Columbia University

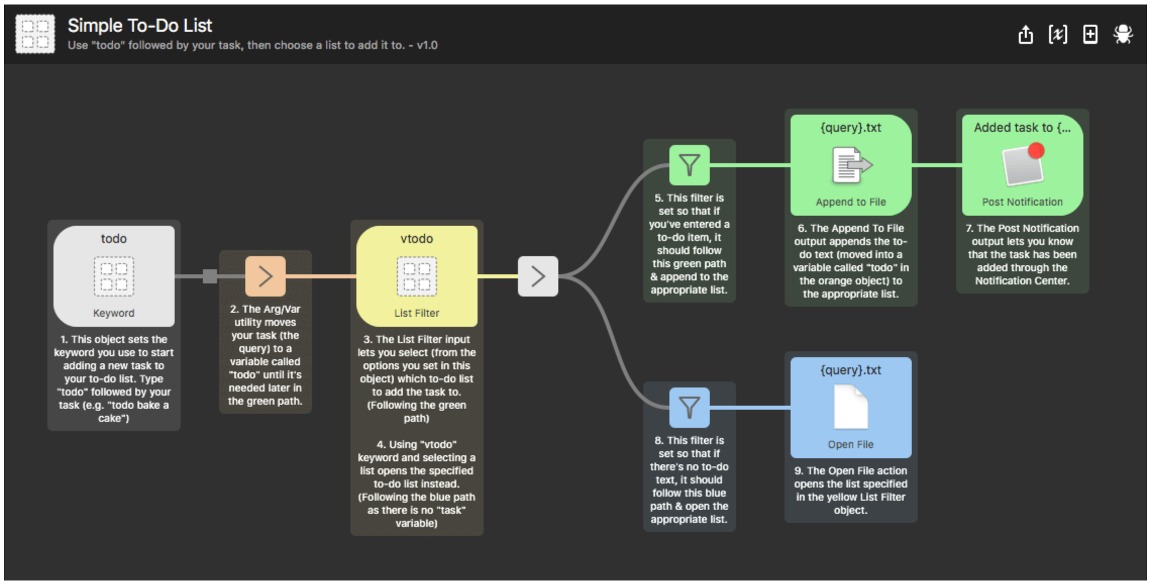
# IRIs Final Report

***Author:***
ShuLan Tang
Xuheng Li
Pinxi Tai
Hanzhou Gu

***Supervisor:***
John Hui

12/19/2018

# Contents



1. This object sets the keyword you use to start adding a new task to your to-do list. Type "todo" followed by your task (e.g. "todo bake a cake")

2. The Arg/Var utility moves your task (the query) to a variable called "todo" until it's needed later in the green path.

3. The List Filter input lets you select (from the options you set in this object) which to-do list to add the task to. (Following the green path)

4. Using "vtodo" keyword and selecting a list opens the specified to-do list instead. (Following the blue path as there is no "task" variable)

5. This filter is set so that if you've entered a to-do item, it should follow this green path & append to the appropriate list.

6. The Append To File output appends the to-do text (moved into a variable called "todo" in the orange object) to the appropriate list.

7. The Post Notification output lets you know that the task has been added through the Notification Center.

8. This filter is set so that if there's no to-do text, it should follow this blue path & open the appropriate list.

9. The Open File action opens the list specified in the yellow List Filter object.

Figure 1: TODO workflow

## 1 Introduction

### 1.1 Inspiration

The inspiration of our language comes from the MacOS Alfred, a simple yet extremely useful tool every mac user loves. It helps users to complete simple tasks easily. Alfred introduce an interesting concept called workflow, which inspires our design of IRIs. Typically, a workflow is constructed by a number of actions, each action takes an input, do some simple process and then present the output to another action, through utilities. The final action in the workflow outputs the desired result that completes the task user wants. Figure 1 shows a workflow to add a new todo. It starts with a trigger key: todo, followed by the note user wants to add. The actions in the workflow do most of the work, and results in the task added into the list.

### 1.2 IRIs Introduction

IRIs not only stands for the flower but also stands for the reverse of the Apple so-called smart assistant Siri. IRIs is a programming language designed for writing programs conducting simple tasks as workflows. It helps the programmer better design and implement the flow as the programs need to be written more vertically compared with the ones written in other programming languages. When you want to assign a value to a variable, you pipe the value into a variable. Similarly, when you assign a parameter to a function, you pipe the values into it. As the program flows to the end of the the function, it accomplish the task the program runner wants to complete.

# 2 Language Tutorial

In this section, we present a simple guide to set up environment and use our language.

## 2.1 Environment Setup

IRIs requires the installation of the OCaml llvm library. Use Ubuntu 16.04 LTS for ease of use. Then download the following packages.

```
sudo apt-get install -y ocaml m4 llvm opam
opam init
opam install llvm.3.6 ocamlfind
eval ‘opam config env
```

Then you need to cd into path/to/project/src and run

```
make all
```

Then the binary of IRIs compiler will be generated as iris.native. To compile and run the program, you can cd into the root of the project and run the iris.sh:

```
./iris.sh yourcode.ir
```

Then the target binary file will be output to the project root directory with name main.

## 2.2 Simple Programs

**Hello World** The IRIs program begins from the main function. You need to declare all of the local variable at the beginning of the function and then you can use them.

```
int main()
    string str
    "Hello World\n"
    |
    str
    |
    printi
Siri
```

**Arithmetic** IRIs using pipe as the assign operation

```
int main()
    int a, b, c
    1
    |
    a
    2
    |
    b
    a
    |
    +b
    |
    c
    |
    printi
Siri
```

**Function Call** In IRIs, you can call a function in 2 ways, on is using "arguments pipe fun_name" and the other is using "fun_name(args)".

```
int add1(int i)
    return i+1
Siri

int main()
    5
    |
    add1
    |
    printi

    add1(5)
    |
    printi

    printi(add1(5))
Siri
```

## 3 Project Plan

### 3.1 Overview

Our team met between 2-5 times every week throughout the semester. We met with our TA every Wednesday, updated our progress and set goals for the next seven days. During the week, we communicated through WeChat when we have questions regarding project implementation and design. When we had some major implementation and design decisions to make, we met in person and discuss the solution collectively.

## 3.2 Design and Implementation

We started with the design of our language. This phase involved lots of mind storming and discussions on what syntax should IRIs support, what kind of programs should be suitable to write in IRIs and what different data structures we need to support. The milestone of this phase are our project proposal and Language Reference Manual.

After that, we started to implementing our scanner. We used our token specified in the scanner to construct our parser and abstract syntax tree. We verified our parser and scanner using the pretty printer of our AST. After we made sure everything has been parsed correctly, we started to do the semantic checking of our language. The semantic checking process produces SAST, if the program is semantically right, otherwise reports error information. Finally, we implemented our LLVM code generation to generate executable programs.

We collaborate through GitHub. In order to avoid merge conflicts, we have a single branch for all the commits. Before each push, the each member first execute:

```
git pull --rebase
```

If there exists conflicts, we fix it in our local machine, then conduct:

```
git push origin
```

## 3.3 Programming Style Guide

Our programming style guide is quite simple but efficient to make sure that codes are easy to understand, and helps to avoid unnecessary bugs. There are five rules we applied:

1. Use spaces instead of tabs for indentation.
2. *let in* keywords should be at separate lines with codes in between.
3. Two spaces for each *let in* indentation.
4. Matching symbol | at the same column for each OCaml match statement. Each one should be followed by a space.
5. No code should be exceed 80 columns with rare exceptions.

## 3.4 Project Timeline

Figure 2 shows the timeline of our project, including different delopment periods and major millstones. We started with a relatively vague idea of IRIs, and our proposal. Then, after a number of thinking and discussion, as well as taking advice from our TA, John, we came up with our LRM, specifying the detailed data types, syntax and other functionality we

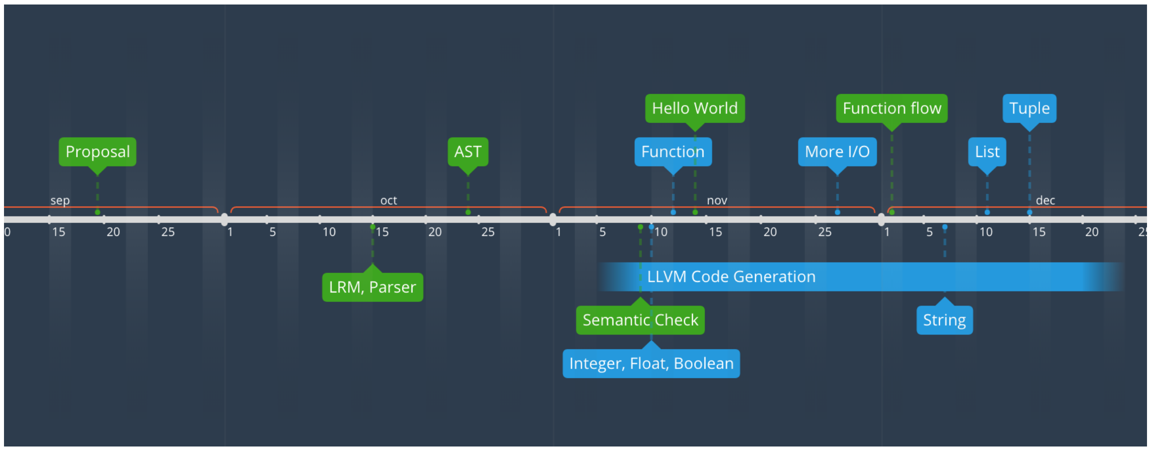


Figure 2: Project Timeline

want to implement. The LRM served as the implementation guideline for the team, and if there's any design change was made, we updated it accordingly. We built our scanner and parser according to our LRM, as well as our abstract syntax tree. The correctness of our those three components are verified through the pretty printer in the AST to make sure the program is parsed as we expected. After that, we started to implement the semantic checking incrementatally and generated SAST for IRIs. As for the code generation part, we started with the MicroC code generator, modified it so that it fits our language. The project continued as we started to progressively improve our code generator to support more advanced data types and syntax.

## 3.5 Team Member Roles and Responsibilities

| Shulan Tang | Xuheng Li | Hanzhou Gu | Pinxi Tai |
|---|---|---|---|
| Manager | System Architect | Language Guru | Tester |
| Parser, Semantic checking, Code Generation | Parser, Scanner, Code Generation | Semantic Checking, Code Generation | Makefile, Tests |

As for the codebase, Shulan and Xuheng were in charge of Scanner and Parser construction, Shulan constructed the semantic checking file and Hanzhou is the one initialize the code generation file. Pinxi created and maintained our testing suite. After the overall architecture was established, everyone worked as a ”full-stack” developer, to create and update each file to implement new features in our project. The detailed contribution can be found in our project log.

## 3.6 Project Development Environment

We use the virtual machine provided by course instructor as our development environment. The platform of the virtual machine is Ubuntu 16.04 with Ocaml 4.05.0 and LLVM 6.0. We use Github for collaboration and Sublime Text as the editor.

## 3.7 Project Log

---

```
commit b36d6c678ef66a0ca887c003f8b1271075f311c8
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Mon Dec 17 01:25:41 2018 -0500

    debug demos

commit 03560915f1a6a8c5c6574e4747039231c1cbe904
Author: Serena <zy16373@gmail.com>
Date:   Mon Dec 17 00:53:54 2018 -0500

    demos

commit 1e0a56d2fa9acfbd466827adc0952087f48d740c
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Mon Dec 17 00:45:59 2018 -0500

    List Length

commit b1875719a87e4a894f0a86a10385d1471317a628
Author: guhanzhou163 <496495139@qq.com>
Date:   Sun Dec 16 23:58:38 2018 -0500

    demo 1

commit 993d462424545dd6655a178ac9ebd440b86c572a
Author: guhanzhou163 <496495139@qq.com>
Date:   Sun Dec 16 23:57:31 2018 -0500

    demo1

commit ad5cebee2ba4af275bec9ceecacd66a7e0ca0fab
Author: guhanzhou163 <496495139@qq.com>
Date:   Sun Dec 16 23:44:54 2018 -0500

    demo send email

commit e4bcc6e5a45d93d778d49412b0276dd7b274f07e
```

```
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Dec 16 22:33:48 2018 -0500

    some bugs

commit d0dd32f81972c2eaa4f3cc78c5bec3ce83b361f6
Author: guhanzhou163 <496495139@qq.com>
Date:   Sun Dec 16 22:10:33 2018 -0500

    functions

commit 25e120333ce955fe4c3865bde7c25f901b0fdedf
Author: guhanzhou163 <496495139@qq.com>
Date:   Sat Dec 15 23:04:44 2018 -0500

    writefile

commit 9d255f16fc971f616485a7106694975b1b6f73aa
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Dec 16 21:47:20 2018 -0500

    delete some junks

commit 68df88c6d0c0524e541c3983e08d1a30fb1f2905
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Dec 16 21:31:45 2018 -0500

    tuple unfinished

commit c5a6d61539c2e3a8d1dac7d1271762de04df6ea8
Author: Serena TANG <zy16373@gmail.com>
Date:   Sun Dec 16 16:18:57 2018 -0500

    test_flow

commit d684190b5286f767d0a3dfebfb77ecb619e2d9cc
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sat Dec 15 18:46:12 2018 -0500

    string alloc

commit 2c37333d56358a18e4201efacba92164e6d1714d
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sat Dec 15 17:38:50 2018 -0500

    list setn
```

```
commit 83d8828ce2e9ad670ba57d240c6df6d319aecf87
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sat Dec 15 15:33:15 2018 -0500

    list code gen

commit 62b91305dda9669e4abb6e3819821f7c7855674b
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sat Dec 15 10:57:19 2018 -0500

    list ops

commit 2c968148d9c9bf8b35676f7891a1f6f6fe0f71d4
Author: pousT <zy16373@gmail.com>
Date:   Fri Dec 14 23:06:12 2018 -0500

    add calloc and free in sast

commit 6fe2fd62ac0b7900100a834418a39be8101159f7
Author: pousT <zy16373@gmail.com>
Date:   Fri Dec 14 22:20:23 2018 -0500

    test suite setup

commit 58f580789fbdd9e349da3b4366675d4e3a590010
Author: Serena <zy16373@gmail.com>
Date:   Fri Dec 14 22:20:19 2018 -0500

    add free and calloc

commit 37f145f9888684c18d0dad64976a48e41537e45b
Merge: f79fd7a c6c3130
Author: Serena <zy16373@gmail.com>
Date:   Fri Dec 14 21:54:48 2018 -0500

    Merge branch 'master' of github.com:XuhengLi/IRIs

commit f79fd7ad303ddefaf520aaa064c857a1cb7d2a4d
Author: Serena <zy16373@gmail.com>
Date:   Fri Dec 14 21:54:43 2018 -0500

    pip support for arithmetic

commit 2ea0568529e02e34eeb996246ed49b473d2966bd
Author: guhanzhou163 <496495139@qq.com>
```

```
Date:   Sun Dec 9 00:29:43 2018 -0500

    first link

commit c5d96f7ba7fa6397c58096892cfff4a9f8284568
Author: guhanzhou163 <496495139@qq.com>
Date:   Sun Dec 9 00:27:14 2018 -0500

    first linker

commit 72cc9370084939a5144ab80700abd64309b61cf9
Author: Serena <zy16373@gmail.com>
Date:   Wed Dec 5 17:08:17 2018 -0500

    strcat and strcpy

commit 8b6ac6263d4af1b40a3fec64fb9712b0e8fec703
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Wed Dec 5 15:31:58 2018 -0500

    strcmp and strlen

commit a47f50a1576076fd3bae4b374d8911ee136cccd1
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Wed Dec 5 14:06:20 2018 -0500

    list lib init

commit 273edcedd8a4acd27f474d88a39061ac79c327bb
Author: Serena <zy16373@gmail.com>
Date:   Wed Dec 5 13:16:46 2018 -0500

    add string functions

commit 1be9b0f65e574652b8e1fae7d07714300431559b
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Wed Nov 28 16:31:12 2018 -0500

    setn / getn tests

commit 60ec52b95da923d2b804db6dcefc317f4b4886f8
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Wed Nov 28 14:38:42 2018 -0500

    some bugs
```

```
commit 9717cfe0b9010e7a434e80ca5a1ee6f9cc9e74cb
Author: Serena <zy16373@gmail.com>
Date:   Wed Nov 28 14:34:37 2018 -0500

    semantic checking for list get and set

commit 7d73443a925715b27d1b848d93c509bee436ae07
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Wed Nov 28 14:26:59 2018 -0500

    setn sast

commit e2540b9104109fb3a8db93533b0affe471ebee90
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Wed Nov 28 13:40:12 2018 -0500

    update sacnner

commit 0abb5da3e6d01ecd10ed4b7393565a0c5721dc88
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Wed Nov 28 00:35:58 2018 -0500

    indentation

commit ffba300e2125703b5f1cff7fe5a96d7391a53536
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Tue Nov 27 23:40:41 2018 -0500

    add test cases

commit a335055f38e38b8d7029671909f53b2b4eff9076
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Tue Nov 27 23:31:54 2018 -0500

    add setn

commit 2922baea54310cac6638c524a46a7c34abf41ff2
Author: Serena <zy16373@gmail.com>
Date:   Tue Nov 27 21:00:34 2018 -0500

    add list access and replace

commit afc6c7c8328b2625fa7664d403dd206b079a97e9
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Mon Nov 12 16:08:32 2018 -0500
```

```
    modify indents

commit 429d2897d2e7f2e209aa29a02491fae33bca6ea8
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Mon Nov 12 01:46:02 2018 -0500

    list staff

commit 2e5c8eb4bcf543ec9371300391ce25d6654636b9
Merge: 296e062 472e534
Author: guhanzhou163 <496495139@qq.com>
Date:   Mon Nov 12 01:00:40 2018 -0500

    Merge branch 'master' of https://github.com/XuhengLi/IRIs

commit 296e06292adec7f72aafa70270382aa508c18846
Author: guhanzhou163 <496495139@qq.com>
Date:   Mon Nov 12 01:00:18 2018 -0500

    test.sh

commit 472e534deeb627b46d1e60b19578fe28bf8e24b5
Merge: 7368390 0081f7c
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Mon Nov 12 00:12:57 2018 -0500

    merge

commit 0081f7c06c2727504382232fddaa3b92546083b7
Author: guhanzhou163 <496495139@qq.com>
Date:   Mon Nov 12 00:08:30 2018 -0500

    codegen

commit 7368390fff51abaa1b43df9b02b1bb09c55fbbbd
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Nov 11 22:20:38 2018 -0500

    add "debug" make option

commit ad6653f11136362a43e77fd34dfd052a2904c550
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Nov 11 18:50:45 2018 -0500

    fix bugs of sast and semant
```

```
commit 6fc855a5328b232c1ab6a81857579e2610f732b9
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sat Nov 10 12:13:45 2018 -0500

    remove .output

commit b718ccdc696fe11a03773a56ce829acda0b685cc
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sat Nov 10 12:12:46 2018 -0500

    modify parser

commit ec450df8c5263006de32e26c987cd5dbd4bcf99f
Author: Serena <zy16373@gmail.com>
Date:   Sat Nov 10 10:34:11 2018 -0500

    add string literal in sast

commit b6874c78f236af990d988a39d7e77c8f23b9094f
Author: Serena <zy16373@gmail.com>
Date:   Sat Nov 10 10:29:22 2018 -0500

    init sementice checking and sast

commit 3065562c181cb6fb893f3200be49c8b451058c8e
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Fri Nov 2 15:37:27 2018 -0400

    fix bug in scanner

commit 01983c44109ff65820de1bb3c3646ba7d19f9e0d
Merge: 8369521 9d0a241
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Fri Nov 2 14:27:05 2018 -0400

    resolve conflict

commit 83695211dcc70d48a06e87083a9c456a1a8f353a
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Fri Nov 2 14:24:27 2018 -0400

    clean files

commit c49bc913c827277825fc1ccaacb661cf94931211
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Fri Nov 2 14:19:46 2018 -0400
```

```
    first version of interpreter

commit 9d0a241fd769533e75392513c746aeb1fd827407
Author: Serena <zy16373@gmail.com>
Date:   Thu Nov 1 18:16:26 2018 -0400

    updated parser during compilation

commit fa8f241247e955c505d81556de9ea2795d5d25a9
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Thu Nov 1 17:48:36 2018 -0400

    update pretty-printing

commit 99d20c464482c265579d7051c24ab38ff297aef1
Author: Serena <zy16373@gmail.com>
Date:   Thu Nov 1 17:27:10 2018 -0400

    updated scanner paser and ast

commit 51bdd01ee4653320e43df72ecfc7fee68c993599
Author: Serena <zy16373@gmail.com>
Date:   Thu Nov 1 17:17:07 2018 -0400

    init iris

commit 28487548424e167daff2f430774b7530d11573ab
Author: Serena <zy16373@gmail.com>
Date:   Thu Nov 1 17:10:06 2018 -0400

    add semant

commit c7ec7b895e501c13ba4e91c14637d551f194b4b8
Merge: d9d9413 a4b24c8
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Thu Nov 1 17:04:57 2018 -0400

    Merge branch 'master' of https://github.com/XuhengLi/IRIs

commit d9d9413c76561502f778381edee21a0f4630293d
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Thu Nov 1 17:04:45 2018 -0400

    modify parser
```

```
commit a4b24c8cea284587e9aab2617e3577cb09b3efbb
Author: Serena <zy16373@gmail.com>
Date:   Wed Oct 31 19:03:15 2018 -0400

    add test case

commit 50c6af42e2a57d73f20cd7a465a5bb3c05acfecd
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Wed Oct 31 11:25:56 2018 -0400

    partly finished ast

commit 6cc281bdb0cb5d8736ddff8bf1e845d6619c3203
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Mon Oct 15 21:09:41 2018 -0400

    yacc parser

commit 6111717eaaeef768f09e2cfa420ab2787adc033d
Author: Serena <zy16373@gmail.com>
Date:   Sun Oct 14 21:24:35 2018 -0400

    modified parser

commit 4052e1a6819530476827ec7bc7d2a538d61c91fe
Author: Serena <zy16373@gmail.com>
Date:   Sun Oct 14 20:08:14 2018 -0400

    init parser

commit 9d0dc0c6655086c7865a615c04c3b524f387cc7f
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Oct 14 14:23:45 2018 -0400

    resolve symbol conflict

commit 259aa3c916f4b45a00c97fdd3793126026e2e0f5
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Oct 14 14:12:38 2018 -0400

    move scanner.mll

commit aed4ac602967037747d1ed3f4061a0dbaa385437
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Oct 14 14:11:41 2018 -0400
```

```
    init version of scanner.ml

commit 4a9b2d07176d9510be8805c522cc6c0e912f9b54
Merge: edd910a 7f79c8f
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Oct 14 14:09:42 2018 -0400

    Merge branch 'master' of https://github.com/XuhengLi/IRIs

commit edd910afde69e1d8850b9caaaa9d824b14c094b8
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Sun Oct 14 14:08:30 2018 -0400

    scanner

commit 7f79c8fee66486bf9195ac42894a7d1186de9858
Author: Serena <zy16373@gmail.com>
Date:   Sun Oct 14 11:58:05 2018 -0400

    init workspace for parser and scanner

commit 795784256ef27fe222a34280a4921b1f494d95c2
Author: XuhengLi <42981800+XuhengLi@users.noreply.github.com>
Date:   Wed Sep 19 19:40:04 2018 -0400

    update readme

commit 5d241867e7f926faf996e6f198014da844033d7c
Author: Serena TANG <zy16373@gmail.com>
Date:   Wed Sep 19 19:20:48 2018 -0400

    Update readme.md

commit aafdb545b19e71a17904adeedffb27eacf1fbde4
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Wed Sep 19 15:28:26 2018 -0400

    update readme

commit 3d20e91d5e888aa1fbbe24e0b37fa23097ae754a
Author: Serena TANG <zy16373@gmail.com>
Date:   Tue Sep 18 22:42:00 2018 -0400

    Update readme.md

    typo and gramma correction
```

```
commit e0a4107672051b76205b35e1b1f3900bdd7e21d5
Author: Xuheng Li <xl2784@columbia.edu>
Date:   Tue Sep 18 21:41:14 2018 -0400

    First commit with a readme file
```

## 4 Architectural Design

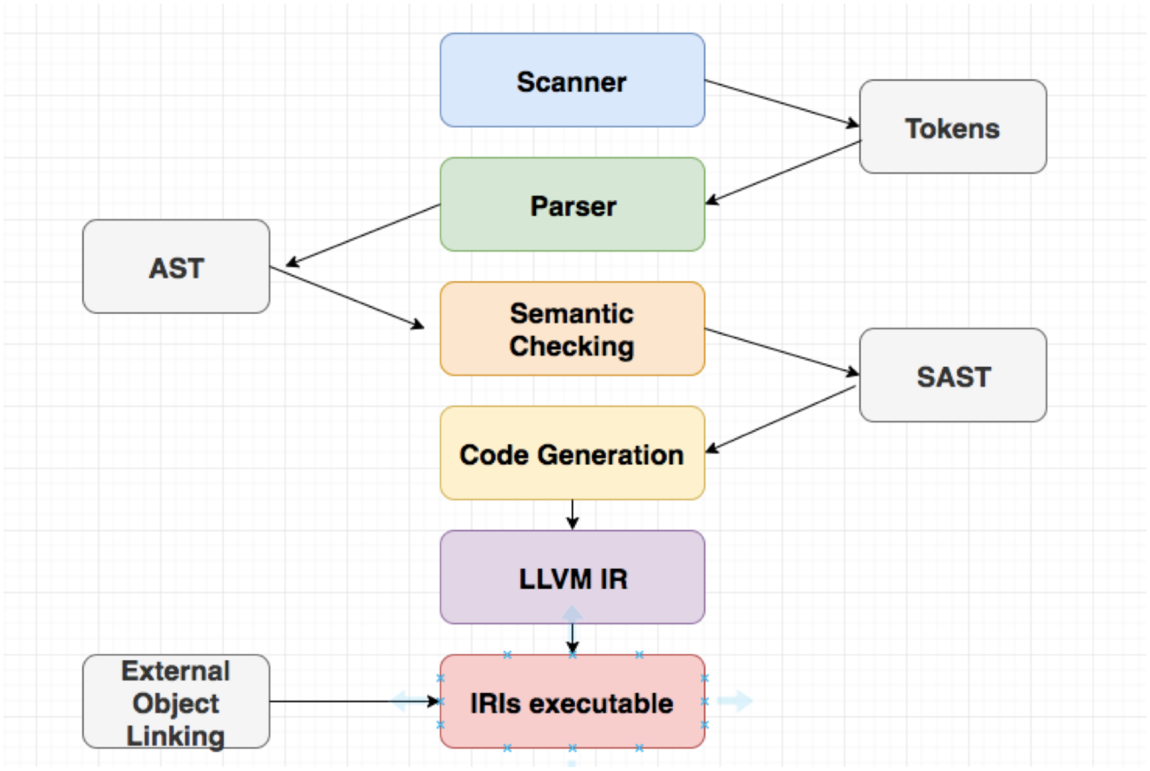


Figure 3: Project Architecture

Figure 3 shows our overall project architecture. The scanner is the first step our of compiler, when translate the read-in program text as a sequence of IRIs tokens.

**scanner.ml** scanner.mll is the top level program of our compiler. It is a ocamllex program and will generate the scanner.ml by ocamllex. It will take the source code of IRIs file and output the tokenized code.

**parser.mly** parser.mly is the ocamlyacc file for parsing. It will generate the parser.ml which will take the tokenized code produced by the scanner and output the ast of the source code.

**ast.ml** ast.ml is the definition of ocaml type of IRIs ast with a set of pretty printing functions for debug and output the generate ast result.

**semant.ml** semant.ml is the semantic checking program for IRIs. It takes the ast generated by the parser and produce the sast.

**sast.ml** sast.ml is the semantic-checked ast definition of IRIs with a set of pretty printing functions.

**codegen.ml** codegen.ml is the backend of the IRIs complier. It takes the sast and using Ocaml-LLVM library to generate the target LLVM-IR.

**iris.ml** iris.ml is the main file of the IRIs compiler which will call each of the above files to generate the LLVM-IR code for IRIs.

**lib/** The lib directory contains the external C programs for IRIs.

## 5 Test Plan

### 5.1 Representative Programs

#### 5.1.1 Calculate_Tips

**IRIs Source Code** A program to calculate tip based on user input bill.

```
int main()
    int i, n, id
    float bill
    float[] tip_rate
    0
    |
    i

    [0.12, 0.15, 0.18, 0.20]
    |
    tip_rate
    length(tip_rate)
    |
    n
    "Please input your bill:\n"
    |
    inputfloat
    |
    bill
    while(i<n)
        1
        |
        +i
        |
        printi
        ": "
        |
        print
        tip_rate[i]
        |
        printf
        "\n"
        |
        print
        1
        |
        +i
        |
        i
    end
    "Choose the tip rate:\n"
    |
    inputint
    |
    id
    "Your tip is: "
    |
    print
    tip_rate[id]
    |
    *bill
    |
    printf
    "\n"
    |
    print
    return 0
Siri
```

**LLVM IR** The generated LLVM IR for calc_tips is as bellow

```
; ModuleID = 'IRIs'
source_filename = "IRIs"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@str = private unnamed_addr constant [25 x i8] c"Please input your bill:\0A\00"
@str.2 = private unnamed_addr constant [3 x i8] c": \00"
@str.3 = private unnamed_addr constant [2 x i8] c"\0A\00"
@str.4 = private unnamed_addr constant [22 x i8] c"Choose the tip rate:\0A\00"
@str.5 = private unnamed_addr constant [14 x i8] c"Your tip is: \00"
@str.6 = private unnamed_addr constant [2 x i8] c"\0A\00"

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

declare i32 @printbig(i32)

declare i32 @inputint(i8*)

declare i8* @inputstring(i8*)

declare double @inputfloat(i8*)

declare i8* @inputfile(i8*, i32)

declare i32 @cmd(i8*)

declare i32 @inputgui(i32)

declare i32 @sendmail(i8*, i8*)

declare i32 @strlen(i8*)

declare i32 @strcmp(i8*, i8*)

declare i8* @strcat(i8*, i8*)

declare i8* @strcpy(i8*, i8*)

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

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

declare i32 @getn_int(i8*, i32)
declare double @getn_double(i8*, i32)

declare void @setn_int(i8*, i32, i32)

declare void @setn_double(i8*, i32, double)

declare i8* @sassign(i8*, i8*)

declare i32 @length(i8*)

declare i8* @calloc(i32, i32)

declare i8* @free(i8*)

define i32 @main() {
entry:
  %id = alloca i32
  store i32 0, i32* %id
  %n = alloca i32
  store i32 0, i32* %n
  %i = alloca i32
  store i32 0, i32* %i
  %bill = alloca double
  store double 0.000000e+00, double* %bill
  %tip_rate = alloca i8*
  store i8* null, i8** %tip_rate
  store i32 0, i32* %i
  %append_double = call i8* (i8*, i32, ...) @append_double(i8* null, i32 4,
      double 1.200000e-01, double 1.500000e-01, double 1.800000e-01, double
      2.000000e-01)
  store i8* %append_double, i8** %tip_rate
  %tip_rate1 = load i8*, i8** %tip_rate
  %length = call i32 @length(i8* %tip_rate1)
  store i32 %length, i32* %n
  %inputfloat = call double @inputfloat(i8* getelementptr inbounds ([25 x i8],
      [25 x i8]* @str, i32 0, i32 0))
  store double %inputfloat, double* %bill
  br label %while

while:                                            ; preds = %while_body, %entry
  %i10 = load i32, i32* %i
  %n11 = load i32, i32* %n
  %tmp12 = icmp slt i32 %i10, %n11
  br i1 %tmp12, label %while_body, label %merge

while_body:                                       ; preds = %while
```

```
  %i2 = load i32, i32* %i
  %tmp = add i32 1, %i2
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4
      x i8]* @fmt, i32 0, i32 0), i32 %tmp)
  %printf3 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3
      x i8]* @str.2, i32 0, i32 0))
  %i4 = load i32, i32* %i
  %tip_rate5 = load i8*, i8** %tip_rate
  %get_double = call double @getn_double(i8* %tip_rate5, i32 %i4)
  %printf6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4
      x i8]* @fmt.1, i32 0, i32 0), double %get_double)
  %printf7 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2
      x i8]* @str.3, i32 0, i32 0))
  %i8 = load i32, i32* %i
  %tmp9 = add i32 1, %i8
  store i32 %tmp9, i32* %i
  br label %while

merge:                                            ; preds = %while
  %inputint = call i32 @inputint(i8* getelementptr inbounds ([22 x i8], [22 x
      i8]* @str.4, i32 0, i32 0))
  store i32 %inputint, i32* %id
  %printf13 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([14 x i8],
      [14 x i8]* @str.5, i32 0, i32 0))
  %id14 = load i32, i32* %id
  %tip_rate15 = load i8*, i8** %tip_rate
  %get_double16 = call double @getn_double(i8* %tip_rate15, i32 %id14)
  %bill17 = load double, double* %bill
  %tmp18 = fmul double %get_double16, %bill17
  %printf19 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8],
      [4 x i8]* @fmt.1, i32 0, i32 0), double %tmp18)
  %printf20 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8],
      [2 x i8]* @str.6, i32 0, i32 0))
  ret i32 0
}
```

### 5.1.2 Binary_Search

**IRIs Source Code** An IRIs program takes a target number and a sorted list, and returns the index of the target number if found in the list, -1 otherwise.

```
int binary_search(int target, int[] nums)
    int n, l, r, mid
    0
    |
    l
    length(nums)
    |
    n
    while(l<r)
        l
        |
        +r
        |
        /2
        |
        mid
        if(target=nums[mid])
            return mid
        fi
        if(target>nums[mid])
            1
            |
            +mid
            |
            l
        else
            mid
            |
            -1
            |
            r
        fi
    end
    return -1
Siri

int main()
    {0, [1, 2, 3, 4, 5, 6]}
    |
    binary_search
    |
   printi
Siri
```

**LLVM IR** Generated LLVM IR for binary search by our compiler.

```
; ModuleID = 'IRIs'
source_filename = "IRIs"

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

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

declare i32 @printbig(i32)

declare i32 @inputint(i8*)

declare i8* @inputstring(i8*)

declare double @inputfloat(i8*)

declare i8* @inputfile(i8*, i32)

declare i32 @cmd(i8*)

declare i32 @inputgui(i32)

declare i32 @sendmail(i8*, i8*)

declare i32 @strlen(i8*)

declare i32 @strcmp(i8*, i8*)

declare i8* @strcat(i8*, i8*)

declare i8* @strcpy(i8*, i8*)

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

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

declare i32 @getn_int(i8*, i32)
declare double @getn_double(i8*, i32)

declare void @setn_int(i8*, i32, i32)

declare void @setn_double(i8*, i32, double)

declare i8* @sassign(i8*, i8*)

declare i32 @length(i8*)

declare i8* @calloc(i32, i32)

declare i8* @free(i8*)

define i32 @main() {
entry:
  %append_int = call i8* (i8*, i32, ...) @append_int(i8* null, i32 6, i32 1, i32
      2, i32 3, i32 4, i32 5, i32 6)
  %binary_search_result = call i32 @binary_search(i32 0, i8* %append_int)
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4
      x i8]* @fmt, i32 0, i32 0), i32 %binary_search_result)
  ret i32 0
}

define i32 @binary_search(i32 %target, i8* %nums) {
entry:
  %target1 = alloca i32
  store i32 %target, i32* %target1
  %nums2 = alloca i8*
  store i8* %nums, i8** %nums2
  %mid = alloca i32
  store i32 0, i32* %mid
  %r = alloca i32
  store i32 0, i32* %r
  %l = alloca i32
  store i32 0, i32* %l
  %n = alloca i32
  store i32 0, i32* %n
  store i32 0, i32* %l
  %nums3 = load i8*, i8** %nums2
  %length = call i32 @length(i8* %nums3)
  store i32 %length, i32* %n
  br label %while

while:                                            ; preds = %merge17, %entry
  %l24 = load i32, i32* %l
  %r25 = load i32, i32* %r
  %tmp26 = icmp slt i32 %l24, %r25
  br i1 %tmp26, label %while_body, label %merge27

while_body:                                       ; preds = %while
  %l4 = load i32, i32* %l
  %r5 = load i32, i32* %r
  %tmp = sdiv i32 %r5, 2
  %tmp6 = add i32 %l4, %tmp
  store i32 %tmp6, i32* %mid
  %target7 = load i32, i32* %target1
  %mid8 = load i32, i32* %mid
  %nums9 = load i8*, i8** %nums2
  %getn_int = call i32 @getn_int(i8* %nums9, i32 %mid8)
  %tmp10 = icmp eq i32 %target7, %getn_int
  br i1 %tmp10, label %then, label %else

merge:                                            ; preds = %else
  %target12 = load i32, i32* %target1
  %mid13 = load i32, i32* %mid
  %nums14 = load i8*, i8** %nums2
  %getn_int15 = call i32 @getn_int(i8* %nums14, i32 %mid13)
  %tmp16 = icmp sgt i32 %target12, %getn_int15
  br i1 %tmp16, label %then18, label %else21

then:                                             ; preds = %while_body
  %mid11 = load i32, i32* %mid
  ret i32 %mid11

else:                                             ; preds = %while_body
  br label %merge

merge17:                                          ; preds = %else21, %then18
  br label %while

then18:                                           ; preds = %merge
  %mid19 = load i32, i32* %mid
  %tmp20 = add i32 1, %mid19
  store i32 %tmp20, i32* %l
  br label %merge17

else21:                                           ; preds = %merge
  %mid22 = load i32, i32* %mid
  %tmp23 = sub i32 %mid22, 1
  store i32 %tmp23, i32* %r
  br label %merge17
```

```
merge27:                                          ; preds = %while
  ret i32 -1
}
```

## 5.2 Testing Phases

### 5.2.1 Unit Testing

We test our lexer, parser and code generation individually throughout the semester. We put different case into the scanner and print them at the parser level to test our lexer. We test our parser by writing IRIs code and verifying it with our AST. We create "hello world", "calculate tips" and "binary search" to test our code generation.

### 5.2.2 Integration Testing

We test the following syntax and lexical conventions in order to check the compiler of our language from lexer to code generation. The generation code is then compiled and we run it to compare the actual results with our expected results.

**Identifiers** Testing the rule of name of variables, which is similar with C language. The name of variables for each type should start with a letter or underscore followed by alphanumeric or underscore. e.g. int _a, float abc123.

**Keywords** Many of our keywords are similar with C language but there are also some differences. For example, we have "fi", "elif", "end". Sufficient testing was constructed to verify that keywords should not be used as the name of variables.

**Control Flow** Control flow was tested in the following way: if, if/else, nested if, while, nested while. Our language doesn't support for loop. "fi" is the end of "if" and "end" is the end of the "while" loop. Both positive and negative cases were tested to verify that lacking one of them is invalid.

**Types** In our language, we support integer, float, bool, string, list and tuple. We test that assigning a value of a specific type to a variable of a different type is incorrect. And we also test to verify that list is mutable and tuple is immutable. For lists and tuples, we test usage of them of different types to verify the correctness.

**Comments** We don't have single-line comment like "//". Our symbol of comments are "/*" and "*/". Comments can be placed randomly in the code.

**Operators** We test addition, subtraction, multiplication and division on different data types. For Boolean, we test the operation of "and", "or" and "not". Operation on different types is also tested, which is invalid, e.g. an integer plus a float.

**Built-in Functions** We have many built-in functions and we test all of them, such as inputint, inputfloat, inputstring, sendmail. We have different kinds of print function to print value of different types, e.g. printi to print integer, ”printf” to print float. We test to verify that variables of corresponding types must be passed as arguments to each of them.

**Variables and Function Declarations** We test multiple variable declarations on one line. We try different parameters for functions and different return types.

**Pipe** This is the biggest feature in our language. We test different methods of assigning values to variables to verify that the former variable/value, the pipe and the latter variable must be placed on different lines. And we test synchronous assignment of values to many variables. We also test continuous passing values to different variables using pipe.

**Pipe with Print** There are two ways to print a value, one is like printi(1), another way is using pipe to pass value to print function so that no parenthesis is needed, just like

```
1
|
printi
```

We test printing values in this way for each type.

### 5.2.3 System Testing

We test more meaningful test cases here. We test our scoping rules and function closures. ”Hello world” was tested firstly. We wrote a program to calculate tip and also wrote a program to implement binary search algorithm to verify that our language can support implementation of some algorithms.

## 5.3 Automation

Our compile script called ”test.sh” can be used to test a specific test case. And the script called ”testall.sh” can take in a directory name and compile all the files in that directory and run them. The script ”testall.sh” not only runs every test but also checks the test name and identify each test if it is intended to succeed or fail.

## 5.4 Test Suites

As we developed the compiler, we wrote tests every time that we implemented a new feature to verify that it works. There are two test suites, one intended to pass, and one intended to fail. All programs that are intended to succeed are named with the prefix ”test_”, and those that are intended to fail are named with the prefix ”fail_”. The former test suite can generate ".out" file that stores the output and the latter suite can generate ".err" file that stores the error information. There are about 80 test cases in total.

We create test suites since we need to test the following features of our language:

- Primitive data assignment and operations: We test single assignment to one variable, synchronous assignments to multiple variables and operations on variables/values of same and different data types
- Control flow: We test "if", "if/else", "while". We test that "if" control flow is invalid without "fi" and "while" loop is invalid without "end". Nested version of them are also tested.
- Lists and tuples: We test to verify that list is immutable and tuple is mutable. And we also test that the elements in lists should be of same data type and for tuple they can be different.
- Function: We check our function declaration, try different data types for parameters and return types of each specific function.
- Pipe: The two variables connected with pipe and pipe itself should be placed in different lines. And we also test that pipe can be used with our print function.

## 5.5 Testing Roles

Pinxi Tai designed the test cases, created the testing infrastructure including automation of tests and reported bugs to the member responsible for the code (Xuheng Li or Shulan Tang), who would in turn find and solve the problems. Hanzhou Gu helped design the negative test cases and test some built-in functions.

# 6 Language Manual

## 6.1 Data type

IRIs is a language with strict variable type so that each variale has one fixed type and differen typs cannot be casted implicitly. Variables have to be declared at the beginning of the function and the scope of the varible is within the function. Variables cannot be used without declaration.

### 6.1.1 Integer

**Declaration** An integer variable is declared with an **int** keyword, and an value can be assigned with a pipe: | operator:

```
int var
6
|
var
```

### 6.1.2 Float

**Declaration** A float variable is declared with an **float** keyword, and an value can be assigned with a pipe: | operator

```
float var
0.6
|
var
```

### 6.1.3 Boolean

**Declaration** A boolean variable is declared with the keyword **bool**. A boolean variable can be a value of true or false.

```
bool var
true
|
var
```

### 6.1.4 String

String is mutable and heap-based. The string variable is a pointer to the heap where the string is stored.

**Declaration** A string variable is declared with the keyword **string**.

```
string var
"hello world"
|
var
```

**Operations** IRIs provide several build-in functions to manipulate the string. They are strlen to get the lenght of the string, strcmp takes 2 strings as arguments and compare them in dictionary order and return 0 if their contents are equal, negative number if the second one is bigger and positive if the first one is bigger, strcat to concatenate 2 strings and save the result in the first string (the first string must have enough space), strcpy to copy the second string to the fisrt one (the first string must have enough space) and calloc to allocate the memory of the given size in bytes and return the address of that space.

### 6.1.5 List

List is a mutable set of the same type of elements on the heap. The type of members of a list can be int, float or string.

**Declaration** To define a list, you can use type[] and the list list literal is wrapped by a pair of brackets and the elementes are separated by commas.

```
int[] li
[1,2,3]
|
li
```

Then li will be a list of integers.

**Operation**

**Append elements to list** One can append a list with the **append** function.

```
int[] li
[1,2,3]
|
li
append(li, 4)
```

Then li will be [1,2,3,4].

**Length** Get the length of a list with **length** function.

```
int[] li
int a
[1,2,3,4,5]
|
li
length(li)
|
a
```

a will equal to 5.

**Element Accessing** Element in the list can be retrieved with **[id]**. The index of list start from 0.

```
int[] li
int ele
[1,2,3,4,5]
|
li

li[2]
|
ele
```

The variable ele should be 3.

### 6.1.6 Tuple(experimental)

A tuple is a set of elements. The type of elements in a tuple can be different, they can be integers, floats, strings, lists and booleans. Tuple is immutable. Once created, it can't be updated.

**Declaration** To declare a varible of tuple, you can use key word tuple and the literal of tuple is warpped by a pair of curly brackets and the elements are seperated by commas.

```
tuple a
{1,2,3,"hello","world"}
```

```
|
a
```

### Operation

**Element Accessing** Element in the tuple can be retrieved with **[id]**. The index of tuple start from 0.

```
tuple t
string ele
{1,2,3,"hello","world"}
|
a

t[4]
|
ele
```

The variable ele should be ”hello”.

## 6.2 Operators

### 6.2.1 Basic Operators

IRIs has 4 basic arithmeitc operators and 2 logic operators. Arithmetic operators can be 2 forms: one is the operator self solely and the other is following a pipe operators.

```
int a, b
1
|
+2
|
a

1+2
|
b
```

Logic operators are && and ||.

```
bool b
true && false
|
b
```

## 6.3 Pipe

In IRIs, pipe stands for the flow a data. The data can flow from the begin end to the terminal end. It has 2 usage. One is for assign and working with the operators.

```
int a, b
1
|
+2
|
a
|
b
```

This code means add one and two and assign them to a, then assign a to b. The other usage for pipe is for pass the argument to a function which can take one argument:

```
1
|
a

a
|
fun1
```

Here fun1 is a function takes one argument and it will take a as its arguments.

## 6.4 Control Flow

### 6.4.1 If-else

The if-else statement is used to express decisions. The syntax of If-else in IRIs language is

```
if (expression)
  (statements)
else
  (statements)
fi
```

The end of if - else is denoted by the keyword fi. Each if-else statement can only have zero or one else. The expression is evaluated; if it is true (expression has a non-zero value), statement will be executed.

**Nested if-else** Because each if-else statement need to be ended with an fi keyword, there is no ambiguity if the else statement is omitted in the inner if-else statement.

### 6.4.2 while loop

While is the only loop implementation in our language. The syntax of while in our language is

```
while (expression)
  (statement)
end
```

The expression is evaluated first; If it is true, statement is executed, and then the expression is re-evaluated. The cycle continues until the expression is False.

## 6.5 Function

In IRIs, every program should have a main function and will start at main function. In IRIs, you can declare a function in such a form:

```
type fun_name((para_list))
(function body)
return (value)
Siri
```

A function declaration starts with type fun_name((para_list)) and end with keyword Siri. For example,

```
int fun(int a)
  int b

    a
    |
    +1
    |
    b

    return b
Siri
```

This function will increment a by 1 and assign its value to b and return b. Any IRIs statement must be used in a function. But IRIs does not support nested function which means you cannot declare a function in a function. In IRIs, each function must have a return value and arguments. When reached to a return keyword, the function will terminate immediately. To call a function, you can just use: fun_name(a,b,c...). For example,

```
fun(5)
|
c
```

c will be 5 after execution. You can call a function inside a function.

## 6.6 Tuple Arguments (experimental)

You can also use a tuple to pass arguments to a function with multiple arguments. This is only an experimental features. For now, you can only pass tuple literals to a function.

```
int foo(int i, string s)
    s
    |
    print
    i
    |
    printi
    return 0
Siri
int main()
    (1, "Hello")
    |
    foo
Siri
```

This snippet of code will print ”Hello1”

## 6.7 Standard Input and Output

### 6.7.1 Input

The input mechanism is to read one string at a time from the standard input, normally the user input. IRIs provide a set of input functions to handle user input. The input family functions can take one string argument as the prompt and wait for user input. There 3 input functions: inputint/inputfloat/inputstring.

```
int i
inputint("int put your int")
|
i
```

### 6.7.2 Output

There are also a set of output functions called print, printi, printf which are for print string, integer and float respectively.

```
string s1
"hello"
|
print

"world"
|
print
```

Then "hello world" will be printed.

**GUI Input (experimental)** GUI input is an experimental function in IRIs. You can use inputgui function to call a gui input interface. But this need complicate configurations.

## 7 Lessons Learned

### 7.1 Shulan Tang

This project is definitely challenging, nauseating but somewhat appealing. The excitement I had when our language finally did something overwhelmed other happiness in the past few months. Our language has new concepts and requires a lot of independent thinking in the design and implementation phases, which pushes us to really understand the working principles of a compiler. As for the teamwork, it is essential to keep communicating and support others on the team. One advice I want to give is to think thoroughly about every design and implementation, and know what you are doing.

### 7.2 Xuheng Li

I learned the functional ways of programming from this class which is a brand new mind-blowing stuff for me. This special way of thinking gives me another way of solving problems.

### 7.3 Hanzhou Gu

What I learn from this project: From this project experience, I feel that writing a good software is not that easy. When we have finished some milestone, we will find that there are still some more interesting and challenging things which should also be done. So the first thing I learned is that, no matter what we are dealing with, we should start as early as possible. Enough time will give us the opportunity to product a better work, not just a finished work although it can also work. I admit that we have truly gained a lot from this project, for both language compiler itself and the team developing. But it is really a pity, at least for me, that I have not enough time to contribute to all components of our project. By now, I have a thorough understanding about the programming language and compiler, and I know how to make our language better. As an system architect, I divided the whole project into several modules. I think this kind of thing should be encouraged in software engineering, especially for the team developing. Otherwise, putting nearly everything in a single file means that all of us should work on that single file, which further means that there is huge possibility that we will tough others codes when we solve the problem of ourselves, bringing more bugs. I have experienced this in part of our development, and I definitely believe that an appropriate developing style can enable us to avoid this kind of problem. We should keep different codes with different functions in different modules, and keep their interface between the modules as consistent as possible. When someone is responsible for one specific module, that person should guarantee that he can use the input from last module appropriately, and generate the right output format for the following module. By doing this, we can keep the wasting time of the team as less as possible. And for each member itself, he should have the ability to separately handle the problem from the module himself. We can make whole bunch of things more easily work in order by doing this. Actually, it is not a good thing to start programming right after you get an idea. A whole project is not just a programming practice. It is user-oriented and domain-specific. We should schedule the structure of our project first and figure out critical feature of our project. We should also get a general idea about what other features we should implement, otherwise in the future, when we want to implement a new idea, maybe we should go through the whole project, from scanner to compiler in the present case. A good start is half done, and we should never forget that. And we dont regret to spend such seemly wasteful time on design our language before implementation. Programming is very interesting, and exciting. I cannot describe the happiness after I write several effective codes, or find out some bugs in my program. During the finals week, we start a lot of time doing this programming project, even during my sleeping. I dont care so much about the results but try my best to finish our well-designed language. I think I should do the right thing as a 23 young man, a futures computer engineer. Unfortunately, I am still not a clever problem-solver some time and need help from other group members, at least by now. But it doesnt matter. I believe I will become one in the future. At last, I thank my teammates, Xuheng Li, Shulan Tang and Pinxi Tai, very much. You are all good programmers. You taught me a lot about programming, and teamwork. Thank you for accompanying me for a whole term at Columbia. I hope we can all be better in the near future!

## 7.4 Pinxi Tai

The functional programming language OCaml is really a challenge for me, but I find it is an interesting language and easy to use. Working with good teammates throughout the whole semester, I learned lessons from both the technical aspects of writing a compiler and working effectively with others in a team. I learned the way to design a programming language and the principles from lexer to code generation. Creating test cases each time we implemented a new feature, debugging and solving shift/reduce conflicts are really important. Communicating with each other often and make sure that each team member is on the same page is also important. In addition, I got more experience about Linux system.

Advice: Consider more details when designing our language at the beginning of the semester. Meeting a lot and regularly to make sure that each team member is on the same page. Make an appropriate project time plan at the beginning. Asking questions to others frequently and understand the code written by others as early as possible. Keep testing along the way and solve problems in time.

## 8 Appendix

### iris.sh

```
#!/bin/sh


./src/iris.native $1> source.llvm
llc source.llvm -o source.s

clang source.s src/lib/*.o -o main
rm source.s source.llvm
./main
```

### readme.md

```
# IRIs

## UNIs
xl2784 <br />
st3174 <br />
hg2498 <br />
pt2508 <br />

## Introduction

Inspired by the powerful macOS tool Alfred, we decide to design a workflow
    description language called IRIs. IRIs not only stands for the flower but
    stands for the reverse of the Apple so-called smart assistant Siri.

IRIs is used to describe and generate a workflow that with the input of the
    user, completes a sequence of tasks. IRIs will mainly focus on the macOS, and
    some of its code can also work on Linux distributions.
```

~~~
## Ingredients

### Basic Structure (Not Syntax Tree)

IRIs is mainly made up of two key structures: the expression and the workflow.

#### Expression
The expression is similar to what other awful or terrific languages which can be
    defined as a single instruction. But it is slightly different that
    expressions in IRIs are connected by a flow symbol (or pipe, whatever) `|`
    and the value of an expression will flow down to the next line if connected
    so that you do not need to use ~~silly~~ temporary variables to transmit
    information and will also make your program subtle and neat.

For example, in C++, if we want to receive an input from a keyboard and add 1 to
    that input, what we needto do is:

```c
int a;
a << stdin;
a = a + 1;
cout << a << endl;
```

But in IRIs, it will be:

```
input()
|
+ 1
|
ShowResult()
```

We have to admit that the C++ way has many advantages, especially in big
    programming projects. However, what most of us really need is a simple way to
    handle an "input -> process -> output" flow. In most circumstances, the
    process could be not much complicated that a "plus 1" operation.

So here we will introduce the second principal structure of our language - the
    workflow.

#### Workflow
A workflow just likes the function part in other imperative languages. It can
    accept an argument, process a sequence of instructions and finally, give an
~~~

```
    output. What is more, a workflow can read an input from the result or the
    output from an expression or just another workflow if they are connected by
    the flow symbol. It just likes the pipe in the Linux shell. Also, a workflow
    can be called recursively to complete the repeat process of other imperative
    languages.

A workflow should start with the keyword ‘HeySiri‘ followed by a workflow name
    and end with the keyword ‘GoodbyeSiri‘.

### Data Types

- **num**: Integer or floating number
- **text**: just likes the string
- **list**: a list of option that can be chosen
- **dictionary**: a set of key-value pairs

### Key Words
- ‘HeySiri‘: <br /> Define a workflow
- ‘GoodbyeSiri‘: <br /> Mark the end of a workflow
- ‘repeat‘: <br /> Mark the start of a loop
- ‘RepeatIndex‘: <br /> A repeat counter. It will count the number of times a
    loop runs. If there is no appendix number behind, it stands for the innermost
    loop; if there is a number appended, such as RepeatIndex2, it stands for the
    second layer of the nested loop.
- ‘end‘: <br /> Mark the end of a loop
- ‘if/elif/fi‘: <br /> Those set of keywords are used to perform a condition
    expression just as a Bash script does.
- ‘$‘: <br /> A placeholder for output. Explain below.
### Basic Workflows
- ‘input()‘: <br /> Just like the input function of other languages, it accepts
    the input from a user and will pass it to others by the flow symbol.
- ‘list[]‘:<br /> The ‘list[]‘ keyword defines a basic data type in IRIs. This
    data type is an indispensable part of the IRIs. A list is not only an array
    like data type that stores a set of data but also connect with the user
    interaction tightly.
- ‘choose{}‘: <br /> It likes the switch-case expression of other languages, but
    slightly different. In other languages, the switch-case expression likes
    another form of an if expression (although the low-level implementation is
    different, I know). In IRIs, ‘choose{}‘ is more of an interactive
    instruction. It accepts an input of a list, and prompts the items from the
    list to the user, waits user choosing an item (or an option), then output
    flow of data.
- ‘ShowResult()‘: <br /> Like the print function of other languages. It accepts
    input from a flow, too. And inside the parenthesis, it uses the keyword ‘$‘
    as a placeholder of the input.
```

~~~~
### Sample Code
Caclculate the tip
‘‘‘
HeySiri CalcTip()

input()
|
bill

list[0.12, 0.15, 0.18, 0.20]
|
choose{
    1:
        0.12
    2:
        0.15
    3:
        0.18
    4:
        0.20
}
|
rate
|
* bill
|
tip
|
+ bill
|
total

ShowResult("Your tip is ", tip, " and your total is ", total)

GoodbyeSiri
‘‘‘
**Explanation:**
At first, the program starts with a ‘HeySiri‘ keywords to define a workflow.
    There is no concept of the main function as the entry of the program. The
    program will begin at the very first line.

Then, there is an ‘input()‘ workflow connecting to a variable ‘bill‘ which means
    "read the users input and put it in the variable."

Then, there is a list within the ‘[]‘ symbol. This is one of the basic data type
    called list, and it will flow into a ‘choose expression‘ with a pair of ‘{}‘.
~~~~

These few lines will prompt the list and ask the user to choose one of the items in the list, then match the value within the ‘{}‘ and pass it to the next line. Here, the number ‘1/2/3/4‘ stands for the index of the items in the list, and the number behind the ‘:‘ stands for the result of the ‘choose expression‘.

Notice that the result of a ‘choose expression‘ does not have to identify to the input list. For example, the input list could be ‘["large", "medium", "little"]‘ and the result of the ‘choose expression‘ could be ‘10, 5 and 1‘.

After the choice, there is a long flow a calculate. In IRIs, there is no ‘=‘ expression, the operator ‘=‘ only means *equal to*. All of the assignments will be done by using the flow symbol ‘|‘. Line 19 to 28 of code above means - "assign the result to variable ‘rate‘, and then multiply it by the variable ‘bill‘ and assign the result to variable ‘tip‘(but the value of the rate will not change), then add the value of ‘tip‘ and ‘bill‘, which we get from the input, and assign the result to variable ‘total‘. Similarly, the value of ‘tip‘ will not change."

Then, the ‘ShowResult()‘ will show the content inside the parenthesis and the workflow will terminate after the ‘Goodbye Siri‘ keyword.

GCD
‘‘‘
```
HeySiri GCD()

intput()
|
a

input()
|
b

if (a != b)
    if (a > b)
        a - b
        |
        a
        GCD(a, b)
    elif (a < b)
        b - a
        |
        b
        GCD(a, b)
    fi
```

```
fi

GoodbyeSiri
‘‘‘

99-Bottles-of-Beer
‘‘‘
HeySiri _99_Bottles_of_Beer()

input()
|
repeat

99 - RepeatIndex + 1
|
left
|
ShowResult($, "bottles of beer on the wall, ", $, " bottles of beer.")

left - 1
|
ShowResult("Take one down and pass it around, ", $, " bottles of beer on the
    wall.")

end

ShowResult("No more bottles of beer on the wall, no more bottles of beer.")
ShowResult("Go to the store and buy some more, 99 bottles of beer on the wall.")

GoodbyeSiri
‘‘‘

## TODOs:

Add clarifications about naming scope rules
```

### testall.sh

```
#!/bin/sh

# Regression testing script for MicroC
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test
# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc"

# Path to the C compiler
CC="cc"

# Path to the iris compiler. Usually "./iris.native"
# Try "_build/iris.native" if ocamlbuild was unable to create a symbolic link.
IRIS="./src/iris.native"
#IRIS="_build/iris.native"

# Set time limit for all operations
ulimit -t 30

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

keep=0

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

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

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

```
   SignalError "$1 differs"
   echo "FAILED $1 differs from $2" 1>&2
    }
}

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

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

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

    echo -n "$basename..."

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

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

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

    echo -n "$basename..."

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

    generatedfiles=""

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

    # Report the status and clean up the generated files

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

```
}

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

shift `expr $OPTIND - 1`

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

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

if [ $# -ge 1 ]
then
    files=$@
else
    files="tests/test_*.ir tests/fail_*.ir"
fi

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

```
exit $globalerror
```

## ./src/Makefile

```
.PHONY : all
all : iris.native lib/inputint.o lib/inputfile.o lib/inputstring.o lib/cmd.o
    lib/inputfloat.o lib/liblist.o lib/libstr.o

.PHONY : GUI
GUI : iris.native lib/inputint.o lib/inputfile.o lib/inputstring.o lib/cmd.o
    lib/inputfloat.o lib/inputgui.o

iris.native:
   opam config exec -- \
   ocamlbuild -use-ocamlfind -pkg llvm -package llvm.analysis iris.native

# inputint: inputint.c
#   clang -o inputint inputint.c

# inputstring: inputstring.c
#   clang -o inputstring inputstring.c

# inputfile: inputfile.c
#   clang -o inputfile inputfile.c

# cmd: cmd.c
#   clang -o cmd cmd.c

# inputgui.o: inputgui.c
#   clang ‘pkg-config --cflags gtk+-3.0‘ -c inputgui.c ‘pkg-config --libs
    gtk+-3.0‘=


# "make clean" removes all generated files

.PHONY : clean
clean :
   ocamlbuild -clean
   rm -rf testall.log ocamlllvm *.diff parser.ml parser.mli
   rm -rf *.o

.PHONY : debug
debug:
   opam config exec -- \
   ocamlbuild -use-ocamlfind -pkg llvm -package llvm.analysis iris.d.byte
```

```
.PHONY : prettyp
parser:
   ocamllex scanner.mll
   ocamlyacc parser.mly
   ocamlc -c ast.mli
   ocamlc -c parser.mli
   ocamlc -c scanner.ml
```

## ./src/ast.ml

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

type uop = Neg | Not

type typ = Int | Bool | Float | String | List of typ | Tuple of typ list

type bind = typ * string

type expr =
    Lint of int
  | Lfloat of string
  | Lbool of bool
  | Lstring of string
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Llist of expr list
  | Ltuple of expr list
  | Getn of string * expr
  | Length of expr

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | While of expr * stmt
  (* list *)
  | Setn of string * expr * expr

type func_decl = {
    typ : typ;
    fname : string;
    formals : bind list;
    locals : bind list;
    body : stmt list;
  }

type program = bind list * func_decl list

(* Pretty-printing functions *)

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

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

let rec string_of_expr = function
    Lint(l) -> string_of_int l
  | Lfloat(l) -> l
  | Lbool(true) -> "true"
  | Lbool(false) -> "false"
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> string_of_expr e ^ " | " ^ v
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  (* | Noexpr -> "" *)
  | Lstring(s) -> s
  | Llist(l) -> "[" ^ String.concat ", " (List.map string_of_expr l) ^ "]"
  | Ltuple(l) -> "(" ^ String.concat ", " (List.map string_of_expr l) ^ ")"
  | Getn(e1,e2) -> e1 ^ "[" ^ string_of_expr e2 ^ "]"
  | Length(s) -> "length of " ^ string_of_expr s

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

let rec string_of_typ = function
    Int -> "int"
  | Bool -> "bool"
  | Float -> "float"
  | String -> "string"
  | List(t) -> string_of_typ t ^ "[]"
  | Tuple(l) -> "tuple(" ^ String.concat ", " (List.map string_of_typ l) ^ ")"

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

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "Siri\n"

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

## ./src/codegen.ml

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

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

http://llvm.org/docs/tutorial/index.html
Detailed documentation on the OCaml LLVM library:

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

*)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

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

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

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

  let str_t     = L.pointer_type i8_t
  and list_t    = L.pointer_type i8_t
  and tuple_t    = L.pointer_type i8_t
  in

  (* Return the LLVM type for a MicroC type *)
  let ltype_of_typ = function
      A.Int  -> i32_t
    (* TODO: Modify Bool type *)
    | A.Bool -> i8_t
    | A.Float -> float_t
    | A.String -> str_t
    (* TODO: Add list type *)
    | A.List l -> list_t
    | A.Tuple (_) -> tuple_t
```

```
in

(* Create a map of global variables after creating each *)
let global_vars : L.llvalue StringMap.t =
  let global_var m (t, n) =
    let init = match t with
        A.Float -> L.const_float (ltype_of_typ t) 0.0
      | _ -> L.const_int (ltype_of_typ t) 0
    in StringMap.add n (L.define_global n init the_module) m
  in List.fold_left global_var StringMap.empty globals in

let printf_t : L.lltype =
    L.var_arg_function_type i32_t [| L.pointer_type i8_t |]
in
let printf_func : L.llvalue =
    L.declare_function "printf" printf_t the_module
in
let printb_func : L.llvalue =
    L.declare_function "printf" printf_t the_module
in
let printbig_t : L.lltype =
    L.function_type i32_t [| i32_t |]
in
let printbig_func : L.llvalue =
    L.declare_function "printbig" printbig_t the_module
in
let inputint_t : L.lltype =
    L.function_type i32_t [| str_t |]
in
let inputint_func : L.llvalue =
    L.declare_function "inputint" inputint_t the_module
in
let inputstring_t : L.lltype =
    L.function_type str_t [| str_t |]
in
let inputstring_func : L.llvalue =
    L.declare_function "inputstring" inputstring_t the_module
in
let inputfloat_t : L.lltype =
    L.function_type float_t [| str_t |]
in
let inputfloat_func : L.llvalue =
    L.declare_function "inputfloat" inputfloat_t the_module
in
let inputfile_t : L.lltype =
    L.function_type str_t [| str_t;i32_t|]
```

```
in
let inputfile_func : L.llvalue =
   L.declare_function "inputfile" inputfile_t the_module
in
let cmd_t : L.lltype =
   L.function_type i32_t [| str_t|]
in
let cmd_func : L.llvalue =
   L.declare_function "cmd" cmd_t the_module
in
let inputgui_t : L.lltype =
   L.function_type i32_t [| i32_t |]
in
let inputgui_func : L.llvalue =
   L.declare_function "inputgui" inputgui_t the_module
in
let sendmail_t : L.lltype =
   L.function_type i32_t [|str_t;str_t|]
in
let sendmail_func : L.llvalue =
   L.declare_function "sendmail" sendmail_t the_module
in
(* Declare the built-in strlen() function *)
let strlen_t : L.lltype =
   L.function_type i32_t [| str_t |]
in
let strlen_func : L.llvalue =
   L.declare_function "strlen" strlen_t the_module
in

(* Declare the built-in strcmp() function *)
let strcmp_t : L.lltype =
   L.function_type i32_t [| str_t; str_t|]
in
let strcmp_func : L.llvalue =
   L.declare_function "strcmp" strcmp_t the_module
in
(* Declare the built-in strcat() function *)
let strcat_t : L.lltype =
   L.function_type str_t [| str_t; str_t|]
in

let strcat_func : L.llvalue =
   L.declare_function "strcat" strcat_t the_module
in
```

```
(* Declare the built-in strcpy() function *)
let strcpy_t : L.lltype =
   L.function_type str_t [| str_t; str_t|]
in
let strcpy_func : L.llvalue =
   L.declare_function "strcpy" strcpy_t the_module
in
(* Declare the list_append() functions *)
let append_double_t : L.lltype =
   L.var_arg_function_type list_t [| list_t; i32_t |]
in
let append_double_func : L.llvalue =
   L.declare_function "append_double" append_double_t the_module
in
let append_int_t : L.lltype =
   L.var_arg_function_type list_t [| list_t; i32_t |]
in
let append_int_func : L.llvalue =
   L.declare_function "append_int" append_int_t the_module
in
let getn_int_t : L.lltype =
   L.function_type i32_t [| list_t; i32_t |]
in
let getn_int_func : L.llvalue =
   L.declare_function "getn_int" getn_int_t the_module
in
let getn_double_t : L.lltype =
   L.function_type float_t [| list_t; i32_t |]
in
let getn_double_func : L.llvalue =
   L.declare_function "getn_double" getn_double_t the_module
in
let setn_int_t : L.lltype =
   L.function_type void_t [| list_t; i32_t; i32_t|]
in
let setn_int_func : L.llvalue =
   L.declare_function "setn_int" setn_int_t the_module
in
let setn_double_t : L.lltype =
   L.function_type void_t [| list_t; i32_t; float_t|]
in
let setn_double_func : L.llvalue =
   L.declare_function "setn_double" setn_double_t the_module
in
let sassign_t : L.lltype =
   L.function_type str_t [| str_t; str_t;|]
```

```
in
let sassign_func : L.llvalue =
    L.declare_function "sassign" sassign_t the_module
in
let length_t : L.lltype =
    L.function_type i32_t [| list_t |]
in
let length_func : L.llvalue =
    L.declare_function "length" length_t the_module
in
(* Declare heap storage function *)
let calloc_t = L.function_type str_t [| i32_t ; i32_t|] in
let calloc_func = L.declare_function "calloc" calloc_t the_module in

(* Declare free from heap *)
let free_t = L.function_type str_t [| str_t |] in
let free_func = L.declare_function "free" free_t the_module in
(* Define each function (arguments and return type) so we can
   call it even before we've created its body *)
let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
  let function_decl m fdecl =
    let name = fdecl.sfname
    and formal_types =
        Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.sformals)
    in
    let ftype = L.function_type (ltype_of_typ fdecl.styp) formal_types
    in StringMap.add name (L.define_function name ftype the_module, fdecl) m
  in List.fold_left function_decl StringMap.empty functions
in

(* Fill in the body of the given function *)
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.sfname function_decls
  in
  let builder = L.builder_at_end context (L.entry_block the_function)
  in

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

    (* Allocate space for any locally declared variables and add the
     * resulting registers to our map *)
      and add_local m (t, n) =
        let local_var = L.build_alloca (ltype_of_typ t) n builder
        in (match t with
            Int -> ignore (L.build_store (L.const_int i32_t 0) local_var
                builder); StringMap.add n local_var m
          | Float -> ignore (L.build_store (L.const_float float_t 0.0) local_var
              builder); StringMap.add n local_var m
          | String
          | List(_) -> ignore (L.build_store (L.const_pointer_null
              (L.pointer_type i8_t)) local_var builder); StringMap.add n
              local_var m
          | Bool -> ignore (L.build_store (L.const_int i8_t 0) local_var
              builder); StringMap.add n local_var m
          | Tuple _ -> ignore (L.build_store (L.const_pointer_null
              (L.pointer_type i8_t)) local_var builder); StringMap.add n
              local_var m)
    in

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

  (* Return the value for a variable or formal argument.
     Check local names first, then global names *)
  let lookup n = try StringMap.find n local_vars
                with Not_found -> StringMap.find n global_vars
  in

  let build_string e builder =
    let str = L.build_global_stringptr e "str" builder
    in
    let null = L.const_int i32_t 0
    in L.build_in_bounds_gep str [| null |] "str" builder
  in
  (* Construct code for an expression; return its value *)
  let rec expr builder ((sx, e) : sexpr) = match e with
      SLint i -> L.const_int i32_t i
    | SLbool b -> L.const_int i1_t (if b then 1 else 0)
    | SLfloat l -> L.const_float_of_string float_t l
    | SId s      -> L.build_load (lookup s) s builder
    | SLstring s -> build_string s builder
    (* TODO: List support *)
    | SLlist l -> let list_builder l = Array.of_list (List.map (expr builder) l)
                 in
                 (match sx with
                 List(Int) -> L.build_call append_int_func (Array.append
                     [|L.const_pointer_null list_t; L.const_int i32_t
                     (List.length l);|] (list_builder l)) "append_int" builder
                 | List(Float) -> L.build_call append_double_func (Array.append
                     [|L.const_pointer_null list_t; L.const_int i32_t
                     (List.length l);|] (list_builder l)) "append_double" builder
                 | _ -> raise (Failure "List type error"))
    | SLtuple l -> let t = L.const_struct context (Array.of_list (List.map
        (expr builder) l))
                 in
                 let null = L.const_int i32_t 0
                 in L.build_in_bounds_gep t [| null |] "tpl" builder
    | SAssign (s, e) -> let n = try lookup s
                              with Not_found ->

                              let (fdef, fdecl) = StringMap.find s function_decls
                              in
                              let llargs = List.rev (List.map (expr builder)
                                  (List.rev [e]))
                              in
                              let result = (match fdecl.styp with
                                      | _ -> s ^ "_result")
                              in L.build_call fdef (Array.of_list llargs) result
                                  builder

                     in
                    (match e with
                      (String, SLstring(l)) -> let e' = L.build_call
                          sassign_func [|L.build_load n s builder; expr builder
                          e;|] "sassign" builder
                               in ignore(L.build_store e' n builder); e'
                      | _ -> let e' = expr builder e
                             in ignore(L.build_store e' n builder); e')
    | SGetn(s, e) -> (match sx with
                    Int -> L.build_call getn_int_func [|L.build_load (lookup s)
                        s builder; expr builder e;|] "getn_int" builder
                    | Float -> L.build_call getn_double_func [|L.build_load
                        (lookup s) s builder; expr builder e;|] "get_double"
                        builder
                    | _ -> raise (Failure "List type error"))
    | SBinop ((A.Float,_ ) as e1, op, e2) ->
      let e1' = expr builder e1
      and e2' = expr builder e2
      in
      (match op with
        A.Add    -> L.build_fadd
      | A.Sub    -> L.build_fsub
      | A.Mult   -> L.build_fmul
      | A.Div    -> L.build_fdiv
      | A.Equal  -> L.build_fcmp L.Fcmp.Oeq
      | A.Neq    -> L.build_fcmp L.Fcmp.One
      | A.Less   -> L.build_fcmp L.Fcmp.Olt
      | A.Leq    -> L.build_fcmp L.Fcmp.Ole
      | A.Greater -> L.build_fcmp L.Fcmp.Ogt
      | A.Geq    -> L.build_fcmp L.Fcmp.Oge
      | A.And
      | A.Or ->
        raise (Failure "internal error: semant should have rejected and/or on
            float")
        ) e1' e2' "tmp" builder
      | SBinop (e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2
        in
        (match op with
          A.Add    -> L.build_add
        | A.Sub    -> L.build_sub
        | A.Mult   -> L.build_mul
        | A.Div    -> L.build_sdiv
        | A.And    -> L.build_and
        | A.Or     -> L.build_or
        | A.Equal  -> L.build_icmp L.Icmp.Eq
        | A.Neq    -> L.build_icmp L.Icmp.Ne
        | A.Less   -> L.build_icmp L.Icmp.Slt
        | A.Leq    -> L.build_icmp L.Icmp.Sle
        | A.Greater -> L.build_icmp L.Icmp.Sgt
        | A.Geq    -> L.build_icmp L.Icmp.Sge
        ) e1' e2' "tmp" builder
        | SUnop(op, ((t, _) as e)) ->
          let e' = expr builder e
          in
          (match op with
           A.Neg when t = A.Float -> L.build_fneg
          | A.Neg                 -> L.build_neg
          | A.Not                 -> L.build_not) e' "tmp" builder
        | SLength(e) ->
            L.build_call length_func [| (expr builder e) |] "length" builder
          | SCall ("print", [e]) ->
            L.build_call printf_func [|(expr builder e)|] "printf" builder
          | SCall ("printb", [e]) ->
            L.build_call printb_func [|int_format_str; (expr builder e)|]
            "printf" builder
          | SCall ("printi", [e]) ->
            L.build_call printf_func [|int_format_str ; (expr builder e)|]
            "printf" builder
          | SCall ("printbig", [e]) ->
            L.build_call printbig_func [| (expr builder e) |]
            "printbig" builder
          | SCall ("printf", [e]) ->
            L.build_call printf_func [| float_format_str ; (expr builder e) |]
            "printf" builder
          | SCall ("strlen", [e]) ->
            L.build_call strlen_func [|(expr builder e)|] "strlen" builder
          | SCall ("inputint", [e]) ->
            L.build_call inputint_func [| (expr builder e) |] "inputint" builder
          | SCall ("inputfloat", [e]) ->
            L.build_call inputfloat_func [| (expr builder e) |] "inputfloat"
                builder
          | SCall ("cmd", [e]) ->
            L.build_call cmd_func [| (expr builder e) |] "cmd" builder
          | SCall ("inputstring", [e]) ->
            L.build_call inputstring_func [| (expr builder e) |] "inputstring"
                builder
          | SCall ("sendmail", [e1; e2]) ->
            L.build_call sendmail_func [| (expr builder e1);(expr builder e2) |]
                "sendmail" builder
          | SCall ("inputgui", [e]) ->
            L.build_call inputgui_func [| (expr builder e) |] "inputgui" builder
          | SCall ("inputfile", [e1; e2]) ->
            L.build_call inputfile_func [|(expr builder e1); (expr builder e2)|]
            "inputfile" builder
          | SCall ("strcmp", [e1; e2]) ->
            L.build_call strcmp_func [|(expr builder e1); (expr builder e2)|]
            "strcmp" builder
          | SCall ("strcat", [e1; e2]) ->
            L.build_call strcat_func [|(expr builder e1); (expr builder e2)|]
            "strcat" builder
          | SCall ("strcpy", [e1; e2]) ->
            L.build_call strcpy_func [|(expr builder e1); (expr builder e2)|]
            "strcpy" builder
          | SCall("calloc", [e]) ->
            L.build_call calloc_func [|L.const_int i32_t 1; (expr builder e)|]
```

```
                "calloc" builder
        | SCall("free", [e]) ->
          L.build_call free_func [| (expr builder e) |] "free" builder
        | SCall (f, args) ->
          let (fdef, fdecl) = StringMap.find f function_decls
          in
          let llargs = List.rev (List.map (expr builder) (List.rev args))
          in
          let result = (match fdecl.styp with
                 | _ -> f ^ "_result")
          in L.build_call fdef (Array.of_list llargs) result builder
in

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

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

let rec stmt builder = function
    SBlock sl -> List.fold_left stmt builder sl
  | SExpr e -> ignore(expr builder e); builder
  | SReturn e -> ignore(match fdecl.styp with
                        (* Special "return nothing" instr *)

                        (* Build return statement *)
                      | _ -> L.build_ret (expr builder e) builder );
                builder
  | SSetn(t, s, e1, e2) -> ignore((match t with
                          List(Int) -> L.build_call setn_int_func
                                      [|L.build_load (lookup s) s builder;
                                      expr builder e1; expr builder e2;|]
                                      "" builder
                        | List(Float) -> L.build_call setn_double_func
                                        [|L.build_load (lookup s) s builder;
                                        expr builder e1; expr builder e2;|]
                                        "" builder
                        | _ -> raise (Failure "List type error"))); builder
  | SIf (predicate, then_stmt, else_stmt) ->
      let bool_val = expr builder predicate
      in
      let merge_bb = L.append_block context "merge" the_function
      in
      let build_br_merge = L.build_br merge_bb
      in (* partial function *)

      let then_bb = L.append_block context "then" the_function
      in add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
                     build_br_merge;

      let else_bb = L.append_block context "else" the_function
      in add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
                     build_br_merge;

        ignore(L.build_cond_br bool_val then_bb else_bb builder);
        L.builder_at_end context merge_bb

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

    let body_bb = L.append_block context "while_body" the_function
    in add_terminal (stmt (L.builder_at_end context body_bb) body)
                   (L.build_br pred_bb);

    let pred_builder = L.builder_at_end context pred_bb
    in
    let bool_val = expr pred_builder predicate
    in

    let merge_bb = L.append_block context "merge" the_function
    in ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
      L.builder_at_end context merge_bb

  (* Implement for loops as while loops *)

in

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

(* Add a return if the last block falls off the end *)
add_terminal builder (match fdecl.styp with
       (* A.Void -> L.build_ret_void*)
      | A.Float -> L.build_ret (L.const_float float_t 0.0)
      | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body functions;
  the_module
```

## ./src/iris.ml

```
type action = Ast | Sast | Compile | IR
open Ast

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a", Arg.Unit (set_action Ast), "Print the AST");
    ("-s", Arg.Unit (set_action Sast), "Print the SAST");
     ("-l", Arg.Unit (set_action 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: ./iris.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
    match !action with
      Ast     -> ()
    | Sast    -> print_string (Sast.string_of_sprogram sast)
    | IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
    | Compile -> let m = Codegen.translate sast in
  Llvm_analysis.assert_valid_module m;
  print_string (Llvm.string_of_llmodule m)
```

## ./src/outputfile.c

```
#include<stdio.h>
```

```
#include<stdlib.h>

char* outputfile( char* fname, char* fcontent) {
   FILE *fp=fopen(fname, "w");
   fprintf(fp,"%s\n", fcontent);
   fclose(fp);
   return fcontent;
 }
```

## ./src/parser.mly

```
%{
open Ast
%}

%token COMMA LPAR RPAR LSQR RSQR ZDKH YDKH
%token IF ELIF ENDIF ELSE ENDFUN NEWLINE
%token WHILE ENDLOOP BREAK CONTINUE
%token TRUE FALSE
%token PLUS MINUS TIMES DIVIDE MOD LENGTH
%token EQ NEQ LEQ GEQ LT GT OR AND NOT
%token EOF PIPE
%token INT FLOAT BOOL STRING TUPLE
%token RETURN
%token <string> LSTR ID LFLT
%token <int> LINT
%token <bool> LBOOL

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%left PIPE
%left OR
%left AND
%left EQ NEQ
%left LEQ GEQ LT GT
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT
%left LSQR RSRQ

%%
```

```
/* functions are */
program:
  decls EOF { $1 }

decls:
    { [], [] }
    | decls vdecl NEWLINE      { ($2 @ fst $1), snd $1 }
    | decls fdecl NEWLINE      { fst $1, ($2 :: snd $1) }

/* start of decls */
fdecl:
    TYPE ID LPAR param RPAR NEWLINE vdecl_list stmt_list ENDFUN
    { { typ = $1;
        fname = $2;
        formals = List.rev $4;
        locals = List.rev $7;
        body = List.rev $8 } }

param:
                  { [] }
    | param_list  { $1 }

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

TYPE:
    basic_type        { $1 }
    | basic_type LSQR RSQR   { List($1)}
    | TUPLE ZDKH basic_type_list YDKH              { Tuple(List.rev $3) }

basic_type_list:
        {[]}
    | basic_type_list basic_type   {$2 :: $1}
basic_type:
    FLOAT             { Float }
    | INT             { Int}
    | STRING          { String }
    | BOOL            { Bool }


/* end of decls */

/* end of rule_list */

stmt_list:
    {[]}
    | stmt_list stmt {$2 :: $1}

stmt:
    | expr NEWLINE                                  { Expr $1 }
    | RETURN expr NEWLINE                           { Return($2) }
    | IF LPAR expr RPAR NEWLINE stmt_list %prec NOELSE ENDIF NEWLINE { If($3,
      Block(List.rev $6), Block([])) }
    | IF LPAR expr RPAR NEWLINE stmt_list ELSE NEWLINE stmt_list ENDIF NEWLINE {
      If($3, Block(List.rev $6), Block(List.rev $9)) }
    | WHILE LPAR expr RPAR NEWLINE stmt_list ENDLOOP NEWLINE       { While($3,
      Block(List.rev $6)) }
    | expr PIPE ID LSQR expr RSQR NEWLINE                 { Setn($3, $5, $1) }
    /*| LPAR expr_list RPAR PIPE TYPE id_list NEWLINE { VarMulDecl($2, $5, $6)}*/

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

/* TODO: Modify TYPE id_list return value */
vdecl:
    /* TYPE id_list NEWLINE { $1, $2 } */
    TYPE id_list NEWLINE { List.map (fun id -> ($1, id)) $2}
    /*| expr PIPE TYPE ID NEWLINE { Vdecl($3, $4, $1) }*/

id_list:
     ID { [$1] }
    | ID COMMA id_list { $1 :: $3 }

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


expr:
      LSTR                  { Lstring($1) }
    | LFLT                  { Lfloat($1) }
    | LINT                  { Lint($1) }
    | LBOOL                 { Lbool($1) }
    | ID                    { Id($1) }
    | LSQR expr_list RSQR   { Llist($2) }
    | ZDKH expr_list YDKH   { Ltuple($2) }
    | LENGTH LPAR expr RPAR { Length($3)}
    | expr PLUS  expr  { Binop($1, Add, $3) }
    | expr PIPE PLUS expr { Binop($1, Add, $4) }
    | expr MINUS expr  { Binop($1, Sub, $3) }
    | expr PIPE MINUS expr { Binop($1, Sub, $4) }
    | expr TIMES expr  { Binop($1, Mult, $3) }
    | expr PIPE TIMES expr { Binop($1, Mult, $4) }
    | expr DIVIDE expr { Binop($1, Div, $3) }
    | expr PIPE DIVIDE expr { Binop($1, Div, $4) }
    | expr EQ    expr  { Binop($1, Equal, $3) }
    | expr NEQ   expr  { Binop($1, Neq, $3) }
    | expr LT    expr  { Binop($1, Less, $3) }
    | expr LEQ   expr  { Binop($1, Leq, $3) }
    | expr GT    expr  { Binop($1, Greater, $3) }
    | expr GEQ   expr  { Binop($1, Geq, $3) }
    | expr AND   expr  { Binop($1, And, $3) }
    | expr OR    expr  { Binop($1, Or,  $3) }
    | MINUS      expr  { Unop(Neg, $2) }
    | NOT expr         { Unop(Not, $2) }
    | expr PIPE ID     { Assign($3, $1) }
    | ID LSQR expr RSQR      { Getn($1, $3) }
    | ID LPAR args_opt RPAR  { Call($1, $3) }
    | LPAR expr RPAR         { $2 }

args_opt:
    { [] }
    | args_list { List.rev $1 }

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

## ./src/sast.ml

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

open Ast

type sexpr = typ * sx
and sx =
    SLint of int
  | SLfloat of string
  | SLbool of bool
  | SLstring of string
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of string * sexpr
  | SCall of string * sexpr list
  (* TODO: List relavant sast *)
  | SLlist of sexpr list
  | SLtuple of sexpr list
  | SGetn of string * sexpr
  | SLength of sexpr
  (* | SNoexpr *)

type sstmt =
    SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SWhile of sexpr * sstmt
  | SSetn of typ * string * sexpr * sexpr

type sfunc_decl = {
    styp : typ;
    sfname : string;
    sformals : bind list;
    slocals : bind list;
    sbody : sstmt list;
  }

type sprogram = bind list * sfunc_decl list

(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
    SLint(l) -> string_of_int l
  | SLbool(true) -> "true"
  | SLbool(false) -> "false"
  | SLfloat(l) -> l
  | SLstring(s) -> s
  | SId(s) -> s
  | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_sexpr e2
  | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
  | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
  | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^ ")"
  | SLlist l -> "[" ^ String.concat ", " (List.map string_of_sexpr l) ^ "]"
  | SLtuple l -> "(" ^ String.concat ", " (List.map string_of_sexpr l) ^ ")"
  | SGetn (e1, e2) -> "SGetn(" ^ e1 ^ ", " ^ string_of_sexpr e2 ^ ")"
  | SLength(e) -> "SLengthof" ^ string_of_sexpr e
  (* | SNoexpr -> "" *)
```

```
          ) ^ ")"

let rec string_of_sstmt = function
    SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^ "}\n"
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) -> "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^ string_of_sstmt s
  | SSetn(t, s, e1, e2) -> "Setn(" ^ string_of_typ t ^ ": " ^ s ^ ", " ^
      string_of_sexpr e1 ^ ", " ^ string_of_sexpr e2 ^ ")"

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

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

## ./src/scanner.mll

```
{ open Parser }

rule token =
    parse [' ' '\t' '\r']     { token lexbuf } (* whitespace *)
    | "||"                    { OR }
    | [' ' '\t' '\r' '\n']+'|'[' ' '\t' '\r' '\n']+ { PIPE }
    | "/*"                    { comment lexbuf }(* start comment *)
    (* operators*)
    | '+'
    | [' ' '\t' '\r' '\n']*'|'[' ' '\t' '\r' '\n']*'+'             { PLUS }
    | '-'
    | [' ' '\t' '\r' '\n']*'|'[' ' '\t' '\r' '\n']*'-'             { MINUS }
    | '*'
    | [' ' '\t' '\r' '\n']*'|'[' ' '\t' '\r' '\n']*'*'             { TIMES }
    | '/'
    | [' ' '\t' '\r' '\n']*'|'[' ' '\t' '\r' '\n']*'/'             { DIVIDE}
    | '%'
    | [' ' '\t' '\r' '\n']*'|'[' ' '\t' '\r' '\n']*'%'               { MOD }
    | ['\n']+                 { NEWLINE }
    (* parathen *)
    | '['                     { LSQR }
    | ']'                     { RSQR }
    | ','                     { COMMA }
    | '('                     { LPAR }
    | ')'                     { RPAR }
    | '{'                     { ZDKH}
    | '}'                     { YDKH }
    (* logic *)
    | '='                     { EQ }
    | "!="                    { NEQ }
    | "<="                    { LEQ }
    | ">="                    { GEQ }
    | '<'                     { LT }
    | '>'                     { GT }
    | "&&"                    { AND }
    | '!'                     { NOT }
    | ("true"|"false") as bl  { LBOOL(bool_of_string bl) }
    (* num *)
    | ['0' - '9']+ as lit     { LINT(int_of_string lit) }
    | ['0' - '9']*'.'['0' - '9']+ as lit { LFLT(lit) }
    | ['0' - '9']+'.'['0' - '9']* as lit { LFLT(lit) }
    (* key words *)
    | "int"                   { INT }
    | "float"                 { FLOAT }
    | "string"                { STRING }
    | "bool"                  { BOOL }
    | "tuple"                 { TUPLE }
    | "if"                    { IF }
    | "else"                  { ELSE }
    | "elif"                  { ELIF }
    | "fi"                    { ENDIF }
    | "while"                 { WHILE }
    | "return"                { RETURN }
    | "end"                   { ENDLOOP }
    | "continue"              { CONTINUE }
    | "break"                 { BREAK }
    | "Siri"                  { ENDFUN }
    | "length"                { LENGTH }
    | eof                     { EOF }
    (* String *)
    | '"'                     {
                                  let buffer = Buffer.create 1
                              in LSTR(str_of_string buffer lexbuf)
                          }
    (* ID *)
    | ['a' - 'z' 'A' - 'Z']['a' - 'z' 'A' - 'Z' '0' - '9' '_']* as id { ID(id) }
    | _ as ch { raise (Failure("invalid character detected " ^ Char.escaped ch)) }
    and comment =
    parse "*/" { token lexbuf } (* end comment *)
    | _ { comment lexbuf }
    and str_of_string buffer = (* can be more functional *)
    parse '"' { Buffer.contents buffer}
    | "\\t"  { Buffer.add_char buffer '\t'; str_of_string buffer lexbuf }
    | "\\n"  { Buffer.add_char buffer '\n'; str_of_string buffer lexbuf }
    | "\\\"" { Buffer.add_char buffer '"'; str_of_string buffer lexbuf }
    | "\\\\" { Buffer.add_char buffer '\\'; str_of_string buffer lexbuf }
    | _ as ch { Buffer.add_char buffer ch; str_of_string buffer lexbuf }
    (*
       ref:https://stackoverflow.com/questions/5793702/using-ocamllex-for-lexing-strings-the-tiger-
       *)
```

## ./src/semant.ml

```
open Ast
open Sast

module StringMap = Map.Make(String)
let check (globals, functions) =
    (* Verify a list of bindings has no duplicate names *)
  let check_binds (kind : string) (binds : bind list) =
    let rec dups = function
        [] -> ()
      | ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
        raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
      | _ :: t -> dups t
    in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
  in
    (**** Check global variables ****)

  check_binds "global" globals;

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

    (* Collect function declarations for built-in functions: no bodies *)
  let built_in_decls =
    let add_bind map (typ, name, fm) = StringMap.add name {
      typ = typ;
      fname = name;
      formals = fm;
      locals = []; body = [] } map
    in List.fold_left add_bind StringMap.empty
       [(Int, "print", [(String, "x")]);
        (Int, "printi", [(Int, "x")]);
        (Int, "printf", [(Float, "x")]);
        (Int, "printb", [(Bool, "x")]);
        (Int, "strlen", [(String, "x")]);
        (Int, "strcmp", [(String, "x");(String, "x1")]);
        (String, "strcat", [(String, "x1");(String, "x2")]);
        (String, "strcpy", [(String, "x");(String, "x1")]);
        (Int, "inputint", [(String, "x")]);
        (Float, "inputfloat", [(String, "x")]);
        (String, "inputstring", [(String, "x")]);
        (String, "calloc", [(Int, "x1")]);
        (Int, "free", [(String, "x")]);
        (Int, "inputgui", [(Int, "x")]);
        (Int, "cmd", [(String, "x")]);
        (Int, "sendmail", [(String, "x1");(String, "x2")]);
        (String, "inputfile", [(String, "x1");(Int, "x")]);
        (*add built-in function here*)]
  in
    (* Add function name to symbol table *)
  let add_func map fd =
    let built_in_err = "function " ^ fd.fname ^ " may not be defined"
    and dup_err = "duplicate function " ^ fd.fname
    and make_err er = raise (Failure er)
    and n = fd.fname (* Name of the function *)
    in match fd with (* No duplicate functions or redefinitions of built-ins *)
         _ when StringMap.mem n built_in_decls -> make_err built_in_err
       | _ when StringMap.mem n map -> make_err dup_err
       | _ -> StringMap.add n fd map
  in
    (* Collect all function names into one symbol table *)
  let function_decls = List.fold_left add_func built_in_decls functions
  in
      (* Return a function from our symbol table *)
  let find_func s =
    try StringMap.find s function_decls
    with Not_found -> raise (Failure ("unrecognized function " ^ s))
  in

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

```
let check_function func =
  (* Make sure no formals or locals are duplicates *)
  check_binds "formal" func.formals;
  check_binds "local" func.locals;
    (* Raise an exception if the given rvalue type cannot be assigned to
       the given lvalue type *)
  let check_assign lvaluet rvaluet err =
    if lvaluet = rvaluet then lvaluet else raise (Failure err)
  in
  (* Build local symbol table of variables for this function *)
  let symbols = List.fold_left (fun m (ty, name) -> StringMap.add name ty m)
              StringMap.empty (globals @ func.formals @ func.locals )
  in
  (* Return a variable from our local symbol table *)
  let type_of_identifier s =
  try StringMap.find s symbols
  with Not_found -> raise (Failure ("undeclared identifier " ^ s))
  in
  let list_access_type = function
     List(t) -> t
   | _ -> raise (Failure ("illegal list access") )
  in

  let list_type s = match (List.hd s) with
     Lint _ -> List(Int)
   | Lfloat _ -> List(Float)
   | Lbool _ -> List(Bool)
   | Lstring _ -> List(String)
   (* TODO | (* nested list *) *)
   | _ -> raise ( Failure ("Cannot instantiate a list of that type"))
  in

  let rec check_all_list_literal m ty idx =
   let length = List.length m
   in
   match (ty, List.nth m idx) with
     (List(Int), Lint _) -> if idx == length - 1 then List(Int) else
         check_all_list_literal m (List(Int)) (succ idx)
   | (List(Float), Lfloat _) -> if idx == length - 1 then List(Float) else
       check_all_list_literal m (List(Float)) (succ idx)
   | (List(Bool), Lbool _) -> if idx == length - 1 then List(Bool) else
       check_all_list_literal m (List(Bool)) (succ idx)
   | (List(String), Lstring _) -> if idx == length - 1 then List(String) else
       check_all_list_literal m (List(String)) (succ idx)
   (* TODO | (* nested list *) *)
   | _ -> raise (Failure ("illegal list literal"))
```

```
in
 (* Return a semantically-checked expression, i.e., with a type *)
let rec expr = function
    Lint  l -> (Int, SLint l)
  | Lfloat l -> (Float, SLfloat l)
  | Lbool l -> (Bool, SLbool l)
  | Lstring s -> (String, SLstring s)
  | Id s      -> (type_of_identifier s, SId s)
  | Llist l -> ((check_all_list_literal l (list_type l) 0), SLlist (List.map
      expr l))
  | Ltuple l -> (Tuple(List.map (fun e -> fst (expr e)) l), SLtuple (List.map
      expr l))
  | Assign(var, e) as ex ->
        (match var with
          _ when StringMap.mem var function_decls ->
               let args = (match e with
                             Ltuple l -> l
                           | _ -> [e]
                           )
               and fname = var
               in
               expr (Call(fname, args))
        | _ ->
          let lt = type_of_identifier var
          and (rt, e') = expr e
          in
          let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
           string_of_typ rt ^ " in " ^ string_of_expr ex
          in (check_assign lt rt err, SAssign(var, (rt, e')))
        )

  | Getn(s, e) -> let e' = expr e in
                 let _ = (match e' with
                      (Int, _) -> e'
                     | _ -> raise (Failure ("attempting to access with a
                         non-integer type"))) in
                     ( list_access_type (type_of_identifier s), SGetn(s, e') )
  | Unop(op, e) as ex ->
    let (t, e') = expr e
    in
    let ty = match op with
        Neg when t = Int || t = Float -> t
      | Not when t = Bool -> Bool
      | _ -> raise (Failure ("illegal unary operator " ^
                 string_of_uop op ^ string_of_typ t ^
                 " in " ^ string_of_expr ex))
    in (ty, SUnop(op, (t, e')))
  | Binop(e1, op, e2) as e ->
    let (t1, e1') = expr e1
    and (t2, e2') = expr e2
    in
      (* All binary operators require operands of the same type *)
    let same = t1 = t2
    in
      (* Determine expression type based on operator and operand types *)
    let ty = match op with
        Add | Sub | Mult | Div when same && t1 = Int -> Int
      | Add | Sub | Mult | Div when same && t1 = Float -> Float
      | Equal | Neq          when same              -> Bool
      | Less | Leq | Greater | Geq
                 when same && (t1 = Int || t1 = Float) -> Bool
      | And | Or when same && t1 = Bool -> Bool
      | _ -> raise (
     Failure ("illegal binary operator " ^
                 string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
                 string_of_typ t2 ^ " in " ^ string_of_expr e))
    in (ty, SBinop((t1, e1'), op, (t2, e2')))
  | Call(fname, args) as call ->
      let fd = find_func fname
      in
      let param_length = List.length fd.formals
      in
        if List.length args != param_length then
          raise (Failure ("expecting " ^ string_of_int param_length ^
                         " arguments in " ^ string_of_expr call))
        else
          let check_call (ft, _) e =
            let (et, e') = expr e
            in
            let err = "illegal argument found " ^ string_of_typ et ^
              " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e
            in (check_assign ft et err, e')
          in
          let args' = List.map2 check_call fd.formals args
          in (fd.typ, SCall(fname, args'))
  | Length(e) as len ->
    let e' = expr e in (match e' with
        (List(_), SId _)
        | (_, SLlist(_)) -> (Int, SLength e')
        | _ -> raise(Failure ("length can only applied to list" ^
            string_of_expr len))
    )
```

```
    in

    let check_bool_expr e =
      let (t', e') = expr e
      and err = "expected Boolean expression in " ^ string_of_expr e
      in if t' != Bool then raise (Failure err) else (t', e')
    in
(* Return a semantically-checked statement i.e. containing sexprs *)
    let rec check_stmt = function
        Expr e -> SExpr (expr e)
      | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1, check_stmt b2)
      | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)
      | Return e -> let (t, e') = expr e in
         if t = func.typ then SReturn (t, e')
         else raise (
              Failure ("return gives " ^ string_of_typ t ^ " expected " ^
              string_of_typ func.typ ^ " in " ^ string_of_expr e))

      (* A block is correct if each statement is correct and nothing
         follows any Return statement. Nested blocks are flattened. *)
      | Block sl ->
          let rec check_stmt_list = function
              [Return _ as s] -> [check_stmt s]
            | Return _ :: _ -> raise (Failure "nothing may follow a return")
            | Block sl :: ss -> check_stmt_list (sl @ ss) (* Flatten blocks *)
            | s :: ss        -> check_stmt s :: check_stmt_list ss
            | []             -> []
         in SBlock(check_stmt_list sl)
      | Setn(s, e1, e2) ->
        SSetn(type_of_identifier s, s, expr e1, expr e2)
    in (* body of check_function *)
    {
      styp = func.typ;
      sfname = func.fname;
      sformals = func.formals;
      slocals = func.locals;
      sbody = match check_stmt (Block func.body) with
        SBlock(sl) -> sl
        | _ -> raise (Failure ("internal error: block didn't become a block?"))
    }
  in (globals, List.map check_function functions)
```

## ./src/lib/cmd.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>


int cmd(char* word)
{

  //printf("%s",str);
  system(word);


  return 0;


}
```

## ./src/lib/inputfile.c

```
#include<stdio.h>
#include<stdlib.h>

 char *inputfile(char * str, int length) {
  char *text = malloc(length * sizeof(char));
  FILE *fp=fopen(str, "r");
  int i=0;
  while(!feof(fp)) {
     text[i++] = fgetc(fp);
  }
  text[i]='\0';
  fclose(fp);
  return text;
}
```

## ./src/lib/inputfloat.c

```
#include<stdio.h>
#include<stdlib.h>

double inputfloat(char *str) {
   double a;
   printf("%s", str);
```

```
  scanf("%lg",&a);
  return a;
}
```

## ./src/lib/inputgui.c

```
#include <stdio.h>
#include <gtk/gtk.h>


int inputgui(int b)
{

  GtkWidget *window;
  GtkWidget *vbox;

  GtkWidget *menubar;
  GtkWidget *fileMenu;
  GtkWidget *fileMi;
  GtkWidget *quitMi;

  gtk_init(NULL,NULL);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 300, 200);
  gtk_window_set_title(GTK_WINDOW(window), "Simple menu");

  vbox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(window), vbox);

  menubar = gtk_menu_bar_new();
  fileMenu = gtk_menu_new();

  fileMi = gtk_menu_item_new_with_label("File");
  quitMi = gtk_menu_item_new_with_label("Quit");

  gtk_menu_item_set_submenu(GTK_MENU_ITEM(fileMi), fileMenu);
  gtk_menu_shell_append(GTK_MENU_SHELL(fileMenu), quitMi);
  gtk_menu_shell_append(GTK_MENU_SHELL(menubar), fileMi);
  gtk_box_pack_start(GTK_BOX(vbox), menubar, FALSE, FALSE, 0);

  g_signal_connect(G_OBJECT(window), "destroy",
       G_CALLBACK(gtk_main_quit), NULL);
  g_signal_connect(G_OBJECT(quitMi), "activate",
        G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

## ./src/lib/inputint.c

```
#include <stdio.h>

int inputint(char *str)
{
   int a;
   printf("%s", str);
   scanf("%d",&a);
   return a;
}
```

## ./src/lib/inputstring.c

```

#include<stdio.h>
#include<stdlib.h>

#define BUFFER 100

char *inputstring(char *str) {
  char *text = malloc(BUFFER * sizeof(char));

  printf("%s", str);
  scanf("%s",text);
  return text;
}
```

## ./src/lib/liblist.c

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
```

```c
#include <errno.h>
#include <stdarg.h>
#include "liblist_type.h"

static inline void init_list(struct iris_list_head *head)
{
    head->length = 0;
    head->size = 0;
    head->data.i = NULL;
}

int iris_list_resize(struct iris_list_head *head, int newsize, int unit)
{
    int new_allocated;

    new_allocated = (newsize >> 3) + (newsize < 9 ? 3 : 6);
    if (new_allocated * unit > IRIS_SIZE_MAX - newsize * unit) {
        errno = -ENOMEM;
        return errno;
    } else
        new_allocated += newsize;

    head->data.i = realloc(head->data.i, new_allocated * unit);

    if (head->data.i == NULL)
        return errno;

    head->size = new_allocated;
    return 0;
}

struct iris_list_head *__new_list()
{
    struct iris_list_head *head;

    head = malloc(sizeof(struct iris_list_head));
    init_list(head);

    if (head == NULL)
        return NULL;


    return head;
}

int __append1(struct iris_list_head **head, int unit)
```

```
{
    int err = 0;

    if (*head == NULL)
        *head = __new_list();

    if (*head == NULL)
        return errno;

    if ((*head)->size < (*head)->length + 1)
        err = iris_list_resize(*head, (*head)->length + 1, unit);

    return err;
}

struct iris_list_head *append_int(struct iris_list_head *head, int num, ...)
{
    int err, i, value;
    va_list valist;

    va_start(valist, num);
    for (i = 0; i < num; i++) {
        value = va_arg(valist, int);
        err = __append1(&head, sizeof(int));
        if (err)
            perror("IRIs error: ");

        memcpy(&(head->data.i[head->length]),
                                &value, sizeof(int));
        head->length += 1;
    }
    va_end(valist);

    return head;
}

struct iris_list_head *append_double(struct iris_list_head *head, int num, ...)
{
    int err, i;
    double value;
    va_list valist;

    va_start(valist, num);
    for (i = 0; i < num; i++) {
        value = va_arg(valist, double);
        err = __append1(&head, sizeof(double));
```

```c
        if (err)
            perror("IRIs error: ");

        memcpy(&(head->data.f[head->length]),
                                &value, sizeof(double));
        head->length += 1;
    }
    va_end(valist);

    return head;
}

struct iris_list_head *append_char(struct iris_list_head *head, int num, ...)
{
    int err, i;
    char value;
    va_list valist;

    va_start(valist, num);
    for (i = 0; i < num; i++) {
        value = va_arg(valist, int);
        err = __append1(&head, sizeof(char));
        if (err)
            perror("IRIs error: ");

        memcpy(&(head->data.c[head->length]),
                                &value, sizeof(char));
        head->length += 1;
    }
    va_end(valist);

    return head;
}

int getn_int(struct iris_list_head *head, int n)
{
    if (n >= head->length) {
        errno = EFAULT;
        perror("IRIs error: ");
        return 0;
    }
    return head->data.i[n];
}

double getn_double(struct iris_list_head *head, int n)
{
    if (n >= head->length) {
        errno = EFAULT;
        perror("IRIs error: ");
        return 0;
    }
    return head->data.f[n];
}

char getn_char(struct iris_list_head *head, int n)
{
    if (n >= head->length) {
        errno = EFAULT;
        perror("IRIs error: ");
        return 0;
    }
    return head->data.c[n];
}

void setn_int(struct iris_list_head *head, int n, int value)
{
    if (n >= head->length) {
        errno = EFAULT;
        perror("IRIs error: ");
        return ;
    }
    head->data.i[n] = value;
}

void setn_double(struct iris_list_head *head, int n, double value)
{
    if (n >= head->length) {
        errno = EFAULT;
        perror("IRIs error: ");
        return ;
    }
    head->data.f[n] = value;
}

void setn_char(struct iris_list_head *head, int n, char value)
{
    if (n >= head->length) {
        errno = EFAULT;
        perror("IRIs error: ");
        return ;
    }
    head->data.c[n] = value;
```

```
}

int length(struct iris_list_head *head)
{
    return head->length;
}
//     return head;
// }
```

### ./src/lib/liblist_type.h

```
#ifndef _LIBLIST_TYPE_H
#define _LIBLIST_TYPE_H

#define LIST_POISON1  ((void *) 0x100)
#define LIST_POISON2  ((void *) 0x200)
#define IRIS_SIZE_MAX 0x10000

struct iris_list_head *append_int(struct iris_list_head *head, int num, ...);
struct iris_list_head *append_double(struct iris_list_head *head, int num, ...);
struct iris_list_head *append_char(struct iris_list_head *head, int num, ...);
int getn_int(struct iris_list_head *head, int n);
double getn_double(struct iris_list_head *head, int n);
char getn_char(struct iris_list_head *head, int n);
void setn_int(struct iris_list_head *head, int n, int value);

struct list_head {
    struct list_head *prev, *next;
};

struct iris_list_head {
    union data {
        int *i;
        double *f;
        char *c;
        char **s;
        struct iris_list_node *p;
    } data;
    int length;
    int size;
};

#endif
```

## ./src/lib/libstr.c

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <stdarg.h>

char* sassign(char *dst, char *src)
{
    free(dst);
    dst = malloc(sizeof(char) * (strlen(src) + 1));

    dst = strcpy(dst, src);
    return dst;
}
```

## ./tests/fail_identifier.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_identifier.ir

```
int main()
   String 12a
        "hello world!"
   |
   12a
   printi(12a)

Siri
```

## ./tests/fail_intplusfloat.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_intplusfloat.ir

```
int main()
   int a
   float b
   float c
   1
   |
   a

   2.2
   |
   b

   a+b
   |
   c
Siri
```

### ./tests/fail_list_element.err

```
Fatal error: exception Failure("illegal list literal")
```

### ./tests/fail_list_element.ir

```
int main()
    int[] a
    [1, 2, 3.4]
    |
    a
Siri
```

### ./tests/fail_manytoone_assign.err

```
Fatal error: exception Parsing.Parse_error
```

### ./tests/fail_manytoone_assign.ir

```
int main()
    int a
    1,2,3
    |
    a
    printi(a)
    printi(b)
    printi(c)
Siri
```

## ./tests/fail_missfi.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_missfi.ir

```
int main()
    int a,b,c
    1
    |
    a
    if(a>5)
        2
        |
    b
    else
    3
    |
    b
Siri
```

## ./tests/fail_multiassign_char.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_multiassign_char.ir

```
int main()
    char a,b,c
    'x','y','z'
    |
    a,b,c


Siri
```

## ./tests/fail_multiassign_float.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_multiassign_float.ir

```
int main()
    float a,b,c
    1.1,2.2,3.3
    |
    a,b,c

Siri
```

## ./tests/fail_multiassign_int.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_multiassign_int.ir

```
int main()
    int a,b,c
    1,2,3
    |
    a,b,c
    printi(a)
    printi(b)
    printi(c)

Siri
```

## ./tests/fail_multiassign_string.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_multiassign_string.ir

```
int main()
    string a,b,c
    "thank","you","!"
    |
    a,b,c
    printi(a)
    printi(b)
    printi(c)
```

```
Siri
```

## ./tests/fail_onetomany_assign.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_onetomany_assign.ir

```
int main()
    int a,b,c
    1
    |
    a,b,c
    printi(a)
    printi(b)
    printi(c)
Siri
```

## ./tests/fail_operate_bool.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_operate_bool.ir

```
int main()
    print(1<2)
    print(1>2)
    print(1=2)
    print("1<2")
    print(1*2)
    print("1*2")
    print("7/3")
    print("7/3)
    print(true and true)
    print(true and false)
    print(true or true)
    print(true or false)
    print(false and false)
    print(false or fase)
    print("true and false")
Siri
```

## ./tests/fail_pass_argu_fun.err

```
Fatal error: exception Failure("illegal argument found string expected int in
    hello")
```

## ./tests/fail_pass_argu_fun.ir

```
int fun(int a)
    a+1
    |
    a
Siri

int main()
    fun("hello")
Siri
```

## ./tests/fail_print_bool.err

```
Fatal error: exception Failure("illegal argument found bool expected int in
    true")
```

## ./tests/fail_print_bool.ir

```
int main()
    printi(false)
    printi(true)
Siri
```

## ./tests/fail_print_char.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_print_char.ir

```
int main()
    char a
    'h'
    |
    a
    print(a)

Siri
```

## ./tests/fail_recursion.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/fail_recursion.ir

```
int fib(int n)
    if(n==0 or n==1)
        return 1
    else
        return fib(n-1)
Siri

int main()
    printi(fib(5))
Siri
```

## ./tests/fail_undeclared.err

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

## ./tests/fail_undeclared.ir

```
int main()
    5
    |
    a
    printi(a)

Siri
```

## ./tests/fail_while_missend.err

```
Fatal error: exception Parsing.Parse_error
```

**./tests/fail_while_missend.ir**

```
int main()
    int a
    1
    |
    a
    while(a<5)
    a+1
    |
    a
Siri
```

**./tests/pass_test_cmd.ir**

```
int main()
    string content, mail, command
    inputstring("Please input the content:\n")
    |
    content
    inputstring("Please input the email address:\n")
    |
    mail
    calloc(80)
    |
    command
    strcpy(command, "echo ")
    |
    command
    strcat(command, content)
    |
    command
    strcat(command," | mail -s \"noreply at ‘date‘\" ")
    |
    command
    strcat(command, mail)
    |
    command
    |
    cmd
Siri
```

**./tests/pass_test_cmdsendmail.ir**

```
int main()
    string content
    int a
    inputstring(0)
    |
    content
    cmd(content)
    |
    a
    |
    printi
Siri
```

## ./tests/pass_test_inputfile.ir

```
int main()

    string str1


    inputfile("chatbot.txt",100)
    |
    str1

    print(str1)
    return 0
Siri
```

## ./tests/pass_test_inputgui.ir

```
int main()
    int a

    inputgui(1)
    |
    a

    printi(a)
Siri
```

## ./tests/pass_test_inputstring.ir

```
int main()
    string content
    inputstring(0)
    |
    content
    print(content)
Siri
```

## ./tests/pass_test_sendmail.ir

```
int main()
    string content,email
    int a
    inputstring(0)
    |
    content
    inputstring(0)
    |
    email
    sendmail(content,email)
    |
    a
    printi(a)
Siri
```

## ./tests/test_bin_search.ir

```
int binary_search(int target, int[] nums)
    int n, l, r, mid
    0
    |
    l
    length(nums)
    |
    n
    n
    |
    -1
    |
    r
    while(l<r)
        (l
        |
        +r)
        |
        /2
        |
        mid
        if(target=nums[mid])
            return mid
        fi
        if(target>nums[mid])
            1
            |
            +mid
            |
            l
        else
            mid
            |
            r
        fi
    end
    return -1
Siri

int main()
    {2, [1, 2, 3, 4, 5, 6]}
    |
    binary_search
    |
    printi
Siri
```

---

## ./tests/test_calculate_float.ir

---

```
int main()
    float a,b,c
    5.2
    |
    a


    a+1.3
    |
    b

    b+2.5
    |
    c

    c-b
    |
    a

    printf(a)
    printf(b)
    printf(c)
Siri
```

### ./tests/test_calculate_float.out

```
2.5
6.5
9
```

### ./tests/test_calculate_int.ir

```
int main()
    int a,b,c
    5
    |
    a


    a+1
    |
    b

    b+1
    |
    c

    c-b
    |
    a

    printi(a)
    printi(b)
    printi(c)
Siri
```

**./tests/test_calculate_int.out**

```
1
6
7
```

**./tests/test_calculate_tips.ir**

```
int main()
    int i, n, id
    float bill
    float[] tip_rate
    0
    |
    i

    [0.12, 0.15, 0.18, 0.20]
    |
    tip_rate
    length(tip_rate)
    |
    n
    "Please input your bill:\n"
    |
    inputfloat
    |
    bill
    while(i<n)
        1
        |
        +i
        |
        printi
        ": "
        |
        print
        tip_rate[i]
        |
        printf
        "\n"
        |
        print
        1
        |
        +i
        |
        i
    end
    "Choose the tip rate:\n"
    |
    inputint
    |
    id
    "Your tip is: "
    |
    print
    tip_rate[id]
    |
    *bill
    |
    printf
    "\n"
    |
    print
    return 0
Siri
```

## ./tests/test_cat.ir

```
int main()
    calloc(1, 20)
    |
    string catedString
    strcpy(catedString, "first ")

    strcat(catedString, "second\n")
    print(catedString)

    return 0
Siri
```

## ./tests/test_comment.ir

```
int main()
    /* hello world */printi(1)
Siri
```

## ./tests/test_comment.out

```
1
```

## ./tests/test_continuous_assign_bool.ir

```
int main()
    bool a,b,c
    true
    |
    a
    |
    b
    |
    c
Siri
```

## ./tests/test_continuous_assign_float.ir

```
int main()
    float a,b,c
    1.1
    |
    a
    |
    b
    |
    c
    printf(a)
    printf(b)
    printf(c)
Siri
```

## ./tests/test_continuous_assign_float.out

```
1.1
1.1
1.1
```

## ./tests/test_continuous_assign_int.ir

```
int main()
    int a,b,c
    1
    |
    a
    |
    b
    |
    c
    printi(a)
    printi(b)
    printi(c)
Siri
```

### ./tests/test_continuous_assign_int.out

```
1
1
1
```

### ./tests/test_continuous_assign_string.ir

```
int main()
    string a,b,c
    "hello"
    |
    a
    |
    b
    |
    c
    print(a)
    print(b)
    print(c)
Siri
```

### ./tests/test_continuous_assign_string.out

```
hello
hello
hello
```

## ./tests/test_continuous_operate_float.ir

```
int main()
   float a,b,c
   1.23
   |
   a
   |
   b
   |
   +2.12
   |
   c
   printf(a)
   printf(b)
   printf(c)
Siri
```

## ./tests/test_continuous_operate_float.out

```
1.23
1.23
3.35
```

## ./tests/test_continuous_operate_int.ir

```
int main()
   int a,b,c
   1
   |
   a
   |
   b
   |
   +2
   |
   c
   printi(a)
   printi(b)
   printi(c)
Siri
```

## ./tests/test_continuous_operate_int.out

```
1
1
3
```

### ./tests/test_continuous_operate_string.ir

```
int main()
    int a
    string str,mystr
    "abcd"
    |
    + "hello"
    |
    mystr

    print(str)
    print(mystr)
Siri
```

### ./tests/test_continuous_operate_string.out

```
Fatal error: exception Failure("illegal binary operator string+string in abcd +
    "hello"")
```

### ./tests/test_def_fun_aftermain.ir

```
int main()
    int a
    add(5)
    |
    a
    printi(a)
Siri

int add(int n)
    n+1
    |
    n
Siri
```

## ./tests/test_def_fun_aftermain.out

```
0
```

## ./tests/test_different_assign.ir

```
int main()
    int a, b, c
    2
    |
    a

    3|b

    4
    |c

    printi(a)
    printi(b)
    printi(c)

Siri
```

## ./tests/test_different_assign.out

```
2
3
4
```

## ./tests/test_elseif.err

```
FAtal error: exception Parsing.Parse_error
```

## ./tests/test_elseif.ir

```
int main()
    int a
    3
    |
    a
    if (a==1)
        10
    |
    b
    elif(a==2)
        100
    |
    b
    else
    1000
    |
    b
    fi

    printi(b)
Siri
```

### ./tests/test_escape.ir

```
int main()
    print("Hello \"world\"\n")
    return 0
Siri
```

### ./tests/test_escape.out

```
Hello "world"
```

### ./tests/test_fail_pass_argu_fun.err

```
Fatal error: exception Failure("illegal argument found string expected int in
    "hello"")
```

### ./tests/test_fail_pass_argu_fun.ir

```
int fun(int a)
    a+1
    |
    a
Siri

int main()
    fun("hello")
```

## ./tests/test_flow2.ir

```
int start_order()
    return 0
Siri

int add_rice(int calorie)
    string[] rices
    int num
    int cur

    0
    |
    cur
    ["1: White Rice", "2: Brown Rice"]
    |
    rices
    "What kinda rice do you want?\n"
    |
    print

    rices
    |
    length
    |
    len

    while(cur<len)
        (rices[cur], "\n")
        |
        strcat
        |
        print

        1
        +cur
        |
        cur
    endwhile

    inputi()
    |
    rice
    if (rice=1)
        return calorie + 120
    elif (rice=2)
        return calorie + 120
    else
        return calorie
    fi

    return calorie



int print_order(int calorie)
    calorie
    |
    printi
    return 0
Siri

int main()
    start_order
    |
    add_rice
    |
    print_order

    return 0
Siri
```

## ./tests/test_fun_flow.ir

```
int bar(int v)
    return v + 2
Siri

int foo(int a)
    return a + 1
Siri

int main()
    1
    |
    foo
    |
    bar
    |
    printi
Siri
```

### ./tests/test_func_call.ir

```
int fun(int a)
    int n
    a
    |
    +1
    |
    n
    return(n)
Siri

int main()
    int b
    fun(5)
    |
    b
    printi(b)
Siri
```

### ./tests/test_func_call.out

```
6
```

### ./tests/test_helloworld.ir

```
int main()
    print("Hello world\n")
    return 0
Siri
```

### ./tests/test_helloworld.out

```
Hello world
```

## ./tests/test_identifier.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_identifier.ir

```
int main()
    String 12a
        "hello world!"
    |
    12a
    printi(12a)

Siri
```

## ./tests/test_if.ir

```
int main()
    int a
    3
    |
    a
    if(a<5)
    10
    |
    a
    fi
    printi(a)
Siri
```

## ./tests/test_if.out

```
10
```

## ./tests/test_ifelse.ir

```
int main()
    int a
    8
    |
    a
    if (a > 5)
```

```
        6|a
    else
        4|a
    fi
    printi(a)
Siri
```

### ./tests/test_ifelse.out

```
6
```

### ./tests/test_inputint.ir

```
int main()
    int a

    inputint(1)
    |
    a

    printi(a)
Siri
```

### ./tests/test_inputint.out

```
1
```

### ./tests/test_intplusfloat.err

```
Fatal error: exception Parsing.Parse_error
```

### ./tests/test_intplusfloat.ir

```
int main()
    int a
    float b
    float c
    1
    |
    a
    2.2
    |
    b

    a+b
    |
    c
Siri
```

### ./tests/test_list_access.ir

```
int main()
    int[] a
    int b
    [1,2,3]
    |
    a
    a[0]
    |
    b
    printi(b)
    a[1]
    |
    b
    printi(b)
    a[2]
    |
    b
    printi(b)
Siri
```

### ./tests/test_list_access.out

```
1
2
3
```

### ./tests/test_list_access_double.ir

```
int main()
    float[] a
    float b
    [1.1,2.1,3.1]
    |
    a
    a[0]
    |
    b
    printf(b)
    a[1]
    |
    b
    printf(b)
    a[2]
    |
    b
    printf(b)
Siri
```

## ./tests/test_list_access_double.out

```
1.1
2.1
3.1
```

## ./tests/test_list_basic.ir

```
int main()
    int[] a
    float[] b
    [1,2,3]
    |
    a

    [1.1, 1.2, 1.3]
    |
    b
Siri
```

## ./tests/test_list_overflow.ir

```
int main()
    int[] a
    int b
    [1,2,3]
    |
    a
    a[3]
    |
    b
    printi(b)
Siri
```

## ./tests/test_list_overflow.out

```
IRIS error: Bad address
0
```

## ./tests/test_list_setn.ir

```
int main()


    int[] a
    int b
    [1,2,3]
    |
    a
    a[1]
    |
    a[0]
    printi(a[0])
Siri
```

## ./tests/test_list_setn_double.ir

```
int main()
    float[] a
    [1.1,2.1,3.1]
    |
    a
    4.1
    |
    a[0]
    printf(a[0])
Siri
```

## ./tests/test_list_setn_double.out

```
4.1
```

## ./tests/test_manytoone_assign.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_manytoone_assign.ir

```
int main()
    int a
    1,2,3
    |
    a
    printi(a)
    printi(b)
    printi(c)
Siri
```

## ./tests/test_multiassign_char.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_multiassign_char.ir

```
int main()
    char a,b,c
    'x','y','z'
    |
    a,b,c


Siri
```

## ./tests/test_multiassign_float.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_multiassign_float.ir

```
int main()
    float a,b,c
    1.1,2.2,3.3
    |
    a,b,c

Siri
```

## ./tests/test_multiassign_int.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_multiassign_int.ir

```
int main()
    int a,b,c
    1,2,3
    |
    a,b,c
    printi(a)
    printi(b)
    printi(c)

Siri
```

## ./tests/test_multiassign_string.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_multiassign_string.ir

```
int main()
    string a,b,c
    "thank","you","!"
    |
    a,b,c
    printi(a)
    printi(b)
    printi(c)
```

```
Siri
```

## ./tests/test_mutable_string.ir

```
int main()
string as
calloc(30)
|
as
strcpy(as, "hello\n")
print(as)
free(as)
return 0
Siri
```

## ./tests/test_mutable_string.out

```
hello
```

## ./tests/test_nested_if.ir

```
int main()
    int a, b
    3
    |
    a
    if(a<5)
        if(a<3)
        10
        |
        a
        else
            100
            |
            a
        fi
    else
        1000
        |
        a
    fi
    printi(a)
Siri
```

## ./tests/test_nested_if.out

```
100
```

## ./tests/test_nested_while.ir

```
int main()
    int a,b
    1
    |
    a

    2
    |
    b
    while(a<5)
        a+1
        |
        a
    while(b<10)
        b+1
        |
        b
    end
    end
    printi(a)
    printi(b)
Siri
```

## ./tests/test_nested_while.out

```
5
10
```

## ./tests/test_onetomany_assign.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_onetomany_assign.ir

```
int main()
    int a,b,c
    1
    |
    a,b,c
    printi(a)
    printi(b)
    printi(c)
Siri
```

## ./tests/test_operate_bool.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_operate_bool.ir

```
int main()
    print(1<2)
    print(1>2)
    print(1=2)
    print("1<2")
    print(1*2)
    print("1*2")
    print("7/3")
    print("7/3)
    print(true and true)
    print(true and false)
    print(true or true)
    print(true or false)
    print(false and false)
    print(false or fase)
    print("true and false")
Siri
```

## ./tests/test_pipe.ir

```
int main()
    int rate, bill, tip, total

    2
    |
    rate
    3
    |
    bill

    rate
    |
    * bill
    |
    tip
    |
    +bill
    |
    total

    printi(tip)
    printi(total)
Siri
```

### ./tests/test_pipe.out

```
6
9
```

### ./tests/test_pipeprint_float.ir

```
int main()
    float a,b,c
    1.234
    |
    a
    |
    b
    |
    c
    |
    printf
Siri
```

### ./tests/test_pipeprint_float.out

```
1.234
```

## ./tests/test_pipeprint_int.ir

```
int main()
    int a,b,c
    1
    |
    a
    |
    b
    |
    c
    |
    printi
Siri
```

## ./tests/test_pipeprint_int.out

```
1
```

## ./tests/test_pipeprint_string.ir

```
int main()
    string a,b,c
    "hello"
    |
    a
    |
    b
    |
    c
    |
    print
Siri
```

## ./tests/test_pipeprint_string.out

```
hello
```

## ./tests/test_print_bool.err

```
Fatal error: exception Failure("illegal argument found bool expected int in
    true")
```

## ./tests/test_print_bool.ir

```
int main()
    printi(false)
    printi(true)
Siri
```

## ./tests/test_print_char.err

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_print_char.ir

```
int main()
    char a
    'h'
    |
    a
    print(a)

Siri
```

## ./tests/test_print_float.ir

```
int main()
    float a
    1.23
    |
    a

    printi(a)
Siri
```

## ./tests/test_print_float.out

```
1.23
```

## ./tests/test_print_int.ir

```
int main()
    int a
    2
    |
    a

    printi(a)
Siri
```

## ./tests/test_print_int.out

```
2
```

## ./tests/test_print_string.ir

```
int main()
    string a
    "hello world"
    |
    a

    print(a)
Siri
```

## ./tests/test_print_string.out

```
hello world
```

## ./tests/test_recursion.ir

```
int fib(int n)
    if(n==0 or n==1)
        return 1
    else
        return fib(n-1)
Siri

int main()
    printi(fib(5))
Siri
```

## ./tests/test_recursion.out

```
Fatal error: exception Parsing.Parse_error
```

## ./tests/test_sort_n_search.ir

```
int print_list(int[] l)
    int i

    0
    |
    i

    while(i<length(l))
        printi(l[i])
        i
        |
        +1
        |
        i
    end
    return 0
Siri

int binary_search(int target, int[] nums)
    int n, l, r, mid
    0
    |
    l
    length(nums)
    |
    n
    n
    |
    -1
    |
    r
    while(l<r)
        (l
        |
        +r)
        |
        /2
        |
        mid
        if(target=nums[mid])
            return mid
        fi
        if(target>nums[mid])
            1
            |
            +mid
            |
            l
        else
            mid
            |
            r
        fi
    end
    return -1
Siri

int[] sort(int [] arr)
    int n, idx, i, j, tmp
    length(arr)
    |
    n
    0
    |
    idx
    0
    |
    i
    while(i<n)
        0
        |
        j
        while(j<(n - i - 1))
            if(arr[j+1]<arr[j])
                arr[j]
                |
                tmp
                arr[j+1]
                |
                arr[j]
                tmp
                |
                arr[j+1]
            fi
            1
            |
            +j
            |
            j
        end
        1
        |
        +i
        |
        i
    end
    return arr
Siri

int main()
    int[] sorted_list
    sort([3,5,2,6,1])
    |
    sorted_list

    binary_search(2, sorted_list)
    |
    printi
    return 0
Siri
```

### ./tests/test_str_assign.ir

```
int main()
    string as
    "calloc(30)\n"
    |
    as
    print(as)
    free(as)
    return 0
Siri
```

### ./tests/test_str_cat.ir

```
int main()
```

```
    string str1, str2, str3
    "abcd"
    |
    str1
    "1234"
    |
    str2
    strcpy(str1, str2)
    print(str1)
    print("\n")

    return 0
Siri
```

### ./tests/test_strcmp.ir

```
int main()
    int a
    string str1, str2
    "abcd"
    |
    str1
    "abcd"
    |
    str2
    strcmp(str1, str2)
    |
    a

    printi(a)
    print(str1)
    print("\n")
    print(str2)
    print("\n")

    return 0
Siri
```

### ./tests/test_strlen.ir

```
int main()
    int a
    string str
    "abcd"
    |
    str
    strlen(str)
    |
    a

    printi(a)

    return 0
Siri
```

---

## ./tests/test_tuple_assign.ir

---

```
int print3(string s1, string s2)
    s1
    |
    print

    s2
    |
    print

Siri

int main()
    tuple(string string) a

    ("Hello", " World.\n")
    |
    a
    |
    print3
Siri
```

---

## ./tests/test_tuple_basic.ir

---

```
int print3(string s1, string s2, int i)
    s1
    |
    print

    s2
    |
    print
    i
    |
    printi
Siri

int main()
    tuple a

    ("Hello", " World.\n", 123)
    |
    print3

Siri
```

### ./tests/test_uindeclared.err

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

### ./tests/test_undeclared.ir

```
int main()
    5
    |
    a
    printi(a)

Siri
```

### ./tests/test_while.ir

```
int main()
    int a
    0
    |
    a
    while(a<10)
    a+1
    |
    a
    end
    printi(a)
Siri
```