CSE304 : Compiler Design : Project

Spring 2025
Project Overview

Over the course of the semester, you will be implementing a compiler for a very small subset of the C language. The project will be assigned in phases matching the major phases of compilation presented in class.

Phases

The phases include:

- Lexical Analysis
- Symbol Table Management
- Syntax Analysis (parsing)
- Semantic Analysis/Intermediate Code Generation
- Target Code Generation

Language Description

This paper will describe the features of the language (called *RASCL*, 'Realistically, A Small C Language') you are expected to implement.

The language to be implemented is similar in structure and semantics to C. It should support the following:

- Comments (/* … */)
- Integer and floating point variables
- Integer and floating point constants
- Simple assignment
- Arithmetic expressions available in C including the following operators:
    - Addition (+)
    - Subtraction (-)
    - Multiplication (*)
    - Division (/)
- 'if/else' statements
- 'while' loops
- Statement groups enclosed in braces ({})
- 'print' and 'read' statements that print or read a single integer or floating point value

- Relational expressions including the following operators:
    - Equal (==)
    - Less Than (<)
    - Less Than or Equal (<=)
    - Greater Than (>)
    - Greater Than or Equal (>=)
    - Not Equal (!=)
- Logical expressions including the following operators:
    - Not (!)
    - Or (||)
    - And (&&)
- Arrays
- Functions and function calls

As each project phase is assigned, more specific requirements will be provided as appropriate to the phase.

Example Code

Following are some short programs for which your final compiler should be capable of parsing and generating code:

```
/* Prog 1 */

int a, b;
float c, d;

main()
{
  a = 5;
  b = a + 2 * (b + a);
  print bignum
}
```

```
/* Prog 2 */

int a;
int b;
int bignum;
int sqr;
float avg;
int numbers[5];
int even;
int index;

function int isEven (int val) {
   if (val == 0) {
     return 1
   } else {
    if (val == 1) {
     return 0
     } else {
       return isEven((val - 2))
     }
   }
}


function void outputArray(int nums[], int length) {
   int index;
   index = 0;
   while (index < length) {
     print nums[index]
  }
}
function int dummy()
{
  index = 0;
  even = 0;

  numbers[0] = 50;
  numbers[1] = -52;
  numbers[2] = -12;
  numbers[3] = 31;
  numbers[4] = -17;
  call outputArray(numbers, 5);

  index = 0;
  while !(numbers[index] == -101) && (index < 5) {
```

```
    if (numbers[index] > 0) && (isEven(numbers[index]) == 1) {
       print numbers[index]
    }
  }

}

main() {
  print even
}
```

Note that the above samples are only to give you a feel for rascl and its differences from the C language. I will give a more detailed grammar with the phase on parsing (syntax analysis).

The final code generated will depend on whether we reach final code generation or not. If we only have time to implement intermediate code generation, then the output of a compile will comply with an intermediate form that I will share a little later in the semester. If we implement a code generator, it should output MIPS assembler that can be run in the MARS or QtSpim simulators.

Development Environment/Implementation Languages

Regarding languages you may use to write your compiler… I am allowing any language that I have a basic understanding of and can build on my laptop. (Among languages I understand are: C, C++, Java, Javascript, Typescript, Python, Perl (not recommended), and APL (don't ask)). Again, I must be able to easily build and test your code. To that end, I'm hoping to present a short intro to Docker which is a system used to set up thin, light weight virtual machines. I strongly recommend you learn to use this and then build your code in a Docker Container. Then submit a Dockerfile along with your source that builds the container environment. That will make it easy for us to assure I can test your code and there will be less delays in grading. More about this later on.

Rough Schedule for the Project phases [all dates approximate]

Phase	Issue Date	Due Date
Lexical Analyzer	7-Mar-2025	23-Mar-2025
Symbol Table Manager	24-Mar-2025	2-Apr-2025
Parser	3-Apr-2025	23-Apr-2025
Semantic Analzyer/ICG	24-Apr-2025	13-May-2025
Code Generator	14-May-2025	28-May-2025