SCIC

Scientific Calculation Language

Yucen Sun (ys3393)
Zhengyi Li (zl3029)
Zhengyuan Dong (zd2216)

A report presented for the COMS4115 Programming Languages and Compiler course.

Computer Science Department
Columbia University
Date 04/25/2021

Contents

Chapter 1

Introduction

Historically, there have been more than one catastrophic industrial accident due to code that didn't have proper units. For example, Mars Climate Orbiter's failure is due to the failure of auto-conversion from pound-force seconds to newton seconds. Motivated by that, we created SCIC to incorporate unit system in the programming language.

The SCIC language a general purpose language with support of units. It is a statically scoped, statically typed, C-like language for science. Data types and units are explicitly specified for each variable. SCIC has a unit system that supports arithmetic operations and functions with unit. In addition, it allows users to define units based on our basic units and operators. It also supports automatic unit conversion and high-order unit manipulation.

Chapter 2

Language Tutorial

2.1 Environment Setup

Prior to installing SCIC, OCaml, LLVM and gcc have to be installed.

To install Ocaml and LLVM on Ubuntu 20.04, follow the steps below.

```
sudo apt install ocaml llvm-10.0 llvm-10.0-dev llvm-runtime m4 opam
opam init
opam install llvm.10.0.0
eval `opam config env`
```

To install Ocaml and LLVM on MacOS, use the following commands.

```
brew install opam
brew install llvm
opam install llvm
```

To install GCC on your system, refer to the gcc website `https://gcc.gnu.org/install/index.html`.

2.2 Download and Build SCIC

Then we can download SCIC, and use the make command to do the compiling and run test suits.

```
git clone https://github.com/constancedongg/SCIC.git
cd SCIC
```

```
make
```

2.3 Write your first SCIC program and run

Once the SCIC is successfully compiled, you can write your own SCIC program and run it. Here is a example code of a simple SCIC program. You can refer to the Language Reference Manual in the next chapter for more syntax details.

```
|'{km} = 0.001 '{m}|;

float '{km} func foo(float '{m} x) {
    printf(x); /* 30 (m) */
    float '{m} y = x;
    return y;   /* at return: convert to return unit km */
}

int func main() {
    float '{cm} tmp = 3000.0;
    printf(foo(tmp));  /* 0.03 (km)*/
    return 0;
}
```

You can paste the code into myprogram.sc file. Then, use the compiled scic.native to run the program.

```
> ./scic.native myprogram.sc
30
0.03
```

The program run should output two lines 30 and 0.03. For debugging, you can also use options to print out middle steps -a for AST, -s for SAST, -u for USAST, -l for LLVM IR code.

Chapter 3

Language Reference Manual

3.1 Lexical conventions

3.1.1 Comments

SCIC does multi-line comments starting with /* and ending with */.

```
/* my first program in SCIC */
```

3.1.2 Identifiers

An identifier is composed of ascii letters, digits, and underscore ' _'; the first character must be letters.

3.1.3 Keywords

The following identifiers are reserved as keywords and cannot be used to define units or name variables:
int *float* *bool* *string* *if* *else* *for* *return* *print* *true* *false* *func*

3.1.4 Constants

Integer constants

The integer constant is a sequence of digits in decimal; the first digit cannot be 0.

```
int x = 2;   /* allowed */
int x = 02;  /* not allowed to start with 0  */
```

Float constants

The float constant consists of 1 bit for sign, 1 bit for sign of exponent, 7 bits of expoenent and 23 bits of mantissa.

```
float t = 2.078;
```

Boolean constants

The boolean constant is true or false.

```
bool x = false;
```

String constants

The string constant is a sequence of characters enclosed in ” ” ”.

```
string s = "hello world";
```

3.2 Data Types

Data Type	Description
int	32-bit signed integer
float	single-precision 32-bit floating point
bool	true or false
string	immutable array of characters

Data Structure	Description
int[ ]/float[ ]	integer/float array

3.3 Unit System

SCIC features arithmetic operations and formula calculations with units. The prototype supports a part of SI units as base units and their common conversions as built-in units.

SCIC also allows users to define units. User can define own base units and and conversion of base units as well. User-defined unit names must only contain English letters. Derived units can be

Base Units	Description	Quantity
'{s}	second	time
'{m}	meter	length
'{kg}	kilogram	mass
'{A}	ampere	electric current
'{K}	kelvin	thermodynamic temperature
'{mol}	mole	amount of substance
'{cd}	candela	luminous intensity
Non-Base Units	**Description**	**Quantity**
'{cm}	centimeter	length

defined as multiplication, division, and power of defined units. Unit declaration is global.

SCIC supports user-defined functions and equations with units. It also supports automatic unit conversion.

In SCIC, all units begin with identifier " ' " and wrapped by curly braces {} such as '{m} and '{s}. Only defined unit identifiers and their power can go into {}, and constants shall not be placed inside. For example, derived unit $'\{m/s\}$ can be used because m and s are both defined, and we can also define cm per second using $|'\{cm/s\} := 100.0 * '\{m/s\}|$; (more details follow). '{100 * m}, '{undefined-unit / defined-unit} are not allowed.

Variable declaration

with unit:

```
float '{m} dist = 5.5;
float '{s} t1 = 1.0;
float[] '{m/s} speed_list = [1.09, 0.92, 0.883];
```

Define a unit

based on base units. On the right of assignment symbol, constant calculation expression should be separated from the expression in the curly bracket only involving units .

```
|'{mm} = 1000 * '{m}|
/* Define current unit mm as  1m = 1000mm */

|'{g} = 10^(-3) * '{kg} |
```

```
/* Define microampere as uA */
```

3.4 Conversions

3.4.1 Unit conversion

SCIC supports automatic unit conversion if the pre-convertion unit and post-convertion units are both defined and correctly related.

```
|'{cm/s} := 100.0 * '{m/s}|
float '{m/s} dist2 = 0.123;
float '{cm/s} dist1 = dist2;

/* value of dist1: 12.3, unit of dist1: cm/s */
```

3.5 Expressions

The precedence of expressions is descending in the order of the following subsections.

3.5.1 Primary expressions

identifier

Identifier is a primary expression specified in its declaration.

constants

Constant is a primary expression which can be an integer, a float number, a character, a string, and a boolean.

(expression)

An expression with parentheses is a primary expression. The type and value of the expression remains unchanged.

primary expression [expression]

The expression enclosed by square bracket after a primary expression is a primary expression. It is used to access an element in the array.

```
int[] x = [2,3,6,4];    /* initialize list */
int num = x[2];    /* the index of array starts from 0, num is at index 2, which is 6*/
x[1] = 0;  /* x becomes [0,3,6,4] */
```

---

primary expression (expression list)

A primary expression followed by empty or a list of comma-separated expressions enclosed by parentheses is a function call. The expression lists are the parameters passed into the function.

3.5.2 Unary Operators

- expression

The result is negative of the expression.

```
int x = -5;     /* x is negative 5 */
```

! expression

The result is logically negative of the expression.

```
bool x = true;
bool y = !x;     /* y is false */
```

3.5.3 Multiplicative operators

The multiplicative operators are * for multiplication, / for division, evaluated from left to right. For division, if two literals are integers, the result is integer truncated to 0; if one or both literals are float numbers, the result is float.

```
int x = 2;
int y = 3;
float z = 4.0;
int mul = x * y;          /* mul = 6 */
int div = y / x;          /* div = 1*/
float res = z / x;        /* res = 2.0 */
```

3.5.4 Additive operators

The additive operators are + for addition, - for subtraction, evaluated from left to right. If one of operands is float, the result is float. If both operands are integer, the result is integer.

```
int x = 2;
float y = 3.0;
x + y;          /* 5.0 */
```

3.5.5 Relational operators

The relational operators are < (less than), > (greater than), <= (no greater than), >= (no less than). They yield true if the relation between two operands is true, otherwise, false.

3.5.6 Equality operators

The equality operators are == (equal to) and ! = (not equal to). They compare values of two operands and yield true if two values are the same.

3.5.7 Logical operators

The logical operators are && (logical and) and || (logical or). && yields true only if operands are true. || yields false only if both sides are false.

3.5.8 Assignment operators

Assignment is identifier = expression. Two operands should be the same type.

```
int x = 2;          /* correct */
int x = 2.0;        /* not allowed */
float y = 0;        /* not allowed */
```

3.6 Declarations

3.6.1 Variable Declaration

Variable declaration follows the form

type unit identifier where type is limited to float and int.

or

type identifier without unit.

```
/* declare variable y of float type and unit meter*/
float '{m} y = 1.0;      /* y is 1.0 meter*/
int x = -5;              /* x is negative 5 */
```

3.6.2 Function Declaration

A function takes a list of arguments and returns a value. The body of a function is wrapped by curly braces. All arguments and returned value in a function declaration should have legal unit; or none of them should have units.
SCIC enforces function declaration and definition at the same time, and it follows the form:

type (unit) func identifier(arguments) {statements}

Examples of the two types of function declarations are as follows:

```
/* Function declaration with units */
float '{m} func dist_speed(float '{m} X1 , float '{m/s} Va , float '{s} T ) {
    return X1 + Va * T;
}

/* Function declaration with no units */
int func add2 (int a) {
    return a + 2;
}
```

*More details of the expression to call equations in SCIC in Statements section.

3.7 Statements

statement are executed in sequence

3.7.1 Expression statement

expression
most of statement is an expression, and expression can be unit type define, variable assignment, function calls, or equation calls

3.7.2 Conditional Statement

if (expression) statement

if (expression) statement else statement

```
int x = 0;
if (true) x = 5;
if (true) x = 10 else x = 20;
```

the "else" is connected to the last "if" if there is ambiguity

3.7.3 For Statement

for (expression 1; expression 2; expression 3) statement

the first expression set up the initial state, the second expression check if loop can continue, and the third expression specifies the incrementation after each iteration

```
int x = 0;
int i = 0;
for (; i <= 10; i = i+1){
    x = i+1;
}
```

int i = 0 is expression for setting up initial state. $i <= 10$ is the expression for checking loop stop. **i=i+1** is what variable will be increment after each loop.

3.7.4 Return Statement

return expression;

the first statement will have no value return. the second statement will return value when function calls it.

```
/* return statement */
int func add(int x){
    return x+2;
}
```

3.8 Scope rules

Scope means when creating a function or equation, scope was created with hierarchy.

When **Scope** was form, then lower level of **Scope** cannot access higher level of **Scope**

The lower level scope can access variables and functions in higher level of scope

Default **Scope** is global scope, which is lowest level of scope

---

```
/* create a variable in global scope */
int y;

/* creating a function or equation will form a new scope */
int func add() {
    /* here is a higher level of scope */
    /* access and modify the variable in lower level */
    y = 2;
    int x = 5;
    return x + y;
}

int func test() {
    /* error: cannot access x in different function */
    return x;
}
```

3.9 Library Functions

print, printf, printl, printb would print integer, float, string, boolean correspondingly.

```
int x =10;
print(x);
/* console log: 10 */

float x =0.1;
printf(x);
/* console log: 0.1 */

string = "hello world";
printl(string);
/* console log: "hello world"*/

```

```
bool predicate = true;
printb(predicate);
/* console log: 1*/
```

3.9.1 Array Operation

Initialization

Array can be initialized with expressions surrounded by square brackets.

```
int x = 5;
int[] arr = [2 * x, x * x, (10 - 3) * 2, 5];

/* arr = [10, 25, 14, 5] */
```

Access

Array operation supports using array[index] to find the value at index position and doing modification

```
int[] a = [1,2,3,4];
a[0] = 0;
print(a[0]);       /* 0 */
```

3.10 Examples

```
/* example of recording experiment result */
|'{mm} = 1000.0 '{m}|;

void func foo(float '{m/s} base) {
    int i = 0;
    float[] '{m} dx = [2.3, 4.5, 3.4, 0.7];
    float[] '{s} dt = [0.5, 0.2, 1.7, 0.5];
    float[] '{mm/s} res = [0.0, 0.0, 0.0, 0.0];
    for (; i < 4; i = i + 1) {
        res[i] = dx[i] / dt[i] + base ;   /* 2.3m / 0.5s = 4.6 m/s + 0.0122 m/s = 4.6122 m
```

```
11          printf(res[i]);
12      }
13  }
14
15  /* constant acceleration formula dx = at^2 */
16  int func main() {
17      float '{m} start1 = 1.05;
18      float '{m} end1 = 2.05;
19      float '{m} dx1 = end1 - start1;    /* dx = 1.0 */
20
21      float '{m} start2 = 2.05;
22      float '{m} end2 = 4.05;
23      float '{m} dx2 = end2 - start2;     /* dx = 2.0 */
24
25      float '{s} dt = 0.5;
26
27      float '{m/s*s} acceleration = (dx2 - dx1) / (dt * dt);
28      printf(acceleration);
29
30      float '{cm/s} base = 1.22;
31      foo(base);                  /* 1.22 cm/s = 0.0122 m/s */
32      return 0;
33  }
```

Chapter 4

Project Plan

SCIC was from the beginning proposed to be general purpose language with support of units.

Overall, we achieved our goal. The proposal has some minor variations from the current version of SCIC. In terms of delivery, the goal was not to strictly meet the original proposal, but to learn the compile architecture and flow.

The proposal raised up three key ideas, unit auto-conversion, user self-defined units, and solving equations automatically. After discussing with TA Evan, we found it might be hard to realize equation solving in our language except very simple equations. Therefore, we focus on unit and auto-conversions, and leave out equations in this project.

4.1 Project management

With a group of only 3 members, the project management and communication is quite efficient.

We have a group chat channel for arranging meeting and daily communications. For programming, we did sole programming and group programming alternatively. Tiny group size makes both modes effective. Due to the fact that members are in different time zones, we typically had a short summary meeting in the morning/night, then did our own work until next meeting. We typically met twice a week during active weeks and once a week during busy weeks. The meeting typically revolved around updates, merging new results together, and delegating tasks. If there were bugs that cannot be solved by a single team member, we extended the meeting to make it group programming. We used Liveshare in VSCode to code together and github to control version and source code.

4.2 Programming Style Guide

SCIC's programming style is similar to C syntax with some subtle differences.

- Indentation: tabs with a length of 4 characters are used for the indentation. Braces go in the same line.

- Comment: multiple-line comment /* */
- Main function: main must return 0 for normal exit
- Snake case variable, function names
- Curly braces indicating scopes

4.3 Timeline

Date	Milestone
Feb 3	Proposal
Feb 22	Language Reference Manual
Feb 24	Parser
Mar 24	Hello World Milestone
Mar 26	Library functions
Mar 27	Operators
Apr 9	Unit initialization
Apr 10	Local declare and assign
Apr 24	Unit check layer
Apr 25	Test suite
Apr 25	Presentation

4.4 Roles and Responsibilities

Member	Role	Responsibility
Zhengyuan Dong	Project Manager/Tester	Code merge, array, loops, basic unit conversion, tests
Zhengyi Li	System Architect/Tester	compiler architecture, local dec-assign, unit propagation, tests
Yucen Sun	Language Guru/Tester	language design, parser, unit declaration, derived unit exprs, tests

4.5 software development environment

We used the following programming and development environments:

1. **Libraries and Tools:** Ocaml Version 4.08.1 (including Ocamlyacc and Ocammllex)
   LLVM Version 10.0.0
   gcc Version 9.3.0 (Ubuntu 9.3.0-17ubuntu1 20.04)
   clang Version 10.0.0-4ubuntu1

2. **System:** macOS 11.0.1, WSL Ubuntu 20.04.2 LTS

3. **Container environment:** Docker image provided by the course

4. **Software:** VScode
5. **Version Control:** github

4.6 Git log

```
commit de6025b14554623a8b7439fc11d723534a81cbc3
Merge: b0fb759 2823ac5
Author: yucensun <kallla@umich.edu>
Date:   Mon Apr 26 17:38:30 2021 -0400

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit b0fb759a143a781598940ead02102be0e0a0a674
Author: yucensun <kallla@umich.edu>
Date:   Mon Apr 26 17:38:15 2021 -0400

    change test case

commit 2823ac5c9ef2af6c09a4d3696287de11a275c9a5
Merge: 4cfeafa b4d23a8
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Mon Apr 26 16:55:51 2021 +0800

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit 4cfeafa65b65e2ce0630d65bb19a6254c8299fce
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Mon Apr 26 16:55:27 2021 +0800

    add one array test case

commit b4d23a8bf9f2d44ace9c89c384cd33d604404f5d
Author: yucensun <kallla@umich.edu>
Date:   Mon Apr 26 00:51:14 2021 -0400

    mod readme

commit d6833b0b810043d2447dc27071ca8180c91610b5
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Mon Apr 26 12:33:08 2021 +0800
```

```
    modify README

commit a2f4334377a444f324700bcd05cae3344918838f
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Mon Apr 26 10:30:36 2021 +0800

    delete scic.native

commit 9319087104ed1d2351e6e73d2b24e800035012a4
Author: yucensun <kallla@umich.edu>
Date:   Sun Apr 25 22:25:01 2021 -0400

    rename

commit 7ffffcbe8eaf56f7ab600bbcb8964a6f66f3219d
Author: yucensun <kallla@umich.edu>
Date:   Sun Apr 25 20:26:08 2021 -0400

    before dm

commit 669decd5414adf9a8ce1ed981dcb7961b7e2209a
Merge: 63a3f50 0177fab
Author: yucensun <kallla@umich.edu>
Date:   Sun Apr 25 18:13:53 2021 -0400

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit 63a3f502812cca000aeea3d60dc44ff5fa3da571
Author: yucensun <kallla@umich.edu>
Date:   Sun Apr 25 18:13:33 2021 -0400

    change binop return unit

commit 0177fabd30d62eb1be96202873aed25b515e02cc
Merge: 03d14cc 58662c9
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sun Apr 25 12:07:01 2021 -0400
```

```
    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit 03d14ccbdbf38e4d666d4806e2010338b114e851
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sun Apr 25 12:02:36 2021 -0400

    fix the test case

commit 58662c94fe7608d664acea2c1a85d05712f47ddc
Author: yucensun <kallla@umich.edu>
Date:   Sun Apr 25 12:02:15 2021 -0400

    change tc

commit 5dcdfbc9299a4be1213ab8612e1d7131bf79acb7
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sun Apr 25 12:02:05 2021 -0400

    fix the test case

commit 9f9c6dd8aad4ae5753a37b689d733ca7a5b2dd88
Merge: 97331f1 989e665
Author: yucensun <kallla@umich.edu>
Date:   Sun Apr 25 11:59:31 2021 -0400

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit 989e665164d608b1ac9be10850d841a459aebcd7
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sun Apr 25 11:56:38 2021 -0400

    fix if else bug

commit 97331f1cb492f445fe7393fa3a21502abf642735
Merge: e64838a 8ac1ec4
Author: yucensun <kallla@umich.edu>
Date:   Sun Apr 25 09:39:41 2021 -0400

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main
```

```
commit 8ac1ec4599bec090b3b24e4224d58591a11e6c74
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sun Apr 25 20:09:08 2021 +0800

    clean codes, add more test cases, fix function call formals order bug

commit e64838a699fa1cb05413205ddb0b09bd4cd2e640
Merge: 0b02956 36f87f2
Author: yucensun <kallla@umich.edu>
Date:   Sun Apr 25 00:04:44 2021 -0400

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit 0b029567e7c0d31c9860a73de0725a7bdeb6b637
Merge: 7034ec7 c4592af
Author: yucensun <kallla@umich.edu>
Date:   Sun Apr 25 00:04:40 2021 -0400

    m?

commit 36f87f2bb68b159c308bc691167e93933b8922f0
Author: Zhengyuan Dong <47376081+constancedongg@users.noreply.github.com>
Date:   Sat Apr 24 21:02:42 2021 -0700

    add unit array test cases to support array with unit (#13)

    * derived unit comp

    * mixed unit manipulation, unit binop, stringmap of units

    * add unit array test cases to support array with unit

    Co-authored-by: yucensun <kallla@umich.edu>

commit c4592af56b43e02eb53963902dfd4485850c3cdd
Author: Zhengyuan Dong <47376081+constancedongg@users.noreply.github.com>
Date:   Sat Apr 24 20:13:28 2021 -0700
```

```
    mixed unit manipulation, unit binop, stringmap of units (#12)

    * derived unit comp

    * mixed unit manipulation, unit binop, stringmap of units

    Co-authored-by: yucensun <kallla@umich.edu>

commit 7034ec73e0910929ba0b316bfa65a5990a0a4d12
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 21:27:53 2021 -0400

    derived unit comp

commit 5589c54a4cf9f00667cd70ec1e6f4d448b5993c5
Merge: 7a38109 c2f5b08
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 16:29:41 2021 -0400

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit 7a38109009fc7929d9bf48db555d1d1eb1977f14
Merge: 9f3a4ed eebeffc
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 16:29:28 2021 -0400

    add funcall test

commit c2f5b08f4a78071ad8b6319598db9830912b6ddb
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sat Apr 24 16:26:09 2021 -0400

    change funcall4.out last value

commit 9f3a4edb478d871d38d603fdb38fd00c505edbbb
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 16:17:13 2021 -0400

    wht chg
```

```
commit eebeffcb5d5c39f58e1949142715b8f51a71ad8a
Merge: e4be520 a5e5e13
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sat Apr 24 16:15:02 2021 -0400

    nwew Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit e4be520c5abce5fc7ecbb036e1ee1d560cf97e70
Merge: 5ead5fb 05bbc6e
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sat Apr 24 16:13:57 2021 -0400

    Merge branch 'unit-add-minus' into main

commit a5e5e1397af7b70a265cc6ab1cdbd8bead44abc4
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 13:11:40 2021 -0400

    add test funcall4

commit 5ead5fb71b63bcc5f5aa4dd63bc7fdfd2624b43a
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 13:00:30 2021 -0400

    funcallUC2 problem

commit f907b0fa1e872e669d219fd1b8de5f89a3ac2fc8
Merge: d148910 847351f
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 12:40:41 2021 -0400

    Merge remote-tracking branch 'refs/remotes/origin/main' into main

commit d14891009b4d5b4d8f5795c343612d3e9b33745e
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 12:40:36 2021 -0400

    convert ub2nb
```

```
commit 05bbc6e0f76fa187d700d160fce5b4b58f2c8e92
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sat Apr 24 12:29:55 2021 -0400

    pass add and sub

commit 847351fba283271e6fc54e63ef5e7d6b49bb5782
Merge: bd288e5 00d9b17
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Apr 24 22:37:39 2021 +0800

    add test cases

commit bd288e521e22cf0f61c471ae2d0d196ef7e021a1
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Apr 24 22:34:21 2021 +0800

    add unit return, formals, declare assign test cases, fix return lunit runit bug

commit 00d9b1741bf1763c3c3fc90479ccfba28e1064c8
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 10:03:41 2021 -0400

    clean

commit 04ef5ae01d440222c5a29423baec7b53dad14a6c
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 10:03:21 2021 -0400

    tests and target

commit 10e0b75acbcd869289bec00130a9d37a189da97d
Merge: 5117989 a1a4ba7
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Apr 24 21:36:00 2021 +0800

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main
```

```
commit 51179893bf548c4f7512365ecea029438d9a7edd
Merge: e1739ac 7f7101d
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Apr 24 21:35:29 2021 +0800

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit a1a4ba72f5788e21cda0084bf61cb03cc4e2d156
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 09:35:26 2021 -0400

    clean

commit 7f7101d536b51a53db5cb004448d7fdb3d30cedc
Merge: 0c0ba66 74d56e7
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 09:32:29 2021 -0400

    float print

commit e1739ac84317726c61a316cdaa33133369fb0478
Merge: 2a7a1cc 74d56e7
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Apr 24 21:30:13 2021 +0800

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit 74d56e76967b12da6c3bdca0078c089623a18531
Author: Zhengyuan Dong <47376081+constancedongg@users.noreply.github.com>
Date:   Sat Apr 24 06:29:39 2021 -0700

    merge unit-base to main (#11)

    * inital unicheck

    * add some comments

    * part debug & problems in expr
```

```
    * debug most unit

    * somechanges

    * llc

    * fix tons of unit convert bugs

    Co-authored-by: ElevenLe <zl1499@nyu.edu>
    Co-authored-by: yucensun <kallla@umich.edu>

commit 2a7a1cc5b0293043c308fc53562c94f227c61218
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Apr 24 13:45:21 2021 +0800

    llc

commit 8bd2ee547923c46058250a96403732c055155513
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Apr 24 13:44:21 2021 +0800

    modify parser to acccept only float data with unit

commit 096117eeca1e6c90a11c6b7e49e4a19460c8ef0f
Author: Zhengyuan Dong <47376081+constancedongg@users.noreply.github.com>
Date:   Fri Apr 23 22:17:17 2021 -0700

    unit decl w only base (#10)

    Co-authored-by: yucensun <kallla@umich.edu>

commit 0c0ba66eddcdf975308d701c40bf75413d042a50
Author: yucensun <kallla@umich.edu>
Date:   Sat Apr 24 00:50:53 2021 -0400

    unit decl w only base

commit 4e4feff56a4384e649d16cc2907cd73c6df8ed43
Author: Zhengyuan Dong <constancedongg@gmail.com>
```

```
Date:   Fri Apr 23 12:56:35 2021 +0800

    delete trash files

commit 7a042695efba569d6ff8e15ba063cc3050bf5303
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Fri Apr 23 12:42:31 2021 +0800

    add draft

commit 0ec917faa8e4b727171890fedbdbdb680735008d
Author: yucensun <kallla@umich.edu>
Date:   Thu Apr 22 21:35:42 2021 -0400

    local dassign + unit (dummy)

commit 4db070609c37da59fae0d6bb622ab17888a0f36c
Merge: 4c6a3ca 12bf2ce
Author: yucensun <kallla@umich.edu>
Date:   Thu Apr 22 17:06:53 2021 -0400

    merge

commit 4c6a3caf11edae79b71dd08b61d5c5f6aa5cf2e3
Author: yucensun <kallla@umich.edu>
Date:   Wed Apr 14 16:56:08 2021 -0400

    add unitcheck & fix pow lib

commit 12bf2ce2331a831e34d9f8f42b5120d98b2405bf
Author: ElevenLe <zl1499@nyu.edu>
Date:   Mon Apr 12 23:19:42 2021 -0400

    some test case

commit ad3fccfd6ee823f9c5058a7eeeb7bd6d971ee2fe
Author: ElevenLe <zl1499@nyu.edu>
Date:   Mon Apr 12 21:34:10 2021 -0400
```

```
    while debug

commit d99ebb0edf8f0e6f80499203b919cacf735e4f4c
Author: ElevenLe <zl1499@nyu.edu>
Date:   Mon Apr 12 21:29:19 2021 -0400

    more debug

commit 82caf161e9f36e7b0c2e4cb91c2bf568c87fb179
Author: yucensun <kallla@umich.edu>
Date:   Mon Apr 12 21:13:02 2021 -0400

    before binop

commit 24f0e8f172ade4cb5f9f8cf1a5f495d6c8433bd9
Author: ElevenLe <zl1499@nyu.edu>
Date:   Mon Apr 12 21:02:02 2021 -0400

    debug unit

commit 5d617ce6895eba8727acf1b9904402b06ab758b1
Author: ElevenLe <zl1499@nyu.edu>
Date:   Mon Apr 12 20:53:10 2021 -0400

    pri

commit fd911f855b3aacc48e2183b300ac7efe0d1ee254
Author: ElevenLe <zl1499@nyu.edu>
Date:   Mon Apr 12 20:52:11 2021 -0400

    complied

commit a82e9d7496ab550c66f6f6324061ad4694da2bd1
Merge: f8d79e2 32195c4
Author: ElevenLe <zl1499@nyu.edu>
Date:   Mon Apr 12 20:41:52 2021 -0400

    Merge branch 'unit-init' into main
```

```
commit 32195c4d7ac99bb996e6ba83139e48b5669f8995
Author: ElevenLe <zl1499@nyu.edu>
Date:   Mon Apr 12 20:23:13 2021 -0400

    make clean

commit f8d79e2f575f35631b079b0cb34cf61031d01e76
Author: Zhengyuan Dong <47376081+constancedongg@users.noreply.github.com>
Date:   Sun Apr 11 02:49:00 2021 -0700

    Zhengyuan list (#8)

    * add array initial & assign feature, test cases, modify Makefile

    * add for & while feature, add test cases, modify makefile

commit 59af52746e72caabef34e0d87211eb866ff737ea
Author: Zhengyuan Dong <47376081+constancedongg@users.noreply.github.com>
Date:   Fri Apr 9 17:59:39 2021 -0700

    add funcall test cases (#7)

commit f2545e21a868a725b14b983635d2416e914026fa
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Apr 10 08:49:09 2021 +0800

    modify .gitignore

commit 1a85fa46600498a09b61b3fab26a6cbf94737260
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Apr 10 08:45:40 2021 +0800

    remove trash

commit 8734626227951414c2ddd411a06eca757d46f4e7
Merge: 3fe956a 2904341
Author: Zhengyuan Dong <47376081+constancedongg@users.noreply.github.com>
Date:   Fri Apr 9 17:18:23 2021 -0700
```

```
    Merge pull request #6 from constancedongg/assign-decl-int

    Assign decl

commit 5f42032e04d40b98f8b332a144d298d8ec903b70
Author: yucensun <kallla@umich.edu>
Date:   Fri Apr 9 20:00:53 2021 -0400

    unit init f

commit 290434151464e3f07751596944895e21eec3d6a9
Author: ElevenLe <zl1499@nyu.edu>
Date:   Fri Apr 9 12:31:22 2021 -0400

    codgen pass

commit 6aeac26fd01005213adca77a6b3cc92a7342d531
Author: ElevenLe <zl1499@nyu.edu>
Date:   Fri Apr 9 12:31:13 2021 -0400

     codegen pass

commit 548d2ba430dad28621287be4e0656d6dd45d3a8f
Author: ElevenLe <zl1499@nyu.edu>
Date:   Fri Apr 9 12:24:22 2021 -0400

    semant pass variable

commit dccf08599b6a4c757b911704a58d8c4b429eabcb
Author: ElevenLe <zl1499@nyu.edu>
Date:   Wed Apr 7 20:16:28 2021 -0400

    Co-authored-by: yucensun <yucensun@users.noreply.github.com>

commit 8b20f07e120a6c222ff36a559a65862e8e16ccc7
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sun Apr 4 12:24:23 2021 -0400

    bugs
```

```
commit a7013dfdbb3731c99dbbfba5cd0515b147285911
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sat Apr 3 17:31:17 2021 -0400

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit 990dd44fb118d99aec3f025d408172acc73c9ee1
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sat Apr 3 14:43:03 2021 -0400

    add gloable var

commit 3fe956a55aeb26c66018b2597f3b2c523c650d5f
Author: yucensun <43123355+yucensun@users.noreply.github.com>
Date:   Sat Mar 27 17:27:11 2021 -0400

    merged

commit db7f7418342d94784876374a6e7a4886d93a333f
Merge: 5dbfe00 8bb3373
Author: yucensun <43123355+yucensun@users.noreply.github.com>
Date:   Sat Mar 27 17:23:29 2021 -0400

    Merge branch 'string-and-char' into main

    merge 3

commit 8bb3373c7cd9a076a495b99c335ebf4648222635
Merge: e62d5a6 5dbfe00
Author: yucensun <43123355+yucensun@users.noreply.github.com>
Date:   Sat Mar 27 17:23:13 2021 -0400

    commit to avoid overwrite

commit 5dbfe00c825d187c09e891184d19ea144d0fd031
Merge: 7412d9b 82b0a7d
Author: yucensun <43123355+yucensun@users.noreply.github.com>
Date:   Sat Mar 27 16:13:58 2021 -0400
```

```
    Merge pull request #1 from constancedongg/binops

    Binops

commit e62d5a6e49185e5cf182014245dfb8dda1a63157
Author: ElevenLe <zl1499@nyu.edu>
Date:   Fri Mar 26 23:36:18 2021 -0400

    both print float and print string works

commit 27a426d3e5710c9277ee11e893e54d036323ffb5
Author: ElevenLe <zl1499@nyu.edu>
Date:   Fri Mar 26 21:44:31 2021 -0400

    print string works

commit 0efd36155ca48d85082a40e62b4b6ea478495f0e
Author: ElevenLe <zl1499@nyu.edu>
Date:   Fri Mar 26 16:57:53 2021 -0400

    encouter problem with string and char

commit 132ea764ee317bb7bddeb308e6112d43b9130023
Author: ElevenLe <zl1499@nyu.edu>
Date:   Fri Mar 26 16:54:00 2021 -0400

    change the parser

commit 3e28c3382957a7fe03432c20ac93175ca74234ce
Author: ElevenLe <zl1499@nyu.edu>
Date:   Fri Mar 26 12:26:30 2021 -0400

    add char

commit 1dc1b075bcc1280133c301b7da7662580bdccdfd
Author: ElevenLe <zl1499@nyu.edu>
Date:   Thu Mar 25 20:05:18 2021 -0400
```

```
    illegal character

commit 50da88b3cf0f39a9841183a02ab03d1e02c52d86
Author: ElevenLe <zl1499@nyu.edu>
Date:   Thu Mar 25 20:03:41 2021 -0400

    Adding char and string parser

commit 82b0a7d5c4d1b9ff8ffcb7ce0ccf198f69ff2b02
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Thu Mar 25 18:22:15 2021 -0400

    add tests

commit b4d599ac329718d73442e6a5c1da2a910ca12648
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Thu Mar 25 18:18:09 2021 -0400

    add unop & binop

commit ec72ee559123195b3dca7be95a25c6f7be37f5c5
Author: ElevenLe <zl1499@nyu.edu>
Date:   Thu Mar 25 16:54:45 2021 -0400

    remove tar

commit ba7b5682c9f467fd68f6fea501e0e860068cce4d
Author: ElevenLe <zl1499@nyu.edu>
Date:   Thu Mar 25 16:28:58 2021 -0400

    one make

commit aad40e26e4089f2246a95f1cbfdd92742ba6fe40
Author: ElevenLe <zl1499@nyu.edu>
Date:   Thu Mar 25 16:20:11 2021 -0400

    make works

commit 3bd937c82a119f72bb028f366d9caee5651a5775
```

```
Author: ElevenLe <zl1499@nyu.edu>
Date:   Thu Mar 25 14:51:28 2021 -0400

    adding test case for C and String

commit 7412d9ba4d51dd48d6f425179bd8acc2a78e917b
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Tue Mar 23 16:44:52 2021 -0400

    add submit tar

commit 526d3d7496974bd710c59dbef58f649c533c0964
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Mon Mar 22 17:31:48 2021 -0400

    hello worl

commit 79d9ee200153e437f9786ec736846b16e7b6c16c
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Sun Mar 21 11:20:28 2021 -0400

    complied but error

commit db0e2aa87f63780193f988f8eadff7a939afd00f
Merge: 3c936b6 8bad0aa
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Sat Mar 20 18:23:52 2021 -0400

    ?

commit 3c936b6998b1018d341c9c31e90567bd4636685b
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Sat Mar 20 18:22:31 2021 -0400

    aborttt

commit 8bad0aa7ca2183eb9087b93daff65b399ff343a4
Author: ElevenLe <zl1499@nyu.edu>
Date:   Sat Mar 20 18:21:50 2021 -0400
```

```
    Co-authored-by: yucensun <yucensun@users.noreply.github.com>

commit 20632dc2e240b2b4b7055d70b03cea834363ac48
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Fri Mar 19 21:08:43 2021 -0400

    0319/first-commit

commit 3e1e08b4dac6a9e407850d7c62ca72e77eb7718f
Author: ElevenLe <zl1499@nyu.edu>
Date:   Wed Feb 24 23:09:27 2021 -0500

    no append

commit f6ca7178a95198302ca61d52fb554559692c837a
Author: ElevenLe <zl1499@nyu.edu>
Date:   Wed Feb 24 23:01:24 2021 -0500

    add function call

commit d52fdbb9b7d5631f8ee7a48e2dee4eda1b7a60d7
Author: ElevenLe <zl1499@nyu.edu>
Date:   Wed Feb 24 22:28:30 2021 -0500

    more libs!

    add i2f, f2i, ceil, floor

commit d926c9c004fe8471648c7c944e1f7c1c25a5e2c4
Author: ElevenLe <zl1499@nyu.edu>
Date:   Wed Feb 24 22:12:31 2021 -0500

    Move print to statement

    aovid print(print(print()))

commit cb6f0bf092daeb140549a3b1a1a73140e226555f
Author: ElevenLe <zl1499@nyu.edu>
```

```
Date:   Wed Feb 24 21:52:22 2021 -0500

    useless

commit 3c8d5a60ddec082a069d5157ff062e51a26ae782
Author: ElevenLe <zl1499@nyu.edu>
Date:   Wed Feb 24 21:52:10 2021 -0500

    Compiled with list and lib

    Add some list operation and libs

commit a91b7dc0bcaf4e013d892e0151bea832f8def637
Merge: 9f881cd 4912174
Author: ElevenLe <zl1499@nyu.edu>
Date:   Wed Feb 24 21:02:24 2021 -0500

    Merge branch 'main' of https://github.com/constancedongg/SCIC into main

commit 9f881cd65dfba2f902a57e4a04ac1147f9291422
Author: ElevenLe <zl1499@nyu.edu>
Date:   Wed Feb 24 21:02:12 2021 -0500

    Update parser.mly

commit 4912174c61bb83681018085a3c07d1fc580834b3
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Wed Feb 24 17:56:11 2021 -0500

    corrected stupid err

commit 525f7d7b5e62bda71a5a0b28e0f06dc68392435f
Author: Yucen Sun <yucensun@Yucens-MacBook-Pro.local>
Date:   Wed Feb 24 16:10:04 2021 -0500

    unit & decl

commit a972c857c6a19853c51b71160fe0e801a9fba9e9
Author: ElevenLe <zl1499@nyu.edu>
```

```
Date:   Wed Feb 24 16:06:47 2021 -0500

    Update parser.mly

commit e0371b61b28c8ca73f9b627fff4c63f4db01dc38
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Feb 20 20:27:11 2021 -0800

    add token, type, precedence, expr

commit 8740fda9c1ba0a4d5a104f5ef4903f5584b86aba
Author: Zhengyuan Dong <constancedongg@gmail.com>
Date:   Sat Feb 20 16:11:15 2021 -0800

    first commit
```

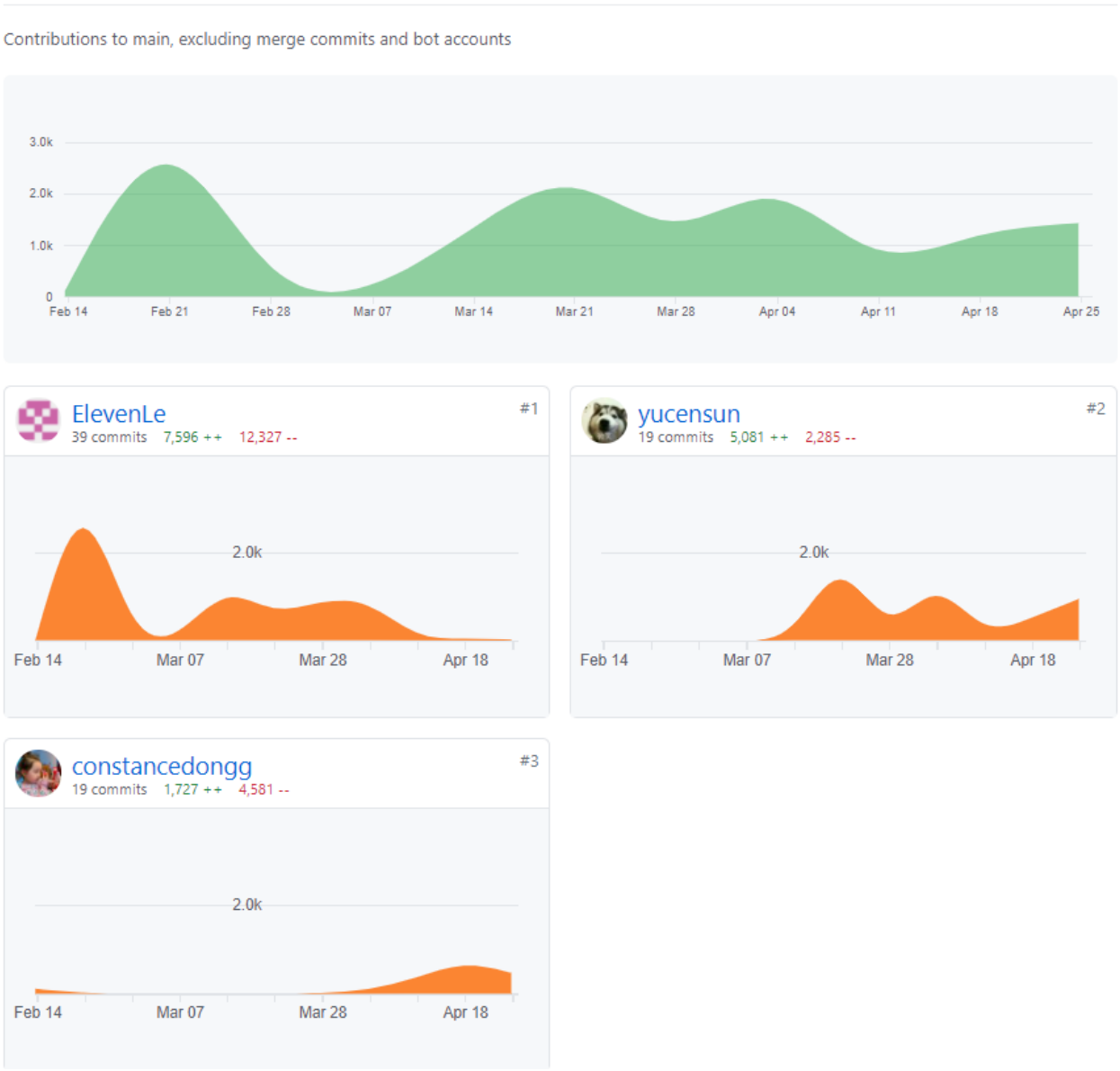

Figure 4.1: Git commits plot

Chapter 5

Architectural Design

The compiler is mainly coded in OCaml. This takes the SCIC .sc source code and compiles it into LLVM IR.

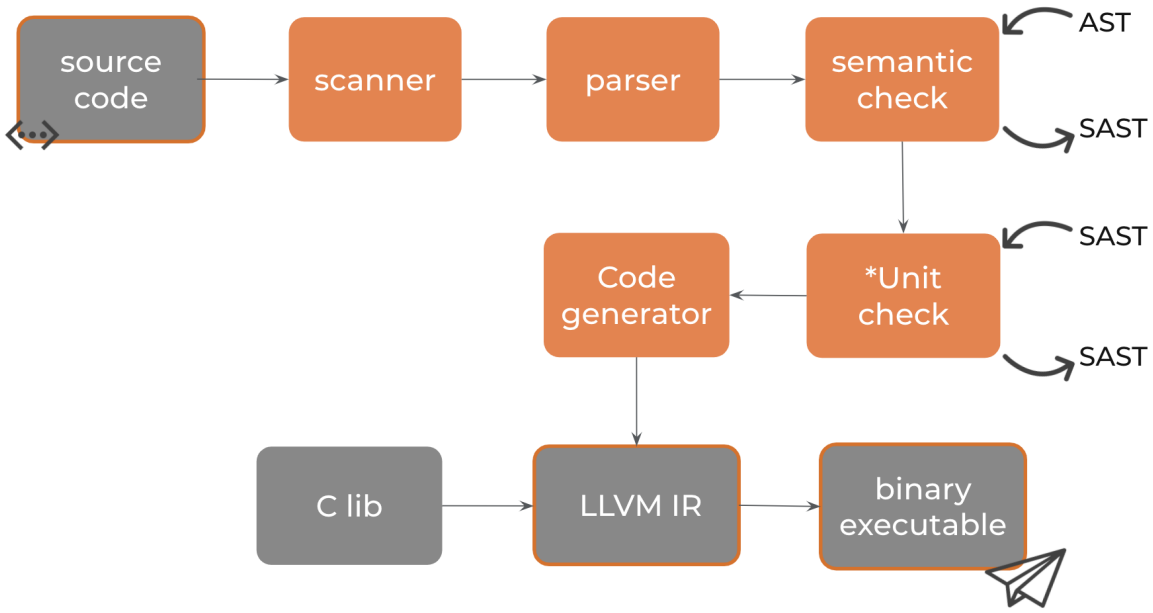

Figure 5.1: Compiler architecture

5.1 Front-end (Scanner and parser)

The SCIC source code is first tokenized by scanner (scanner.mll) and then past to parser (parser.mly) to generate an abstract syntax tree following the ast.ml as template.

5.2 Semantic checker

Sement.ml focus on the normal semantic check including type check, expression validation, and variable and function declarations. It walks through the AST and generate an SAST if successful.

5.3 Unit checker

Unit checker (unicheck.ml) is an additional layer dedicated to check all rules about units including unit declarations, units for variables, expressions and functions; Unitchecker is also responsible to apply unit conversion by modifying semantically checked expressions (sast) before feeding to code generator. Because we modify sexprs to do unit conversion, there is not any change in type and we don't need to preserve units for later. Therefore, the unit checker Walks through the SAST from semantic checker and return an SAST.

5.4 Code generator

Codegen.ml takes in the sast generated from unit-checker, link C library and generate LLVM IR. Apart from stdlib.c, we also link C math library for power calculation. Our code generator is modified and added on the base of microC code generation.

5.5 Work division

All three members of our team are involved in every part of the architecture, which is due to the nature of ocaml programming. We basically assign features to each member, and one need to code for every stage of the pipeline.

Chapter 6

Test Plan

6.1 Development Process

Test suite is enriched during the process of development of whole language. Every feature is accompanied and tested by a bunch of unit test cases and integrated test cases. At the beginning, the purpose of test guided the development process of our language. For example, we develop build-in functions like print first and the declarations first because they are supposed to be used by most test cases. Then arithmetic expressions, statements, etc. In addition to "test" cases (cases whose names start with 'test'), we also developed many failing tests. Testing to fail entails pushing our language to its limits by running it through a range of demanding scenarios.

A total of 70 test cases are included in the next section. The test cases includes unit tests for every little feature, and the integrated tests, and also real-world application tests for scientific calculation. The name of test cases and the comments at the beginning shows their main focus.

6.2 Representative test cases

6.2.1 Test 1

An integrated test case that mimics program for experiments, testing array manipulation and unit conversion, with function.

SCIC code

```
1  /* example of recording experiment result */
2  |'{mm} = 1000.0 '{m}|;
3
4  void func foo(float '{m/s} base) {
5      int i = 0;
6      float[] '{m} dx = [2.3, 4.5, 3.4, 0.7];
7      float[] '{s} dt = [0.5, 0.2, 1.7, 0.5];
8      float[] '{mm/s} res = [0.0, 0.0, 0.0, 0.0];
```

```
    for (; i < 4; i = i + 1) {
        res[i] = dx[i] / dt[i] + base ;   /* 2.3m / 0.5s = 4.6 m/s + 0.0122 m/s =
↪   4.6122 m/s = 4612.2 mm/s */
        printf(res[i]);
    }
}

int func main() {
    float '{cm/s} base = 1.22;
    foo(base);                  /* 1.22 cm/s = 0.0122 m/s */
    return 0;
}
```

LLVM IR

```
; ModuleID = 'scic'
source_filename = "scic"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.4 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.5 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1

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

declare i32 @printbig(i32)

define i32 @main() {
entry:
  %base = alloca double, align 8
  store double 1.220000e+00, double* %base, align 8
  %base1 = load double, double* %base, align 8
  %tmp = fmul double %base1, 1.000000e-02
  call void @foo(double %tmp)
  ret i32 0
}

define void @foo(double %base) {
```

```
entry:
  %base1 = alloca double, align 8
  store double %base, double* %base1, align 8
  %i = alloca i32, align 4
  store i32 0, i32* %i, align 4
  %dx = alloca double*, align 8
  %malloccall = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr
   ↪  (double, double* null, i32 1) to i32), i32 4))
  %0 = bitcast i8* %malloccall to double*
  %1 = getelementptr double, double* %0, i32 0
  store double 2.300000e+00, double* %1, align 8
  %2 = getelementptr double, double* %0, i32 1
  store double 4.500000e+00, double* %2, align 8
  %3 = getelementptr double, double* %0, i32 2
  store double 3.400000e+00, double* %3, align 8
  %4 = getelementptr double, double* %0, i32 3
  store double 0x3FE6666666666666, double* %4, align 8
  store double* %0, double** %dx, align 8
  %dt = alloca double*, align 8
  %malloccall2 = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr
   ↪  (double, double* null, i32 1) to i32), i32 4))
  %5 = bitcast i8* %malloccall2 to double*
  %6 = getelementptr double, double* %5, i32 0
  store double 5.000000e-01, double* %6, align 8
  %7 = getelementptr double, double* %5, i32 1
  store double 2.000000e-01, double* %7, align 8
  %8 = getelementptr double, double* %5, i32 2
  store double 1.700000e+00, double* %8, align 8
  %9 = getelementptr double, double* %5, i32 3
  store double 5.000000e-01, double* %9, align 8
  store double* %5, double** %dt, align 8
  %res = alloca double*, align 8
  %malloccall3 = tail call i8* @malloc(i32 mul (i32 ptrtoint (double* getelementptr
   ↪  (double, double* null, i32 1) to i32), i32 4))
  %10 = bitcast i8* %malloccall3 to double*
  %11 = getelementptr double, double* %10, i32 0
  store double 0.000000e+00, double* %11, align 8
  %12 = getelementptr double, double* %10, i32 1
  store double 0.000000e+00, double* %12, align 8
  %13 = getelementptr double, double* %10, i32 2
  store double 0.000000e+00, double* %13, align 8
```

```
64    %14 = getelementptr double, double* %10, i32 3
65    store double 0.000000e+00, double* %14, align 8
66    store double* %10, double** %res, align 8
67    br label %while
68
69  while:                                            ; preds = %while_body, %entry
70    %i18 = load i32, i32* %i, align 4
71    %tmp19 = icmp slt i32 %i18, 4
72    br i1 %tmp19, label %while_body, label %merge
73
74  while_body:                                       ; preds = %while
75    %dx4 = load double*, double** %dx, align 8
76    %i5 = load i32, i32* %i, align 4
77    %15 = getelementptr double, double* %dx4, i32 %i5
78    %16 = load double, double* %15, align 8
79    %dt6 = load double*, double** %dt, align 8
80    %i7 = load i32, i32* %i, align 4
81    %17 = getelementptr double, double* %dt6, i32 %i7
82    %18 = load double, double* %17, align 8
83    %tmp = fdiv double %16, %18
84    %base8 = load double, double* %base1, align 8
85    %tmp9 = fmul double %base8, 1.000000e+00
86    %tmp10 = fadd double %tmp, %tmp9
87    %tmp11 = fmul double %tmp10, 1.000000e+03
88    %res12 = load double*, double** %res, align 8
89    %i13 = load i32, i32* %i, align 4
90    %19 = getelementptr double, double* %res12, i32 %i13
91    store double %tmp11, double* %19, align 8
92    %res14 = load double*, double** %res, align 8
93    %i15 = load i32, i32* %i, align 4
94    %20 = getelementptr double, double* %res14, i32 %i15
95    %21 = load double, double* %20, align 8
96    %printf = call i8* (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
      ↪  i8]* @fmt.4, i32 0, i32 0), double %21)
97    %i16 = load i32, i32* %i, align 4
98    %tmp17 = add i32 %i16, 1
99    store i32 %tmp17, i32* %i, align 4
100   br label %while
101
102 merge:                                            ; preds = %while
103   ret void
```

```
}

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

6.2.2 Test 2

A simple test case for unit conversion upon function call and return.

SCIC Code

```
|'{km} = 0.001 '{m}|;

float '{km} func foo(float '{m} x) {
    printf(x); /* 30 (m) */
    float '{m} y = x;
    return y;   /* at return: convert to return unit km */
}

int func main() {
    float '{cm} tmp = 3000.0;
    printf(foo(tmp));  /* 0.03 (km)*/
    return 0;
}
```

LLVM IR

```
; ModuleID = 'scic'
source_filename = "scic"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@fmt.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.4 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.5 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@str = private unnamed_addr constant [12 x i8] c"Hello World\00", align 1

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

```

```
14 declare i32 @printbig(i32)
15
16 define i32 @main() {
17 entry:
18   call void @foo()
19   ret i32 0
20 }
21
22 define void @foo() {
23 entry:
24   %printf = call i8* (i8*, ...) @printf(i8* getelementptr inbounds ([4 xs i8], [4 x
     ↪  i8]* @fmt.5, i32 0, i32 0), i8* getelementptr inbounds ([12 x i8], [12 x i8]*
     ↪  @str, i32 0, i32 0))
25   ret void
26 }
```

6.2.3 Test 3

A test case for unit declaration, unit conversion upon multiple function call and return.

SCIC code

```
1  |'{km} = 0.001 '{m}|;
2  |'{ms} = 1000.0 '{s}|;
3
4  /* return value will be auto-converted */
5  float '{m/s} func foo() {
6      float '{cm} x = 250.0;
7      float '{s} t = 2.5;
8      return x / t;
9  }
10
11 float '{km/s} func bar(float '{cm/ms} a) {
12     float '{cm/ms} b = 31.2 ^ 2;
13     printf(a);
14     printf(b);
15     if (a > b) {
16         return a + b;
17     }
18     else {
19         return b - a;
```

```
20      }
21  }
22
23  int func main() {
24      printf(foo());
25      printf(bar(foo()));
26      return 0;
27  }
```

LLVM IR

```
1   ; ModuleID = 'scic'
2   source_filename = "scic"
3
4   @fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
5   @fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
6   @fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
7   @fmt.3 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
8   @fmt.4 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
9   @fmt.5 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
10  @fmt.6 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
11  @fmt.7 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
12  @fmt.8 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
13
14  declare i8* @printf(i8*, ...)
15
16  declare i32 @printbig(i32)
17
18  define i32 @main() {
19  entry:
20    %foo_result = call double @foo()
21    %printf = call i8* (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
      ↪  i8]* @fmt.1, i32 0, i32 0), double %foo_result)
22    %foo_result1 = call double @foo()
23    %tmp = fmul double %foo_result1, 1.000000e-01
24    %bar_result = call double @bar(double %tmp)
25    %printf2 = call i8* (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
      ↪  i8]* @fmt.1, i32 0, i32 0), double %bar_result)
26    ret i32 0
27  }
```

```
28
29 define double @bar(double %a) {
30 entry:
31   %a1 = alloca double, align 8
32   store double %a, double* %a1, align 8
33   %b = alloca double, align 8
34   %powi32 = call double @llvm.powi.f64(double 3.120000e+01, i32 2)
35   store double %powi32, double* %b, align 8
36   %a2 = load double, double* %a1, align 8
37   %printf = call i8* (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
     ↪  i8]* @fmt.4, i32 0, i32 0), double %a2)
38   %b3 = load double, double* %b, align 8
39   %printf4 = call i8* (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
     ↪  i8]* @fmt.4, i32 0, i32 0), double %b3)
40   %a5 = load double, double* %a1, align 8
41   %b6 = load double, double* %b, align 8
42   %tmp = fcmp ogt double %a5, %b6
43   br i1 %tmp, label %then, label %else
44
45 merge:                                            ; No predecessors!
46   ret double 0.000000e+00
47
48 then:                                             ; preds = %entry
49   %a7 = load double, double* %a1, align 8
50   %b8 = load double, double* %b, align 8
51   %tmp9 = fmul double %b8, 1.000000e+00
52   %tmp10 = fadd double %a7, %tmp9
53   %tmp11 = fmul double %tmp10, 1.000000e-02
54   ret double %tmp11
55
56 else:                                             ; preds = %entry
57   %b12 = load double, double* %b, align 8
58   %a13 = load double, double* %a1, align 8
59   %tmp14 = fmul double %a13, 1.000000e+00
60   %tmp15 = fsub double %b12, %tmp14
61   %tmp16 = fmul double %tmp15, 1.000000e-02
62   ret double %tmp16
63 }
64
65 define double @foo() {
66 entry:
```

```
67    %x = alloca double, align 8
68    store double 2.500000e+02, double* %x, align 8
69    %t = alloca double, align 8
70    store double 2.500000e+00, double* %t, align 8
71    %x1 = load double, double* %x, align 8
72    %t2 = load double, double* %t, align 8
73    %tmp = fdiv double %x1, %t2
74    %tmp3 = fmul double %tmp, 1.000000e-02
75    ret double %tmp3
76  }
77
78  ; Function Attrs: nounwind readnone speculatable willreturn
79  declare double @llvm.powi.f64(double, i32) #0
80
81  attributes #0 = { nounwind readnone speculatable willreturn }
```

6.3 Test suite

The set of 70 test cases are printed in Appendix.

6.4 Automation in testing

We automate our testing in a shell script modified the testall.sh from the mircoC [3]. The script runs all test cases with names test-*.sc and fail-*.sc, compare with .out or .err file respectively and print result to testall.log file. Makefile calls the testall.sh script, so everytime executing make command, the test suits are automatically run.

Chapter 7

Lessons Learned

7.1 From Zhengyuan Dong

PLT project makes realize how trivial previous CS projects I have done are. This was a challenging and large scale project in my final semester. We made the decision about the design of language after many iterations of discussions. During the whole project process, we kept raising up new design ideas and modifying the direction. Design is always the hardest part. With a group of only 3 members, I found that programming alone and group programming both effective. Due to the fact that members are in different time zones, we typically had a short summary meeting in the morning, then did our own work, and repeated the process daily in the final period. Also, I learned the importance of test-driven development to cover all possibilities of what can happen. Some test cases make us discover bugs in the program. For example, we didn't find if-else codegen.ml bug until we tested the program.

Aside from group part, I did spend quite long time figuring out syntax of ocaml, compiler architecture, relationships among each layers. The MicroC codes and ocaml documentation help a lot in the process.

7.2 From Zhengyi Li

I have been using many programming languages and have seen many different syntaxes, despite how complicated and weird they are. However, I have never thought about how those languages achieved their syntaxes features; moreover, I have never viewed such matter from an aspect of design and how it may greatly impact the speed of whole language. This PLT project is challenging and inspiring for me to peek into programming language design, which started from scratch and involved a great amount of teamwork and communications. More importantly, I can understand the purpose and functionalities of some syntax and features based on the design decision. This is one of the most inspiring lesson I have learned, since now I have grasped the common traits and universally applicable characteristics of language syntax and features, which will help me to learn a new language much more quickly. I can also understand how a programming language syntax breaking down to the assembly language and allocating space in the computer for variables.

Throughout the entire process, we took a lot of efforts on designing the Unit system, but the solution is not ultimate since adding many calculation layers to the number would drop the speed of the language, which is a limitation that I have realized and would like to dive into in the future. There might be a better solution, which is why there will always be a new language born.

7.3 From Yucen Sun

During this project, I learned a lot about the way to think from the perspective of a compiler architect. When we were brainstorming for ideas of new language and even when we were writing LRM, I was totally in a language user's angle. It was after several lectures explaining the details of microC pipeline that I gradually form the picture of how to approach the problem. It was a great experience figuring out how to realize the features we want with the progress of learning. However, I do think that if the proposal phase is put to a bit later, we can come up with more challenging ideas with confidence, as well as making many little decisions more wisely in LRM phase.

The debugging and testing for compiler programs are quite interesting. I truly learned how things work fine individually but break subtly on some integrated test cases. My skills of functional programming and debugging the pipeline definitely have been improved.

Also, I would say for this kind of large-scope term-long project, the teammate matching is vital. At the beginning of the semester we did talk with several people before finally decide to form the team. Although our team only has three members and all busy with loads of tasks, we are indeed working towards the same goal and supporting each other during the project.

Chapter 8

Appendix

8.1 scic.ml

```
(* Top-level of the SCIC 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 | Usast | 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");
    ("-u", Arg.Unit (set_action Usast), "Print the USAST");
    ("-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: ./scic.native [-a|-s|-l|-c] [file.mc]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

  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 in
         let usast = Unitcheck.check sast in
    match !action with
```

```
29        Ast     -> ()
30      | Sast    -> print_string (Sast.string_of_sprogram sast)
31      | Usast   -> print_string (Usast.string_of_usprogram usast)
32      | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate usast))
33      | Compile -> let m = Codegen.translate usast in
34          Llvm_analysis.assert_valid_module m;
35          print_string (Llvm.string_of_llmodule m)
```

8.2 scanner.mll

```
1  (* Ocamllex scanner for SCIC *)
2
3  { open Parser }
4
5  let digit = ['0' - '9']
6  let digits = digit+
7
8  rule token = parse
9    [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
10 | "/*"     { comment lexbuf }           (* Comments *)
11 | '('      { LPAREN }
12 | ')'      { RPAREN }
13 | '{'      { LBRACE }
14 | '}'      { RBRACE }
15 | '['      { LBRACK }
16 | ']'      { RBRACK }
17 | '|'      { BAR }
18 | ';'      { SEMI }
19 | ','      { COMMA }
20 | '+'      { PLUS }
21 | '-'      { MINUS }
22 | '*'      { TIMES }
23 | '/'      { DIVIDE }
24 | '^'      { POW }
25 | '='      { ASN }
26 | "=="     { EQ }
27 | "!="     { NEQ }
28 | '<'      { LT }
29 | "<="     { LEQ }
30 | ">"      { GT }
31 | ">="     { GEQ }
32 | "&&"     { AND }
```

```
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "bool"   { BOOL }
| "float"  { FLOAT }
| "void" { VOID }
| "string"  { STRING }
| "int[]"   { INTARR }
| "float[]" { FLOATARR }
| "func"   { FUNC }
| "equa"   { EQUA }
| "true"   { BOOL_LITERAL(true) }
| "false"  { BOOL_LITERAL(false) }
| digits as lxm { INT_LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm { FLOAT_LITERAL(lxm) }
| '\"' ([^'\"']* as lxm ) '\"' {STRING_LITERAL(lxm)}
| '\'' '{' (['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '^' '/' '*']*['a'-'z' 'A'-'Z'])* as
↪  lxm)'}'   { UNIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm { ID(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

8.3 parser.mly

```
%{ open Ast %}

%token SEMI COLASN COMMA LPAREN RPAREN LBRACK RBRACK LBRACE RBRACE BAR PRIME
%token NEG NOT
%token PLUS MINUS TIMES DIVIDE POW
%token EQ NEQ LT GT LEQ GEQ AND OR
%token ASN DASN
%token BOOL INT FLOAT CHAR STRING INTARR FLOATARR VOID
%token FUNC EQUA
```

```
10 %token IF ELSE NOELSE FOR WHILE RETURN
11 %token METER SEC KGRAM AMP CMETER HERTZ GRAM NEWTON
12
13
14 /* literals */
15 %token <string> ID /* identifier for variable and function names */
16 %token <string> UNIT
17 %token <int> INT_LITERAL
18 %token <string> FLOAT_LITERAL
19 %token <string> STRING_LITERAL
20 %token <bool> BOOL_LITERAL
21 %token EOF
22
23 /* precedence */
24 %nonassoc NOELSE
25 %nonassoc ELSE
26
27
28 %right ASN
29 %left OR
30 %left AND
31 %left EQ NEQ
32 %left LT GT LEQ GEQ
33 %left PLUS MINUS
34 %left TIMES DIVIDE
35 %left POW
36 %right NOT NEG
37
38 %nonassoc LPAREN LBRACK LBRACE
39 %nonassoc RPAREN RBRACK RBRACE
40
41
42 %start program
43 %type <Ast.program> program
44
45 %%
46
47 program:
48    decls EOF { $1 }
49
50 decls:
```

```
51    /* nothing */   { {globals=[]; udecls=[]; fdecls=[];} }
52    | decls var_decl {{
53                  globals = $2 :: $1.globals;
54                  udecls = $1.udecls;
55                  fdecls = $1.fdecls;
56                                          }}
57    | decls unit_decl {{
58                  globals = $1.globals;
59                  udecls = $2 :: $1.udecls;
60                  fdecls = $1.fdecls;
61                                          }}
62    | decls func_decl {{
63                  globals = $1.globals;
64                  udecls = $1.udecls;
65                  fdecls = $2 ::$1.fdecls;
66                                          }}
67
68
69  /***************** unit declarations *******************/
70    /* |'{mm} = 0.001 '{m}|; */
71  unit_decl:
72    BAR UNIT ASN FLOAT_LITERAL UNIT BAR SEMI { ($2, $5, $4) }
73
74
75  /*************** variable declarations ********************/
76  var_decl:
77    unit_typ UNIT ID SEMI { ($1, $2, $3) }
78   |typ ID SEMI { ($1, "1", $2) }
79
80  typ:
81    INT     { Int   }
82   | FLOAT  { Float }
83   | STRING { String}
84   | BOOL   { Bool  }
85   | VOID   { Void  }
86   | INTARR { IntArr }
87   | FLOATARR { FloatArr }
88
89  unit_typ:
90    FLOAT    { Float }
91   | FLOATARR { FloatArr }
```

```
92
93
94  /*************** func_decl ******************/
95  func_decl:
96     unit_typ UNIT FUNC ID LPAREN opt_formals RPAREN LBRACE stmt_list RBRACE{{
97        return_type       = $1;
98        return_unit       = $2;
99                  func_identifier   = $4;
100                 func_formals      = $6;
101                 func_stmts        = List.rev $9;
102    }}
103 |  typ FUNC ID LPAREN opt_formals RPAREN LBRACE stmt_list RBRACE{{
104       return_type       = $1;
105       return_unit       = "1";
106                 func_identifier   = $3;
107                 func_formals      = $5;
108                 func_stmts        = List.rev $8;
109    }}
110
111
112 opt_formals:
113    /* nothing */ {[]}
114    | formals_list {List.rev $1 }
115
116
117 formals_list:
118     typ ID { [($1, "1", $2)] }
119    | unit_typ UNIT ID { [($1, $2, $3)] }
120    | formals_list COMMA unit_typ UNIT ID { ($3, $4, $5) :: $1 }
121    | formals_list COMMA typ ID { ($3, "1", $4) :: $1 }
122
123
124
125 stmt_list:
126    { [] }
127    | stmt_list stmt { $2 :: $1 }
128
129 stmt:
130    expr SEMI {Expr $1}
131    | typ ID ASN expr SEMI                  { DAssign($1, "1", $2, $4) }
132    | unit_typ UNIT ID ASN expr SEMI                  { DAssign($1, $2, $3, $5) }
```

```
133     | RETURN expr_opt SEMI                                    {Return $2}
134     | LBRACE stmt_list RBRACE              { Block(List.rev $2) }
135         | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
136           | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7)        }
137           | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt { For($3, $5, $7,
              ↪  $9)   }
138     | WHILE LPAREN expr RPAREN stmt           { While($3, $5)   }
139
140 expr_opt:
141     /* nothing */ { Noexpr }
142           | expr          { $1 }
143
144 expr:
145     INT_LITERAL
        ↪                                                                  {
        ↪  IntLit($1) }
146     | FLOAT_LITERAL                                                      {
        ↪  FloatLit($1) }
147         | STRING_LITERAL                                        { StringLit($1) }
148         | BOOL_LITERAL
            ↪                                                              {
            ↪  BoolLit($1) }
149         | ID
            ↪
            ↪  Id($1) }
150         | expr PLUS expr
            ↪                                                          {
            ↪  Binop($1, Add, $3) }
151         | expr MINUS expr                                            {
            ↪  Binop($1, Sub, $3) }
152         | expr TIMES expr                                            {
            ↪  Binop($1, Mult, $3) }
153         | expr DIVIDE expr                                           {
            ↪  Binop($1, Div, $3) }
154         | expr POW expr
            ↪                                                          {
            ↪  Binop($1, Pow, $3) }
155         | expr EQ expr
            ↪                                                              {
            ↪  Binop($1, Equal, $3) }
```

```
156         | expr NEQ expr
            ↪                                                    {
            ↪  Binop($1, Neq, $3) }
157         | expr LT expr
            ↪                                                        {
            ↪  Binop($1, Less, $3) }
158         | expr LEQ
            ↪  expr                                                      {
            ↪  Binop($1, Leq, $3) }
159         | expr GT expr
            ↪                                                        {
            ↪  Binop($1, Greater, $3) }
160         | expr GEQ expr
            ↪                                                    {
            ↪  Binop($1, Geq, $3) }
161         | expr AND expr
            ↪                                                    {
            ↪  Binop($1, And, $3) }
162         | expr OR expr
            ↪                                                        {
            ↪  Binop($1, Or, $3) }
163         | MINUS expr %prec NEG
            ↪                                                { Unop(Neg, $2) }
164         | NOT expr
            ↪                                                            {
            ↪  Unop(Not, $2) }
165     | expr ASN expr                                                {
        ↪  Assign($1, $3) }
166         | LPAREN expr RPAREN                                         {
            ↪  $2 }
167     | ID LPAREN args RPAREN            { FunctionCall($1, $3) }
168     | LBRACK opt_lst RBRACK                 { Array($2)}
169     | expr LBRACK expr RBRACK           { ArrayAccess($1, $3)}
170
171
172 opt_lst:
173    {[]}
174     | lst { List.rev $1 }
175
176 lst:
177       expr { [$1]}
```

```
178     | lst COMMA expr { $3 :: $1 }
179
180
181 args:
182     /* nothing */ { [] }
183   | args_list  { List.rev $1 }
184
185 args_list:
186     expr                    { [$1] }
187   | args_list COMMA expr { $3 :: $1 }
```

8.4 ast.ml

```
1  (* Abstract Syntax Tree and functions for printing it *)
2
3  type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
4            And | Or | Pow
5
6  type uop = Neg | Not
7
8  type typ = Int | Bool | Float | String | Void | IntArr | FloatArr
9
10 type uni = string
11
12
13 type bind = typ * string
14
15 type ubind = typ * string * string
16
17 (* new unit, existing unit * scale in floatlit *)
18 type unit_decl = string * string * string
19
20 type expr =
21   IntLit of int
22   | FloatLit of string
23   | StringLit of string
24   | BoolLit of bool
25   | Id of string
26   | Binop of expr * op * expr
27   | Unop of uop * expr
28   | Assign of expr * expr
29   | FunctionCall of string * expr list
```

```
30    | Array of expr list
31    | ArrayAccess of expr * expr
32    | Noexpr
33
34  type stmt =
35      Block of stmt list
36    | Expr of expr
37    | Return of expr
38    | If of expr * stmt * stmt
39    | For of expr * expr * expr * stmt
40    | While of expr * stmt
41    | DAssign of typ * string * string * expr
42
43  type func_decl = {
44      return_type : typ;
45      return_unit : uni;
46      func_identifier : string;
47      func_formals : ubind list;
48      func_stmts : stmt list;
49    }
50
51
52  type program = {
53    udecls: unit_decl list;
54    globals: ubind list;
55    fdecls: func_decl list;
56  }
57
58
59
60  (* Pretty-printing functions *)
61
62  let string_of_op = function
63      Add -> "+"
64    | Sub -> "-"
65    | Mult -> "*"
66    | Div -> "/"
67    | Equal -> "=="
68    | Pow -> "^"
69    | Neq -> "!="
70    | Less -> "<"
```

```
71    | Leq -> "<="
72    | Greater -> ">"
73    | Geq -> ">="
74    | And -> "&&"
75    | Or -> "||"
76
77    let string_of_typ = function
78    Int -> "int"
79  | Bool -> "bool"
80  | Float -> "float"
81  | String -> "string"
82  | Void -> "void"
83  | IntArr -> "int[]"
84  | FloatArr -> "float[]"
85
86  let string_of_uop = function
87      Neg -> "-"
88    | Not -> "!"
89
90
91  let rec string_of_expr = function
92    IntLit(l) -> string_of_int l
93    | FloatLit(l) -> l
94    | BoolLit(true) -> "true"
95    | BoolLit(false) -> "false"
96    | StringLit(l) -> l
97    | Id(s) -> s
98    | Binop(e1, o, e2) ->
99        string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
100   | Unop(o, e) -> string_of_uop o ^ string_of_expr e
101   | Assign(v, e) -> string_of_expr v ^ " = " ^ string_of_expr e
102   | FunctionCall(f, el) ->
103       f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
104   | Array(el) -> String.concat ", " (List.map string_of_expr el)
105   | ArrayAccess(s, e) -> string_of_expr s ^ "[" ^ string_of_expr e ^ "]"
106
107   | Noexpr -> ""
108
109 let rec string_of_stmt = function
110     Block(stmts) ->
111       "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
```

```
112   | Expr(expr) -> string_of_expr expr ^ ";\n";
113   | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
114   | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
115   | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
116       string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
117   | For(e1, e2, e3, s) ->
118       "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
119       string_of_expr e3  ^ ") " ^ string_of_stmt s
120   | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
121   | DAssign(t, u, v, e) -> string_of_typ t ^ " " ^ u ^ " " ^ v ^ " = " ^
      ↪  string_of_expr e ^ ";\n"
122
123 let string_of_vdecl (t, u, id) = string_of_typ t ^ " " ^ u ^ " " ^ id ^ ";\n"
124
125 let string_of_udecl (u1, c, u2) = "'{" ^ u1 ^ "} = " ^ c ^ " * '{" ^ u2 ^ "}; \n"
126
127 let get_3_3 t = match t with
128     (_, _, e) -> e;;
129
130 let string_of_fdecl fdecl =
131   string_of_typ fdecl.return_type ^ " " ^ "func " ^
132   fdecl.func_identifier ^ "(" ^ String.concat ", " (List.map get_3_3
      ↪  fdecl.func_formals) ^
133   ")\n{\n" ^
134   (* String.concat "" (List.map string_of_vdecl fdecl.locals) ^ *)
135   String.concat "" (List.map string_of_stmt fdecl.func_stmts) ^
136   "}\n"
137
138 let string_of_program program =
139   String.concat "" (List.map string_of_udecl program.udecls) ^ "\n" ^
140   String.concat "" (List.map string_of_vdecl program.globals) ^ "\n" ^
141   String.concat "\n" (List.map string_of_fdecl program.fdecls)
```

8.5 semant.ml

```
1 (* Semantic checking for the SCIC compiler *)
2
3 open Ast
4 open Sast
5
6 module StringMap = Map.Make(String)
7 module SS = Set.Make(String)
```

```
8
9  (* Semantic checking of the AST. Returns an SAST if successful,
10    throws an exception if something is wrong.
11
12    Check each global variable, then check each function *)
13
14 let check program =
15
16
17   (* SCIC does not have void type, which here only check for duplicate names *)
18   (* Verify a list of bindings has no void types or duplicate names *)
19   let check_ubinds (kind : string) (ubinds : ubind list) =
20     List.iter (function
21     (Void, _, b) -> raise (Failure ("illegal void " ^ kind ^ " " ^ b))
22     | _ -> ()) ubinds;
23     let rec dups = function
24         [] -> ()
25       |        ((_, _, n1) :: (_,_, n2) :: _) when n1 = n2 -> raise (Failure
       ↪  ("duplicate " ^ kind ^ " " ^ n1))
26       | _ :: t -> dups t
27     in dups (List.sort (fun (_,_,a) (_,_,b) -> compare a b) ubinds)
28   in
29
30   (**** Check global variables ****)
31
32   check_ubinds "global" program.globals;
33
34   (**** Check functions ****)
35
36
37   (* Collect function declarations for built-in functions: no bodies *)
38   let built_in_decls =
39     let add_bind map (name, ty) = StringMap.add name {
40       return_type = Void;
41       return_unit = "1";
42       func_identifier = name;
43       func_formals = [(ty,"1","x")];
44       func_stmts = [] } map
45     in List.fold_left add_bind StringMap.empty [ ("print", Int); ("printl", String);
46                                                  ("printb", Bool);
47                                ("printf", Float);
```

```
48                                                  ("printbig", Int) ]
49    in
50
51    (* Add function name to symbol table *)
52    let add_func map fd =
53      let built_in_err = "function " ^ fd.func_identifier ^ " may not be defined"
54      and dup_err = "duplicate function " ^ fd.func_identifier
55      and make_err er = raise (Failure er)
56      and n = fd.func_identifier (* Name of the function *)
57      in match fd with (* No duplicate functions or redefinitions of built-ins *)
58           _ when StringMap.mem n built_in_decls -> make_err built_in_err
59         | _ when StringMap.mem n map -> make_err dup_err
60         | _ ->  StringMap.add n fd map
61    in
62
63    (* Collect all function names into one symbol table *)
64    let function_decls = List.fold_left add_func built_in_decls program.fdecls
65    in
66
67    (* Return a function from our symbol table *)
68    let find_func s =
69      try StringMap.find s function_decls
70      with Not_found -> raise (Failure ("unrecognized function " ^ s))
71    in
72
73    let _ = find_func "main" in (* Ensure "main" is defined *)
74
75    let check_function func =
76      (* Make sure no formals or locals are void or duplicates *)
77      check_ubinds "formal" func.func_formals;
78
79      (* Raise an exception if the given rvalue type cannot be assigned to
80         the given lvalue type *)
81      let check_assign lvaluet rvaluet err =
82         if lvaluet = rvaluet then lvaluet else raise (Failure err)
83      in
84
85
86      (* Build local symbol table of variables for this function *)
87      let symbols = List.fold_left (fun m (ty, _, name) -> StringMap.add name ty m)
88                           StringMap.empty (program.globals @ func.func_formals )
```

```
89      in
90
91      (* Return a variable from our local symbol table *)
92      let type_of_identifier s table =
93        try StringMap.find s table
94        with Not_found -> raise (Failure ("undeclared identifier " ^ s))
95      in
96
97      (* type_to_array converts type to corresponding array type for array handling *)
98      let type_to_array = function
99        | Int -> IntArr
100       | Float -> FloatArr
101       | _ as x -> x
102     in
103
104     (* array_to_type converts array types to their corresponding non-array types *)
105     let array_to_type = function
106       | IntArr -> Int
107       | FloatArr -> Float
108       | _ as x -> x
109     in
110
111     (* Return a semantically-checked expression, i.e., with a type *)
112     let rec expr table = function
113         IntLit  l -> (Int, SIntLit l)
114       | FloatLit l -> (Float, SFloatLit l)
115       | BoolLit l  -> (Bool, SBoolLit l)
116       | StringLit l -> (String, SStringLit l)
117       | Noexpr     -> (Void, SNoexpr)
118       | Id s       -> (type_of_identifier s table, SId s)
119       | Assign(e1, e2) as ex ->
120           let (lt, e1') = match e1 with
121                 Id(s) -> (type_of_identifier s table, SId s)
122               | _ -> expr table e1
123           and (rt, e2') = expr table e2 in
124           let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
125             string_of_typ rt ^ " in " ^ string_of_expr ex
126           in (check_assign lt rt err, SAssign((lt, e1'), (rt, e2')))
127       | Unop(op, e) as ex ->
128           let (t, e') = expr table e in
129           let ty = match op with
```

```
130             Neg when t = Int || t = Float -> t
131           | Not when t = Bool -> Bool
132           | _ -> raise (Failure ("illegal unary operator " ^
133                                  string_of_uop op ^ string_of_typ t ^
134                                  " in " ^ string_of_expr ex))
135         in (ty, SUnop(op, (t, e')))
136       | Binop(e1, op, e2) as e ->
137         let (t1, e1') = expr table e1
138         and (t2, e2') = expr table e2 in
139         (* All binary operators require operands of the same type *)
140         let same = t1 = t2 in
141         (* Determine expression type based on operator and operand types *)
142         let ty = match op with
143                   Add | Sub | Mult | Div  when same && t1 = Int   -> Int
144                 | Add | Sub | Mult | Div | Pow when same && t1 = Float -> Float
145                 | Pow when (t1 = Float && t2 = Int) -> Float
146                 | Equal | Neq            when same               -> Bool
147                 | Less | Leq | Greater | Geq
148                           when same && (t1 = Int || t1 = Float) -> Bool
149                 | And | Or when same && t1 = Bool -> Bool
150                 | _ -> raise (Failure ("illegal binary operator " ^
151                                        string_of_typ t1 ^ " " ^ string_of_op op ^ " "
                                       ↪  ^
152                                        string_of_typ t2 ^ " in " ^ string_of_expr e))
153         in (ty, SBinop((t1, e1'), op, (t2, e2')))
154       | FunctionCall(fname, args) as call ->
155         let fd = find_func fname in
156         let param_length = List.length fd.func_formals in
157         if List.length args != param_length then
158           raise (Failure ("expecting " ^ string_of_int param_length ^
159                           " arguments in " ^ string_of_expr call))
160         else let check_call (ft, _,_) e =
161           let (et, e') = expr table e in
162           let err = "illegal argument " ^ string_of_typ et ^
163             " found, " ^ string_of_typ ft ^ " in " ^ string_of_expr e ^ " expected"
164           in (check_assign ft et err, e')
165         in
166         let args' = List.map2 check_call fd.func_formals args
167         in (fd.return_type, SFunctionCall(fname, args'))
168
169       | Array(el) ->
```

```
170         let rec helper typ out  = function
171           | [] -> (type_to_array typ, SArray(List.rev out))
172           | a :: tl ->
173             let (t, e) = expr table a in
174             if t = typ then helper t ((t, e) :: out) tl
175             else raise (Failure ("multiple types in an array not allowed"))
176         in (match el with
177             | a :: tl ->  let (t,e) = expr table a in helper t [(t, e)] tl
178             | [] -> raise (Failure ("empty array init not allowed")))
179
180       | ArrayAccess(e1, e2) ->
181         let (t1, e1') = expr table e1 in
182         let (t2, e2') = expr table e2 in
183         if (t1 = IntArr || t1 = FloatArr) && t2 = Int
184           then (array_to_type t1, SArrayAccess((t1, e1'), (t2, e2')))
185         else raise (Failure ("invalid type for array access"))
186
187     in
188
189     let check_bool_expr table e =
190       let (t', e') = expr table e
191       and err = "expected Boolean expression in " ^ string_of_expr e
192       in if t' != Bool then raise (Failure err) else (t', e')
193     in
194
195     (* Return a semantically-checked statement i.e. containing sexprs *)
196     let rec check_stmt table = function
197         Expr e -> (table, SExpr (expr table e))
198       | DAssign(lt, unt, var, e) as ex ->
199         let (rt, e') = expr table e in
200         let err = "illegal declare and assignment " ^ string_of_typ lt ^ " = " ^
201             string_of_typ rt ^ " in " ^ string_of_stmt ex in
202         let lt2 = check_assign lt rt err in
203         let new_table = StringMap.add var lt2 table in
204         (new_table, SDAssign(lt2, unt, var, (rt, e')))
205       | If(p, b1, b2) -> let (table_b1, st_b1) = check_stmt table b1 in
206                          let (table_b2, st_b2) = check_stmt table_b1 b2 in
207                          (table_b2, SIf(check_bool_expr table p, st_b1, st_b2))
208       | For(e1, e2, e3, st) -> let (new_table, new_st) = check_stmt table st in
209                                (new_table, SFor(expr new_table e1, check_bool_expr
                                 ↪  new_table e2, expr new_table e3, new_st))
```

```
210       | While(p, st) -> let (new_table, new_st) = check_stmt table st in
211                 (new_table, SWhile(check_bool_expr new_table p, new_st))
212       | Return e -> let (t, e') = expr table e in
213         if t = func.return_type then (table, SReturn (t, e'))
214         else raise (
215                 Failure ("return gives " ^ string_of_typ t ^ " expected " ^
216                       string_of_typ func.return_type ^ " in " ^ string_of_expr e))
217
218             (* A block is correct if each statement is correct and nothing
219                follows any Return statement.  Nested blocks are flattened. *)
220       | Block sl ->
221           let rec check_stmt_list table = function
222               [Return _ as s] -> let (new_table, st) = check_stmt table s in
                ↪  (new_table, [st])
223             | Return _ :: _   -> raise (Failure "nothing may follow a return")
224             | Block sl :: ss  -> check_stmt_list table (sl @ ss) (* Flatten blocks *)
225             | s :: ss         -> let (one_table, one_s) = check_stmt table s in
226                                  let (list_table, list_s) = check_stmt_list one_table
                                    ↪  ss in
227                                    (list_table, one_s :: list_s)
228             | []              -> (table, [])
229           in let (new_table, listS) = check_stmt_list table sl in (new_table,
            ↪  SBlock(listS))
230
231     in (* body of check_function *)
232     { sreturn_type = func.return_type;
233       sreturn_unit = func.return_unit;
234       sfunc_identifier = func.func_identifier;
235       sfunc_formals = func.func_formals;
236       sfunc_stmts = match check_stmt symbols (Block func.func_stmts) with
237         (_, SBlock(sl)) -> sl
238         | _ -> raise (Failure ("internal error: block didn't become a block?"))
239       }
240   in (program.udecls, program.globals, List.map check_function program.fdecls)
```

8.6 sast.ml

```
1  (* Semantically-checked Abstract Syntax Tree and functions for printing it *)
2
3  open Ast
4
5  type sunt =
```

```
6     SUnit of string
7
8
9   type sexpr = typ * sx
10  and sx =
11    SIntLit of int
12    | SFloatLit of string
13    | SBoolLit of bool
14    | SStringLit of string
15    | SId of string
16    | SBinop of sexpr * op * sexpr
17    | SUnop of uop * sexpr
18    | SAssign of sexpr * sexpr
19    | SFunctionCall of string * sexpr list
20    | SArray of sexpr list
21    | SArrayAccess of sexpr * sexpr
22    | SNoexpr
23
24  type sstmt =
25      SBlock of sstmt list
26    | SExpr of sexpr
27    | SReturn of sexpr
28    | SIf of sexpr * sstmt * sstmt
29    | SFor of sexpr * sexpr * sexpr * sstmt
30    | SWhile of sexpr * sstmt
31    | SDAssign of typ * string * string * sexpr
32
33  type sfunc_decl = {
34      sreturn_type : typ;
35      sreturn_unit : uni;
36      sfunc_identifier : string;
37      sfunc_formals : ubind list;
38      sfunc_stmts: sstmt list;
39    }
40
41
42  type sprogram = unit_decl list * ubind list * sfunc_decl list
43
44  (* Pretty-printing functions *)
45
46  let rec string_of_sexpr (t, e) =
```

```
47    "(" ^ string_of_typ t ^ " : " ^ (match e with
48      SIntLit(l) -> string_of_int l
49    | SBoolLit(true) -> "true"
50    | SBoolLit(false) -> "false"
51    | SFloatLit(l) -> l
52    | SStringLit(l) -> l
53    | SId(s) -> s
54    | SBinop(e1, o, e2) ->
55        string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
56    | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
57    | SAssign(v, e) -> string_of_sexpr v ^ " = " ^ string_of_sexpr e
58    | SFunctionCall(f, el) ->
59        f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
60    | SArray(el) -> String.concat ", " (List.map string_of_sexpr el)
61    | SArrayAccess(e1, e2) -> string_of_sexpr e1 ^ "[" ^ string_of_sexpr e2 ^ "]"
62
63    | SNoexpr -> ""
64                                    ) ^ ")"
65
66  let rec string_of_sstmt = function
67      SBlock(stmts) ->
68        "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
69    | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
70    | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
71    | SIf(e, s, SBlock([])) ->
72        "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
73    | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
74        string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
75    | SFor(e1, e2, e3, s) ->
76        "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ " ; " ^
77        string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
78    | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s
79    | SDAssign(t, u, v, e) -> string_of_typ t ^ " " ^ u ^ " " ^ v ^ " = " ^
      ↪  string_of_sexpr e ^ ";\n"
80
81
82  let get_3_3 t = match t with
83  (_, _, e) -> e;;
84
85  let string_of_sfdecl fdecl =
86    string_of_typ fdecl.sreturn_type ^ " " ^ "func " ^
```

```
87    fdecl.sfunc_identifier ^ "(" ^ String.concat ", " (List.map get_3_3
      ↪  fdecl.sfunc_formals) ^
88    ")\n{\n" ^
89    String.concat "" (List.map string_of_sstmt fdecl.sfunc_stmts) ^
90    "}\n"
91
92  let string_of_sprogram (udecls, vars, funcs) =
93    String.concat "" (List.map string_of_udecl udecls) ^ "\n" ^
94    String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
95    String.concat "\n" (List.map string_of_sfdecl funcs)
```

8.7 unicheck.ml

```
1   (* Unit Semantic checking for the SCIC compiler *)
2
3   open Ast
4   open Sast
5
6   module StringMap = Map.Make(String)
7   module SS = Set.Make(String);;
8
9
10
11  let check (udecls, globals, functions) =
12    (* base unit set - static *)
13    let base_units =
14      List.fold_right SS.add ["m"; "s"; "kg"; "A"; "K"; "mol"; "cd"; "1";] SS.empty
15    in
16
17    (* unit maping key: non-base unit, value: (base unit, scale)*)
18    let units =
19      StringMap.add "cm" ("m", 100.0) StringMap.empty
20    in
21
22
23    let nonbase_unit_check u table =
24      match StringMap.find_opt u table with
25        Some (bu, l) -> ()
26      | None ->  raise (Failure ("units cannot be found in table " ^ u))
27    in
28
29
```

```
30  let unit_check u set table =
31    match SS.find_opt u set with
32      Some bu -> ()
33    | None -> nonbase_unit_check u table
34  in
35
36  let base_unit_check u set =
37    match SS.find_opt u set with
38    Some bu -> ()
39  | None -> raise (Failure ("units cannot be found in set " ^ u))
40  in
41
42  let nonbase_unit_reverse_check u table =
43    match StringMap.find_opt u table with
44      None -> ()
45    | Some (bu, l) ->  raise (Failure ("units already in table " ^ u))
46  in
47
48  (* raise failure if the unit already exists in table/set*)
49  let unit_reverse_check u set table =
50    match SS.find_opt u set with
51     Some bu -> raise(Failure ("unit already exists in set " ^ u))
52    | None -> nonbase_unit_reverse_check u table
53  in
54
55  let check_udecls (kind : string) (unit_decls : unit_decl list) =
56    List.iter (function (u1, u2, _) -> ignore(unit_check u2 base_units units);
57                                       ignore(unit_reverse_check u1 base_units
                                           ↪  units);
58    ) unit_decls;
59
60    (* check duplication in new units *)
61    let rec dups = function
62        [] -> ()
63      |        ((u1, _, _) :: (u1',_, _) :: _) when u1 = u1' -> raise (Failure
        ↪  ("duplicate unit definition " ^ kind ^ " " ^ u1))
64      | _ :: t -> dups t
65    in dups (List.sort (fun (a,_,_) (b,_,_) -> compare a b) unit_decls)
66  in
67
68  let add_unit table (u1, u2, c)  =
```

```
69      match SS.find_opt u2 base_units with
70        Some bu -> StringMap.add u1 (u2, float_of_string c) table
71      | None -> match StringMap.find_opt u2 table with
72                  Some (bu, c2) -> StringMap.add u1 (bu, c2 *. (float_of_string c)) table
73                | None -> raise (Failure ("The reference unit not existed " ^ u2))
74    in
75
76    let add_unit_decls (unit_decls: unit_decl list) table =
77      ignore(check_udecls "new unit" unit_decls);
78      List.fold_left add_unit table unit_decls
79    in
80
81    let units = add_unit_decls udecls units in
82
83    let increment_count counter u =
84      match StringMap.find_opt u counter with
85      Some i -> let i = i+1 in StringMap.add u i counter
86      | None -> StringMap.add u 1 counter
87    in
88
89    let decompose_unit u set table =
90      let l = Str.split (Str.regexp "/") u in
91      let (ln, ld) = match l with
92                [e1] ->  (Str.split (Str.regexp "*") e1, [])
93              | [e1; e2] -> (Str.split (Str.regexp "*") e1, Str.split( Str.regexp "*")
                ↪  e2)
94              | _ -> ([], [])
95      in
96      List.iter (fun s -> unit_check s set table) ln;
97      List.iter (fun s -> unit_check s set table) ld;
98      let sn = List.fold_left increment_count StringMap.empty ln
99        and sd = List.fold_left increment_count StringMap.empty ld
100     in (sn, sd)
101   in
102
103   let derived_unit_check u set table =
104     let l = Str.split (Str.regexp "/") u in
105     let (ln, ld) = match l with
106               [e1] ->  (Str.split (Str.regexp "*") e1, [])
107             | [e1; e2] -> (Str.split (Str.regexp "*") e1, Str.split( Str.regexp "*")
                ↪  e2)
```

```
108             | _ -> ([], [])
109       in
110       List.iter (fun s -> unit_check s set table) ln;
111       List.iter (fun s -> unit_check s set table) ld;
112     in
113
114     let all_unit_check u set table =
115       match SS.find_opt u set with
116       Some bu -> ()
117     | None -> match StringMap.find_opt u table with
118               Some (bu, c) -> ()
119             | None -> derived_unit_check u set table
120     in
121
122     (* check global variable unit exists*)
123     let var_unit_check (ubinds : ubind list) =
124       List.iter (function (_, u, _) ->ignore(all_unit_check u base_units units)
125                   ) ubinds
126     in
127     var_unit_check globals;
128
129
130
131     let rec resemble lst =
132         match lst with
133       |  [] -> []
134       | (t, u, n)::tl -> (t, n)::(resemble tl)
135     in
136
137
138     let built_in_decls =
139       let add_bind map (name, ty) = StringMap.add name {
140         sreturn_type = Void;
141         sreturn_unit = "1";
142         sfunc_identifier = name;
143         sfunc_formals = [(ty,"1","x")];
144         sfunc_stmts = [] } map
145       in List.fold_left add_bind StringMap.empty [ ("print", Int); ("printl", String);
146                                    ("printb", Bool);
147                                    ("printf", Float);
148                                    ("printbig", Int) ]
```

```
149   in
150
151
152   (***** build function id lookup table *****)
153   (* Add function name to symbol table *)
154    let add_func map fd =
155       let n = fd.sfunc_identifier (* Name of the function *)
156       in StringMap.add n fd map
157     in
158     (* Collect all function names into one symbol table *)
159     let function_decls = List.fold_left add_func built_in_decls functions
160     in
161
162     (* Return a function from our symbol table *)
163     let find_func s =
164       try StringMap.find s function_decls
165       with Not_found -> raise (Failure ("unrecognized function " ^ s))
166     in
167
168   (* func is current function scope *)
169   let check_function func =
170
171     var_unit_check func.sfunc_formals;
172
173     (* if right unit is "1", does not multiply anything *)
174     let check_right_unit runit =
175       if runit = "1" then true
176       else false
177     in
178
179     let lookup_base u =
180       match SS.find_opt u base_units with
181       Some bu -> bu
182       | None -> match StringMap.find_opt u units with
183                 Some (bu, c) -> bu
184               | None -> raise (Failure ("Cannot find conversion rule for unit " ^ u))
185     in
186
187     let lookup_base_scale u =
188       match SS.find_opt u base_units with
189       Some bu -> 1.0
```

```
190       | None -> match StringMap.find_opt u units with
191                Some (bu, c) -> c
192              | None -> raise (Failure ("Cannot find conversion rule for unit " ^ u))
193     in
194
195     let reduce_to_base umap =
196       let buset = StringMap.fold (fun u i buset -> StringMap.add (lookup_base u) i
          ↪  buset) umap StringMap.empty
197       in
198       let scale = StringMap.fold (fun u i scale -> scale *. (lookup_base_scale u) **
          ↪  float_of_int i) umap 1.0
199       in
200       (buset, scale)
201     in
202
203     let derived_get_scale lunit runit =
204       let (sn, sd) = decompose_unit lunit base_units units and (sn', sd') =
          ↪  decompose_unit runit base_units units
205       in
206       let (snb, cn) = reduce_to_base sn and (snb',cn') = reduce_to_base sn' and
          ↪  (sdb,cd) = reduce_to_base sd and (sdb',cd') = reduce_to_base sd' in
207       if (StringMap.equal (fun a b -> a = b) snb snb'
208         && StringMap.equal (fun a b -> a = b) sdb sdb')
209         then (cn /. cd )/. (cn' /. cd')
210       else raise( Failure("No conversion rules between unit " ^ lunit ^ " and " ^
          ↪  runit))
211     in
212
213     (* get conversion rate between two units *)
214     let get_scale lunit runit map =
215         if lunit = runit then 1.0
216         else try let (u, r) = StringMap.find lunit map in
217                 if u = runit then r
218                 else let (u', r') = StringMap.find runit map in
219                       if u' = u then r /. r'
220                       else  raise (Failure (lunit ^ " and " ^ runit ^ " is not defined
                          ↪  in the conversion rule"))
221               with Not_found -> derived_get_scale lunit runit;
222                 try let (u, r) = StringMap.find runit map in
223                   if u = lunit then 1.0 /. r
224                   else let (u', r') = StringMap.find lunit map in
```

```
225                         if u' = u then r' /. r
226                         else raise (Failure (lunit ^ " and " ^ runit ^ " is not
                            ↪  defined in the conversion rule"))
227                 with Not_found -> derived_get_scale lunit runit;
228         in
229
230       let scale_expr scaler e t =
231         let e' = (t, SBinop((t, e), Mult, (Float, SFloatLit (Printf.sprintf "%.5f"
          ↪  scaler)))) in e'
232       in
233
234       (* Build local symbol table of variables for this function. key: variable name,
          ↪  value: unit.*)
235       let symbols = List.fold_left (fun m (_, unt, name) -> StringMap.add name unt m)
236         StringMap.empty (globals @ func.sfunc_formals )
237       in
238
239       (* Return a variable from our local symbol table *)
240       let unit_of_identifier s table =
241         try StringMap.find s table
242         with Not_found -> raise (Failure ("undeclared identifier " ^ s))
243       in
244
245
246     let rec expr table (t, e) = match e with
247       SIntLit  l   -> ("1", (t, SIntLit l))
248     | SFloatLit l  -> ("1", (t, SFloatLit l))
249     | SBoolLit l   -> ("1", (t, SBoolLit l))
250     | SStringLit l -> ("1", (t, SStringLit l))
251     | SNoexpr      -> ("1", (t, SNoexpr))
252     | SId s        -> (unit_of_identifier s table, (t, SId s))
253     | SAssign(e1, e2) as ex ->
254         let (lu, (t1, e1')) = expr table e1
255           and (ru, (t2, e2')) = expr table e2 in
256         if check_right_unit ru then (lu, (t, SAssign((t1, e1'), (t2, e2'))))
257         else let scale = get_scale lu ru units in
258         let e2_scale = scale_expr scale e2' t2 in
259         (lu, (t, SAssign((t1, e1'), e2_scale)))
260
261     | SFunctionCall(fname, args) as call ->
262       let fd = find_func fname in
```

```
263     (* check each of the args, to see if it can scale *)
264     let check_args_unit (_,fu,_) e =
265       let (eu, (t, e')) = expr table e in
266         if StringMap.mem fname built_in_decls  || check_right_unit eu
267           then (t, e')
268         else let scale = get_scale fu eu units in
269         let e_scale = scale_expr scale e' t in
270             e_scale
271       in
272     let args' = List.map2 check_args_unit fd.sfunc_formals args
273     in (fd.sreturn_unit, (fd.sreturn_type, SFunctionCall(fname, args')))
274   | SUnop(op, e) as ex ->
275       let (eu, e') = expr table e in
276      (eu, (t, SUnop(op, e')))
277   | SBinop(e1, op, e2) as e ->
278     let (eu1, (t1, e1')) = expr table e1 in
279     let (eu2, (t2, e2')) = expr table e2 in
280     let same = t1 = t2 in
281     let check_binop_unit op e1 e2 = match op with
282       | Add | Sub | Equal | Neq | Less | Leq | Greater | Geq | And | Or ->
283         let (eu1, (t1, e1')) = expr table e1 in
284         let (eu2, (t2, e2')) = expr table e2 in
285         if check_right_unit eu1 || check_right_unit eu2 then
286           e2
287         else
288           let scale = get_scale eu1 eu2 units in
289           let e2_scale = scale_expr scale e2' t2 in
290           e2_scale
291       | _ -> e2 in
292     let muti op eu1 eu2 = match op with
293       | Mult -> if check_right_unit eu1 && check_right_unit eu2 then eu1
294                 else if check_right_unit eu1 then eu2
295                 else if check_right_unit eu2 then eu1
296                 else eu1 ^ "*" ^ eu2
297       | Div -> if check_right_unit eu1 && check_right_unit eu2 then eu1
298                 else if check_right_unit eu1 then eu2
299                 else if check_right_unit eu2 then eu1
300                 else eu1 ^  "/"  ^ eu2
301       | Add | Sub -> if check_right_unit eu1 then eu2
302                      else eu1
303       | _ -> eu1
```

```
304         in
305       let e2_scale = check_binop_unit op e1 e2 in
306       let eu_change = muti op eu1 eu2
307       in (eu_change, (t1, SBinop(e1, op, e2_scale)))
308     | SArray(el) ->
309       ("1", (t, SArray(el)))
310     | SArrayAccess(e1, e2) ->
311       let (u, e1') = expr table e1 in
312       (u, (t, SArrayAccess(e1', e2)))
313 in
314
315   let rec check_stmt table = function
316    SExpr e -> let (u, e') = expr table e in (table, SExpr(e'))
317
318   | SDAssign (lt, unt, var, e) ->
319       (* get e's unit in recursion*)
320       let (ru, (t, e')) = expr table e in
321       let new_table = StringMap.add var unt table in
322       (* here all the expr must be (unit, expr) according to usat *)
323       if check_right_unit ru then
324         (new_table, SDAssign(lt, unt, var, e))
325       else
326         let scale = get_scale unt ru units in
327         let e_scale = scale_expr scale e' t in
328           (new_table, SDAssign(lt, unt, var, e_scale))
329     | SIf(p, b1, b2) ->
330       let (table_b1, st_b1) = check_stmt table b1 in
331       let (table_b2, st_b2) = check_stmt table_b1 b2 in
332       (table_b2, SIf(p, st_b1, st_b2))
333     | SFor(e1, e2, e3, st) ->
334       let (new_table, new_st) = check_stmt table st in
335       let (_, e1') = expr new_table e1 in
336       let (_, e3') = expr new_table e3 in
337       (new_table, SFor(e1', e2, e3', new_st))
338     | SWhile(p, st) ->
339       let (new_table, new_st) = check_stmt table st in
340       (new_table, SWhile(p, new_st))
341     | SReturn e -> let (eu, (t, e')) = expr table e in
342       if check_right_unit eu then  (table, SReturn(t, e'))
343       else let scale = get_scale func.sreturn_unit eu units in
344       let e_scale = scale_expr scale e' t in
```

```
345           (table, SReturn (e_scale))
346   | SBlock sl ->
347     let rec check_stmt_list table = function
348         [SReturn _ as s] -> let (new_table, st) = check_stmt table s in (new_table,
          ↪  [st])
349       | SReturn _ :: _   -> raise (Failure "nothing may follow a return")
350       | SBlock sl :: ss  -> check_stmt_list table (sl @ ss) (* Flatten blocks *)
351       | s :: ss          -> let (one_table, one_s) = check_stmt table s in
352                               let (list_table, list_s) = check_stmt_list one_table ss in
353                                 (list_table, one_s :: list_s)
354       | []               -> (table, [])
355     in let (new_table, listS) = check_stmt_list table sl in (new_table, SBlock(listS))
356
357
358    in
359     { sreturn_type = func.sreturn_type;
360     sreturn_unit = func.sreturn_unit;
361     sfunc_identifier = func.sfunc_identifier;
362     sfunc_formals = func.sfunc_formals;
363     sfunc_stmts = match check_stmt symbols (SBlock func.sfunc_stmts) with
364             (_, SBlock(sl)) -> sl
365       | _ -> raise (Failure ("internal error: block didn't become a block?"))
366     }
367
368 in (resemble globals, List.map check_function functions)
```

8.8 codegen.ml

```
1  (* Code generation: translate takes a semantically checked AST and produces LLVM IR *)
2  module C = Char
3  module L = Llvm
4  module A = Ast
5  open Sast
6
7  module StringMap = Map.Make(String)
8
9  (* translate : Sast.program -> Llvm.module *)
10 let translate (globals, functions) =
11   let context    = L.global_context () in
12
13   (* Create the LLVM compilation module into which
14      we will generate code *)
```

```
15  let the_module = L.create_module context "scic" in
16
17  (* Get types from the context *)
18  let i32_t      = L.i32_type    context
19  and i8_t       = L.i8_type     context
20  and i1_t       = L.i1_type     context
21  and float_t    = L.double_type context
22  and void_t     = L.void_type   context
23  and string_t   = L.pointer_type (L.i8_type context)
24  in
25
26
27  (* Return the LLVM type for a SCIC type *)
28  let rec ltype_of_typ = function
29      A.Int   -> i32_t
30    | A.Bool  -> i1_t
31    | A.Float -> float_t
32    | A.Void  -> void_t
33    | A.String -> string_t
34    | A.IntArr -> L.pointer_type (ltype_of_typ A.Int)
35    | A.FloatArr -> L.pointer_type (ltype_of_typ A.Float)
36  in
37
38  (* Create a map of global variables after creating each *)
39  let global_vars : L.llvalue StringMap.t =
40    let global_var m (t, n) =
41      let init = match t with
42          A.Float -> L.const_float (ltype_of_typ t) 0.0
43        | _ -> L.const_int (ltype_of_typ t) 0
44      in StringMap.add n (L.define_global n init the_module) m in
45    List.fold_left global_var StringMap.empty globals in
46
47  let printf_t : L.lltype =
48      L.var_arg_function_type string_t [| L.pointer_type i8_t |] in
49  let printf_func : L.llvalue =
50      L.declare_function "printf" printf_t the_module in
51
52  let printbig_t : L.lltype =
53      L.function_type i32_t [| i32_t |] in
54  let printbig_func : L.llvalue =
55      L.declare_function "printbig" printbig_t the_module in
```

```
56
57   (* Define each function (arguments and return type) so we can
58      call it even before we've created its body *)
59   let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
60     let function_decl m fdecl =
61       let name = fdecl.sfunc_identifier
62       and formal_types =
63         Array.of_list (List.map (fun (t,_,_) -> ltype_of_typ t) fdecl.sfunc_formals)
64       in let ftype = L.function_type (ltype_of_typ fdecl.sreturn_type) formal_types in
65       StringMap.add name (L.define_function name ftype the_module, fdecl) m in
66     List.fold_left function_decl StringMap.empty functions in
67
68   (* Fill in the body of the given function *)
69   let build_function_body fdecl =
70     let (the_function, _) = StringMap.find fdecl.sfunc_identifier function_decls in
71     let builder = L.builder_at_end context (L.entry_block the_function) in
72
73     let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder
74     and float_format_str = L.build_global_stringptr "%g\n" "fmt" builder
75     and str_format_str = L.build_global_stringptr "%s\n" "fmt" builder in
76
77     (* Construct the function's "locals": formal arguments and locally
78        declared variables.  Allocate each on the stack, initialize their
79        value, if appropriate, and remember their values in the "locals" map *)
80         let local_vars =
81           let add_formal m (t,u,n) p =
82             L.set_value_name n p;
83           let local = L.build_alloca (ltype_of_typ t) n builder in
84                 ignore (L.build_store p local builder);
85           StringMap.add n local m
86           in
87
88           List.fold_left2 add_formal StringMap.empty fdecl.sfunc_formals
89               (Array.to_list (L.params the_function))
90         in
91
92     (* Return the value for a variable or formal argument.
93        Check local names first, then global names *)
94     let lookup n table = try StringMap.find n table
95                   with Not_found -> StringMap.find n global_vars
96     in
```

```
97
98      (* Construct code for an expression; return its value *)
99      let rec expr builder table ((_, e) : sexpr) = match e with
100               SIntLit i  -> L.const_int i32_t i
101       | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
102       | SFloatLit l -> L.const_float_of_string float_t l
103       | SStringLit s -> L.build_global_stringptr s "str" builder
104       | SNoexpr     -> L.const_int i32_t 0
105       | SId s       -> L.build_load (lookup s table) s builder
106       | SAssign((_, SArrayAccess(e1, i)), e2) ->
107             let e2' = expr builder table e2 in
108             let id = expr builder table e1 in
109             let idx = expr builder table i in
110             let ptr = L.build_gep id [| idx |] "" builder in
111             ignore(L.build_store e2' ptr builder); e2'
112       | SAssign (e1, e2) -> let e' = expr builder table e2 in
113                             let s = match (let (_, e) = e1  in e) with
114                                           SId(s)  -> s
115                                         | _ -> raise (Failure "invalid assignment
                                          ↪  id")
116                                in
117                            ignore(L.build_store e' (lookup s table) builder); e'
118       | SBinop (e1, op, ((A.Float, _) as e2)) ->
119         let e1' = expr builder table e1
120         and e2' = expr builder table e2 in
121           (match op with
122             A.Add     -> L.build_fadd e1' e2' "tmp" builder
123           | A.Sub     -> L.build_fsub e1' e2' "tmp" builder
124           | A.Mult    -> L.build_fmul e1' e2' "tmp" builder
125           | A.Div     -> L.build_fdiv e1' e2' "tmp" builder
126           | A.Pow     ->
127             let pow32 = L.declare_function "llvm.pow.f64" (L.function_type float_t
              ↪  [|float_t;float_t;|]) the_module in
128             L.build_call pow32 [| e1'; e2' |] "pow32" builder
129           | A.Equal   -> L.build_fcmp L.Fcmp.Oeq e1' e2' "tmp" builder
130           | A.Neq     -> L.build_fcmp L.Fcmp.One e1' e2' "tmp" builder
131           | A.Less    -> L.build_fcmp L.Fcmp.Olt e1' e2' "tmp" builder
132           | A.Leq     -> L.build_fcmp L.Fcmp.Ole e1' e2' "tmp" builder
133           | A.Greater -> L.build_fcmp L.Fcmp.Ogt e1' e2' "tmp" builder
134           | A.Geq     -> L.build_fcmp L.Fcmp.Oge e1' e2' "tmp" builder
135           | A.And | A.Or ->
```

```
136             raise (Failure "internal error: semant should have rejected and/or on
                ↪ float")
137           )
138       | SBinop (e1, op, e2) ->
139         let e1' = expr builder table e1
140         and e2' = expr builder table e2 in
141         (match op with
142           A.Add     -> L.build_add e1' e2' "tmp" builder
143         | A.Sub     -> L.build_sub e1' e2' "tmp" builder
144         | A.Mult    -> L.build_mul e1' e2' "tmp" builder
145         | A.Div     -> L.build_sdiv e1' e2' "tmp" builder
146         | A.Pow     ->
147           let powi32 = L.declare_function "llvm.powi.f64" (L.function_type float_t
            ↪ [|float_t;i32_t;|]) the_module in
148         L.build_call powi32 [| e1'; e2' |] "powi32" builder
149         | A.And     -> L.build_and e1' e2' "tmp" builder
150         | A.Or      -> L.build_or e1' e2' "tmp" builder
151         | A.Equal   -> L.build_icmp L.Icmp.Eq e1' e2' "tmp" builder
152         | A.Neq     -> L.build_icmp L.Icmp.Ne e1' e2' "tmp" builder
153         | A.Less    -> L.build_icmp L.Icmp.Slt e1' e2' "tmp" builder
154         | A.Leq     -> L.build_icmp L.Icmp.Sle e1' e2' "tmp" builder
155         | A.Greater -> L.build_icmp L.Icmp.Sgt e1' e2' "tmp" builder
156         | A.Geq     -> L.build_icmp L.Icmp.Sge e1' e2' "tmp" builder
157         )
158       | SUnop(op, ((t, _) as e)) ->
159         let e' = expr builder table e in
160                 (match op with
161                 A.Neg when t = A.Float -> L.build_fneg
162                 | A.Neg                  -> L.build_neg
163           | A.Not                  -> L.build_not) e' "tmp" builder
164       | SFunctionCall ("print", [e])  |SFunctionCall ("printb", [e]) ->
165           L.build_call printf_func [| int_format_str ; (expr builder table e) |]
166             "printf" builder
167       | SFunctionCall("printl", [e]) -> L.build_call printf_func [| str_format_str ;
          ↪ (expr builder table e) |] "printf" builder
168       | SFunctionCall ("printbig", [e]) ->
169                 L.build_call printbig_func [| (expr builder table e) |] "printbig"
                  ↪ builder
170       | SFunctionCall ("printf", [e]) ->
171                 L.build_call printf_func [| float_format_str ; (expr builder table e)
                  ↪ |]
```

```
172                 "printf" builder
173       | SFunctionCall (f, args) ->
174         let (fdef, fdecl) = StringMap.find f function_decls in
175         let llargs = List.rev (List.map (expr builder table) (List.rev args)) in
176         let result = (match fdecl.sreturn_type with
177                               A.Void -> ""
178                             | _ -> f ^ "_result") in
179               L.build_call fdef (Array.of_list llargs) result builder
180
181       | SArray (el) ->
182           let ls = List.map (fun e -> expr builder table e) el in
183           let typ = L.type_of (List.hd ls) in
184           let n = List.length ls in
185           let ptr = L.build_array_malloc typ (L.const_int i32_t n) "" builder  in
186           ignore (List.fold_left (fun i elem ->
187                                     let idx = L.const_int i32_t i in
188                                     let eptr = L.build_gep ptr [|idx|] "" builder in
189                                     let cptr = L.build_pointercast eptr
190                                         (L.pointer_type (L.type_of elem)) "" builder
                                          ↪  in
191                                     let _ = (L.build_store elem cptr builder) in i +
                                      ↪  1) 0 ls); ptr
192
193       | SArrayAccess(e1, e2) ->
194           let id = expr builder table e1 in
195           let i = expr builder table e2 in
196           let ptr =  L.build_load (L.build_gep id [| i |] "" builder) "" builder in
            ↪  ptr
197
198       in
199     (* LLVM insists each basic block end with exactly one "terminator"
200        instruction that transfers control.  This function runs "instr builder"
201        if the current block does not already have a terminator.  Used,
202        e.g., to handle the "fall off the end of the function" case. *)
203     let add_terminal builder instr =
204       match L.block_terminator (L.insertion_block builder) with
205             Some _ -> ()
206       | None -> ignore (instr builder) in
207
208     (* Build the code for the given statement; return the builder for
209        the statement's successor (i.e., the next instruction will be built
```

```
210         after the one generated by this call) *)
211
212     let rec stmt builder table = function
213               SBlock sl -> List.fold_left (fun (builder, table) s -> stmt builder
                ↪  table s) (builder, table) sl
214       | SExpr e -> ignore(expr builder table e); (builder, table)
215       | SDAssign (t, u, s, e) -> let add_local m (t, n) =
216                                    let local_var = L.build_alloca (ltype_of_typ t) n
                                     ↪  builder
217                                      in StringMap.add n local_var m in
218                                  let new_table = add_local table (t, s) in
219                                  let e' = expr builder new_table e in
220                                  L.build_store e' (lookup s new_table) builder;
221                                  (builder, new_table)
222       | SReturn e -> ignore(match fdecl.sreturn_type with
223                                  (* Special "return nothing" instr *)
224                                  A.Void -> L.build_ret_void builder
225                                  (* Build return statement *)
226                                | _ -> L.build_ret (expr builder table e) builder );
227                     (builder, table)
228       | SIf (predicate, then_stmt, else_stmt) ->
229           let bool_val = expr builder table predicate in
230           let merge_bb = L.append_block context "merge" the_function in
231           let build_br_merge = L.build_br merge_bb in (* partial function *)
232
233           let then_bb = L.append_block context "then" the_function in
234           let (then_builder, table) = stmt (L.builder_at_end context then_bb) table
            ↪  then_stmt in
235             add_terminal then_builder build_br_merge;
236
237           let else_bb = L.append_block context "else" the_function in
238           let (else_builder, table) = stmt (L.builder_at_end context else_bb) table
            ↪  else_stmt in
239             add_terminal else_builder build_br_merge;
240
241                 ignore(L.build_cond_br bool_val then_bb else_bb builder);
242               L.builder_at_end context merge_bb, table
243
244       | SWhile (predicate, body) ->
245           let pred_bb = L.append_block context "while" the_function in
246           ignore(L.build_br pred_bb builder);
```

```
247
248           let body_bb = L.append_block context "while_body" the_function in
249           let (builder, _) = (stmt (L.builder_at_end context body_bb) table body) in
250             add_terminal builder (L.build_br pred_bb);
251
252           let pred_builder = L.builder_at_end context pred_bb in
253           let bool_val = expr pred_builder table predicate in
254
255           let merge_bb = L.append_block context "merge" the_function in
256           ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
257           (L.builder_at_end context merge_bb, table)
258
259       (* Implement for loops as while loops *)
260       | SFor (e1, e2, e3, body) -> stmt builder table
261             ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
262     in
263
264     (* Build the code for each statement in the function *)
265     let (builder, _) = stmt builder local_vars (SBlock fdecl.sfunc_stmts) in
266
267     (* Add a return if the last block falls off the end *)
268     add_terminal builder (match fdecl.sreturn_type with
269         A.Void -> L.build_ret_void
270       | A.Float -> L.build_ret (L.const_float float_t 0.0)
271       | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
272   in
273
274   List.iter build_function_body functions;
275   the_module
```

8.9 Makefile and shell scripts

8.9.1 Makefile

```
1  # "make test" Compiles everything and runs the regression tests
2
3  .PHONY : test
4  test : all testall.sh
5          ./testall.sh
6
7  # "make all" builds the executable as well as the "printbig" library designed
8  # to test linking external code
9
```

```
10 .PHONY : all
11 all : scic.native printbig.o
12
13 # "make microc.native" compiles the compiler
14 #
15 # The _tags file controls the operation of ocamlbuild, e.g., by including
16 # packages, enabling warnings
17 #
18 # See https://github.com/ocaml/ocamlbuild/blob/master/manual/manual.adoc
19
20 scic.native :
21         opam config exec -- \
22         ocamlbuild -use-ocamlfind scic.native -pkgs str
23
24 #
25
26 # "make clean" removes all generated files
27
28 .PHONY : clean
29 clean :
30         ocamlbuild -clean
31         rm -rf testall.log ocamlllvm *.diff *.ll *.exe *.out *.s *.err printbig.o
32
33 # Testing the "printbig" example
34
35 printbig : printbig.c
36         cc -o printbig -DBUILD_TEST printbig.c
37
38 # Building the tarball
39
40 TESTS = \
41   printbig helloWorld printftest printb printC
42
43 FAILS = \
44 #   assign1 assign2 assign3 dead1 dead2 expr1 expr2 expr3 float1 float2 \
45 #   for1 for2 for3 for4 for5 func1 func2 func3 func4 func5 func6 func7 \
46 #   func8 func9 global1 global2 if1 if2 if3 nomain printbig print \
47 #   return1 return2 while1 while2
48
49 TESTFILES = $(TESTS:%=test-%.sc) $(TESTS:%=test-%.out) \
50             $(FAILS:%=fail-%.sc) $(FAILS:%=fail-%.err)
```

```
51
52 TARFILES = ast.ml sast.ml codegen.ml Makefile _tags scic.ml parser.mly \
53         README scanner.mll semant.ml testall.sh \
54         printbig.c  \
55         Dockerfile \
56         $(TESTFILES:%=tests/%)
57
58 microc.tar.gz : $(TARFILES)
59         cd .. && tar czf SCIC/SCIC.tar.gz \
60                 $(TARFILES:%=SCIC/%)
```

8.9.2 testall.sh

```
1  #!/bin/sh
2
3  # Regression testing script for SCIC
4  # Step through a list of files
5  #  Compile, run, and check the output of each expected-to-work test
6  #  Compile and check the error of each expected-to-fail test
7
8  # Path to the LLVM interpreter
9  # LLI="/usr/local/opt/llvm/bin/lli"
10 LLI="/usr/bin/lli"
11 # LLI="lli"
12
13 # Path to the LLVM compiler
14 # LLC="/usr/local/opt/llvm/bin/llc"
15 LLC="/usr/bin/llc"
16 # LLC="llc"
17
18 # Path to the C compiler
19 CC="cc"
20
21 # Path to the microc compiler.  Usually "./microc.native"
22 # Try "_build/microc.native" if ocamlbuild was unable to create a symbolic link.
23 SCIC="./scic.native"
24 #MICROC="_build/microc.native"
25
26 # Set time limit for all operations
27 ulimit -t 30
28
29 globallog=testall.log
```

```
30 rm -f $globallog
31 error=0
32 globalerror=0
33
34 keep=0
35
36 Usage() {
37     echo "Usage: testall.sh [options] [.sc files]"
38     echo "-k    Keep intermediate files"
39     echo "-h    Print this help"
40     exit 1
41 }
42
43 SignalError() {
44     if [ $error -eq 0 ] ; then
45         echo "FAILED"
46         error=1
47     fi
48     echo "  $1"
49 }
50
51 # Compare <outfile> <reffile> <difffile>
52 # Compares the outfile with reffile.  Differences, if any, written to difffile
53 Compare() {
54     generatedfiles="$generatedfiles $3"
55     echo diff -b $1 $2 ">" $3 1>&2
56     diff -b "$1" "$2" > "$3" 2>&1 || {
57         SignalError "$1 differs"
58         echo "FAILED $1 differs from $2" 1>&2
59     }
60 }
61
62 # Run <args>
63 # Report the command, run it, and report any errors
64 Run() {
65     echo $* 1>&2
66     eval $* || {
67         SignalError "$1 failed on $*"
68         return 1
69     }
70 }
```

```
71
72  # RunFail <args>
73  # Report the command, run it, and expect an error
74  RunFail() {
75      echo $* 1>&2
76      eval $* && {
77          SignalError "failed: $* did not report an error"
78          return 1
79      }
80      return 0
81  }
82
83  Check() {
84      error=0
85      basename=`echo $1 | sed 's/.*\\///
86                              s/.sc//'`
87      reffile=`echo $1 | sed 's/.sc$//'`
88      basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
89
90      echo -n "$basename..."
91
92      echo 1>&2
93      echo "###### Testing $basename" 1>&2
94
95      generatedfiles=""
96
97      generatedfiles="$generatedfiles ${basename}.ll ${basename}.s ${basename}.exe
        ↪  ${basename}.out" &&
98      Run "$SCIC" "$1" ">" "${basename}.ll" &&
99      Run "$LLC" "-relocation-model=pic" "${basename}.ll" ">" "${basename}.s" &&
100     Run "$CC" "-o" "${basename}.exe" "${basename}.s" "printbig.o" "-lm" &&
101     Run "./${basename}.exe" > "${basename}.out" &&
102     Compare ${basename}.out ${reffile}.out ${basename}.diff
103
104     # Report the status and clean up the generated files
105
106     if [ $error -eq 0 ] ; then
107         if [ $keep -eq 0 ] ; then
108             rm -f $generatedfiles
109         fi
110         echo "OK"
```

```
111         echo "###### SUCCESS" 1>&2
112     else
113         echo "###### FAILED" 1>&2
114         globalerror=$error
115     fi
116 }
117 
118 CheckFail() {
119     error=0
120     basename=`echo $1 | sed 's/.*\\///
121                              s/.sc//'`
122     reffile=`echo $1 | sed 's/.sc$//'`
123     basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."
124 
125     echo -n "$basename..."
126 
127     echo 1>&2
128     echo "###### Testing $basename" 1>&2
129 
130     generatedfiles=""
131 
132     generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
133     RunFail "$SCIC" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
134     Compare ${basename}.err ${reffile}.err ${basename}.diff
135 
136     # Report the status and clean up the generated files
137 
138     if [ $error -eq 0 ] ; then
139         if [ $keep -eq 0 ] ; then
140             rm -f $generatedfiles
141         fi
142         echo "OK"
143         echo "###### SUCCESS" 1>&2
144     else
145         echo "###### FAILED" 1>&2
146         globalerror=$error
147     fi
148 }
149 
150 while getopts kdpsh c; do
151     case $c in
```

```
152         k) # Keep intermediate files
153             keep=1
154             ;;
155         h) # Help
156             Usage
157             ;;
158     esac
159 done
160 
161 shift `expr $OPTIND - 1`
162 
163 LLIFail() {
164   echo "Could not find the LLVM interpreter \"$LLI\"."
165   echo "Check your LLVM installation and/or modify the LLI variable in testall.sh"
166   exit 1
167 }
168 
169 which "$LLI" >> $globallog || LLIFail
170 
171 if [ ! -f printbig.o ]
172 then
173     echo "Could not find printbig.o"
174     echo "Try \"make printbig.o\""
175     exit 1
176 fi
177 
178 if [ $# -ge 1 ]
179 then
180     files=$@
181 else
182     files="tests/test-*.sc tests/fail-*.sc"
183 fi
184 
185 for file in $files
186 do
187     case $file in
188         *test-*)
189             Check $file 2>> $globallog
190             ;;
191         *fail-*)
192             CheckFail $file 2>> $globallog
```

```
193             ;;
194         *)
195             echo "unknown file type $file"
196             globalerror=1
197             ;;
198     esac
199 done
200
201 exit $globalerror
```

8.10 Test suite

fail-arith1.sc

```
1 int func main() {
2     printb(56.13 <= -1);      /* Less */
3     return 0;
4 }
```

fail-arith1.err

```
1 int func main() {
2     printb(56.13 <= -1);      /* Less */
3     return 0;
4 }
```

fail-arith2.sc

```
1 int func main() {
2     print(2 ^ 3);      /* expect float ^ int for pow */
3     return 0;
4 }
```

fail-arith2.err

```
int func main() {
    print(2 ^ 3);      /* expect float ^ int for pow */
    return 0;
}
```

---

fail-arr1.sc

```
int func main() {
    int[] a = [1,2,3,4];
    print(a[1.2]);
    return 0;
}
```

---

fail-arr1.err

```
int func main() {
    int[] a = [1,2,3,4];
    print(a[1.2]);
    return 0;
}
```

---

fail-arr2.sc

```
int func main() {
    float[] x = [2.3, 0, 1.5];
    return 0;
}
```

---

fail-arr2.err

```
int func main() {
    float[] x = [2.3, 0, 1.5];
```

```
3     return 0;
4 }
```

---

fail-assign.sc

```
1 int func main() {
2     float '{m} x = 2;
3     return 0;
4 }
```

---

fail-assign.err

```
1 int func main() {
2     float '{m} x = 2;
3     return 0;
4 }
```

---

fail-for1.sc

```
1 int func main() {
2     int i = 2;
3     for ( ; i; ) {      /* 2nd expression in parentheses should be boolean */
4         print(i);
5     }
6     return 0;
7 }
```

---

fail-for1.err

```
1 int func main() {
2     int i = 2;
3     for ( ; i; ) {      /* 2nd expression in parentheses should be boolean */
4         print(i);
```

```
5       }
6       return 0;
7   }
```

---

fail-for2.sc

```
1   int func main() {
2       int i = 0;
3       for (; j < 10; ) {}
4       return 0;
5   }
```

---

fail-for2.err

```
1   int func main() {
2       int i = 0;
3       for (; j < 10; ) {}
4       return 0;
5   }
```

---

fail-for3.sc

```
1   int func main() {
2       for (int i = 0; i < 5; i = i + 1) {         /* declare and assign in for loop
    ↪   expression not allowed */
3           print(i);
4       }
5       return 0;
6   }
```

---

fail-for3.err

```
1   int func main() {
```

```
    for (int i = 0; i < 5; i = i + 1) {         /* declare and assign in for loop
↪   expression not allowed */
        print(i);
    }
    return 0;
}
```

---

fail-funcall1.sc

```
void func foo() {
    printl("Hello World");
}

int func foo() {}

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

---

fail-funcall1.err

```
void func foo() {
    printl("Hello World");
}

int func foo() {}

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

---

fail-funcall2.sc

```
void func printl() {}

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

---

fail-funcall2.err

```
void func printl() {}

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

---

fail-funcall3.sc

```
int func foo() {
    return 0.25;
}


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

---

fail-funcall3.err

```
int func foo() {
    return 0.25;
}


int func main() {
    foo();
```

```
8       return 0;
9   }
```

---

fail-funcall4.sc

```
1   int func foo() {
2       return 0;
3       2 + 3;
4   }
5
6
7   int func main() {
8       foo();
9       return 0;
10  }
```

---

fail-funcall4.err

```
1   int func foo() {
2       return 0;
3       2 + 3;
4   }
5
6
7   int func main() {
8       foo();
9       return 0;
10  }
```

---

fail-funcall5.sc

```
1   void func foo(int x, bool y, float z) {}
2
3   int func main() {
4       foo(2);
```

```
5       return 0;
6   }
```

---

fail-funcall5.err

```
1   void func foo(int x, bool y, float z) {}
2
3   int func main() {
4       foo(2);
5       return 0;
6   }
```

---

fail-funcall6.sc

```
1   void func foo(float x, int y, bool z) {}
2
3   int func main() {
4       foo(2.3, 5, true);
5
6       foo(3.5, true, 1);
7       return 0;
8   }
```

---

fail-funcall6.err

```
1   void func foo(float x, int y, bool z) {}
2
3   int func main() {
4       foo(2.3, 5, true);
5
6       foo(3.5, true, 1);
7       return 0;
8   }
```

---

fail-if.sc

```
int func main() {
    if (true) {}
    if (false) {}
    if (2) {
        printl("BUG");
    }
    return 0;
}
```

fail-if.err

```
int func main() {
    if (true) {}
    if (false) {}
    if (2) {
        printl("BUG");
    }
    return 0;
}
```

fail-unit-convert1.sc

```
|'{ms} = 1000.0 '{s}|;

int func main() {
    float '{m} x = 2.5;
    float '{ms} y = x;
    printf(y);
    return 0;
}
```

fail-unit-convert1.err

```
|'{ms} = 1000.0 '{s}|;

int func main() {
    float '{m} x = 2.5;
    float '{ms} y = x;
    printf(y);
    return 0;
}
```

---

fail-unit-convert2.sc

```
int func main() {
    float '{m} dx = 2.0;
    float '{s} t = 0.5;
    float '{m/s*s} acc = dx / t;          /* unit in LHS is m/s*s, RHS is m/s */
    return 0;
}
```

---

fail-unit-convert2.err

```
int func main() {
    float '{m} dx = 2.0;
    float '{s} t = 0.5;
    float '{m/s*s} acc = dx / t;          /* unit in LHS is m/s*s, RHS is m/s */
    return 0;
}
```

---

fail-unit-decl1.sc

```
|'{mm} = 0.001 '{m}|;
|'{mm} = 0.01 '{m}|;           /* duplicate declarations not allowed */

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

fail-unit-decl1.err

```
|'{mm} = 0.001 '{m}|;
|'{mm} = 0.01 '{m}|;           /* duplicate declarations not allowed */

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

fail-unit-decl2.sc

```
|'{mm} = 0.001 '{random}|;

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

fail-unit-decl2.err

```
|'{mm} = 0.001 '{random}|;

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

fail-unit-decl3.sc

```
|'{cm} = 0.01 '{m}|;

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

fail-unit-decl3.err

```
|'{cm} = 0.01 '{m}|;

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

fail-unit-decl4.sc

```
int func main() {
    float '{m} x = 0.5;
    float y = x;          /* cannot assign data with unit to data without unit */
    return 0;
}
```

fail-unit-decl4.err

```
int func main() {
    float '{m} x = 0.5;
    float y = x;          /* cannot assign data with unit to data without unit */
    return 0;
}
```

fail-unit-type.sc

```
int func main() {
    int '{m} x = 2;       /* int data with unit not supported */
    return 0;
}
```

fail-unit-type.err

```
int func main() {
    int '{m} x = 2;       /* int data with unit not supported */
    return 0;
}
```

fail-while.sc

```
int func main() {
    while (0) {
        printl("BUG");
    }
    return 0;
}
```

fail-while.err

```
int func main() {
    while (0) {
        printl("BUG");
    }
    return 0;
}
```

test-arr1.sc

```
int func main() {
    int[] a = [1,2,3,4];
    print(a[0]);
    print(a[0 + 1]);     /* a[1] */
    print(a[1 * 2]);     /* a[2] */
    print(a[3 / 1]);     /* a[3] */
    return 0;
}
```

test-arr1.out

```
int func main() {
    int[] a = [1,2,3,4];
    print(a[0]);
    print(a[0 + 1]);     /* a[1] */
    print(a[1 * 2]);     /* a[2] */
    print(a[3 / 1]);     /* a[3] */
    return 0;
}
```

test-arr2.sc

```
int func main() {
    int[] a = [1,2,3,4];
    a[0] = 0;
    print(a[0]);
    a[0] = 5;
    print(a[0]);
    return 0;
}
```

test-arr2.out

```
int func main() {
    int[] a = [1,2,3,4];
    a[0] = 0;
    print(a[0]);
    a[0] = 5;
    print(a[0]);
    return 0;
}
```

test-arr3.sc

```
int func main() {
    int x = 5;
    int[] arr = [2 * x, x * x, (10 - 3) * 2, 5];
    int i = 0;
    for (; i < 4; i = i + 1) {
        print(arr[i]);
    }
    return 0;
}
```

---

test-arr3.out

```
int func main() {
    int x = 5;
    int[] arr = [2 * x, x * x, (10 - 3) * 2, 5];
    int i = 0;
    for (; i < 4; i = i + 1) {
        print(arr[i]);
    }
    return 0;
}
```

---

test-block.sc

```
int func foo() {
    {
        int x = 2;
        {
            int y = 3;
            print(x + y);
            {
                x = 10;
                print(x - y);
            }
        }
    }
    return 1;
```

```
}

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

---

test-block.out

```
int func foo() {
    {
        int x = 2;
        {
            int y = 3;
            print(x + y);
            {
                x = 10;
                print(x - y);
            }
        }
    }
    return 1;
}

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

---

test-boolOp.sc

```
int func main () {
    printb(true || false);
    printb(false && true);

    printb(true && (true || false));
    printb( false == (true || false));
```

```
7
8       printb( !false );
9       printb( ! (3>1));
10      return 0;
11  }
```

---

test-boolOp.out

```
1   int func main () {
2       printb(true || false);
3       printb(false && true);
4
5       printb(true && (true || false));
6       printb( false == (true || false));
7
8       printb( !false );
9       printb( ! (3>1));
10      return 0;
11  }
```

---

test-floatBinop.sc

```
1   int func main() {
2       printf(1.2 - 0.1);          /* Sub */
3       printf(1.2 + 2.876);        /* Add */
4       printf(1.2 - 2.876);        /* Sub */
5       printf(1.2 * 2.876);        /* Mul */
6       printf(1.2 / 2.4);          /* Div */
7       printf(1.2 / 2.874);        /* Div, truncation at the 6th digit (in total) */
8       printb(2.1234567 > 1.5);    /* Greater */
9       printb(1.0 == 1.0);         /* Equal - not necessarily correct? */
10      printb(1.00 == 1.0);        /* Equal - not necessarily correct? */
11      printb(0.99 >= 1.000);      /* Geq */
12      printf(2.0 ^ 3.0);         /* Pow float ^ float*/
13      printf(2.0 ^ 1.5);         /* Pow float ^ float */
14      printf(3.0 ^ 2);         /* Pow float ^ int */
15      printf(5.5 ^ 3);            /* Pow float ^ int */
```

```
16      return 0;
17  }
```

---

test-floatBinop.out

```
1   int func main() {
2       printf(1.2 - 0.1);          /* Sub */
3       printf(1.2 + 2.876);        /* Add */
4       printf(1.2 - 2.876);        /* Sub */
5       printf(1.2 * 2.876);        /* Mul */
6       printf(1.2 / 2.4);          /* Div */
7       printf(1.2 / 2.874);        /* Div, truncation at the 6th digit (in total) */
8       printb(2.1234567 > 1.5);    /* Greater */
9       printb(1.0 == 1.0);         /* Equal - not necessarily correct? */
10      printb(1.00 == 1.0);        /* Equal - not necessarily correct? */
11      printb(0.99 >= 1.000);      /* Geq */
12      printf(2.0 ^ 3.0);         /* Pow float ^ float*/
13      printf(2.0 ^ 1.5);         /* Pow float ^ float */
14      printf(3.0 ^ 2);         /* Pow float ^ int */
15      printf(5.5 ^ 3);            /* Pow float ^ int */
16      return 0;
17  }
```

---

test-for1.sc

```
1   int func main() {
2       int[] arr = [1, 3, 5, 7, 5, 3, 1];
3       int i = 0;
4       for (; i < 7; i = i + 1) {
5           print(arr[i]);
6           arr[i] = i;
7       }
8
9       printl("========");
10
11      for (i = 6; i >= 0; i = i - 1) {
12          print(arr[i]);
```

```
    }
    return 0;
}
```

---

test-for1.out

```
int func main() {
    int[] arr = [1, 3, 5, 7, 5, 3, 1];
    int i = 0;
    for (; i < 7; i = i + 1) {
        print(arr[i]);
        arr[i] = i;
    }

    println("========");

    for (i = 6; i >= 0; i = i - 1) {
        print(arr[i]);
    }
    return 0;
}
```

---

test-for2.sc

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

int func main() {
    int a = 0;
    int b = 1;
    for (; a + b < 15; a = a + b) {
        print(foo(a, b));
        b = b + 1;
    }
    return 0;
}
```

```
14
15 /*  a = 0, b = 1, foo(a, b) = 1, b = 2, a = 2
16     a = 2, b = 2, foo(a, b) = 4, b = 3, a = 5
17     a = 5, b = 3, foo(a, b) = 8, b = 4, a = 9
18     a = 9, b = 4, break             */
```

---

test-for2.out

```
1  int func foo(int a, int b) {
2      return a + b;
3  }
4
5  int func main() {
6      int a = 0;
7      int b = 1;
8      for (; a + b < 15; a = a + b) {
9          print(foo(a, b));
10         b = b + 1;
11     }
12     return 0;
13 }
14
15 /*  a = 0, b = 1, foo(a, b) = 1, b = 2, a = 2
16     a = 2, b = 2, foo(a, b) = 4, b = 3, a = 5
17     a = 5, b = 3, foo(a, b) = 8, b = 4, a = 9
18     a = 9, b = 4, break             */
```

---

test-funcall1.sc

```
1  void func foo() {
2      printl("Hello World");
3  }
4
5  int func main() {
6      foo();
7      return 0;
8  }
```

test-funcall1.out

```
void func foo() {
    printl("Hello World");
}

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

test-funcall2.sc

```
int func foo(int x) {
    x = 3;
    print(x);
    return 10;
}

int func main() {
    print(foo(2));
    return 0;
}
```

test-funcall2.out

```
int func foo(int x) {
    x = 3;
    print(x);
    return 10;
}

int func main() {
    print(foo(2));
```

```
9       return 0;
10  }
```

---

test-funcall3.sc

```
1   float func foo(float x) {
2       float a = x + 0.99;
3       printf(x);
4       printf(a);
5       return a / 0.1 * 3.0;
6   }
7
8   int func main() {
9       printf(foo(0.1));
10      /* assign to a local*/
11      float ua = foo(0.2);
12      printf(ua);
13      return 0;
14  }
```

---

test-funcall3.out

```
1   float func foo(float x) {
2       float a = x + 0.99;
3       printf(x);
4       printf(a);
5       return a / 0.1 * 3.0;
6   }
7
8   int func main() {
9       printf(foo(0.1));
10      /* assign to a local*/
11      float ua = foo(0.2);
12      printf(ua);
13      return 0;
14  }
```

---

test-funcall4.sc

```
bool func foo(int x, int y, bool a, bool b) {
    if (a && b)
        print(x + y);
    else
        print(x - y);
}

int func main() {
    foo(1, 2, true, false);   /* -1 */
    foo(1, 2, true, true);    /*  3 */
    return 0;
}
```

test-funcall4.out

```
bool func foo(int x, int y, bool a, bool b) {
    if (a && b)
        print(x + y);
    else
        print(x - y);
}

int func main() {
    foo(1, 2, true, false);   /* -1 */
    foo(1, 2, true, true);    /*  3 */
    return 0;
}
```

test-helloWorld.sc

```
int func main() {
    printl("Hello world!");
    return 0;
}
```

test-helloWorld.out

```
int func main() {
    printl("Hello world!");
    return 0;
}
```

test-if1.sc

```
void func check(int num) {
    if (num < 0)
        printl("negative");
    else
        printl("non-negative");
}

int func main() {
    int x = 5;
    check(x);
    return 0;
}
```

test-if1.out

```
void func check(int num) {
    if (num < 0)
        printl("negative");
    else
        printl("non-negative");
}

int func main() {
    int x = 5;
    check(x);
```

```
11      return 0;
12  }
```

---

test-if2.sc

```
1   int func main() {
2       int x = 5;
3       if (x > 0) {
4           printl("positive");
5           print(x + 2);
6       }
7
8       return 0;
9   }
```

---

test-if2.out

```
1   int func main() {
2       int x = 5;
3       if (x > 0) {
4           printl("positive");
5           print(x + 2);
6       }
7
8       return 0;
9   }
```

---

test-intAssignGlobal.sc

```
1   int x;
2   int func main(){
3       x = 5;
4       print(x);
5       return 0;
6   }
```

test-intAssignGlobal.out

```
int x;
int func main(){
    x = 5;
    print(x);
    return 0;
}
```

test-intAssignLocal.sc

```
int func main() {
    int x = 3;
    print(x);
    return 0;
}
```

test-intAssignLocal.out

```
int func main() {
    int x = 3;
    print(x);
    return 0;
}
```

test-intBinopUnop.sc

```
int func main() {
    print(10 + 19801);          /* Add */
    print(0 - 1234567);         /* Sub */
    print(100 * 900000);        /* Mul */
    print(90 / 80);             /* Div -> floor */
    print(-90 / 80);            /* Div -> sign abs(floor(res)) */
```

```
7       printb(2  > -2);            /* Greater */
8       printb(1 + 4 == 5);         /* Equal, Add */
9       printb(7 == 30 / 4);       /* Equal, Div */
10      printb(12345678901234 >= 12345678901233);      /* Geq */
11      print(2147483647);
12      print(2147483648);
13      printb(2147483647 > 2147483648);      /* Greater */
14      printb(-100 <= 51-151);       /* Leq */
15      printb(-91 < -1);             /* Less */
16      print(-187);                    /* Neg */
17      return 0;
18  }
```

---

test-intBinopUnop.out

```
1   int func main() {
2       print(10 + 19801);          /* Add */
3       print(0 - 1234567);         /* Sub */
4       print(100 * 900000);        /* Mul */
5       print(90 / 80);             /* Div -> floor */
6       print(-90 / 80);            /* Div -> sign abs(floor(res)) */
7       printb(2  > -2);            /* Greater */
8       printb(1 + 4 == 5);         /* Equal, Add */
9       printb(7 == 30 / 4);       /* Equal, Div */
10      printb(12345678901234 >= 12345678901233);      /* Geq */
11      print(2147483647);
12      print(2147483648);
13      printb(2147483647 > 2147483648);      /* Greater */
14      printb(-100 <= 51-151);       /* Leq */
15      printb(-91 < -1);             /* Less */
16      print(-187);                    /* Neg */
17      return 0;
18  }
```

---

test-localdassign.sc

```
1   /*locally declare and assign variables*/
```

```
2
3  bool func larger(int a, int b) {
4      string ss = "inside compare func";
5      printl(ss);
6      return a > b;
7  }
8
9  int func main() {
10     float '{A} I = 1.2;
11     printl("Second line in main");
12     bool islarger = larger(1, 3);
13     printb(islarger);
14     return 0;
15 }
```

---

test-localdassign.out

```
1  /*locally declare and assign variables*/
2
3  bool func larger(int a, int b) {
4      string ss = "inside compare func";
5      printl(ss);
6      return a > b;
7  }
8
9  int func main() {
10     float '{A} I = 1.2;
11     printl("Second line in main");
12     bool islarger = larger(1, 3);
13     printb(islarger);
14     return 0;
15 }
```

---

test-printb.sc

```
1  int func main () {
2      printb(true);
```

```
3     printb(false);
4 }
```

---

test-printb.out

```
1 int func main () {
2     printb(true);
3     printb(false);
4 }
```

---

test-printbig.sc

```
1  /*
2   * Test for linking external C functions to LLVM-generated code
3   *
4   * printbig is defined as an external function, much like printf
5   * The C compiler generates printbig.o
6   * The LLVM compiler, llc, translates the .ll to an assembly .s file
7   * The C compiler assembles the .s file and links the .o file to generate
8   * an executable
9   */
10
11 int func main()
12 {
13   printbig(72); /* H */
14   printbig(69); /* E */
15   printbig(76); /* L */
16   printbig(76); /* L */
17   printbig(79); /* O */
18   printbig(32); /*   */
19   printbig(87); /* W */
20   printbig(79); /* O */
21   printbig(82); /* R */
22   printbig(76); /* L */
23   printbig(68); /* D */
24 }
```

---

test-printbig.out

```
/*
 * 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 func 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 */
}
```

test-printC.sc

```
int func main() {
    printl("w");
}
```

test-printC.out

```
int func main() {
```

```
    printl("w");
}
```

---

test-printftest.sc

```
int func main() {
    printf(1.2);
    printf(0.8181818);
    printf(1.2345678);
    printf(103.1243);
    return 0;
}
```

---

test-printftest.out

```
int func main() {
    printf(1.2);
    printf(0.8181818);
    printf(1.2345678);
    printf(103.1243);
    return 0;
}
```

---

test-printString.sc

```
int func main() {
    printl("Hello World");
}
```

---

test-printString.out

```
int func main() {
    printl("Hello World");
```

```
}
```

---

test-simplemix.sc

```
/* Unit declaration */
|'{mm} = 1000.0 '{cm}|;

/* Global variable declaration */
float '{m} x;

/* Function declarations */
int func gcd(int a, int b) {
    while(a != b) {
        if(a > b)
            a = a - b;
        else
            b = b - a;
    }
    return a;
}

/* Main */
int func main() {
    print(gcd(12, 32));
    return 0;
}
```

---

test-simplemix.out

```
/* Unit declaration */
|'{mm} = 1000.0 '{cm}|;

/* Global variable declaration */
float '{m} x;

/* Function declarations */
int func gcd(int a, int b) {
```

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

/* Main */
int func main() {
    print(gcd(12, 32));
    return 0;
}
```

---

test-udecl2func.sc

```
int x;
|'{mm} = 1000.0 '{m}|;
|'{km} = 0.001 '{m}|;

float '{mm} p;

float '{km} func foo(float '{km} y) {
    float '{km} x1 = y;
    float '{mm} x2 = 9191.9;
    printf(x2);
    return x1;
}

int func main(){
    float '{km} y = 13.2;
    float '{mm} z = 0.235;
    foo(y);
    return 0;
}
```

---

test-udecl2func.out

```
int x;
|'{mm} = 1000.0 '{m}|;
|'{km} = 0.001 '{m}|;

float '{mm} p;

float '{km} func foo(float '{km} y) {
    float '{km} x1 = y;
    float '{mm} x2 = 9191.9;
    printf(x2);
    return x1;
}

int func main(){
    float '{km} y = 13.2;
    float '{mm} z = 0.235;
    foo(y);
    return 0;
}
```

test-udeclNotBase.sc

```
|'{mm} = 10.00 '{cm}|;

int func main(){
    float '{m} a = 1.23;
    float '{cm} b = 45.67;
    float '{mm} p = a;
    printf(p);
    p = b;
    printf(p);
    return 0;
}
```

test-udeclNotBase.out

```
|'{mm} = 10.00 '{cm}|;

int func main(){
    float '{m} a = 1.23;
    float '{cm} b = 45.67;
    float '{mm} p = a;
    printf(p);
    p = b;
    printf(p);
    return 0;
}
```

---

test-uexprtarget.sc

```
/*
test derived unit
*/
|'{g} = 1000.0 '{kg}|;
float '{m*g} x;
float '{cm*kg} y;
int func main() {
    printl("in main");
    x = 9.98;
    y = x;
    printf(y);
    return 0;
}
```

---

test-uexprtarget.out

```
/*
test derived unit
*/
|'{g} = 1000.0 '{kg}|;
float '{m*g} x;
float '{cm*kg} y;
int func main() {
```

```
8       printl("in main");
9       x = 9.98;
10      y = x;
11      printf(y);
12      return 0;
13  }
```

---

test-uexprTarget2.sc

```
1   |'{g} = 1000.0 '{kg}|;
2   |'{mm} = 10.0 '{cm}|;
3
4   float '{mm/g} x;
5   float '{cm/kg} y;
6
7   int func main() {
8       printl("in main");
9       x = 9.978;
10      printf(x);
11      y = x - 5.973;
12      printf(y);
13      float '{m/kg} z = (x - 5.974) * 0.1;
14      printf(z);
15      float '{m/kg} w = y - z;
16      printf(w);
17      return 0;
18  }
```

---

test-uexprTarget2.out

```
1   |'{g} = 1000.0 '{kg}|;
2   |'{mm} = 10.0 '{cm}|;
3
4   float '{mm/g} x;
5   float '{cm/kg} y;
6
7   int func main() {
```

```
8       printl("in main");
9       x = 9.978;
10      printf(x);
11      y = x - 5.973;
12      printf(y);
13      float '{m/kg} z = (x - 5.974) * 0.1;
14      printf(z);
15      float '{m/kg} w = y - z;
16      printf(w);
17      return 0;
18  }
```

---

test-unit-array1.sc

```
1   /* example of recording experiment result */
2   |'{mm} = 1000.0 '{m}|;
3
4   void func foo(float '{m/s} base) {
5       int i = 0;
6       float[] '{m} dx = [2.3, 4.5, 3.4, 0.7];
7       float[] '{s} dt = [0.5, 0.2, 1.7, 0.5];
8       float[] '{mm/s} res = [0.0, 0.0, 0.0, 0.0];
9       for (; i < 4; i = i + 1) {
10          res[i] = dx[i] / dt[i] + base ;   /* 2.3m / 0.5s = 4.6 m/s + 0.0122 m/s =
    ↪   4.6122 m/s = 4612.2 mm/s */
11          printf(res[i]);
12      }
13  }
14
15  int func main() {
16      float '{cm/s} base = 1.22;
17      foo(base);                  /* 1.22 cm/s = 0.0122 m/s */
18      return 0;
19  }
```

---

test-unit-array1.out

```
/* example of recording experiment result */
|'{mm} = 1000.0 '{m}|;

void func foo(float '{m/s} base) {
    int i = 0;
    float[] '{m} dx = [2.3, 4.5, 3.4, 0.7];
    float[] '{s} dt = [0.5, 0.2, 1.7, 0.5];
    float[] '{mm/s} res = [0.0, 0.0, 0.0, 0.0];
    for (; i < 4; i = i + 1) {
        res[i] = dx[i] / dt[i] + base ;   /* 2.3m / 0.5s = 4.6 m/s + 0.0122 m/s =
↪   4.6122 m/s = 4612.2 mm/s */
        printf(res[i]);
    }
}

int func main() {
    float '{cm/s} base = 1.22;
    foo(base);                  /* 1.22 cm/s = 0.0122 m/s */
    return 0;
}
```

---

test-unit-array2.sc

```
/* test array access */

|'{ms} = 1000.0 '{s}|;

int func main() {
    float[] '{cm} x = [1.1, 2.2, 3.3];
    float[] '{m} y = [1.0, 2.0];
    y[0] = 5.5;
    printf(y[0]);
    y[1] = x[1];
    printf(y[1]);
    return 0;
}
```

---

test-unit-array2.out

```
/* test array access */

|'{ms} = 1000.0 '{s}|;

int func main() {
    float[] '{cm} x = [1.1, 2.2, 3.3];
    float[] '{m} y = [1.0, 2.0];
    y[0] = 5.5;
    printf(y[0]);
    y[1] = x[1];
    printf(y[1]);
    return 0;
}
```

test-unit-array3.sc

```
/* test array access */

|'{ms} = 1000.0 '{s}|;

int func main() {
    float[] '{cm} x = [1.1, 2.2, 3.3];
    float '{m} y = 4.4;
    y = x[1];
    printf(y); /*2.2cm = 0.022m, prints 0.022*/
    return 0;
}
```

test-unit-array3.out

```
/* test array access */

|'{ms} = 1000.0 '{s}|;

int func main() {
    float[] '{cm} x = [1.1, 2.2, 3.3];
```

```
7       float '{m} y = 4.4;
8       y = x[1];
9       printf(y); /*2.2cm = 0.022m, prints 0.022*/
10      return 0;
11  }
```

---

test-unit-convert1.sc

```
1   /* test local var declaration with unit */
2
3   float '{cm} y;
4   int func main(){
5       float '{m} z = 0.05;
6       printf(z); /* 0.05 m*/
7       y = z;     /* auto-conversion cm<-m*/
8       printf(y); /* 5 cm*/
9       printf(z); /* 0.05 m<- z unchanged*/
10      return 0;
11  }
```

---

test-unit-convert1.out

```
1   /* test local var declaration with unit */
2
3   float '{cm} y;
4   int func main(){
5       float '{m} z = 0.05;
6       printf(z); /* 0.05 m*/
7       y = z;     /* auto-conversion cm<-m*/
8       printf(y); /* 5 cm*/
9       printf(z); /* 0.05 m<- z unchanged*/
10      return 0;
11  }
```

---

test-unit-convert2.sc

```
/* constant acceleration formula dx = at^2 */
int func main() {
    float '{m} start1 = 1.05;
    float '{m} end1 = 2.05;
    float '{m} dx1 = end1 - start1;    /* dx = 1.0 */

    float '{m} start2 = 2.05;
    float '{m} end2 = 4.05;
    float '{m} dx2 = end2 - start2;     /* dx = 2.0 */

    float '{s} dt = 0.5;

    float '{m/s*s} acceleration = (dx2 - dx1) / (dt * dt);
    printf(acceleration);
    return 0;
}
```

---

test-unit-convert2.out

```
/* constant acceleration formula dx = at^2 */
int func main() {
    float '{m} start1 = 1.05;
    float '{m} end1 = 2.05;
    float '{m} dx1 = end1 - start1;    /* dx = 1.0 */

    float '{m} start2 = 2.05;
    float '{m} end2 = 4.05;
    float '{m} dx2 = end2 - start2;     /* dx = 2.0 */

    float '{s} dt = 0.5;

    float '{m/s*s} acceleration = (dx2 - dx1) / (dt * dt);
    printf(acceleration);
    return 0;
}
```

---

test-unit-convert3.sc

```
|'{us} = 1.0E+6 '{s}|;

float '{s} func foo(float x) {
    float '{s} a = x + 0.99;
    printf(x);
    printf(a);
    return a;
}

int func main() {
    printf(foo(0.1));
    /* assign to a local*/
    float '{us} ua = foo(0.2);
    printf(ua);
    return 0;
}
```

---

test-unit-convert3.out

```
|'{us} = 1.0E+6 '{s}|;

float '{s} func foo(float x) {
    float '{s} a = x + 0.99;
    printf(x);
    printf(a);
    return a;
}

int func main() {
    printf(foo(0.1));
    /* assign to a local*/
    float '{us} ua = foo(0.2);
    printf(ua);
    return 0;
}
```

---

test-unit-convert4.sc

```
|'{us} = 1.0E+6 '{s}|;
float '{us} ua;

float '{s} func foo(float x) {
    float '{s} a = x + 0.99;
    printf(x);
    printf(a);
    return a;
}

int func main() {
    printf(foo(0.1));
    /* assign to a global*/
    ua = foo(0.2);
    printf(ua);
    return 0;
}
```

---

test-unit-convert4.out

```
|'{us} = 1.0E+6 '{s}|;
float '{us} ua;

float '{s} func foo(float x) {
    float '{s} a = x + 0.99;
    printf(x);
    printf(a);
    return a;
}

int func main() {
    printf(foo(0.1));
    /* assign to a global*/
    ua = foo(0.2);
    printf(ua);
    return 0;
}
```

---

test-unit-convert5.sc

```
|'{km} = 0.001 '{m}|;

float '{cm} x;
float '{m} y;

int func main(){
    y = 1.022;
    float '{km} z = y;
    printf(z);
    z = y * 335.0;
    printf(z);
    float '{km} w = (3.1 + y) * 10.;
    printf(w);
    return 0;
}
```

test-unit-convert5.out

```
|'{km} = 0.001 '{m}|;

float '{cm} x;
float '{m} y;

int func main(){
    y = 1.022;
    float '{km} z = y;
    printf(z);
    z = y * 335.0;
    printf(z);
    float '{km} w = (3.1 + y) * 10.;
    printf(w);
    return 0;
}
```

test-unit-convert6.sc

```
int func main() {
    float '{cm} x = 30.0;
    float '{m} y = x;
    printf(y);
    return 0;
}
```

test-unit-convert6.out

```
int func main() {
    float '{cm} x = 30.0;
    float '{m} y = x;
    printf(y);
    return 0;
}
```

test-unit-convert7.sc

```
|'{km} = 0.001 '{m}|;

float '{km} x;
float '{m} y;

int func main(){
    x = 0.985321;
    float '{cm} z = x;
    printf(z);
    printf(z +1.1);
    return 0;
}
```

test-unit-convert7.out

```
|'{km} = 0.001 '{m}|;
```

```
2
3  float '{km} x;
4  float '{m} y;
5
6  int func main(){
7      x = 0.985321;
8      float '{cm} z = x;
9      printf(z);
10     printf(z +1.1);
11     return 0;
12 }
```

---

test-unit-decl1.sc

```
1  int x;
2  |'{mm} = 1000.0 '{m}|;
3  |'{um} = 1.0E+6 '{m}|;
4
5  float '{mm} p;
6
7  int func main(){
8      float '{km} y = 13.2;
9      float '{um} z = 230.0;
10     printf(y);
11     printf(z);
12     p = z;
13     printf(p);
14     return 0;
15 }
```

---

test-unit-decl1.out

```
1  int x;
2  |'{mm} = 1000.0 '{m}|;
3  |'{um} = 1.0E+6 '{m}|;
4
5  float '{mm} p;
```

```

int func main(){
    float '{km} y = 13.2;
    float '{um} z = 230.0;
    printf(y);
    printf(z);
    p = z;
    printf(p);
    return 0;
}
```

---

test-unit-formal.sc

```
|'{mm} = 1000.0 '{m}|;

void func test (float '{m} start1, float '{m} end1, float '{m} start2, float '{m}
↪  end2) {
    printf(start1); /* 3 */
    printf(end1); /* 0.4 */
    printf(start2);  /* 3.01 */
    printf(end2);   /* 0.41 */
}

int func main() {
    float[] '{mm} s1 = [105.2, 98.4, 100.3];
    float[] '{mm} e1 = [305.8, 309.9, 398.5];
    float[] '{mm} s2 = [105.1, 94.7, 101.1];
    float[] '{mm} e2 = [305.3, 310.1, 399.8];

    float '{mm} tmp = 3000.0;
    float '{mm} tmp2 = 400.0;
    test(s1[0], s1[1], s1[2] + 10.0, tmp2 + 10.0);
    return 0;
}
```

---

test-unit-formal.out

```
|'{mm} = 1000.0 '{m}|;

void func test (float '{m} start1, float '{m} end1, float '{m} start2, float '{m}
↪  end2) {
    printf(start1); /* 3 */
    printf(end1); /* 0.4 */
    printf(start2);  /* 3.01 */
    printf(end2);   /* 0.41 */
}

int func main() {
    float[] '{mm} s1 = [105.2, 98.4, 100.3];
    float[] '{mm} e1 = [305.8, 309.9, 398.5];
    float[] '{mm} s2 = [105.1, 94.7, 101.1];
    float[] '{mm} e2 = [305.3, 310.1, 399.8];

    float '{mm} tmp = 3000.0;
    float '{mm} tmp2 = 400.0;
    test(s1[0], s1[1], s1[2] + 10.0, tmp2 + 10.0);
    return 0;
}
```

---

test-unit-funcall1.sc

```
|'{km} = 0.001 '{m}|;

/* parameters can be auto-converted
    when passed into function */
float '{km} func foo(float '{m} x) {
    printf(x); /* 30 (m) */
    float '{m} y = x;
    return y;   /* at return: convert to return unit km */
}

int func main() {
    float '{cm} tmp = 3000.0;
    printf(foo(tmp));  /* 0.03 (km)*/
    return 0;
}
```

test-unit-funcall1.out

```
|'{km} = 0.001 '{m}|;

/* parameters can be auto-converted
    when passed into function */
float '{km} func foo(float '{m} x) {
    printf(x); /* 30 (m) */
    float '{m} y = x;
    return y;   /* at return: convert to return unit km */
}

int func main() {
    float '{cm} tmp = 3000.0;
    printf(foo(tmp));  /* 0.03 (km)*/
    return 0;
}
```

test-unit-funcall2.sc

```
/* formals with different units */
float '{m/s} func foo(float '{m} dx, float '{s} dt) {
    return dx / dt;
}

int func main() {
    float '{cm} dx = 300.0;
    printf(foo(dx, 5.0));           /* 300cm = 3m, v = 3m/5s = 0.6 m/s */
    return 0;
}
```

test-unit-funcall2.out

```
/* formals with different units */
float '{m/s} func foo(float '{m} dx, float '{s} dt) {
```

```
3       return dx / dt;
4   }
5
6   int func main() {
7       float '{cm} dx = 300.0;
8       printf(foo(dx, 5.0));           /* 300cm = 3m, v = 3m/5s = 0.6 m/s */
9       return 0;
10  }
```

---

test-unit-funcall3.sc

```
1   /* return value will be auto-converted */
2   float '{m} func foo() {
3       float '{cm} x = 250.0;
4       return x;
5   }
6
7   int func main() {
8       printf(foo());
9       return 0;
10  }
```

---

test-unit-funcall3.out

```
1   /* return value will be auto-converted */
2   float '{m} func foo() {
3       float '{cm} x = 250.0;
4       return x;
5   }
6
7   int func main() {
8       printf(foo());
9       return 0;
10  }
```

---

test-unit-funcall4.sc

```
|'{km} = 0.001 '{m}|;
|'{ms} = 1000.0 '{s}|;

/* return value will be auto-converted */
float '{m/s} func foo() {
    float '{cm} x = 250.0;
    float '{s} t = 2.5;
    return x / t;
}

float '{km/s} func bar(float '{cm/ms} a) {
    float '{cm/ms} b = 31.2 ^ 2;
    printf(a);
    printf(b);
    if (a > b) {
        return a + b;
    }
    else {
        return b - a;
    }
}

int func main() {
    printf(foo());
    printf(bar(foo()));
    return 0;
}
```

test-unit-funcall4.out

```
|'{km} = 0.001 '{m}|;
|'{ms} = 1000.0 '{s}|;

/* return value will be auto-converted */
float '{m/s} func foo() {
    float '{cm} x = 250.0;
    float '{s} t = 2.5;
    return x / t;
```

```
}

float '{km/s} func bar(float '{cm/ms} a) {
    float '{cm/ms} b = 31.2 ^ 2;
    printf(a);
    printf(b);
    if (a > b) {
        return a + b;
    }
    else {
        return b - a;
    }
}

int func main() {
    printf(foo());
    printf(bar(foo()));
    return 0;
}
```

---

test-unitBinop.sc

```
float '{cm} x;
float '{m} y;

int func main(){
    y = 1.022;
    x = 100.9;
    float '{m} z = x + y;
    printf(z);
    float '{m} large = 1.1;
    float '{cm} small = 110.0;
    printb(large == small);
    return 0;
}
```

---

test-unitBinop.out

```
float '{cm} x;
float '{m} y;

int func main(){
    y = 1.022;
    x = 100.9;
    float '{m} z = x + y;
    printf(z);
    float '{m} large = 1.1;
    float '{cm} small = 110.0;
    printb(large == small);
    return 0;
}
```

---

test-while.sc

```
int func main() {
    float[] arr = [2.1, 3.2, 4.3];
    int i = 0;
    while (i < 3) {
        printf(arr[i]);
        i = i + 1;
    }
    return 0;
}
```

---

test-while.out

```
int func main() {
    float[] arr = [2.1, 3.2, 4.3];
    int i = 0;
    while (i < 3) {
        printf(arr[i]);
        i = i + 1;
    }
    return 0;
}
```

Bibliography

[1] C Reference Manual, https://www.bell-labs.com/usr/dmr/www/cman.pdf.

[2] Single-precision floating-point format Wikiepdia, https://en.wikipedia.org/wiki/Single-precision_floating-point_format.

[3] MicroC. OCaml source and test cases for the MicroC language, http://www.cs.columbia.edu/~sedwards/classes/2021/4115-spring/microc.tar.gz.