Compilers Project

|  |  |  |
| --- | --- | --- |
| Name | SEC | BN |
| Ahmed Mohamed Mustafa | 1 | 3 |
| Abd El Rahman Ahmed Abbas | 1 | 29 |
| Ali Abd ellatif Elseddeek | 2 | 1 |
| Mohamed Ashraf | 2 | 10 |

**Project Overview**

We have created a mini C language version that supports

* Integer and Float variables and arrays
* Integer and Float Constants
* If, If-Else, While, Do-While, Repeat-Until, For, Switch Statements
* Functions (without recursion)
* Semantic Errors and Warnings Checking

We have supplied our compiler with a simple Text Editor that we have created. It allows the user to edit multiple files in multiple tabs, highlight the reserved keywords and the compilation errors.

The text editor first call syntax.exe to check the syntax of the file, if it is correct it then calls compiler.exe to compile the file into **MIPS assembly.**

**Parse Tree**

The compiler first constructs a tree structure that describes the input program.

**Semantic Analyzer**

The compiler checks for 15 types of semantic errors and 3 semantic warnings (they are very clearly described at semantics.c file).

**Symbol Table**

The compiler has global scope that contains array of nested inner scopes. Each scope has its own symbol table.

**Code Generation**

The compiler makes a DFS of the parse tree to generate the final code.

Our Quadruples format is very close to MIPS assembly.

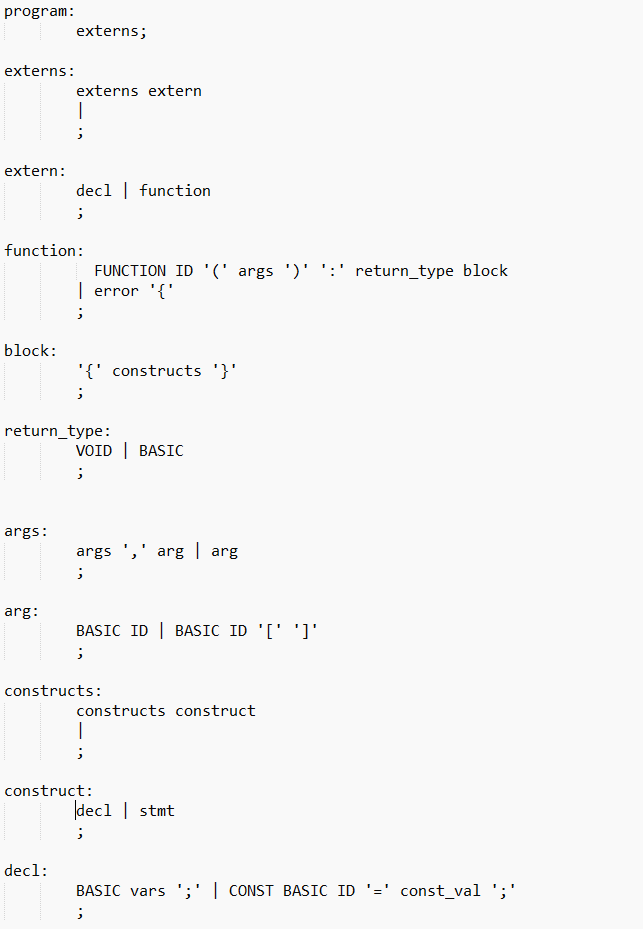
**Tools and Technologies**

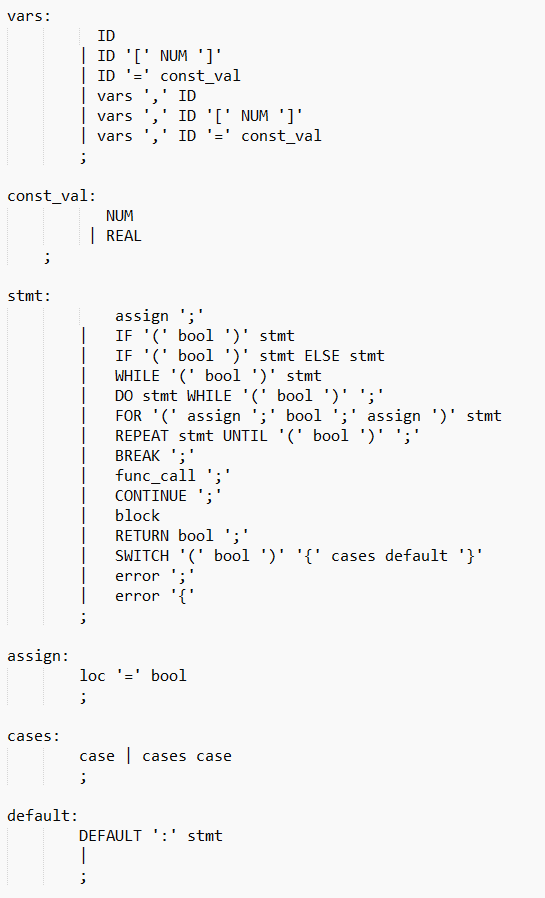
* Windows Operating System
* Lex and Yacc (flex and bison respectively) to create a lexer and a parser in C.
* GNU GCC compiler to compile the generated files and our code.
* Python 3.5 to create the Text Editor (using tkinter GUI library)
* Sublime Text Editor

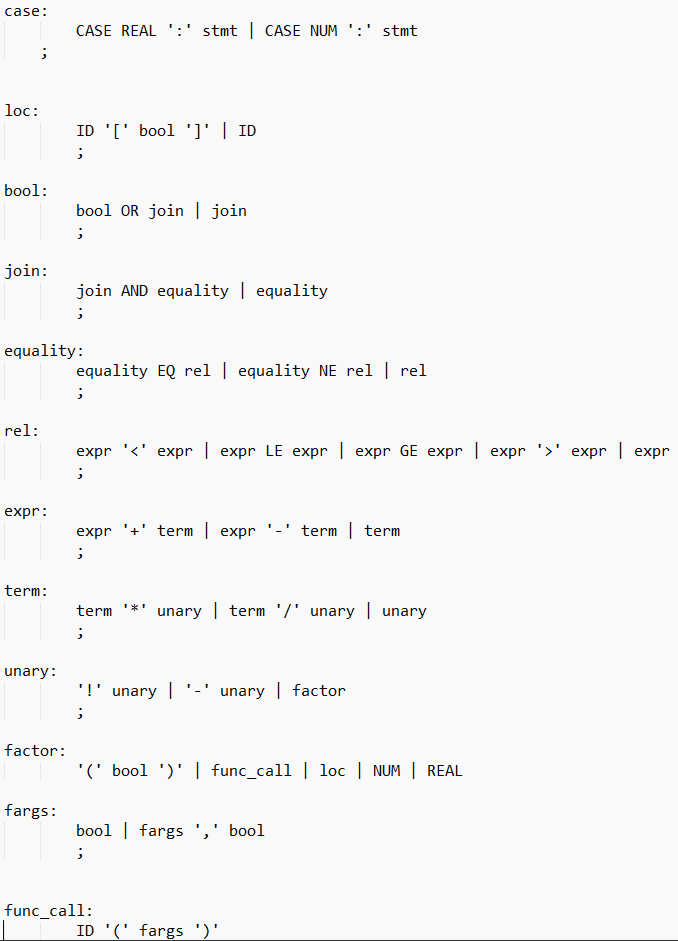
**Tokens**

|  |  |
| --- | --- |
| NUM | Integer Value |
| REAL | Float Value |
| ID | Identifier (string) |
| BASIC | Reserved Words int or float |
| BREAK | Reserved Word break |
| CONTINUE | Reserved Word continue |
| VOID | Reserved Word void |
| IF | Reserved Word if |
| FOR | Reserved Word for |
| WHILE | Reserved Word while |
| DO | Reserved Word do |
| SWITCH | Reserved Word switch |
| CASE | Reserved Word case |
| ELSE | Reserved Word else |
| FUNCTION | Reserved Word function |
| CONST | Reserved Word const |
| DEFAULT | Reserved Word default |
| LE | <= |
| GE | >= |
| EQ | == |
| NE | != |
| AND | && |
| OR | || |

**Grammar**







**Quadruples**

**Registers**

All register names start with dollar sign. Integer registers are $i(number), $a(number). Float registers are $f(number), $d(number), $a, $d are used as accumulators during the evaluation of complex expressions. $sp is the stack pointer. $re.i is a special register for integer return value. $re.f is a special register for float return value.

|  |  |
| --- | --- |
| add $i1, $i2, $i3 | $i1 = $i2 + $i3 |
| sub $i1, $i2, $i3 | $i1 = $i2 - $i3 |
| and $i1, $i2, $i3 | $i1 = $i2 & $i3 |
| or $i1, $i2, $i3 | $i1 = $i2 | $i3 |
| neg $i1, $i2 | $i1 = - $i2 |
| not $i1, $i2 | $i1 = ! $i2 |
| mul $i1, $i2, $i3 | $i1 = $i2 \* $i3 |
| div $i1, $i2, $i3 | $i1 = $i2 / $i3 |
| slt $i1, $i2, $i3 | $i1 = 1 if $i2 < $i3 else 0 |
| sgt $i1, $i2, $i3 | $i1 = 1 if $i2 > $i3 else 0 |
| slte $i1, $i2, $i3 | $i1 = 1 if $i2 <= $i3 else 0 |
| sgte $i1, $i2, $i3 | $i1 = 1 if $i2 >= $i3 else 0 |
| seq $i1, $i2, $i3 | $i1 = 1 if $i2 == $i3 else 0 |
| sne $i1, $i2, $i3 | $i1 = 1 if $i2 != $i3 else 0 |
| beq $i1, $i2, label | Branch to label if $i1 == $i2 |
| bne $i1, $i2, label | Branch to label if $i1 != $i2 |
| j label | Unconditional jump to label |
| mov $i1, $i2 | Copy $i2 into $i1 |
| addi $i1, $i2, imm | $i1 = $i2 + imm |
| call label | Jumps to the function that starts at label |
| li $i1, imm | Load immediate integer value into $i1 |
| cvt.i.f $f1, $i1 | Converts the integer value in $i1 to a float value and stores it in $f1 |
| cvt.i.f $i1, $f1 | Converts the float value in $f1 to a integer value and stores it in $i1 |
| st $1, $2, $3 | Stores the value in $i1 into memory location [$i2 + $i3] |
| ld $1, $2, $3 | Loads the value in memory location [$i2 + $i3] into $i1 |
| ret | Return to the address on top of the stack |

There is another set of similar instruction but for float point operations they operate on float registers and has (.f) suffix.