“ CREATING A COMPILER FROM C TO LLVM ”

Kurisunkal Marcin, Mayank Murali, Atul V john

marcin.kurisunkal2015@vit.ac.in, mayank.murali2015@vit.ac.in, atul.john2015@vit.ac.in

ABSTRACT

The aim of this project is to create a compiler capable of processing C code and converting it into Low Level Virtual Machine (LLVM) assembly code. This is achieved through the use of the LEX Lexical Analysis Package and YACC Parsing Packing on Linux to get to the optimized intermediate three address code. An assembler coded on Python is used to convert the intermediate code to the LLVM assembly.

Keywords: Compiler, C, LLVM, LEX, YACC, Python, Linux,Optimized.

INTRODUCTION

This project primarily uses the Unix softwares named Lex and Yacc (Yet Another Compiler Compiler) as the lexical analysis and parser packages. Yacc is a Look Ahead Left-To-Right (LALR) parser generator which generates a LALR parser based on an analytic grammar written similar to Backus-Naur Form (BNF). Lex is lexical analyser generator, commonly used with Yacc. Lex reads an input stream specifying the lexical anaylser and outputs source code implementing the lexer in C.

We have used these to generate the lexer and parser by tabulating the symbols and syntax of the input language (in this case C) and the output language (in this case LLVM) in the required format, and then compiling using these packages and GCC. Now, C is a very large and complex language. As a consequence, it is entirely likely that there will be errors in what our ‘C language’ compiler accepts. However, as far as a proof-of-concept project goes, it will do it’s job.

IMPLEMENTATION

First the YACC and LEX packages are downloaded from the Ubuntu Library. The code is compiled and run to obtain the output. Finally a python code is written to convert the obtained 3 – Address code to LLVM assemble code.

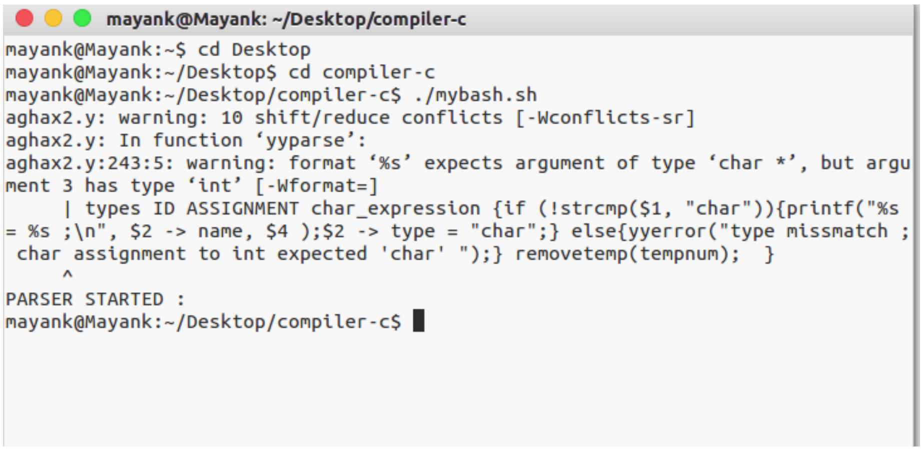
Here is a code snippet to identify types of characters, keywords, tokens etc

```
"int"                                   { yylval.intval = strdup(yytext);return(INT); }

"char"                                  { yylval.intval = strdup(yytext);return(CHAR); }

"if"                                    { return(IF); }

"else"                                  { return(ELSE); }

"and"           { return(AND); }

"not"           { return(NOT); }

"while"          { return WHILE;}

"return"          { return RETURN; }

"or"            { return(OR); }

"("                                     { return PAR_OPEN; }

")"                                     { return PAR_CLOSE; }

";"                                     { return SEMICOLON; }

"{"                                     { return CB_OPEN; }

"}"                                     { return CB_CLOSE; }

"+"                                     { return PLUS; }

"-"                                     { return MINUS; }

"*"                                     { return ASTERISK; }

"/"                                     { return SLASH; }

"="                                     { return ASSIGNMENT; }

"<="                                    { return LESS_EQUAL; }

"<"                                     { return LESS; }

">="                                    { return MORE_EQUAL; }

">"                                     { return MORE; }
```

```
"=="                                    { return EQUAL; }

"!="                                    { return NOT_EQUAL; }

","             { return COMMA;}

"""             {return QUOT;}
```

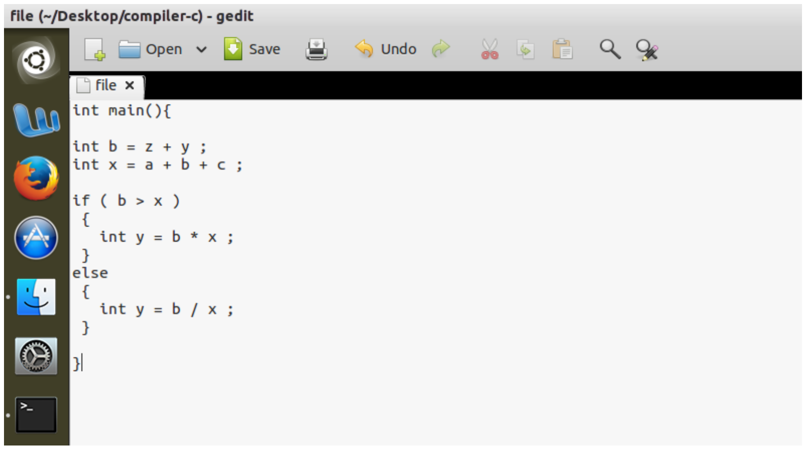
RESULTS

Screenshot #1 : Compiling the code by invoking mybash.sh file.



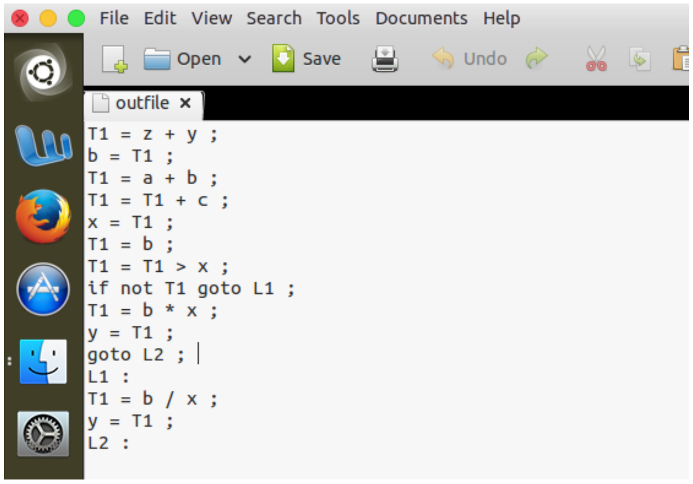
```
mayank@Mayank:~$ cd Desktop
mayank@Mayank:~/Desktop$ cd compiler-c
mayank@Mayank:~/Desktop/compiler-c$ ./mybash.sh
aghax2.y: warning: 10 shift/reduce conflicts [-Wconflicts-sr]
aghax2.y: In function ‘yyparse’:
aghax2.y:243:5: warning: format ‘%s’ expects argument of type ‘char *’, but argu
ment 3 has type ‘int’ [-Wformat=]
     | types ID ASSIGNMENT char_expression {if (!strcmp($1, "char")){printf("%s
= %s ;\n", $2 -> name, $4 );$2 -> type = "char";} else{yyerror("type missmatch ;
 char assignment to int expected 'char' ");} removetemp(tempnum);  }
     ^
PARSER STARTED :
mayank@Mayank:~/Desktop/compiler-c$ 
```

Screenshot #2 : The test code written in C language.



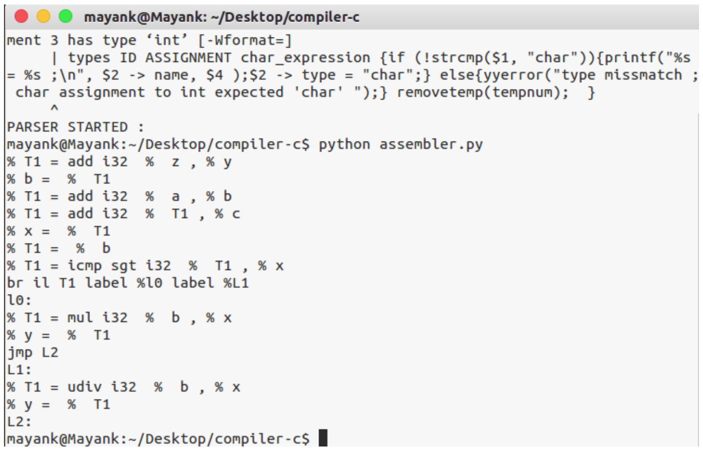
```
int main(){

int b = z + y ;
int x = a + b + c ;

if ( b > x )
 {
   int y = b * x ;
 }
else
 {
   int y = b / x ;
 }

}
```

Screenshot #3 : Output obtained after running the code. The C code has been converted to 3-Address code for further simplification.



```
T1 = z + y ;
b = T1 ;
T1 = a + b ;
T1 = T1 + c ;
x = T1 ;
T1 = b ;
T1 = T1 > x ;
if not T1 goto L1 ;
T1 = b * x ;
y = T1 ;
goto L2 ;
L1 :
T1 = b / x ;
y = T1 ;
L2 :
```

Screenshot #4 : The LLVM assembly code obtained from the 3-Address code which is run in Python.



```
ment 3 has type ‘int’ [-Wformat=]
     | types ID ASSIGNMENT char_expression {if (!strcmp($1, "char")){printf("%s
= %s ;\n", $2 -> name, $4 );$2 -> type = "char";} else{yyerror("type missmatch ;
 char assignment to int expected 'char' ");} removetemp(tempnum);  }
     ^
PARSER STARTED :
mayank@Mayank:~/Desktop/compiler-c$ python assembler.py
% T1 = add i32  %  z , % y
% b =  %  T1
% T1 = add i32  %  a , % b
% T1 = add i32  %  T1 , % c
% x =  %  T1
% T1 =  %  b
% T1 = icmp sgt i32  %  T1 , % x
br il T1 label %l0 label %L1
l0:
% T1 = mul i32  %  b , % x
% y =  %  T1
jmp L2
L1:
% T1 = udiv i32  %  b , % x
% y =  %  T1
L2:
mayank@Mayank:~/Desktop/compiler-c$
```

CONCLUSION

Created a Compiler to convert C to LLVM assembly.

REFERENCES

1) Principles of Compiler Design, by Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman
2) Compiler Construction Principles and Practice, by Kenneth C. Louden