C Plus Minus

Table of Contents

Introduction

The designed language is a `C` like programming language.

Sample program:

```
const int a = 5;
float b = 6;
print ("Operations:");
if (a == 5) {
    print ("a is 5");
}
else {
    if (b == 6) {
        print ("b is 6");
    }
    else {
        print ("b is not 6");
    }
}
exit;
```

Run Steps

- `yacc -d main.y`: create y.tab.h and y.tab.c
- `lex main.l`: create lex.yy.c
- `gcc -g lex.yy.c y.tab.c -o main`: create main
- `./main`: run main

For convenience, the above steps are combined in a makefile. To run the makefile, type `make <test case name>` in the terminal.

Tools and Technologies

1. Lex: It breaks down the input text into a sequence of tokens, which are then passed on to the parser for further processing.
2. Yacc: It takes a sequence of tokens as input and produces a parse tree or an abstract syntax tree (AST) that represents the structure of the input according to the grammar rules.

Tokens

Token	Regex	Description
DIGIT	[0-9]	Number between 0 and 9.

ALPHABET	[a-zA-Z]	Upper case or lower case English letter.
ALPHANUM	[0-9a-zA-Z]	Digit, upper case letter, or lower case letter.
SPACE	[\r\t]	Single space or tab.
NEW_LINE	\n	New line.
INLINE_COMMENT	\\/.*	Single line comment.
MULTILINE_COMMENT	\/*.**\/	Multi-line comment.
arithmeticOps	[/+*%-]	Arithmetic operators (+, -, *, /, %).
bitwiseOps	[&^~|]	Bit-wise operators (AND, OR, NOT, XOR).
endOfStatement	[;]	Semi-colon to mark the end of any statement.
punctuators	[()={}:,]	Language punctuators.
TRUE	[tT]rue | 1 | [yY]es	True value.
FALSE	[fF]alse| 0 | [nN]o	False value.

Syntax

Data Types

Tha language supports the following data types:

- Integer
- Float
- Boolean
- String

It supports modifiers like `const` as well.

```
const int a = 10;
int b = 20;
float c = 10.5;
bool d = true;
string e = "Hello World";
```

Operators

The language supports the common operators in C.

```
// Arithmetic operators
a = b + c;
a = b - c;
a = b * c;
a = b / c;
a = b % c;
```

```
// Bitwise operators
a = b & c;
a = b | c;
a = b ^ c;
a = ~b;
// Logical operators
a = b && c;
a = b || c;
a = !b;
// Relational operators
a = b == c;
a = b != c;
a = b > c;
a = b >= c;
a = b < c;
a = b <= c;
// Shift operators
a = b << c;
a = b >> c;
```

Conditional Statements

The language supports the if-else, if-elif-else, and switch-case statements.

```
int a = 10;
// if statement
if (a == 10) {
    print("if");
    print("another if");
}
elif (a == 11) {
    print("elif");
    print("another elif");
}
else {
    print("else");
    print("another else");
    if (a == 10) {
        print("if");
        print("another if");
    }
    else {
        print("else");
        print("another else");
    }
}
if (a == 10) {
    print("if");
    print("another if");
}
elif(a == 11) {
    print("else");
```

```
    print("another else");
}
// switch-case statement
switch (a) {
    default:
        print("default");
        break;
}
switch (a) {
    case 1:
        print("1");
        break;

    case 2:
        print("2");
        break;

    case 3:
        print("3");
        break;
}

switch (a) {
    case 1:
        print("1");
        break;

    case 2:
        print("2");
        break;

    case 3:
        print("3");
        break;

    default:
        print("default");
        break;
}
```

Loops

The language supports the while, for, and repeat-until loops.

```
// while loop
a = 0;
while (a < 20) {
    print(a);
    a = a + 1;
}
print(a);
while (a < 20) {
```

```
    if (a == 10) {
        print(a);
    }
    a = a + 1;
}
// for loop
for (a=2 ; a<10; a = a+1 ) {
    print(a);
}
for (a=2 ; a<10; a= a+1 ) {
    print(a);
    b = a;
    while (b < 10) {
        if (b == 5) {
            print("hi");
            print(b);
        }

        b = b + 1;
    }
}
// repeat-until loop
a = 0;
repeat {
    print(a);
    a = a + 1;
    print(a);
} until (a == 1);
repeat {
    print(a);
    a = a + 1;
    if (a == 1) {
        print(a);
    }
} until (a == 1);
```

Functions

The language supports functions with and without parameters.

```
int y (){
    print("y");
    return 1;
}
int x(int a, int b) {
    print("add");
    return a + b;
}
x(1, 2); // function call
a = y(); // function call and assignment
```

N.B.: you can't define a function inside any scope.

Enumerations

The language supports enumerations.

```
enum Color{
    RED=10,
    GREEN,
    BLUE=12,
    RED
};
{
    Color c1;
    Color c2=RED;
    Color c3=3+5;
}
```

Quadruples

Procedures

Quadruples	Description	ARG1	ARG2	RES
PROC	Start of a procedure	procedure name		
ENDPROC	End of a procedure	procedure name		
CALL	Calls a procedure, handles all the stuff related to the PC	procedure name		
RET	Return from a procedure, handles all the stuff related to the PC			

Enums

Quadruples	Description	ARG1	ARG2	RES
ENUM	Start of an enum	enum name		
ENDENUM	End of an enum	enum name		

Variables

Quadruples	Description	ARG1	ARG2	RES
PUSH	Push to the stack frame	Identifier/Expr		
POP	Pop from the stack frame	Identifier/Expr		

Quadruples	Description	ARG1	ARG2	RES
CAST	Cast the type of the var on the top of the stack to the type of the var to be pop into			

Branching & Jumps

Quadruples	Description	ARG1	ARG2	RES
JMP	Unconditional jump to the label	label		
JF	Jumps to the label if the result of the last operation was false	label		

Arithmetic Operations

Quadruples	Description	ARG1	ARG2	RES
NEG	Get the opposite sign of an expression			
COMPLEMENT	Get the complement of an expression			
NOT	Get the bitwise not of an expression			
ADD	Add two numbers			
SUB	Subtract two numbers			
MUL	Multiply two numbers			
DIV	Divide two numbers			
MOD	Modulus two numbers			

Bitwise Operations

Quadruples	Description	ARG1	ARG2	RES
BITWISE_OR	Get the bitwise or of two numbers			
BITWISE_AND	Get the bitwise and of two numbers			
BITWISE_XOR	Get the bitwise xor of two numbers			
SHL	Shift left the number			
SHR	Shift right the number			

Logical Operations

Quadruples	Description	ARG1	ARG2	RES
LOGICAL_OR	Get the logical or of two numbers			
LOGICAL_AND	Get the logical or of two numbers			
EQ	Check if two numbers are equal			

Quadruples	Description	ARG1	ARG2	RES
NEQ	Check if two numbers are not equal			
GT	Check if the first number is greater than the second			
GEQ	Check if the first number is greater than or equal the second			
LT	Check if the first number is less than the second			
LEQ	Check if the first number is less than or equal the second			

Production Rules

- program → statements | functionDef | statements program | functionDef program
- statements → statement | codeBlock | controlstatement | statements codeBlock | statements statement | statements controlstatement
- codeBlock → { statements } | { }
- controlstatement → ifCondition | whileLoop | forLoop | repeatUntilLoop | switchCaseLoop
- statement → assignment | exp | declaration | EXIT | BREAK | CONTINUE | RETURN | RETURN exp | PRINT (exp) | PRINT (STRING)
- declaration → dataType IDENTIFIER | dataType assignment | dataIdentifier declaration
- assignment → IDENTIFIER = exp | IDENTIFIER = STRING | enumDeclaration | enumDef
- exp → term | functionCall | - term | '~' term | NOT term | exp '+' exp | exp '-' exp | exp '*' exp | exp '/' exp | exp '%' exp | exp '|' exp | exp '&' exp | exp '^' exp | exp SHL exp | exp SHR exp | exp EQ exp | exp NEQ exp | exp GT exp | exp GEQ exp | exp LT exp | exp LEQ exp | exp AND exp | exp OR exp
- term → NUMBER | FLOAT_NUMBER | TRUE_VAL | FALSE_VAL |IDENTIFIER | (exp)
- dataIdentifier → CONST
- dataType → INT_DATA_TYPE | FLOAT_DATA_TYPE | STRING_DATA_TYPE | BOOL_DATA_TYPE | VOID_DATA_TYPE
- ifCondition → IF (exp) codeBlock | IF (exp) codeBlock ELSE codeBlock | IF (exp) codeBlock ELIF (exp) codeBlock | IF (exp) codeBlock ELIF (exp) codeBlock ELSE codeBlock
- whileLoop → WHILE (exp) codeBlock
- forLoop → FOR (assignment ; exp ; assignment) codeBlock
- repeatUntilLoop → REPEAT codeBlock UNTIL (exp) ;

- case → CASE exp : statements | DEFAULT : statements
- caseList → caseList case | case
- switchCaseLoop → SWITCH (exp) { caseList }
- functionArgs → dataType IDENTIFIER | dataType IDENTIFIER , functionArgs
- functionParams → term | term , functionParams
- functionDef → dataType IDENTIFIER (functionArgs) codeBlock | dataType IDENTIFIER '(' ')' codeBlock
- functionCall → IDENTIFIER (functionParams) | IDENTIFIER ( )
- enumDef → ENUM IDENTIFIER { enumBody }
- enumBody → IDENTIFIER | IDENTIFIER = exp | enumBody , IDENTIFIER | enumBody , IDENTIFIER = exp
- enumDeclaration → IDENTIFIER IDENTIFIER | IDENTIFIER IDENTIFIER = exp

Semantic Errors

Semantic Errors

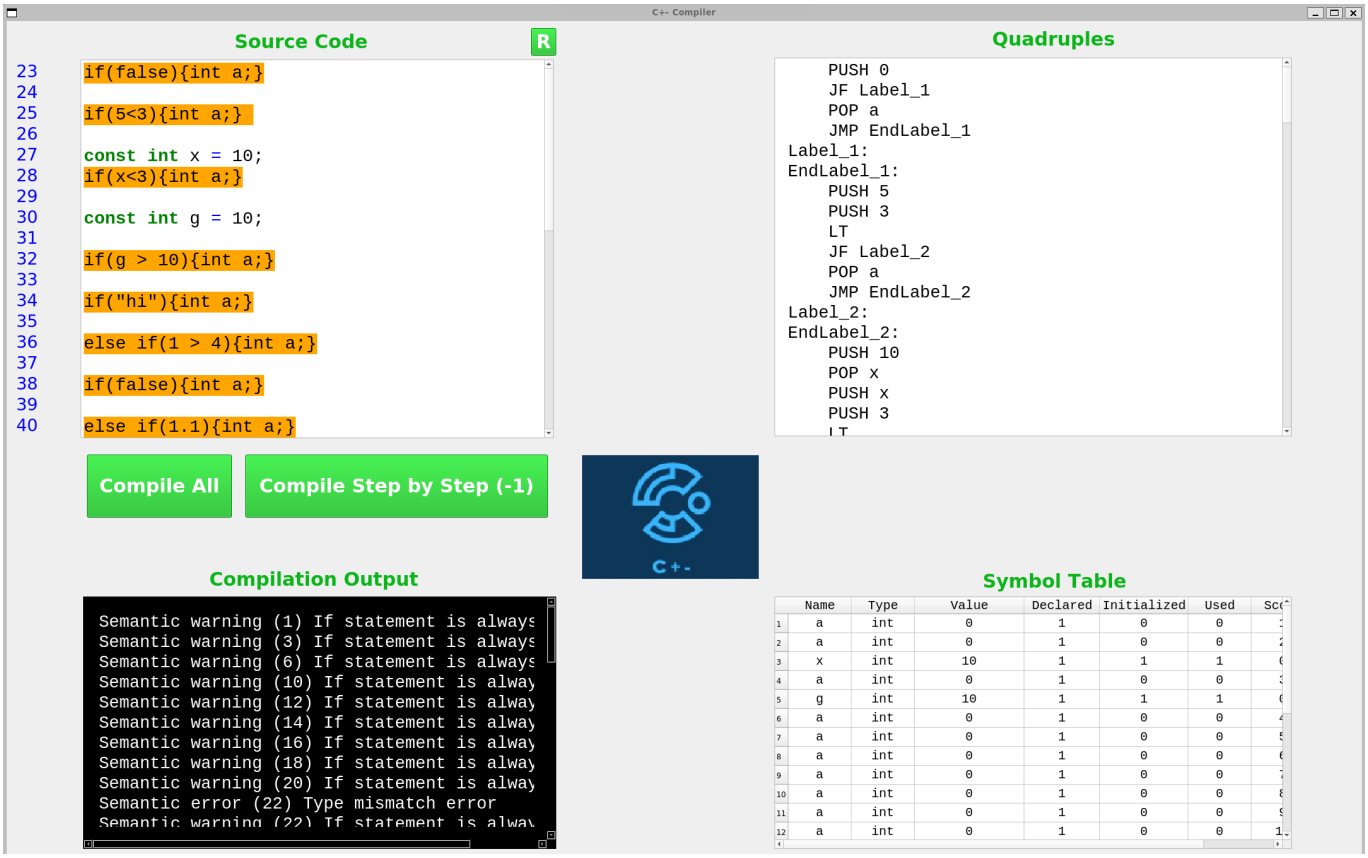
TYPE_MISMATCH UNDECLARED
UNINITIALIZED UNUSED
REDECLARED
CONSTANT
OUT_OF_SCOPE

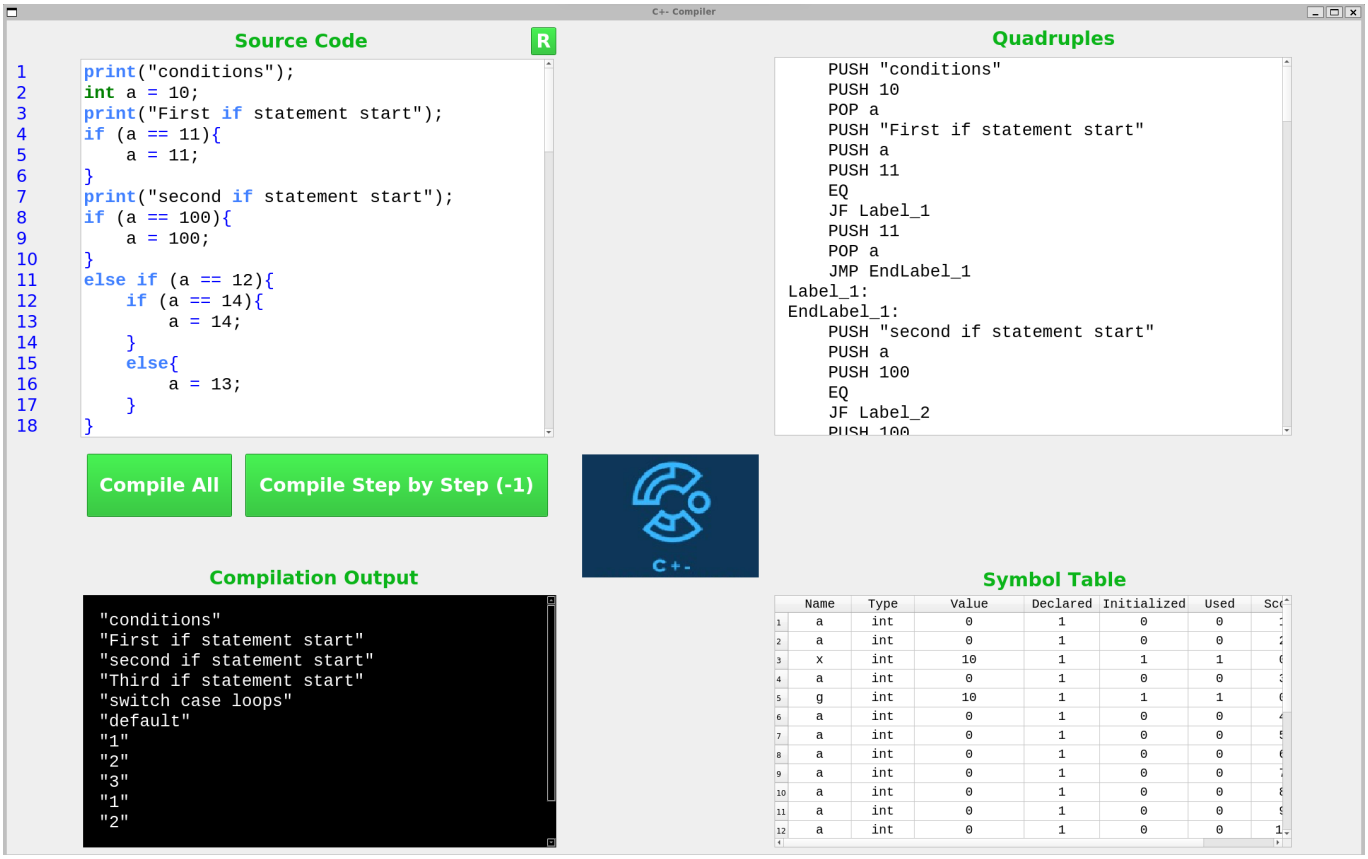
Desktop Application

A desktop application is developed using PyQt5 to provide a user interface for the compiler. The application allows the user to select a file from the file system and compile it. The application will then display the generated quadruples, the symbol table, and the results of the executed code.

- The main functionalities:
    - Open an existing file (Ctrl + O)
    - Write a new file in the text editor
    - Save the file (Ctrl + S)
    - Compile the file in one step
    - Compile the file step by step
    - Display the generated quadruples
    - Display the symbol table
    - Display the results of the executed code
    - Highlight the syntax errors in red
    - Highlight the semantic errors in yellow

- Highlight the semantic warnings in orange
- Remove the highlights (Ctrl + R)





C+- Compiler

Source Code

```
int a;
print("while loops");
a = 0;
while (a < 20)
{
    print(a);
    a = a + 1;
}
print(a);
while (a < 20)
{
    if (a == 10)
    {
        print(a);
    }
    a = a + 1;
}
print(a);
```

Quadruples

```
    PUSH "conditions"
    PUSH 10
    POP a
    PUSH "First if statement start"
    PUSH a
    PUSH 11
    EQ
    JF Label_1
    PUSH 11
    POP a
    JMP EndLabel_1
Label_1:
EndLabel_1:
    PUSH "second if statement start"
    PUSH a
    PUSH 100
    EQ
    JF Label_2
    PUSH 100
```

Compile All | Compile Step by Step (-1)

Compilation Output

```
"conditions"
"First if statement start"
"second if statement start"
"Third if statement start"
"switch case loops"
"default"
"1"
"2"
"3"
"1"
"2"
```

Symbol Table

	Name	Type	Value	Declared	Initialized	Used	Sc
1	a	int	0	1	0	0	1
2	a	int	0	1	0	0	2
3	x	int	10	1	1	1	0
4	a	int	0	1	0	0	3
5	g	int	10	1	1	1	0
6	a	int	0	1	0	0	4
7	a	int	0	1	0	0	5
8	a	int	0	1	0	0	6
9	a	int	0	1	0	0	7
10	a	int	0	1	0	0	8
11	a	int	0	1	0	0	9
12	a	int	0	1	0	0	1

C+- Compiler

Source Code

```
print("arithmetic");
int a = 10;
int b = 10;
int c = 2;
print(c);
print(-a == -10); // -10
print(-3 - 4);
print(a + b == 20);             // 20
print(a - b == 0);              // 0
print(a / b == 1);              // 1
print(a % c == 0);              // 0
print(a *b - b + a / b == 91); // 91
print(100 - 5);
print(-100 - 5 + 5);
//////////////////////////////
print("bitwise");
a = 10;
c = 2;
```

Quadruples

```
    PUSH "arithmetic"
    PUSH 10
    POP a
    PUSH 10
    POP b
    PUSH 2
    POP c
    PUSH a
    NEG
    PUSH 10
    NEG
    EQ
    PUSH 3
    NEG
    PUSH 4
    SUB
    PUSH a
    PUSH b
    ADD
```

Compile All | Compile Step by Step (-1)

Compilation Output

```
"arithmetic"
2
1
-7
1
1
1
1
1
95
-100
```

Symbol Table

	Name	Type	Value	Declared	Initialized	Used	Sc
1	a	int	0	1	0	0	1
2	a	int	0	1	0	0	2
3	x	int	10	1	1	1	0
4	a	int	0	1	0	0	3
5	g	int	10	1	1	1	0
6	a	int	0	1	0	0	4
7	a	int	0	1	0	0	5
8	a	int	0	1	0	0	6
9	a	int	0	1	0	0	7
10	a	int	0	1	0	0	8
11	a	int	0	1	0	0	9
12	a	int	0	1	0	0	1

C+- Compiler

Source Code

```
1  int x = 2;
2
3  int a = 3;
4
5  int y
```

Quadruples

```
PUSH 2
POP x
PUSH 3
POP a
POP y
```

Compile All | Compile Step by Step (-1)

Compilation Output

```
Syntax error (5) Near line 5: syntax error
```

Symbol Table

	Name	Type	Value	Declared	Initialized	Used	Scop
1	x	int	2	1	1	0	0
2	a	int	3	1	1	0	0
3	y	int	0	1	0	0	0

C+- Compiler

Source Code

```
1  int x = 2;
2
3  int a = 3;
4
5  int z;
```

Quadruples

```
PUSH 2
POP x
PUSH 3
POP a
POP z
```

Compile All | Compile Step by Step (-1)

Compilation Output

```
Semantic error (5) Unused variable x
Semantic error (5) Unused variable a
Semantic error (5) Unused variable z
```

Symbol Table

	Name	Type	Value	Declared	Initialized	Used	Scop
1	x	int	2	1	1	0	0
2	a	int	3	1	1	0	0
3	z	int	0	1	0	0	0