-while1.out > test-while1.diff
###### SUCCESS

###### Testing test-while2
./func.native tests/test-while2.fc > test-while2.ll
llc -relocation-model=pic test-while2.ll > test-while2.s
cc -o test-while2.exe test-while2.s printbig.o
./test-while2.exe
diff -b test-while2.out tests/test-while2.out > test-while2.diff
###### SUCCESS

###### Testing fail-anon1
./func.native < tests/fail-anon1.fc 2> fail-anon1.err >>
↪  testall.log
diff -b fail-anon1.err tests/fail-anon1.err > fail-anon1.diff
###### SUCCESS

###### Testing fail-array1
./func.native < tests/fail-array1.fc 2> fail-array1.err >>
↪  testall.log
diff -b fail-array1.err tests/fail-array1.err > fail-array1.diff
###### SUCCESS

###### Testing fail-assign1
./func.native < tests/fail-assign1.fc 2> fail-assign1.err >>
↪  testall.log
diff -b fail-assign1.err tests/fail-assign1.err >
↪  fail-assign1.diff
###### SUCCESS

###### Testing fail-assign2
./func.native < tests/fail-assign2.fc 2> fail-assign2.err >>
↪  testall.log
diff -b fail-assign2.err tests/fail-assign2.err >
↪  fail-assign2.diff
###### SUCCESS

###### Testing fail-assign3
./func.native < tests/fail-assign3.fc 2> fail-assign3.err >>
↪  testall.log
diff -b fail-assign3.err tests/fail-assign3.err >
↪  fail-assign3.diff
###### SUCCESS
```

```
###### Testing fail-dead1
./func.native < tests/fail-dead1.fc 2> fail-dead1.err >>
↪  testall.log
diff -b fail-dead1.err tests/fail-dead1.err > fail-dead1.diff
###### SUCCESS

###### Testing fail-dead2
./func.native < tests/fail-dead2.fc 2> fail-dead2.err >>
↪  testall.log
diff -b fail-dead2.err tests/fail-dead2.err > fail-dead2.diff
###### SUCCESS

###### Testing fail-expr1
./func.native < tests/fail-expr1.fc 2> fail-expr1.err >>
↪  testall.log
diff -b fail-expr1.err tests/fail-expr1.err > fail-expr1.diff
###### SUCCESS

###### Testing fail-expr2
./func.native < tests/fail-expr2.fc 2> fail-expr2.err >>
↪  testall.log
diff -b fail-expr2.err tests/fail-expr2.err > fail-expr2.diff
###### SUCCESS

###### Testing fail-expr3
./func.native < tests/fail-expr3.fc 2> fail-expr3.err >>
↪  testall.log
diff -b fail-expr3.err tests/fail-expr3.err > fail-expr3.diff
###### SUCCESS

###### Testing fail-float1
./func.native < tests/fail-float1.fc 2> fail-float1.err >>
↪  testall.log
diff -b fail-float1.err tests/fail-float1.err > fail-float1.diff
###### SUCCESS

###### Testing fail-float2
./func.native < tests/fail-float2.fc 2> fail-float2.err >>
↪  testall.log
diff -b fail-float2.err tests/fail-float2.err > fail-float2.diff
###### SUCCESS

###### Testing fail-for1
./func.native < tests/fail-for1.fc 2> fail-for1.err >>
↪  testall.log
diff -b fail-for1.err tests/fail-for1.err > fail-for1.diff
###### SUCCESS

###### Testing fail-for2
```

```
./func.native < tests/fail-for2.fc 2> fail-for2.err >>
↪ testall.log
diff -b fail-for2.err tests/fail-for2.err > fail-for2.diff
###### SUCCESS

###### Testing fail-for3
./func.native < tests/fail-for3.fc 2> fail-for3.err >>
↪ testall.log
diff -b fail-for3.err tests/fail-for3.err > fail-for3.diff
###### SUCCESS

###### Testing fail-for4
./func.native < tests/fail-for4.fc 2> fail-for4.err >>
↪ testall.log
diff -b fail-for4.err tests/fail-for4.err > fail-for4.diff
###### SUCCESS

###### Testing fail-for5
./func.native < tests/fail-for5.fc 2> fail-for5.err >>
↪ testall.log
diff -b fail-for5.err tests/fail-for5.err > fail-for5.diff
###### SUCCESS

###### Testing fail-fstcls1
./func.native < tests/fail-fstcls1.fc 2> fail-fstcls1.err >>
↪ testall.log
diff -b fail-fstcls1.err tests/fail-fstcls1.err >
↪ fail-fstcls1.diff
###### SUCCESS

###### Testing fail-fstcls2
./func.native < tests/fail-fstcls2.fc 2> fail-fstcls2.err >>
↪ testall.log
diff -b fail-fstcls2.err tests/fail-fstcls2.err >
↪ fail-fstcls2.diff
###### SUCCESS

###### Testing fail-fstcls3
./func.native < tests/fail-fstcls3.fc 2> fail-fstcls3.err >>
↪ testall.log
diff -b fail-fstcls3.err tests/fail-fstcls3.err >
↪ fail-fstcls3.diff
###### SUCCESS

###### Testing fail-func1
./func.native < tests/fail-func1.fc 2> fail-func1.err >>
↪ testall.log
diff -b fail-func1.err tests/fail-func1.err > fail-func1.diff
###### SUCCESS
```

```
###### Testing fail-func2
./func.native < tests/fail-func2.fc 2> fail-func2.err >>
↪  testall.log
diff -b fail-func2.err tests/fail-func2.err > fail-func2.diff
###### SUCCESS

###### Testing fail-func3
./func.native < tests/fail-func3.fc 2> fail-func3.err >>
↪  testall.log
diff -b fail-func3.err tests/fail-func3.err > fail-func3.diff
###### SUCCESS

###### Testing fail-func4
./func.native < tests/fail-func4.fc 2> fail-func4.err >>
↪  testall.log
diff -b fail-func4.err tests/fail-func4.err > fail-func4.diff
###### SUCCESS

###### Testing fail-func5
./func.native < tests/fail-func5.fc 2> fail-func5.err >>
↪  testall.log
diff -b fail-func5.err tests/fail-func5.err > fail-func5.diff
###### SUCCESS

###### Testing fail-func6
./func.native < tests/fail-func6.fc 2> fail-func6.err >>
↪  testall.log
diff -b fail-func6.err tests/fail-func6.err > fail-func6.diff
###### SUCCESS

###### Testing fail-func7
./func.native < tests/fail-func7.fc 2> fail-func7.err >>
↪  testall.log
diff -b fail-func7.err tests/fail-func7.err > fail-func7.diff
###### SUCCESS

###### Testing fail-func8
./func.native < tests/fail-func8.fc 2> fail-func8.err >>
↪  testall.log
diff -b fail-func8.err tests/fail-func8.err > fail-func8.diff
###### SUCCESS

###### Testing fail-func9
./func.native < tests/fail-func9.fc 2> fail-func9.err >>
↪  testall.log
diff -b fail-func9.err tests/fail-func9.err > fail-func9.diff
###### SUCCESS

###### Testing fail-global1
```

```
./func.native < tests/fail-global1.fc 2> fail-global1.err >>
↪  testall.log
diff -b fail-global1.err tests/fail-global1.err >
↪  fail-global1.diff
###### SUCCESS

###### Testing fail-global2
./func.native < tests/fail-global2.fc 2> fail-global2.err >>
↪  testall.log
diff -b fail-global2.err tests/fail-global2.err >
↪  fail-global2.diff
###### SUCCESS

###### Testing fail-if1
./func.native < tests/fail-if1.fc 2> fail-if1.err >> testall.log
diff -b fail-if1.err tests/fail-if1.err > fail-if1.diff
###### SUCCESS

###### Testing fail-if2
./func.native < tests/fail-if2.fc 2> fail-if2.err >> testall.log
diff -b fail-if2.err tests/fail-if2.err > fail-if2.diff
###### SUCCESS

###### Testing fail-if3
./func.native < tests/fail-if3.fc 2> fail-if3.err >> testall.log
diff -b fail-if3.err tests/fail-if3.err > fail-if3.diff
###### SUCCESS

###### Testing fail-map1
./func.native < tests/fail-map1.fc 2> fail-map1.err >>
↪  testall.log
diff -b fail-map1.err tests/fail-map1.err > fail-map1.diff
###### SUCCESS

###### Testing fail-map2
./func.native < tests/fail-map2.fc 2> fail-map2.err >>
↪  testall.log
diff -b fail-map2.err tests/fail-map2.err > fail-map2.diff
###### SUCCESS

###### Testing fail-map3
./func.native < tests/fail-map3.fc 2> fail-map3.err >>
↪  testall.log
diff -b fail-map3.err tests/fail-map3.err > fail-map3.diff
###### SUCCESS

###### Testing fail-mod1
./func.native < tests/fail-mod1.fc 2> fail-mod1.err >>
↪  testall.log
diff -b fail-mod1.err tests/fail-mod1.err > fail-mod1.diff
```

```
###### SUCCESS

###### Testing fail-nomain
./func.native < tests/fail-nomain.fc 2> fail-nomain.err >>
↪ testall.log
diff -b fail-nomain.err tests/fail-nomain.err > fail-nomain.diff
###### SUCCESS

###### Testing fail-pipe1
./func.native < tests/fail-pipe1.fc 2> fail-pipe1.err >>
↪ testall.log
diff -b fail-pipe1.err tests/fail-pipe1.err > fail-pipe1.diff
###### SUCCESS

###### Testing fail-pipe2
./func.native < tests/fail-pipe2.fc 2> fail-pipe2.err >>
↪ testall.log
diff -b fail-pipe2.err tests/fail-pipe2.err > fail-pipe2.diff
###### SUCCESS

###### Testing fail-print
./func.native < tests/fail-print.fc 2> fail-print.err >>
↪ testall.log
diff -b fail-print.err tests/fail-print.err > fail-print.diff
###### SUCCESS

###### Testing fail-printb
./func.native < tests/fail-printb.fc 2> fail-printb.err >>
↪ testall.log
diff -b fail-printb.err tests/fail-printb.err > fail-printb.diff
###### SUCCESS

###### Testing fail-printbig
./func.native < tests/fail-printbig.fc 2> fail-printbig.err >>
↪ testall.log
diff -b fail-printbig.err tests/fail-printbig.err >
↪ fail-printbig.diff
###### SUCCESS

###### Testing fail-reduce1
./func.native < tests/fail-reduce1.fc 2> fail-reduce1.err >>
↪ testall.log
diff -b fail-reduce1.err tests/fail-reduce1.err >
↪ fail-reduce1.diff
###### SUCCESS

###### Testing fail-reduce2
./func.native < tests/fail-reduce2.fc 2> fail-reduce2.err >>
↪ testall.log
```

```
diff -b fail-reduce2.err tests/fail-reduce2.err >
↪  fail-reduce2.diff
###### SUCCESS

###### Testing fail-reduce3
./func.native < tests/fail-reduce3.fc 2> fail-reduce3.err >>
↪  testall.log
diff -b fail-reduce3.err tests/fail-reduce3.err >
↪  fail-reduce3.diff
###### SUCCESS

###### Testing fail-return1
./func.native < tests/fail-return1.fc 2> fail-return1.err >>
↪  testall.log
diff -b fail-return1.err tests/fail-return1.err >
↪  fail-return1.diff
###### SUCCESS

###### Testing fail-return2
./func.native < tests/fail-return2.fc 2> fail-return2.err >>
↪  testall.log
diff -b fail-return2.err tests/fail-return2.err >
↪  fail-return2.diff
###### SUCCESS

###### Testing fail-while1
./func.native < tests/fail-while1.fc 2> fail-while1.err >>
↪  testall.log
diff -b fail-while1.err tests/fail-while1.err > fail-while1.diff
###### SUCCESS

###### Testing fail-while2
./func.native < tests/fail-while2.fc 2> fail-while2.err >>
↪  testall.log
diff -b fail-while2.err tests/fail-while2.err > fail-while2.diff
###### SUCCESS
```

6.3 Test Automation

Testing is completely automated through a test script, ./testall.sh, which compiles, runs, and compares the output of all the tests under /tests to their predefined output.

Listing 7: testall.sh

```
#!/bin/sh

# Regression testing script for FunC
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work
↪  test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the FunC compiler.  Usually "./func.native"
# Try "_build/func.native" if ocamlbuild was unable to create a
↪  symbolic link.
FUNC="./func.native"
#FUNC="_build/func.native"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0

keep=0

Usage() {
    echo "Usage: testall.sh [options] [.fc files]"
    echo "-k    Keep intermediate files"
    echo "-h    Print this help"
    exit 1
}

SignalError() {
    if [ $error -eq 0 ] ; then
        echo "FAILED"
        error=1
    fi
    echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any,
↪  written to difffile
Compare() {
    generatedfiles="$generatedfiles $3"
    echo diff -b $1 $2 ">" $3 1>&2
    diff -b "$1" "$2" > "$3" 2>&1 || {
        SignalError "$1 differs"
        echo "FAILED $1 differs from $2" 1>&2
    }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
    echo $* 1>&2
    eval $* || {
        SignalError "$1 failed on $*"
        return 1
    }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
    echo $* 1>&2
    eval $* && {
        SignalError "failed: $* did not report an error"
        return 1
    }
    return 0
}

Check() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.fc//'`
    reffile=`echo $1 | sed 's/.fc$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.ll ${basename}.s
    ↪  ${basename}.exe ${basename}.out" &&
    Run "$FUNC" "$1" ">" "${basename}.ll" &&
    Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">"
    ↪  "${basename}.s" &&
    Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o"
    ↪  &&
    Run "./${basename}.exe" > "${basename}.out" &&
    Compare ${basename}.out ${reffile}.out ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}

CheckFail() {
    error=0
    basename=`echo $1 | sed 's/.*\\///
                             s/.fc//'`
    reffile=`echo $1 | sed 's/.fc$//'`
    basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

    echo -n "$basename..."

    echo 1>&2
    echo "###### Testing $basename" 1>&2

    generatedfiles=""

    generatedfiles="$generatedfiles ${basename}.err
    ↪  ${basename}.diff" &&
    RunFail "$FUNC" "<" $1 "2>" "${basename}.err" ">>" $globallog
    ↪  &&
    Compare ${basename}.err ${reffile}.err ${basename}.diff

    # Report the status and clean up the generated files

    if [ $error -eq 0 ] ; then
        if [ $keep -eq 0 ] ; then
            rm -f $generatedfiles
        fi
        echo "OK"
        echo "###### SUCCESS" 1>&2
    else
        echo "###### FAILED" 1>&2
        globalerror=$error
    fi
}
```

```
while getopts kdpsh c; do
    case $c in
        k) # Keep intermediate files
            keep=1
            ;;
        h) # Help
            Usage
            ;;
    esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI
   ↪  variable in testall.sh"
  exit 1
}

which "$LLI" >> $globallog || LLIFail

if [ ! -f printbig.o ]
then
    echo "Could not find printbig.o"
    echo "Try \"make printbig.o\""
    exit 1
fi

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test-*.fc tests/fail-*.fc"
fi

for file in $files
do
    case $file in
        *test-*)
            Check $file 2>> $globallog
            ;;
        *fail-*)
            CheckFail $file 2>> $globallog
            ;;
        *)
            echo "unknown file type $file"
            globalerror=1
            ;;
    esac
```

```
done

exit $globalerror
```

7 Lesson Learned

First of all, I learned that designing a programming language is fun! I've fiddled around with different programming languages over the years, but never really given much thought to how my code gets interpreted by the computer. I was thrilled to finally have a chance to actually look under the hood and see the mechanics of a language and its compiler, and more so because I get to design a new language and implement a compiler that deciphers my language. The project is definitely challenging for a one-person team, but it also means that I get to wear the hats of system architect, language guru, tester and manager. I gained a much deeper understanding of language design and the inner working of a compiler. Here's an itemized list of what I learned:

- Do everything in a recursive manner. In my initial implementation, I had special cases all over the place. For example, other than defining the behavior of array expression by itself, I defined a special case for array under the assignment expression. Soon I realized that I need another special case for piping. This was a disaster because the complexity grew exponentially. I had to refactor my code so that each functionality is self-sufficient. From there, just let recursion handle the rest.
- Learn to read LLVM IR code. LLVM is like a black-box. All you have if a reference manual and it's often unclear what an operation entails. Unlike popular programming languages, the online resources for LLVM is scarce. For me, the best way to know what's going on is to read and learn the LLVM IR code that the compiler generates.
- Debugging a compiler isn't easy. I'm definitely spoiled by the full-blown IDEs that are readily available for mature languages, where I can set up a break point and step through the code. Debugging in OCaml is hard as a start. Debugging the build of a compiler is virtually impossible. Believe it or not, I put more efforts in figuring out how to print variables during run-time than implementing some of the features. I have to give credit to OCAML's pattern matching, which was very helpful in identifying a problem.
- Scope workload appropriately. I initially planned to include closure in my language and even implemented environment in the latest version. But I got stuck on higher-order function and variable-length array for a while and eventually ran out of time.

The learning curve for the project is rather steep. I would recommend future teams to start working on the project as soon as possible. If you aren't familiar with functional programming, homework 1 would be a great opportunity to practice.

8 Appendix

Listing 8: func.ml

```
(* Top-level of the FunC compiler: scan & parse the input,
   check the resulting AST and generate an SAST from it, generate
↪  LLVM IR,
   and dump the module *)

type action = Ast | Sast | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
    ("-l", Arg.Unit (set_action LLVM_IR), "Print the generated
    ↪  LLVM IR");
    ("-c", Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./microc.native [-a|-s|-l|-c]
  ↪  [file.fc]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in
  ↪  filename) usage_msg;

  let built_in_channel = ref (open_in "./built_ins.fc") in
  let built_in_lexbuf = Lexing.from_channel !built_in_channel in
  let (_, built_in_fdecls) = Parser.program Scanner.token
  ↪  built_in_lexbuf in

  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.token lexbuf in
  match !action with
    Ast -> print_string (Ast.string_of_program ast)
  | _ -> let sast = Semant.check ast built_in_fdecls in
    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | LLVM_IR -> print_string (Llvm.string_of_llmodule
    ↪  (Codegen.translate sast))
    | Compile -> let m = Codegen.translate sast in
        Llvm_analysis.assert_valid_module m;
        print_string (Llvm.string_of_llmodule m)
```

Listing 9: scanner.mll

```
(* Ocamllex scanner for FunC *)

{ open Parser }

let digit = ['0' - '9']
let digits = digit+
let letter = ['a'-'z' 'A'-'Z']
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let escape = '\\' ['\\' ''' '"' 'n' 'r' 't']

rule token = parse
(* Whitespace *)
  [' ' '\t' '\r' '\n'] { token lexbuf }

(* Comments *)
| "/*"     { comment lexbuf }

(* Enclosures *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| '{'      { LBRACE }
| '}'      { RBRACE }

(* Delimiters *)
| ';'      { SEMI }
| ','      { COMMA }

(* Operators *)
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MODULO }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "=>"     { ANON }
| "|>"     { PIPE }
| "|"      { BAR }
```

```
(* Control flow *)
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }

(* Types *)
| "int"    { INT }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "void"   { VOID }
| "func"   { FUNC }

(* Malloc *)
| "new"    { NEW }

(* Literals *)
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| digits as lxm { ILIT(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm {
↪  FLIT(lxm) }
| '"' ((ascii | escape)* as lxm) '"' { SLIT(lxm) }
| letter (letter | digit | '_')* as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped
↪  char)) }

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

Listing 10: ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq
↪  | Greater |
          Geq |And | Or

type uop = Neg | Not

type typ =
    Int
  | Bool
  | Float
  | Void
  | String
  | Array of typ * expr option
  | Func of typ * typ list

and bind = typ * string

and expr =
    IntLit of int
  | FloatLit of string
  | BoolLit of bool
  | StringLit of string
  | FuncLit of func_decl
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of expr * expr
  | Call of string * expr list
  | ArrayCreate of typ * expr
  | ArrayInit of expr list
  | ArrayAccess of string * expr
  | Pipe of expr list
  | Noexpr

and stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

and func_decl = {
    rtyp : typ;
    ftype: typ;
    fname : string;
    formals : bind list;
    locals : bind list;
    body : stmt list;
  }

type program = bind list * func_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let rec string_of_expr = function
    IntLit(l) -> string_of_int l
  | FloatLit(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | StringLit(l) -> l
  | FuncLit(fdecl) -> string_of_fdecl fdecl
  | Id(s) -> s
  | Binop(e1, o, 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) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^
       ↪  ")"
  | ArrayCreate(t, l) ->
      "new " ^ string_of_typ t ^ "[" ^ string_of_expr l ^ "]"
  | ArrayInit(el) ->
      "{" ^ String.concat ", " (List.map string_of_expr el) ^ "}"
  | ArrayAccess(v, e) -> v ^ "[" ^ string_of_expr e ^ "]"
  | Pipe(le) -> List.fold_left (fun l e -> string_of_expr e ^ l)
  ↪   "" le
  | Noexpr -> ""

and string_of_stmt = function
    Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^
       ↪  "}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
  ↪   string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^
       ↪  " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
  ↪  string_of_stmt s

and string_of_typ = function
    Int    -> "int"
  | Bool   -> "bool"
  | Float  -> "float"
  | Void   -> "void"
  | String -> "string"
  | Array(t, i) ->
      (match i with
        Some i -> string_of_typ t ^ "[" ^ string_of_expr i ^ "]"
      | None   -> string_of_typ t ^ "[]")
  | Func(t, tl) -> "func <" ^ string_of_typ t ^ "(" ^
      String.concat ", " (List.map string_of_typ tl) ^ ")>"

and string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

and string_of_fdecl fdecl =
  string_of_typ fdecl.rtyp ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd
  ↪  fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

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

Listing 11: parser.mly

```
/* Ocamlyacc parser for FunC */

%{
  open Ast
  let anon_cnt = ref 0
%}

%token SEMI LPAREN RPAREN LBRACKET RBRACKET LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE MODULO ASSIGN ANON PIPE BAR
%token NOT EQ NEQ LT LEQ GT GEQ AND OR
%token RETURN IF ELSE FOR WHILE
%token INT BOOL FLOAT VOID STRING FUNC NEW
%token <int> ILIT
%token <bool> BLIT
%token <string> ID FLIT SLIT
```

```
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left PIPE BAR
%right ANON NEW
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%right NOT
%nonassoc LBRACKET RBRACKET

%%

program:
  decls EOF { $1 }

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

fdecl:
    typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list
     ↪  RBRACE
      { { rtyp = $1;
          fname = $2;
          ftype = Func($1, (List.rev (List.map fst $4)));
          formals = List.rev $4;
          locals = List.rev $7;
          body = List.rev $8 } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { $1 }

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

typ:
    INT                                     { Int
     ↪  }
  | BOOL                              { Bool
  ↪  }
  | FLOAT                             { Float
  ↪  }
  | VOID                              { Void
  ↪  }
  | STRING                            { String
  ↪  }
  | typ LBRACKET RBRACKET             { Array($1, None)
  ↪  }
  | typ LBRACKET expr RBRACKET        { Array($1, Some $3)
  ↪  }
  | FUNC LT typ LPAREN typ_opt RPAREN GT { Func($3, List.rev $5)
  ↪  }

typ_opt:
  /* nothing */ { [] }
  | typ_list    { $1 }

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

vdecl_list:
    /* nothing */    { []       }
  | vdecl_list vdecl { $2 :: $1 }

vdecl:
    typ ID SEMI { ($1, $2) }

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

stmt:
    expr SEMI                              { Expr $1
     ↪  }
  | RETURN expr_opt SEMI                   { Return $2
  ↪  }
  | LBRACE stmt_list RBRACE                { Block(List.rev $2)
  ↪  }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5,
  ↪  Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7)
  ↪  }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
                                           { For($3, $5, $7, $9)
                                            ↪  }
  | WHILE LPAREN expr RPAREN stmt          { While($3, $5)
  ↪  }
```

```
expr_opt:
    /* nothing */ { Noexpr }
  | expr          { $1     }

expr:
    ILIT                             { IntLit($1)            }
  | FLIT                                   { FloatLit($1)
  ↪  }
  | BLIT                             { BoolLit($1)           }
  | SLIT                             { StringLit($1)         }
  | func_lit                         { $1                    }
  | ID                               { Id($1)                }
  | expr PLUS   expr                 { Binop($1, Add,   $3)  }
  | expr MINUS  expr                 { Binop($1, Sub,   $3)  }
  | expr TIMES  expr                 { Binop($1, Mult,  $3)  }
  | expr DIVIDE expr                 { Binop($1, Div,   $3)  }
  | expr MODULO expr                 { Binop($1, Mod,   $3)  }
  | expr EQ     expr                 { Binop($1, Equal, $3)  }
  | expr NEQ    expr                 { Binop($1, Neq,   $3)  }
  | expr LT     expr                 { Binop($1, Less,  $3)  }
  | expr LEQ    expr                 { Binop($1, Leq,   $3)  }
  | expr GT     expr                 { Binop($1, Greater, $3) }
  | expr GEQ    expr                 { Binop($1, Geq,   $3)  }
  | expr AND    expr                 { Binop($1, And,   $3)  }
  | expr OR     expr                 { Binop($1, Or,    $3)  }
  | MINUS expr %prec NOT             { Unop(Neg, $2)         }
  | NOT expr                         { Unop(Not, $2)         }
  | expr ASSIGN expr                 { Assign($1, $3)        }
  | ID LPAREN args_opt RPAREN        { Call($1, $3)          }
  | LPAREN expr RPAREN               { $2                    }
  | NEW typ LBRACKET expr RBRACKET { ArrayCreate($2, $4)   }
  | LBRACKET args_opt RBRACKET       { ArrayInit($2)         }
  | ID LBRACKET expr RBRACKET        { ArrayAccess($1, $3)   }
  | pipe_expr BAR                    { Pipe(List.rev $1)     }

func_lit:
    LPAREN formals_opt RPAREN ANON typ LBRACE vdecl_list
     ↪  stmt_list RBRACE {
      anon_cnt := !anon_cnt + 1;
      FuncLit({
        rtyp = $5;
        fname = "_anon" ^ string_of_int !anon_cnt; (* reserved
         ↪  name *)
        ftype = Func($5, (List.rev (List.map fst $2)));
        formals = List.rev $2;
        locals = List.rev $7;
        body = List.rev $8;
      })
    }
```

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

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

pipe_expr:
| expr PIPE expr      { $3 :: [$1] }
| pipe_expr PIPE expr { $3 :: $1   }
```

Listing 12: sast.ml

```
(* Semantically-checked Abstract Syntax Tree and functions for
 ↪  printing it *)

open Ast

type sexpr = typ * sx
and sx =
    SIntLit of int
  | SFloatLit of string
  | SBoolLit of bool
  | SStringLit of string
  | SFuncLit of sfunc_decl
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of sexpr * sexpr
  | SCall of string * sexpr list
  | SArrayCreate of sexpr
  | SArrayInit of sexpr * sexpr list
  | SArrayAccess of string * sexpr
  | SPipe of sexpr
  | SNoexpr

and  sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt

and sfunc_decl = {
    srtyp : typ;
    sfname : string;
    sftype: typ;
    sformals : bind list;
    slocals : bind list;
    sbody : sstmt list;
  }

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SIntLit(l) -> string_of_int l
  | SBoolLit(true) -> "true"
  | SBoolLit(false) -> "false"
  | SFloatLit(l) -> l
  | SStringLit(l) -> l
  | SFuncLit(sfdecl) -> string_of_sfdecl sfdecl
  | SId(s) -> s
  | SBinop(e1, o, 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) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el)
       ↪  ^ ")"
  | SArrayCreate(l) -> "new [" ^ string_of_sexpr l ^ "]"
  | SArrayInit(_, el) ->
      "{" ^ String.concat ", " (List.map string_of_sexpr el) ^
       ↪  "}"
  | SArrayAccess(v, e) -> v ^ "[" ^ string_of_sexpr e ^ "]"
  | SPipe(e) -> "|> " ^ string_of_sexpr e ^ " |"
  | SNoexpr -> ""
                                  ) ^ ")"

and string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^
       ↪  "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2
       ↪  ^ " ; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^
  ↪  string_of_sstmt s

and string_of_sfdecl fdecl =
  string_of_typ fdecl.srtyp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd
  ↪  fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

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

Listing 13: semant.ml

```
(* Semantic checking for the FunC compiler *)

open Ast
open Sast
(* open Utility *)

module StringMap = Map.Make(String)

(* Struct for environment management: containing local and
↪  external symbol tables *)
type environment = {
  local_vars: typ StringMap.t;
  external_vars: typ StringMap.t;
}

(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions) (built_in_fdecls: func_decl list)
↪  =

  (* Verify a list of bindings has no void types or duplicate
  ↪  names *)
  let check_binds (kind : string) (binds : bind list) =
    List.iter (function
              (Void, b) -> raise (Failure ("illegal void " ^ kind
              ↪  ^ " " ^ b))
      | _ -> ()) binds;
    let rec dups = function
        [] -> ()
      |       ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
                raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
  in

  (* Add a list of bindings to the environment *)
  let add_binds env l = List.fold_left
    (fun env (typ, name) -> {env with local_vars = StringMap.add
     ↪  name typ env.local_vars})
    env
    l
  in

  (**** Check global variables ****)
  check_binds "global" globals;
  let global_vars = add_binds {
      local_vars = StringMap.empty;
      external_vars = StringMap.empty
    } globals in

  (**** Check functions ****)

  (* Collect function declarations for built-in functions: no
   ↪  bodies *)
  let built_in_decls =
    let add_bind map (name, (t, tl)) = StringMap.add name {
      rtyp = Void;
      fname = name;
      ftype = Func(Void, tl);
      formals = [(t, "x")];
      locals = []; body = [] } map
    in List.fold_left add_bind StringMap.empty [
      ("print", (Int, [Int]));
                        ("printb", (Bool, [Bool]));
      ("printf", (Float, [Float]));
      ("prints", (String, [String]));
      ("printbig", (Int, [Int]));
      ("map", (Void, [Void]));
      ("reduce", (Void, [Void]))]
  in

  (* Add function name to symbol table *)
  let add_func map fd =
    let built_in_err = "function " ^ fd.fname ^ " may not be
     ↪  defined"
    and dup_err = "duplicate function " ^ fd.fname
    and make_err er = raise (Failure er)
    and n = fd.fname (* Name of the function *)
    in match fd with (* No duplicate functions or redefinitions
     ↪  of built-ins *)
         _ when StringMap.mem n built_in_decls -> make_err
          ↪  built_in_err
       | _ when StringMap.mem n map -> make_err dup_err
       | _ ->  StringMap.add n fd map
  in

  (* Collect all function names into one symbol table *)
  let global_fdecls =
    (* Add custom built-in functions first so that users can't
     ↪  override them *)
    let all_built_ins =
      List.fold_left (fun map fd -> StringMap.add fd.fname fd
       ↪  map)
      built_in_decls
      built_in_fdecls
    in
    List.fold_left add_func all_built_ins functions
  in

  (* Merge built-in functions with user-defined functions *)
  let functions = List.fold_left (fun l e -> e :: l) functions
   ↪  built_in_fdecls
  in

  let global_functions = StringMap.fold
    (fun n fdecl m -> StringMap.add n fdecl.ftype m)
    global_fdecls
    StringMap.empty
  in

  (* Prepare global environment *)
  let global_env = {
    local_vars = StringMap.empty;
    external_vars = StringMap.merge
      (fun n v1 v2 ->
        match v1, v2 with
          (Some _), (Some _) -> raise (Failure (n ^ " is
           ↪  ambiguous"))
        | (Some v), None -> Some v
        | None, (Some v) -> Some v
        | None, None -> None)
      global_vars.local_vars
      global_functions
  } in

  (* Return a function from our symbol table *)
  let find_func s =
    try StringMap.find s global_fdecls
    with Not_found -> raise (Failure ("unrecognized function " ^
     ↪  s))
  in

  let _ = find_func "main" in (* Ensure "main" is defined *)

  let rec check_function env f =
    (* Make sure no formals or locals are void or duplicates *)
    check_binds "formal" f.formals;
    check_binds "local" f.locals;
    let env = add_binds env f.formals in
    let env = add_binds env f.locals in

    (* Raise an exception if the given rvalue type and the given
     ↪  lvalue
       are not equal *)
    let check_typ_eq lvaluet rvaluet err =
      if lvaluet = rvaluet then lvaluet else raise (Failure err)
    in

    (* More flexible version for array argument: only check type
     ↪  *)
    let check_array_eq lvaluet rvaluet err =
      (match (lvaluet, rvaluet) with
        (Array(lt, _), Array(rt, _)) ->
          if lt = rt then rvaluet else raise (Failure err)
      | _ -> if lvaluet = rvaluet then lvaluet else raise
       ↪  (Failure err))
    in

    (* Return a variable from our local symbol table *)
    let type_of_identifier env s =
      try StringMap.find s env.local_vars
      with Not_found ->
        (try StringMap.find s env.external_vars
         with Not_found -> raise (Failure ("undeclared identifier
          ↪  " ^ s)))
    in

    (* Return a semantically-checked expression, i.e., with a
     ↪  type *)
    let rec check_expr env = function
        IntLit l                    -> (Int, SIntLit l)
      | FloatLit l                  -> (Float, SFloatLit l)
      | BoolLit l                   -> (Bool, SBoolLit l)
      | StringLit l                 -> (String, SStringLit l)
      | FuncLit(fdecl)              -> check_func_literal env fdecl
      | Noexpr                      -> (Void, SNoexpr)
      | Id s                        -> check_id env s
      | Assign(lhs, rhs) as ex    -> check_assign env lhs rhs ex
      | Unop(op, e) as ex         -> check_unop env op e ex
      | Binop(e1, op, e2) as ex   -> check_binop env e1 op e2 ex
      | Call(fname, args) as call -> check_call env fname args
       ↪  call
      | ArrayCreate(t, l)         -> check_array_create env t l
      | ArrayInit(args)           -> check_array_init env args
      | ArrayAccess(s, e)         -> check_array_access env s e
      | Pipe(el)                  -> check_pipe env el

    and check_func_literal env fdecl =
      let nenv = {
        local_vars = StringMap.empty;
        external_vars = StringMap.merge (fun _ v1 v2 ->
          match v1, v2 with
              None, None -> None
            | (Some v), None -> Some v
            | None, (Some v) -> Some v
            | (Some v), (Some _) -> Some v) (* Take the local one
             ↪  if a variable is defined in both *)
          env.local_vars env.external_vars
      } in
      let sfdecl = check_function nenv fdecl in
      (sfdecl.sftype, SFuncLit(sfdecl))

    and check_id env s =
      (type_of_identifier env s, SId s)

    and check_assign env lhs rhs ex =
      match lhs with
      | Id(s) ->
        let lt = type_of_identifier env s
        and (rt, re) = check_expr env rhs in
        let err = "illegal assignment " ^ string_of_typ lt ^ " =
        ↪  " ^
          string_of_typ rt ^ " in " ^ string_of_expr ex
        in
        (match rt with
          Array(_, _) ->
            check_array_eq lt rt err, SAssign((lt, SId(s)), (rt,
            ↪  re))
        | _ -> check_typ_eq lt rt err, SAssign((lt, SId(s)), (rt,
        ↪  re)))
      | ArrayAccess(_, _) ->
          let (lt, le) = check_expr env lhs
          and (rt, re) = check_expr env rhs in
          let err = "illegal array assignment " ^ string_of_typ
          ↪  lt ^
            " = " ^ string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_array_eq lt rt err, SAssign((lt, le), (rt,
           ↪  re)))
      | _ -> raise (Failure ("invalid assignment: " ^
            "LHS must be variable or array"))

    and check_unop env op e ex=
      let (t, e') = check_expr env e in
      let ty = match op with
          Neg when t = Int || t = Float -> t
        | Not when t = Bool -> Bool
        | _ -> raise (Failure ("illegal unary operator " ^
                              string_of_uop op ^ string_of_typ t
                               ↪  ^
                              " in " ^ string_of_expr ex))
      in (ty, SUnop(op, (t, e')))

    and check_binop env e1 op e2 ex =
      let (t1, e1') = check_expr env e1
      and (t2, e2') = check_expr env e2 in
      (* All binary operators require operands of the same type
       ↪  *)
      let same = t1 = t2 in
      (* Determine expression type based on operator and operand
       ↪  types *)
      let ty = match op with
          Add | Sub | Mult | Div | Mod when same && t1 = Int   ->
           ↪  Int
        | Add | Sub | Mult | Div when same && t1 = Float -> Float
        | Mod when same && t1 = Float ->
          raise (Failure("mod is not supported for float"))
        | Equal | Neq            when same               -> Bool
        | Less | Leq | Greater | Geq
                  when same && (t1 = Int || t1 = Float) -> Bool
        | And | Or when same && t1 = Bool -> Bool
        | _ -> raise (Failure ("illegal binary operator " ^
                      string_of_typ t1 ^ " " ^ string_of_op op ^
                       ↪  " " ^
                      string_of_typ t2 ^ " in " ^ string_of_expr
                       ↪  ex))
      in (ty, SBinop((t1, e1'), op, (t2, e2')))

    and check_call env fname args call =
      let check_call_helper ret_typ typ_list =
        let param_length = List.length typ_list in
        if List.length args != param_length then
          raise (Failure ("expecting " ^ string_of_int
           ↪  param_length ^
                          " arguments in " ^ string_of_expr
                           ↪  call))
        else let check_args ft e =
          let (et, e') = check_expr env e in
          let err = "illegal argument found " ^ string_of_typ et
          ↪ ^
            " expected " ^ string_of_typ ft ^ " in " ^
             ↪ string_of_expr e
          in
          (match ft with
            Array(_, _) -> (check_array_eq ft et err, e')
          | _ -> (check_typ_eq ft et err, e'))
        in
        let args' = List.map2 check_args typ_list args
        in (ret_typ, SCall(fname, args'))
      in
      let ftype = type_of_identifier env fname in
      match ftype with
        Func(t, tl) -> check_call_helper t tl
      | _ -> raise (Failure ("invalid type in function call " ^
                    string_of_typ ftype))

    and check_array_create env t l =
      let l' = check_expr env l in
      (Array(t, Some l), SArrayCreate(l'))

    and check_array_init env args =
      let sexpr_list = List.map (fun e -> check_expr env e) args
      ↪ in
      let t_list = List.map (fun (t, _) -> t) sexpr_list in
      let all_eq = (match t_list with
        | []        -> true
        | hd :: tl -> List.for_all ((=) hd) tl) in
      let t = List.hd t_list
      and l = IntLit (List.length sexpr_list) in
      let l' = check_expr env l in
      if not all_eq
      then raise (Failure "inconsistent types in array
       ↪ initialization")
      else (Array(t, Some l), SArrayInit(l', sexpr_list))

    and check_array_access env s e =
      let lt = type_of_identifier env s in
      let (rt, e') = check_expr env e in
      let (lt, l) = (match lt with
          Array(lt, l) -> lt, l
        | _ -> raise (Failure "invalid id type in array access"))
        ↪ in
      (match e' with
        SIntLit(i) ->
          (match l with
            Some (IntLit l) ->
              if i > l - 1 || i < 0
              then raise (Failure "index out of range in array
               ↪  access")
              else (lt, SArrayAccess(s, (rt, e')))
          | _ -> (lt, SArrayAccess(s, (rt, e'))))
      | _ -> (lt, SArrayAccess(s, (rt, e'))))

    and check_pipe env el =
      let check_first_item e = match e with
          Noexpr | FuncLit _ | Assign _ | ArrayCreate _ ->
            raise (Failure "invalid first expression for pipe")
        | _ -> e
      in

      let rec append_arg (hd: expr) (tl: expr list) =
        match tl with
          [] -> hd
        | Call(fname, args) :: tl  ->
            let current_e = Call(fname, args @ [hd]) in
            append_arg current_e tl
        | _ -> raise (Failure "expression needs to be a function
         ↪  call")
      in

      let accum_func = append_arg (check_first_item (List.hd el))
       ↪  (List.tl el) in
      check_expr env accum_func
    in

    let check_bool_expr env e =
      let (t', e') = check_expr env e
      and err = "expected Boolean expression in " ^
       ↪  string_of_expr e
      in if t' != Bool then raise (Failure err) else (t', e')
    in

    (* Return a semantically-checked statement i.e. containing
     ↪  sexprs *)
    let rec check_stmt env = function
        Expr e -> SExpr (check_expr env e)
      | If(p, b1, b2) -> SIf(check_bool_expr env p, check_stmt
       ↪  env b1, check_stmt env b2)
      | For(e1, e2, e3, st) ->
                SFor(check_expr env e1, check_bool_expr env e2,
                 ↪  check_expr env e3, check_stmt env st)
      | While(p, s) -> SWhile(check_bool_expr env p, check_stmt
       ↪  env s)
      | Return e -> let (t, e') = check_expr env e in
          let err = Failure ("return gives " ^ string_of_typ t ^
           ↪  " expected " ^
```

```
                         string_of_typ f.rtyp ^ " in " ^
                         ↪  string_of_expr e) in
          (match (t, f.rtyp) with
            (Array(lt, _), Array(rt, _)) ->
              if lt = rt then SReturn (t, e')
              else raise err
          | _ ->
              if t = f.rtyp then SReturn (t, e')
              else raise err)

            (* A block is correct if each statement is correct
            ↪  and nothing
               follows any Return statement.  Nested blocks are
↪  flattened. *)
        | Block sl ->
          let rec check_stmt_list env = function
              [Return _ as s] -> [check_stmt env s]
            | Return _ :: _   -> raise (Failure "nothing may
            ↪  follow a return")
            | Block sl :: ss  -> check_stmt_list env (sl @ ss) (*
            ↪  Flatten blocks *)
            | s :: ss         -> check_stmt env s ::
            ↪  check_stmt_list env ss
            | []              -> []
          in SBlock(check_stmt_list env sl)

    in (* body of check_function *)
    { srtyp = f.rtyp;
      sfname = f.fname;
      sftype = f.ftype;
      sformals = f.formals;
      slocals  = f.locals;
      sbody = match check_stmt env (Block f.body) with
                      SBlock(sl) -> sl
                | _ -> raise (
                    Failure ("internal error: block didn't become a
                    ↪  block?"))
    }
  in

  (* Check global functions *)
  let sfdecls = List.map (check_function global_env) functions in
  (globals, sfdecls)
```

Listing 14: codegen.ml

```
(* Code generation: translate takes a semantically checked AST
↪  and
produces LLVM IR

LLVM tutorial: Make sure to read the OCaml version of the
↪  tutorial

http://llvm.org/docs/tutorial/index.html

Detailed documentation on the OCaml LLVM library:

http://llvm.moe/
http://llvm.moe/ocaml/

*)

module L = Llvm
module A = Ast
open Sast
(* open Utility *)

module StringMap = Map.Make(String)

(* Struct for environment management: containing local and
↪  external symbol
   tables and the builder *)
type environment = {
  builder: L.llbuilder;
  local_vars: L.llvalue StringMap.t;
  external_vars: L.llvalue StringMap.t;
}

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions) =
  let context    = L.global_context () in

  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "FunC" in

  (* Get types from the context *)
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and float_t    = L.double_type context
  and string_t   = L.pointer_type (L.i8_type context)
  and void_t     = L.void_type   context in

  (* Return the LLVM type for a FunC type *)
  let rec ltype_of_typ = function
      A.Int         -> i32_t
    | A.Bool        -> i1_t
    | A.Float       -> float_t
    | A.String      -> string_t
    | A.Void        -> void_t
    | A.Array(t, _) -> ltype_of_array t
    | A.Func(t, tl) -> ltype_of_func t tl

  (* Return the LLVM type for an array *)
  and ltype_of_array t = L.pointer_type (ltype_of_typ t)

  (* Return the LLVM type for a function type *)
  and ltype_of_func ret_typ typ_list =
    let lret_typ = ltype_of_typ ret_typ in
    let largs = Array.of_list (List.fold_left (fun l t ->
     ↪  ltype_of_typ t :: l)
                                             [] (List.rev
                                               ↪  typ_list))
    in
    L.pointer_type (L.function_type lret_typ largs)
  in

  (* Initialize LLVM objects *)
  let init typ = match typ with
      A.Int | A.Bool | A.Void -> L.const_int (ltype_of_typ typ) 0
    | A.Float                 -> L.const_float (ltype_of_typ typ)
     ↪  0.0
    | A.String                -> L.const_pointer_null
     ↪  (ltype_of_typ typ)
    | A.Array(t, _)           -> L.const_pointer_null
     ↪  (ltype_of_typ t)
    | A.Func(t, tl)           -> L.const_pointer_null
     ↪  (ltype_of_func t tl)
  in

  let printf_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
      L.declare_function "printf" printf_t the_module in

  let printbig_t : L.lltype =
      L.function_type i32_t [| i32_t |] in
  let printbig_func : L.llvalue =
      L.declare_function "printbig" printbig_t the_module in

  (* Define each function (arguments and return type) so we can
     call it even before we've created its body *)
  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
    let function_decl m fdecl =
      let name = fdecl.sfname
      and formal_types =
              Array.of_list (List.map (fun (t, _) -> ltype_of_typ
               ↪  t) fdecl.sformals)
      in let ftype = L.function_type (ltype_of_typ fdecl.srtyp)
       ↪  formal_types in
      StringMap.add name (L.define_function name ftype
       ↪  the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty functions in

  let (main, _) = StringMap.find "main" function_decls in
  let global_builder = L.builder_at_end context (L.entry_block
   ↪  main) in

  let int_format_str = L.build_global_stringptr "%d\n" "fmt"
   ↪  global_builder
  and float_format_str = L.build_global_stringptr "%g\n" "fmt"
   ↪  global_builder
  and string_format_str = L.build_global_stringptr "%s\n" "fmt"
   ↪  global_builder in

  (* Return the value for a variable or formal argument.
      Check local names first, then external names *)
  let lookup env n = try StringMap.find n env.local_vars
                      with Not_found ->
                      (try StringMap.find n env.external_vars
                        with Not_found ->
                        raise (Failure (n ^ " is undefined")))
  in

  (* Construct code for an expression; return its value *)
  let rec build_expr env ((typ, e) : sexpr) = match e with
      SIntLit i -> (env, L.const_int i32_t i)
    | SBoolLit b -> (env, L.const_int i1_t (if b then 1 else 0))
    | SFloatLit l -> (env, L.const_float_of_string float_t l)
    | SStringLit s -> (env, L.build_global_stringptr s "tmp"
     ↪  env.builder)
    | SFuncLit sfdecl ->
        (* Closure implementation: encapsulate parent's
         ↪  environment *)
        let nenv = {
          (* Local variables will be added in build_function_body
           ↪  later *)
          local_vars = StringMap.empty;
          external_vars = StringMap.merge
            (fun _ v1 v2 ->
              match v1, v2 with
                (Some v), (Some _) -> Some v
              | (Some v), None -> Some v
              | None, (Some v) -> Some v
              | None, None -> None)
            env.local_vars
            env.external_vars;
          builder = env.builder
        } in
        let name = sfdecl.sfname
        and formal_types =
          Array.of_list (List.map (fun (t, _) -> ltype_of_typ t)
           ↪  sfdecl.sformals)
        in let ftype = L.function_type (ltype_of_typ
         ↪  sfdecl.srtyp) formal_types in
        let fval = L.define_function name ftype the_module in
        build_function_body nenv (fval, sfdecl);
        (nenv, fval)
    | SNoexpr     -> (env, L.const_int i32_t 0)
    | SId s       -> (env, L.build_load (lookup env s) s
     ↪  env.builder)
    | SAssign(lhs, rhs) ->
        let _, le = lhs in
        let (env', re') = build_expr env rhs in
        let addr = (match le with
            SId s                -> lookup env' s
          | SArrayAccess(s, e) ->
              let var = lookup env' s in
              let arr_ptr = L.build_load var s env'.builder in
              let (env', le') = build_expr env' e in
              L.build_gep arr_ptr [|le'|] s env'.builder
          | _                    -> raise (Failure "invalid type
           ↪  for LHS")) in
        ignore(L.build_store re' addr env'.builder);
        (env', re')
    | SBinop ((A.Float,_ ) as e1, op, e2) ->
      let (_, e1') = build_expr env e1 in
      let (env', e2') = build_expr env e2 in
      (env', (match op with
        A.Add     -> L.build_fadd
      | A.Sub     -> L.build_fsub
      | A.Mult    -> L.build_fmul
      | A.Div     -> L.build_fdiv
      | A.Mod     -> L.build_frem
      | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
      | A.Neq     -> L.build_fcmp L.Fcmp.One
      | A.Less    -> L.build_fcmp L.Fcmp.Olt
      | A.Leq     -> L.build_fcmp L.Fcmp.Ole
      | A.Greater -> L.build_fcmp L.Fcmp.Ogt
      | A.Geq     -> L.build_fcmp L.Fcmp.Oge
      | A.And | A.Or ->
          raise (Failure "internal error: semant should have
           ↪  rejected and/or on float")
      ) e1' e2' "tmp" env'.builder)
    | SBinop (e1, op, e2) ->
      let (_, e1') = build_expr env e1 in
      let (env', e2') = build_expr env e2 in
      (env', (match op with
        A.Add     -> L.build_add
      | A.Sub     -> L.build_sub
      | A.Mult    -> L.build_mul
      | A.Div     -> L.build_sdiv
      | A.Mod     -> L.build_srem
      | A.And     -> L.build_and
      | A.Or      -> L.build_or
      | A.Equal   -> L.build_icmp L.Icmp.Eq
      | A.Neq     -> L.build_icmp L.Icmp.Ne
      | A.Less    -> L.build_icmp L.Icmp.Slt
      | A.Leq     -> L.build_icmp L.Icmp.Sle
      | A.Greater -> L.build_icmp L.Icmp.Sgt
      | A.Geq     -> L.build_icmp L.Icmp.Sge
      ) e1' e2' "tmp" env'.builder)
    | SUnop(op, ((t, _) as e)) ->
      let (env', e') = build_expr env e in
      (env', (match op with
        A.Neg when t = A.Float -> L.build_fneg
      | A.Neg                  -> L.build_neg
      | A.Not                  -> L.build_not) e' "tmp"
       ↪  env'.builder)
    | SCall ("print", [e]) | SCall ("printb", [e]) ->
      let (env', e') = build_expr env e in
      (env', L.build_call printf_func [| int_format_str ; e' |]
                "printf" env'.builder)
    | SCall ("printf", [e]) ->
      let (env', e') = build_expr env e in
      (env', L.build_call printf_func [| float_format_str ; e' |]
                "printf" env'.builder)
    | SCall ("prints", [e]) ->
      let (env', e') = build_expr env e in
      (env', L.build_call printf_func [| string_format_str ; e'
      ↪  |]
                "printf" env'.builder)
    | SCall ("printbig", [e]) ->
      let (env', e') = build_expr env e in
      (env', L.build_call printbig_func [| e' |] "printbig"
       ↪  env'.builder)
    | SCall (f, args) ->
        let fptr = lookup env f in
        let fval = L.build_load fptr "tmp" env.builder in
        let (env', llargs) = List.fold_right
          (fun e (env, llvals) ->
            let (env', llval) = build_expr env e in
            (env', llval :: llvals))
          args
          (env, [])
        in
```

```
      (env', L.build_call fval (Array.of_list llargs) ""
      ↪  env'.builder)
  | SArrayCreate(l) ->
    (match typ with
      A.Array(t, _) ->
        let (env', l') = build_expr env l in
        if L.type_of l' != i32_t
        then raise (Failure "array length arg doesn't evaluate
        ↪  to integer");
        let ptr = L.build_array_malloc
                    (ltype_of_typ t)
                    l'
                    "arrptr"
                    env.builder
        in
        (env', ptr)
    | _ -> raise (Failure "invalid type in array create"))
  | SArrayInit(lexpr, expr_list) ->
    (match typ with
      A.Array(t, l) ->
        let (env', l') = (match l with
            Some _ -> build_expr env lexpr
          | None   -> raise (Failure "array length is None"))
          ↪  in
        if L.type_of l' != i32_t
        then raise (Failure "array length arg doesn't evaluate
        ↪  to integer");
        let env', elems = List.fold_right (fun e (env, llvals)
        ↪  ->
                        let (env', llval) =  build_expr env
                        ↪  e in
                        (env', llval :: llvals))
                        expr_list (env', [])
        in
        let ptr = L.build_array_malloc
                    (ltype_of_typ t)
                    l'
                    "arrptr"
                    env.builder
        in
        ignore(List.fold_left (fun i e ->
            let idx = L.const_int i32_t i in
            let eptr = L.build_gep ptr [|idx|] "tmp"
            ↪  env.builder in
            ignore(L.build_store e eptr env.builder);
            i+1)
                              0 elems);
        (env', ptr)
    | _ -> raise (Failure "invalid type in array init"))
  | SArrayAccess(s, e) ->
```

```
      let var = lookup env s in
      let arr_ptr = L.build_load var s env.builder in
      let (env', e') = build_expr env e in
      let elem_ptr = L.build_gep arr_ptr [|e'|] "tmp"
       ↪  env'.builder in
      (env', L.build_load elem_ptr "tmp" env'.builder)
    | SPipe(e) -> build_expr env e

  (* Fill in the body of the given function *)
  and build_function_body env (the_function, fdecl) =
    let builder = L.builder_at_end context (L.entry_block
     ↪  the_function) in

    (* Construct the function's "locals": formal arguments and
     ↪  locally
        declared variables.  Allocate each on the stack,
↪   initialize their
        value, if appropriate, and remember their values in the
↪   "locals" map *)
    let local_vars =
      let add_formal m (t, n) p =
        L.set_value_name n p;
        let local = L.build_alloca (ltype_of_typ t) n builder in
        ignore (L.build_store p local builder);
        StringMap.add n local m

      (* Allocate space for any locally declared variables and
       ↪  add the
       * resulting registers to our map *)
      and add_local m (t, n) =
        let local_var = L.build_alloca (ltype_of_typ t) n builder
        in StringMap.add n local_var m
      in

      let formals = List.fold_left2 add_formal StringMap.empty
       ↪  fdecl.sformals
          (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals fdecl.slocals
    in

    let external_vars = StringMap.merge (fun _ v1 v2 ->
      match v1, v2 with
          None, None -> None
        | (Some v), None -> Some v
        | None, (Some v) -> Some v
        | (Some v), (Some _) -> Some v) (* Take the local one if
         ↪  a variable is defined in both *)
      env.local_vars env.external_vars
    in
```

```
let env = {
  local_vars = local_vars;
  external_vars = external_vars;
  builder = builder
} in

(* LLVM insists each basic block end with exactly one
 ↪  "terminator"
    instruction that transfers control.  This function runs
↪ "instr builder"
    if the current block does not already have a terminator.
↪ Used,
    e.g., to handle the "fall off the end of the function"
↪ case. *)
let add_terminal env instr =
  match L.block_terminator (L.insertion_block env.builder)
   ↪  with
    Some _ -> ()
  | None -> ignore (instr env.builder) in

(* Build the code for the given statement; return the builder
 ↪  for
    the statement's successor (i.e., the next instruction
↪ will be built
    after the one generated by this call) *)

let rec build_stmt env = function
    SBlock sl -> List.fold_left build_stmt env sl
  | SExpr e -> fst (build_expr env e)
  | SReturn e ->
      let (env', e') = build_expr env e in
      ignore(match fdecl.srtyp with
              (* Special "return nothing" instr *)
              A.Void -> L.build_ret_void env'.builder
              (* Build return statement *)
            | _ -> L.build_ret e' env'.builder);
      env'
  | SIf (predicate, then_stmt, else_stmt) ->
      let (env', bool_val) = build_expr env predicate in
      let merge_bb = L.append_block context "merge"
       ↪  the_function in
      let build_br_merge = L.build_br merge_bb in (* partial
       ↪  function *)

      let then_bb = L.append_block context "then"
       ↪  the_function in
      let builder = L.builder_at_end context then_bb in
      add_terminal (build_stmt {env' with builder = builder}
       ↪  then_stmt) build_br_merge;
```

```
        let else_bb = L.append_block context "else"
         ↪  the_function in
        let builder = L.builder_at_end context else_bb in
        add_terminal (build_stmt {env' with builder = builder}
         ↪  else_stmt) build_br_merge;

        ignore(L.build_cond_br bool_val then_bb else_bb
         ↪  env'.builder);
        {env' with builder = L.builder_at_end context merge_bb}

    | SWhile (predicate, body) ->
        let pred_bb = L.append_block context "while"
         ↪  the_function in
        ignore(L.build_br pred_bb env.builder);

        let pred_builder = L.builder_at_end context pred_bb in
        let (env', bool_val) = build_expr {env with builder =
         ↪  pred_builder} predicate in

        let body_bb = L.append_block context "while_body"
         ↪  the_function in
        add_terminal (build_stmt {env' with builder =
         ↪  (L.builder_at_end context body_bb)} body)
          (L.build_br pred_bb);

        let merge_bb = L.append_block context "merge"
         ↪  the_function in
        ignore(L.build_cond_br bool_val body_bb merge_bb
         ↪  pred_builder);
        {env with builder = L.builder_at_end context merge_bb}

    (* Implement for loops as while loops *)
    | SFor (e1, e2, e3, body) -> build_stmt env
        ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr
         ↪  e3]) ] )
  in

  (* Build the code for each statement in the function *)
  let env = build_stmt env (SBlock fdecl.sbody) in

  (* Add a return if the last block falls off the end *)
  add_terminal env (match fdecl.srtyp 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

(* Create a map of global variables after creating each *)
let global_vars : L.llvalue StringMap.t =
  let global_var m (t, n) =
    StringMap.add n (L.define_global n (init t) the_module) m
     ↪  in
  List.fold_left global_var StringMap.empty globals in

(* Create a map of global functions with pointers to them *)
let global_functions = StringMap.mapi
  (fun fname (fval, _) ->
    let ft = L.type_of fval in
    let fptr = L.define_global (fname ^ "_ptr")
     ↪  (L.const_pointer_null ft) the_module in
    ignore (L.build_store fval fptr global_builder);
    fptr)
  function_decls
in

(* Merge global variables and global functions *)
let globals = StringMap.merge
  (fun n v1 v2 ->
    match v1, v2 with
      (Some _), (Some _) -> raise (Failure (n ^ " is
       ↪  ambiguous"))
    | (Some v), None -> Some v
    | None, (Some v) -> Some v
    | None, None -> None)
  global_vars
  global_functions
in

(* Prepare global environment *)
let global_env = {
  local_vars = StringMap.empty;
  external_vars = globals;
  builder = global_builder
}
in

StringMap.iter (fun _ fdef ->
  build_function_body global_env fdef)
  function_decls;

the_module
```

Listing 15: utility.ml

```
(* Utility functions for the FunC compiler *)

let debug msg =
  prerr_endline msg
```

Listing 16: printbig.c

```
/*
 *  A function illustrating how to link C code to code generated
↪  from LLVM
 */

#include <stdio.h>

/*
 * Font information: one byte per row, 8 rows per character
 * In order, space, 0-9, A-Z
 */
static const char font[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x1c, 0x3e, 0x61, 0x41, 0x43, 0x3e, 0x1c, 0x00,
  0x00, 0x40, 0x42, 0x7f, 0x7f, 0x40, 0x40, 0x00,
  0x62, 0x73, 0x79, 0x59, 0x5d, 0x4f, 0x46, 0x00,
  0x20, 0x61, 0x49, 0x4d, 0x4f, 0x7b, 0x31, 0x00,
  0x18, 0x1c, 0x16, 0x13, 0x7f, 0x7f, 0x10, 0x00,
  0x27, 0x67, 0x45, 0x45, 0x45, 0x7d, 0x38, 0x00,
  0x3c, 0x7e, 0x4b, 0x49, 0x49, 0x79, 0x30, 0x00,
  0x03, 0x03, 0x71, 0x79, 0x0d, 0x07, 0x03, 0x00,
  0x36, 0x4f, 0x4d, 0x59, 0x59, 0x76, 0x30, 0x00,
  0x06, 0x4f, 0x49, 0x49, 0x69, 0x3f, 0x1e, 0x00,
  0x7c, 0x7e, 0x13, 0x11, 0x13, 0x7e, 0x7c, 0x00,
  0x7f, 0x7f, 0x49, 0x49, 0x49, 0x7f, 0x36, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x41, 0x63, 0x22, 0x00,
  0x7f, 0x7f, 0x41, 0x41, 0x63, 0x3e, 0x1c, 0x00,
  0x00, 0x7f, 0x7f, 0x49, 0x49, 0x49, 0x41, 0x00,
  0x7f, 0x7f, 0x09, 0x09, 0x09, 0x09, 0x01, 0x00,
  0x1c, 0x3e, 0x63, 0x41, 0x49, 0x79, 0x79, 0x00,
  0x7f, 0x7f, 0x08, 0x08, 0x08, 0x7f, 0x7f, 0x00,
  0x00, 0x41, 0x41, 0x7f, 0x7f, 0x41, 0x41, 0x00,
  0x20, 0x60, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x7f, 0x7f, 0x18, 0x3c, 0x76, 0x63, 0x41, 0x00,
  0x00, 0x7f, 0x7f, 0x40, 0x40, 0x40, 0x40, 0x00,
  0x7f, 0x7f, 0x0e, 0x1c, 0x0e, 0x7f, 0x7f, 0x00,
  0x7f, 0x7f, 0x0e, 0x1c, 0x38, 0x7f, 0x7f, 0x00,
  0x3e, 0x7f, 0x41, 0x41, 0x41, 0x7f, 0x3e, 0x00,
  0x7f, 0x7f, 0x11, 0x11, 0x11, 0x1f, 0x0e, 0x00,
  0x3e, 0x7f, 0x41, 0x51, 0x71, 0x3f, 0x5e, 0x00,
  0x7f, 0x7f, 0x11, 0x31, 0x79, 0x6f, 0x4e, 0x00,
  0x26, 0x6f, 0x49, 0x49, 0x4b, 0x7a, 0x30, 0x00,
  0x00, 0x01, 0x01, 0x7f, 0x7f, 0x01, 0x01, 0x00,
  0x3f, 0x7f, 0x40, 0x40, 0x40, 0x7f, 0x3f, 0x00,
  0x0f, 0x1f, 0x38, 0x70, 0x38, 0x1f, 0x0f, 0x00,
  0x1f, 0x7f, 0x38, 0x1c, 0x38, 0x7f, 0x1f, 0x00,
  0x63, 0x77, 0x3e, 0x1c, 0x3e, 0x77, 0x63, 0x00,
  0x00, 0x03, 0x0f, 0x78, 0x78, 0x0f, 0x03, 0x00,
  0x61, 0x71, 0x79, 0x5d, 0x4f, 0x47, 0x43, 0x00
};

void printbig(int c)
{
  int index = 0;
  int col, data;
  if (c >= '0' && c <= '9') index = 8 + (c - '0') * 8;
  else if (c >= 'A' && c <= 'Z') index = 88 + (c - 'A') * 8;
  do {
    data = font[index++];
    for (col = 0 ; col < 8 ; data <<= 1, col++) {
      char d = data & 0x80 ? 'X' : ' ';
      putchar(d); putchar(d);
    }
    putchar('\n');
  } while (index & 0x7);
}


#ifdef BUILD_TEST
int main()
{
  char s[] = "HELLO WORLD09AZ";
  char *c;
  for ( c = s ; *c ; c++) printbig(*c);
}
#endif
```

Listing 17: built_ins.fc

```
int[] map(func<int(int)>f, int length, int[] arr)
{
  int[] narr;
  int i;

  narr = new int[length];

  for (i = 0; i < length; i = i + 1)
  {
      narr[i] = f(arr[i]);
  }

  return narr;
}

int reduce(func<int(int, int)> f, int length, int[] arr)
```

```c
{
  int i;
  int result;

  if (length <= 0)
    return 0;

  result = arr[0];
  for (i = 1; i < length; i = i + 1)
  {
      result = f(result, arr[i]);
  }

  return result;
}
```

Listing 18: Makefile

```makefile
# "make test" Compiles everything and runs the regression tests

.PHONY : test
test : all testall.sh
	./testall.sh

# "make all" builds the executable as well as the "printbig"
↪  library designed
# to test linking external code

.PHONY : all
all : func.native printbig.o

# "make func.native" compiles the compiler
#
# The _tags file controls the operation of ocamlbuild, e.g., by
↪  including
# packages, enabling warnings
#
# See
↪  https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc

func.native :
	opam config exec -- \
	ocamlbuild -use-ocamlfind func.native

# "make clean" removes all generated files

.PHONY : clean
clean :
	ocamlbuild -clean
```

```
        rm -rf testall.log ocamlllvm *.diff

# Testing the "printbig" example

printbig : printbig.c
        cc -o printbig -DBUILD_TEST printbig.c

# Building the tarball

TESTS = \
  add1 anon1 arith1 arith2 arith3 array1 array2 array3 array4
   ↪  array5 array6 \
  fib float1 float2 float3 for1 for2 fstcls1 func1 func2 func3
   ↪  func4 func5 \
  func6 func7 func8 func9 gcd2 gcd global1 global2 global3 hello
   ↪  helloworld \
  higherorder1 higherorder2 if1 if2 if3 if4 if5 if6 local1 local2
   ↪  map1 mod1 \
  ops1 ops2 pipe1 pipe2 pipe3 pipe4 pipe5 printbig reduce1 var1
   ↪  var2 while1 \
  while2

FAILS = \
  anon1 array1 assign1 assign2 assign3 dead1 dead2 expr1 expr2
   ↪  expr3 \
  float1 float2 for1 for2 for3 for4 for5 fstcls1 fstcls2 fstcls3
   ↪  func1 func2 \
  func3 func4 func5 func6 func7 func8 func9 global1 global2 if1
   ↪  if2 if3 map1 \
  map2 map3 mod1 nomain pipe1 pipe2 printbig printb print reduce1
   ↪  reduce2 \
  reduce3 return1 return2 while1 while2

TESTFILES = $(TESTS:%=test-%.fc) $(TESTS:%=test-%.out) \
            $(FAILS:%=fail-%.fc) $(FAILS:%=fail-%.err)

TARFILES = ast.ml sast.ml codegen.ml Makefile _tags func.ml
 ↪  parser.mly \
        README scanner.mll semant.ml testall.sh \
        printbig.c \
        Dockerfile \
        $(TESTFILES:%=tests/%)

func.tar.gz : $(TARFILES)
        cd .. && tar czf func/func.tar.gz \
                $(TARFILES:%=func/%)
```

Listing 19: _tags

```
# Include the llvm and llvm.analysis packages while compiling
true: package(llvm), package(llvm.analysis)

# Enable almost all compiler warnings
true : warn(+a-4)

# Instruct ocamlbuild to ignore the "printbig.o" file when it's
↪  building
"printbig.o": not_hygienic
```

9 References and Resources

- The MicroC Compiler, Columbia University Fall 2018, Stephen A. Edwards
- The C Programming Language, 2nd Edition, Brian W. Kernighan, Dennis M. Ritchie.
- Compilers: Principles, Techniques, and Tools (2006), 2nd Edition, Alfred V. Aho, Monica Lam, Ravi Sethi, and Jeffrey D. Ullman
- OCaml Manual, 4.07, Xavier Leroy.
- OCaml.org
- llvm.moe