Lab 8:
Lex:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
void yyerror(char *str);
%}

%%
[0-9]+ {yylval.intval=atoi(yytext); return NUMBER;}
[a-z]+ {yylval.fchar=yytext; return NAME;}
[\t ];
\n return 0;
. { return yytext[0]; }
%%

int yywrap(){
return 1;
}
```

Yacc:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "y.tab.h"
void yyerror(char *str);
int yylex();
%}

%union{
   char* fchar;
   int intval;
   double fval;
};


%token <fchar>NAME
%token <intval>NUMBER
%type <fval>exp
%left '+','-'

%%
stmt: NAME'='exp {printf("=%f\t\n",$3);} | exp {printf("=%f\t\n",$1);};
exp: exp'+'exp {$$ = $1 + $3;} | exp'-'exp {$$ = $1 - $3;} | NUMBER {$$=$1;};
%%
```

```
void yyerror(char *str){
    printf("%s",str);
    return 1;
}

int main(){
    yyparse();
    return 0;


}
```

LAB7:

```
%{
#include<stdio.h>
#include "y.tab.h"
void yyerror(char *str);
extern int yyparse();
%}

%%
"if" return IF;
"(" return OP;
")" return CP;
"<" |
">" |
">=" |
"<=" |
"==" |
"!=" return CMP;
"+" |
"-" |
"*" |
"/"  return OPR;
"=" return ASG;
([a-zA-Z])(["_"a-zA-Z0-9])* return ID;
[0-9]+ return NUM;
";" return SC;
" " {}
%%

int yywrap(){
return 1;
}
```

Yacc:

```
%{
#include <stdio.h>
```

```
extern int yylex();
extern int yywrap();
extern int yyparse();
%}



%token WH IF DO FOR OP CP OCB CCB CMP OPR ASG ID SC COMMA NUM

%%
start: sif;
sif: IF OP cmpn CP stmt {printf("VALID IF STATEMENT\n");};
cmpn: ID CMP ID | ID CMP NUM;
stmt: ID ASG ID OPR ID SC | ID ASG ID OPR NUM SC | ID ASG NUM OPR ID SC | ID ASG
NUM OPR NUM SC | ID ASG ID SC | ID ASG NUM SC;
%%

int yyerror(char *str){
printf("%s", str);
return 1;
}

int main(){
yyparse();
return 0;
}
```

LAB6:

Grammar:

```
E –> T E’
E’ –> + T E’ | e
T –> F T’
T’ –> * F T’ | e
F –> ( E ) | id
```

Main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SUCCESS 1
#define FAILED 0

int E(), Edash(), T(), Tdash(), F();

const char *cursor;
char string[64];
```

```
int main() {
  printf("Enter the string:\n");
  scanf("%s", string);
  cursor = string;
  printf("\nInput       Action");

  if (E() && *cursor == '\0') {
    printf("\nString successfully parsed!");
    return 0;
  } else {
    printf("\nThere was an error parsing the String!");
    return 1;
  }
}

int E() {
  printf("\n%s \tE –> T E' ", cursor);
  if (T()) {
    if (Edash()) {
      return SUCCESS;
    }
    return FAILED;
  } else {
    return FAILED;
  }
}

int Edash() {
  if (*cursor == '+') {
    printf("\n%s \tE' –> + T E' ", cursor);
    cursor++;
    if (T()) {
      if (Edash()) {
        return SUCCESS;
      } else {
        return FAILED;
      }
    } else {
      return FAILED;
    }
  } else {
    printf("\n%s \tE' –> e ", cursor);
    return SUCCESS;
  }
}

int T() {
  printf("\n%s \tT –> F T' ", cursor);
```

```
  if (F()) {
    if (Tdash()) {
      return SUCCESS;
    }
    return FAILED;
  } else {
    return FAILED;
  }
}

int Tdash() {
  if (*cursor == '*') {
    printf("\n%s \tT' –> * F T' ", cursor);
    cursor++;
    if (F()) {
      if (Tdash()) {
        return SUCCESS;
      } else {
        return FAILED;
      }
    } else {
      return FAILED;
    }
  } else {
    printf("\n%s \tT' –> e ", cursor);
    return SUCCESS;
  }
}

int F() {
  if (*cursor == '(') {
    printf("\n%s \tF –> ( E ) ", cursor);
    cursor++;
    if (E()) {
      if (*cursor == ')') {
        cursor++;
        return SUCCESS;
      } else {
        return FAILED;
      }
    } else {
      return FAILED;
    }
  } else if (*cursor == 'i') {
    printf("\n%s  \tF –> id ", cursor);
    cursor++;
    return SUCCESS;
  } else {
```

```
    return FAILED;
  }
}
```

LAB5:

```
%{
#include <stdio.h>
#include <string.h>
int i = 0;
struct SYMTAB {
int id;
char name[20];
char type[20];
} s[20];
int struct_ptr = 0;
%}

letter [a-zA-Z]
digit [0-9]
dtype (int|float|char)
identifier {letter}({letter}|{digit})*
operator "="|"+"|"-"|"*"|"/"
eol [;]
array {identifier}\[
function {identifier}\(

%%

{digit} {printf("\nDigit Matched: %s",yytext);}
{dtype} {printf("\nDatatype Matched: %s",yytext);}
{identifier} {printf("\nIdentifier Found: %s",yytext);
              s[struct_ptr].id = struct_ptr + 1;
              strcpy(s[struct_ptr].name, yytext);
        strcpy(s[struct_ptr].type, "Identifier");
        struct_ptr++;
        }
{operator} {printf("\nOperator Matched: %s",yytext);}
{eol} {printf("\nEOL Matched: %s",yytext);}
{array} {printf("\nArray Found: %s",yytext);
   yytext[strlen(yytext)-1]='\0';
         s[struct_ptr].id = struct_ptr + 1;
         strcpy(s[struct_ptr].name, yytext);
   strcpy(s[struct_ptr].type, "Array");
   struct_ptr++;
   }
{function} {printf("\nFunction Detected: %s",yytext);
   yytext[strlen(yytext)-1]='\0';
```

```
        s[struct_ptr].id = struct_ptr + 1;
        strcpy(s[struct_ptr].name, yytext);
   strcpy(s[struct_ptr].type, "Function");
   struct_ptr++;
   }

%%

int main(){

yylex();
printf("\nSymbol Table");
printf("\nID\tName\tType");
for(int i=0; i<struct_ptr; i++){
printf("\n%d\t%s\t%s\n",s[i].id,s[i].name,s[i].type);
}
return 0;
}
int yywrap(){
return 1;
}
```