CD LAB 7 Intermediate Code Generator using LEX and YACC

Bhushan Sonawane
BE E 66 (BATCH 3)

yacc.y

```
%{
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <ctype.h>
char vartype[10];
char Tempname[10];
int newAssignment = 0;
struct STable{
char label[10];
char type[10];
int size;
int location;
struct STable* next;
}*head;


struct Quad{
struct Quad* prev;
char src1[10];
char op[10];
char src2[10];
char tar[10];
struct Quad* next;
}*Qhead;

void yyerror(const char *st){}


%}
%left '+' '-'
%left '*' '/'
%union {  struct icg{ char code[100]; char label[10]; char type[10];}ICG; }
%token NL
%token <ICG> ID
%token  PR INT FLOAT CHAR DOUBLE IF WHILE EQ
%type <ICG> E Assign S If While Program Condition
%%

Program: S { strcpy($$.code,$1.code); printf("\nGenerated 3 Address Code \n%s\n",$1.code);  }
   ;

S:  S Declare
  | S While {  strcat($$.code,$2.code); }
  | S If  {  strcat($$.code,$2.code); }
  | S Assign { strcat($$.code,$2.code); }
```

CD LAB 7 Intermediate Code Generator using LEX and YACC

```
  |
  ;

Declare: Type List ';'
   ;

Type: INT {strcpy(vartype ,"INT"); }
   | FLOAT {strcpy(vartype ,"FLOAT"); }
   | CHAR {strcpy(vartype ,"CHAR");}
   | DOUBLE {strcpy(vartype ,"DOUBLE");}
   ;

List : List ',' ID { newSYM($3.label,vartype);}
   | ID { newSYM($1.label,vartype);}
   ;

Assign: ID '=' E ';' {

                    strcpy($$.code,$3.code);
                    strcat($$.code,$1.label);
                    strcat($$.code," = ");
                    strcat($$.code,$3.label);
                    strcat($$.code,";\n");
                    newAssignment = 0; }
      ;

E: E '+' E        {   add_Code(&$$,&$1,&$3,"+");   }
   |
   E '-' E       {    add_Code(&$$,&$1,&$3,"-");  }
   |
   E '*' E       {    add_Code(&$$,&$1,&$3,"*");  }
   | ID          {
                    strcpy($$.code,$1.label);
                    strcpy($$.label,$1.label);
                    strcpy($$.type,getType($1.label));
                 }
 ;

If: IF '(' Condition ')' '{' S '}'
                {
                 strcpy($$.code,$3.code);
                 strcat($$.code,"\n");
                 strcat ($$.code,"If.Then:\n");
                 strcat($$.code,$6.code);
                 strcat ($$.code,"If.End:\n");
                }
   ;

Condition: E EQ E    {
                      strcpy($$.code,$1.code);
```

CD LAB 7 Intermediate Code Generator using LEX and YACC

```
                    strcat($$.code,$3.code);
                    strcat($$.code,"EQ ");
                    strcat($$.code,$1.label);
                    strcat($$.code,",");
                    strcat($$.code,$3.label);
                    strcat($$.code,"\n");
                    newAssignment = 0;
                    }
        | E {
              strcpy($$.code,$1.code);
              newAssignment = 0;
            }
        ;
While: WHILE '(' Condition ')' '{' S '}'
                  {
                  strcat($$.code,"WHILE.BEGIN:\n");
                  strcat($$.code,$3.code);
                  strcat($$.code,"WHILE.True:\n");
                  strcat($$.code,$6.code);
                  strcat($$.code,"GOTO WHILE.BEGIN\n");
                  }
    ;

%%

void newSYM(char* lab, char* vartype);
void newQuad(char* src1,char* op,char* src2,char* tar);
void DisplaySTable(struct STable*);

void add_Code(struct icg *tar,struct icg *one, struct icg *two,char* operator){
        char Tempname[10],code[50];
        static int tmpnum = 0;
        sprintf(Tempname,"temp%d",tmpnum++);

        strcpy(tar->label,Tempname );
        strcpy(tar->type,two->type);
        newSYM(Tempname,two->type);
        strcat(Tempname," = ");
        strcpy(code, Tempname);
        strcat(code,one->label);
        strcat (code,operator);
        strcat(code,two->label);
        strcat(code,";\n");

        if(newAssignment == 0){
           strcpy(tar->code , code);
           newAssignment = 1;
        }
        else
           strcat(tar->code , code);
```

```
        newQuad(one->label,operator,two->label,tar->label);
}

int isDeclared(char* lab){
    struct STable *st = head;
    while(st){
     if(strcmp(st->label,lab) == 0){
        return 1;
     }
     st = st->next;
}
return 0;
}
char* getType(char* lab){
    struct STable *st = head;
    while(st){
     if(strcmp(st->label,lab) == 0){
        return st->type;
     }
     st = st->next;
}
}
void DisplayQuad();
int main(){
head =  NULL;
Qhead = NULL;
stdin = fopen("in","r");
freopen("out","w",stdout);

yyparse();
DisplaySTable(head);
DisplayQuad();
}

int getVarSize(char* st){
if(strcmp(st,"INT") == 0 ||strcmp(st,"FLOAT") == 0  )
    return 4;
if(strcmp(st,"CHAR") == 0 )
    return 1;
if(strcmp(st,"DOUBLE") == 0  )
    return 8;
}

void newSYM(char* lab, char* vartype){
struct STable *tnode = head;
int size = getVarSize(vartype);
if( !tnode ){
    struct STable* nnode = (struct STable *)malloc(sizeof(struct STable));
    strcpy(nnode->label ,lab);
    strcpy(nnode->type ,vartype);
```

```
    nnode->size = size;
    nnode->location = 100;
    nnode->next=NULL;
    head = nnode;

}else{
    while(tnode->next){
        if(strcmp (tnode->label,lab) == 0){
            printf("\nError: ReDeclaration of %s Variable %s (Previous Declaration as %s)",vartype,
lab,tnode->type);
            return;
        }
        tnode = tnode->next;
    }
    struct STable* nnode = (struct STable *)malloc(sizeof(struct STable));
    strcpy(nnode->label ,lab);
    nnode->size = size;
    strcpy(nnode->type ,vartype);
    nnode->location = tnode -> location + size;
    nnode->next=NULL;
    tnode->next = nnode;
}
}

void newQuad(char* src1,char* op,char* src2,char* tar){

struct Quad *tnode = Qhead;
if( !tnode ){
    struct Quad* nnode = (struct Quad *)malloc(sizeof(struct Quad));
    strcpy(nnode->src1 ,src1);
    strcpy(nnode->op ,op);
    strcpy(nnode->src2 ,src2);
    strcpy(nnode->tar ,tar);
    nnode->next=NULL;
    Qhead = nnode;

}else{
    while(tnode->next){
        tnode = tnode->next;
    }
    struct Quad* nnode = (struct Quad *)malloc(sizeof(struct Quad));
    strcpy(nnode->src1 ,src1);
    strcpy(nnode->op ,op);
    strcpy(nnode->src2 ,src2);
    strcpy(nnode->tar ,tar);
    nnode->next=NULL;
    tnode->next = nnode;
}
}
```

CD LAB 7 Intermediate Code Generator using LEX and YACC

```
void DisplaySTable(struct STable *st){
int i = 1;
printf("\n\n\t\t\t\t%s\n","SYMBOL TABLE");
printf("\t| %s | Label | size | location |\n","Index");
while(st){
   printf("\t|%7d|%7s|%6d|%10d|\n",i++,st->label,st->size,st->location);
   st = st->next;
}
}

void DisplayQuad(){
int i = 1;
printf("\n\n\t\t\t\t%s\n","Quad TABLE");
printf("\t| %s | SRC1  | OP   |  SRC2 | TARGET |\n","Index");
struct Quad* st = Qhead;
while(st){
   printf("\t|%7d|%7s|%6s|%7s|%8s|\n",i++,st->src1,st->op,st->src2,st->tar);
   st = st->next;
}
}
```

lex.l

```
%{
#include <stdio.h>
#include "y.tab.h"

%}


letter [a-zA-Z]
digit [0-9]

%%
"int" {return INT;}
"float" {return FLOAT;}
"double" {return DOUBLE;}
"char" {return CHAR;}
"if" {return IF;}
"while" {return WHILE;}
{letter}({letter}|{digit})* { strcpy(yylval.ICG.label, yytext); return ID;}
"==" {return EQ;}
","|";"|"="|"+"|"-"|"*"|"/"|"("|")"|"{"|"}" {return yytext[0];}

\n
%%
```

CD LAB 7 Intermediate Code Generator using LEX and YACC

INPUT

```
int a,g,h;
int b,c,d,i;
 a = b + c + d;

while(a+c == b+c){
    a = a+c;
}

if(a) {
    a=h+g;
    h = h + h;
   }
```

OUTPUT

```
Generated 3 Address Code
temp0 = b+c;
temp1 = temp0+d;
a = temp1;
WHILE.BEGIN:
temp2 = a+c;
btemp3 = b+c;
EQ temp2,temp3
WHILE.True:
temp4 = a+c;a
If.Then:
temp5 = h+g;
a = temp5;
temp6 = h+h;
h = temp6;
If.End:
```

SYMBOL TABLE

Index	Label	size	location
1	a	4	100
2	g	4	104
3	h	4	108
4	b	4	112
5	c	4	116
6	d	4	120
7	i	4	124
8	temp0	4	128
9	temp1	4	132
10	temp2	4	136
11	temp3	4	140

CD LAB 7 Intermediate Code Generator using LEX and YACC

12	temp4	4	144
13	temp5	4	148
14	temp6	4	152

Quad TABLE

Index	SRC1	OP	SRC2	TARGET
1	b	+	c	temp0
2	temp0	+	d	temp1
3	a	+	c	temp2
4	b	+	c	temp3
5	a	+	c	temp4
6	h	+	g	temp5
7	h	+	h	temp6