9 Experiment 9

9.1 Aim

Convert the BNF rules into YACC form and write code to generate abstract syntax tree.

9.2 Algorithm

1.Start the Program.
2.Reading an input file line by line.
3.Convert it in to abstract syntax tree using three address code.
4.Represent three address code in the form of quadruple tabular form.
5.Stop the Program.

9.3 Program

lex code

```
%{
#include"y.tab.h"
#include<stdio.h>
#include<string.h>
int LineNo=1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
```

```
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
return VAR;}
{number} {strcpy(yylval.var,yytext);
return NUM;}
\< |
\> |
\>= |
\<= |
== {strcpy(yylval.var,yytext); return RELOP;}
[ \t] ;
\n LineNo++;
. return yytext[0];
%%
```

yacc code

```
%{
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
int yyerror();
int yylex();
struct quad
{
char op[5];
char arg1[10];
char arg2[10];
char result[10];
}QUAD[30];
struct stack
{
int items[100];
```

```
int top;
}stk;
int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
void AddQuadruple(char op[5],char arg1[10],char
   arg2[10],char result[10]);
int pop();
void push(int data);
%}
%union
{
char var[10];
}
%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE
%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-' '+'
%left '*' '/'
%%
PROGRAM : MAIN BLOCK
;
BLOCK: '{' CODE '}'
;
CODE: BLOCK
| STATEMENT CODE
| STATEMENT
;
STATEMENT: DESCT ';'
| ASSIGNMENT ';'
| CONDST
| WHILEST
;
```

```
DESCT: TYPE VARLIST
;
VARLIST: VAR ',' VARLIST
| VAR
;
ASSIGNMENT: VAR '=' EXPR{
strcpy(QUAD[Index].op,"=");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,$1);
strcpy($$,QUAD[Index++].result);
}
;
EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}
| EXPR '*' EXPR { AddQuadruple("*",$1,$3,$$);}
| EXPR '/' EXPR { AddQuadruple("/",$1,$3,$$);}
| '-' EXPR { AddQuadruple("UMIN",$2,"",$$);}
| '(' EXPR ')' {strcpy($$,$2);}
| VAR
| NUM
;
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
| IFST ELSEST
;
IFST: IF '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
```

```
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
};
ELSEST: ELSE{
tInd=pop();
Ind=pop();
push(tInd);
sprintf(QUAD[Ind].result,"%d",Index);
}
BLOCK{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
StNo=Index-1;
}
| VAR
| NUM
;
WHILEST: WHILELOOP{
Ind=pop();
```

```
sprintf(QUAD[Ind].result,"%d",StNo);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
;
WHILELOOP: WHILE '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
;
%%
extern FILE *yyin;
int main(int argc,char *argv[])
{
FILE *fp;
int i;
if(argc>1)
{
fp=fopen(argv[1],"r");
if(!fp)
```

```
{
printf("\n File not found");
exit(0);
}
yyin=fp;
}
yyparse();
printf("\n\n\t\t ----------------------------\n\t\t Pos
   Operator Arg1 Arg2 Result\n\t\t--------------------");
for(i=0;i<Index;i++)
{
printf("\n\t\t %d\t %s\t %s\t %s\t
   %s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);
}
printf("\n\t\t ----------------------");
printf("\n\n");
return 0;
}
void push(int data)
{
stk.top++;
if(stk.top==100)
{
printf("\n Stack overflow\n");
exit(0);
}
stk.items[stk.top]=data;
}
int pop()
{
int data;
if(stk.top==-1)
```

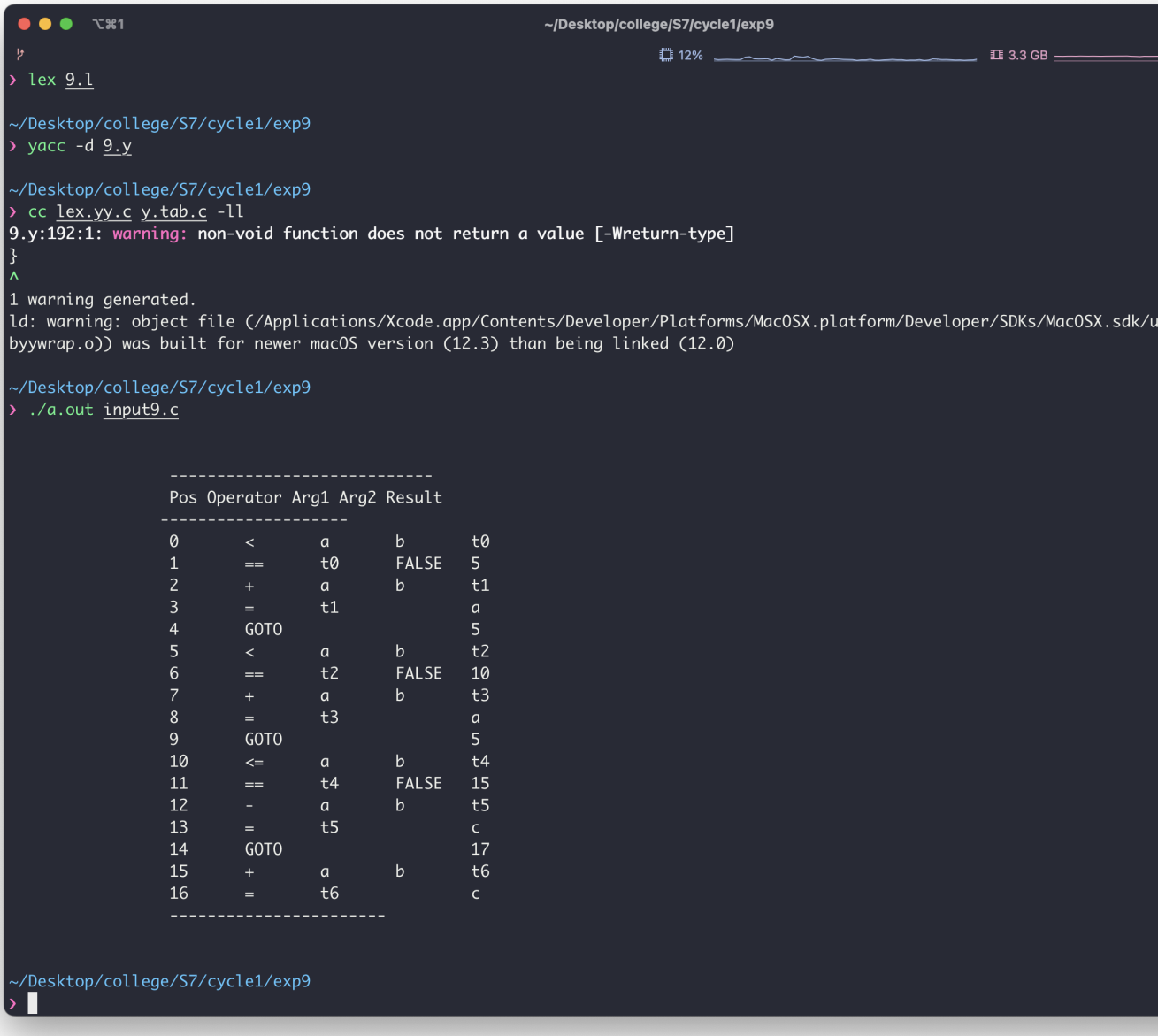
```
{
printf("\n Stack underflow\n");
exit(0);
}
data=stk.items[stk.top--];
return data;
}
void AddQuadruple(char op[5],char arg1[10],char
   arg2[10],char result[10])
{
strcpy(QUAD[Index].op,op);
strcpy(QUAD[Index].arg1,arg1);
strcpy(QUAD[Index].arg2,arg2);
sprintf(QUAD[Index].result,"t%d",tIndex++);
strcpy(result,QUAD[Index++].result);
}
int yyerror()
{
printf("\n Error on line no:%d",LineNo);
}
```

9.4 Input file

```
 main()
{
   int a, b, c;
   if (a < b)
   {
      a = a + b;
   }
   while (a < b)
   {
```

```
      a = a + b;
   }
   if (a <= b)
   {
      c = a - b;
   }
   else
   {
      c = a + b;
   }
}
```

9.5 Output



9.6 Result