3 ADDRESS CODE GENERATION

REPORT SUBMITTED FOR SESSIONAL TEST II OF

COMPILER DESIGN

CO308

IN THE SCHOOL OF ENGINEERING
TEZPUR UNIVERSITY

By

Adil Bin Bhutto (CSB17016)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
TEZPUR UNIVERSITY
TEZPUR, INDIA

APRIL 2020

Question: Describe briefly in English a simple loop-construct in a hypothetical programming language, and describe some types of statements such as assignment statement, arithmetic expression, use of simple variables and constants. Write a Context Free Grammar (CFG) for the above. Write a *bison* program to parse inputs in that language. Include semantic actions to produce 3-address code intermediate code.

Description in English: A while loop will begin with the keyword *while* followed by a conditional statement enclosed with open braces and bunch of arithmetic expression assigned to a variable. These arithmetic expressions will be enclosed with curly braces and this code block is subjected to iterate depending on the condition given. Opening curly brace shows the starting of while loop and closing curly brace separates the inside code block from the rest part.

e.g.

```
while (a <= 10 + c ) {
        c = 45 / 5 + d;
        a = 8 + c / 3;
}
```

At the end of this report we shall see how this loop construct gets converted to **3AC** code with the help of bison and flex.

Context Free Grammar: We have taken an ambiguous grammar but here trick is that we explicitly write the precedence in bison program to eliminate any kind of ambiguity.

D → w(C){S}
C → iXE
X → == | != | <= | >= | < | >
S → i=E;S | ε
E → E+E | E-E | E*E | E/E | (E) | i | n

Lex file: After lexical analysis, these tokens as per written program are passed into bison program for further processing. Bison constructs a parse tree for syntax analysis and in our case, we are producing 3AC code as output.

```
%%
while                       { return WHILE; }
[A-Za-z]([A-Za-z]|[0-9])*   { return ID; }
[0-9]+                      { return NUM; }
"=="                        { return EQUAL; }
"!="                        { return NOT_EQUAL; }
">="                        { return GT_EQUAL; }
"<="                        { return LT_EQUAL; }
"<"                         { return LT; }
">"                         { return GT; }
"$"                         { yyterminate(); }
[ \t\n\r]                   { ; }
.                           { return yytext[0]; }
%%
```

Bison Program with Semantic Actions:

```
/* Tokens are separated by a single space */
%token   WHILE ID NUM EQUAL NOT_EQUAL LT_EQUAL GT_EQUAL LT GT
%right  '='
%left   '+' '-'
%left   '*' '/'

%%
 /* The top of the parser stack is pointed by variable top and push function pushes the content of yytext
  into the stack. insideVariable keeps track of temporary variable which to be generated */

D : WHILE { printf("\nStart: \n");} '('C')' { printf("\t%s = %s %s %s\n",insideVariable, Stack[top-2],
    Stack[top-1], Stack[top]);  update_stack();} '{' S '}' { printf("\tgoto Start\n"); printf("End:\n\t end of
    while loop\n\n");}
C : ID {push();} X E { printf("\t%s = %s %s %s\n",insideVariable, Stack[top-2], Stack[top-1],
    Stack[top]); update_stack(); }
X : EQUAL          {push();}
   | NOT_EQUAL {push();}
   | LT_EQUAL    {push();}
   | GT_EQUAL    {push();}
   | LT                {push();}
   | GT                {push();}
   ;
S  : ID {push();} '=' {push();} E {    printf("\t%s = %s\n", Stack[top-2], Stack[top]); top-=2; } ';' S
   |; /* empty */
   ;
E  : E '+' {push();} E {GenerateCode();}
    | E '-' {push();}  E {GenerateCode();}
    | E '*' {push();} E {GenerateCode();}
    | E '/' {push();}  E {GenerateCode();}
    | '(' E ')'
    | ID   {push();}
    | NUM {push();}
    ;
%%

void GenerateCode() {
   printf("\t%s = %s %s %s\n", insideVariable, Stack[top-2], Stack[top-1], Stack[top]);
   update_stack()
}
void update_stack() {
   top-=2;
   strcpy(Stack[top], insideVariable);
   insideVariable[1]++;
}
void push() {
   strcpy(Stack[++top],yytext);
}
```

Result:

e.g.

Given input program:

```
while (a <= 10 + c ) {
        c = 45 / 5 + d;
        a = 8 + c / 3;
}$
```

3AC output:

```
Start:
        t0 = 10 + c
        t1 = a <= t0
        t2 = not t1
        if t2 goto End
        t3 = 45 / 5
        t4 = t3 + d
        c = t4
        t5 = c / 3
        t6 = 8 + t5
        a = t6
        goto Start
End:
        end of while loop
```