ANNA UNIVERSITY::CHENNAI 600025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

B.E. (CSE) V Semester Batch: 2020-2024 Group: P AUG-DEC 2022

CS 6109 – COMPILER DESIGN LABORATORY

Instructor: Dr. Arockia Xavier Annie R, Asst. Prof, DCSE, CEG, Anna University

Register No.: **2020103016**
Name : **Kathir R M**

S.No	Date	Exercise Title	Max Marks	Marks Scored			Signature of Faculty	Signature of the HOD
				15	5	5		
1.	23.08.2022	Study of LEX and sample programs	25					
2.	30.08.2022	Simple word handling using Lex	25					
3.	06.09.2022	Conversion infix to postfix expression	25					
4.	13.09.2022	Conversion of for to while syntax	25					
5.	20.09.2022	Conversion of struct to class syntax	25					
6.	27.09.2022	Study of YACC and sample programs	25					
7.	11.10.2022	Conversion infix to prefix expression using Yacc	25					
8.	15.10.2022	While syntax validation using Yacc	25					
9.	05.11.2022	Three Address Code for mathematical expressions	25					
10.	08.11.2022	Three Address Code for control statements	25					
11.	15.11.2022	Test	25					
12.	06.12.2022	Three Address Code to Assembly Code	25					

Study of LEX and sample programs

EX NO: 1 **NAME: Kathir R M**

DATE: 23.08.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

General Steps for LEX compilation and execution:

1. lex filename.l
   -this would translate the lex specification that we defined into a C source file(lex.yy.c)
2. gcc lex.yy.c –ll
   -we compile and link with lex library –ll
3. ./a.out

Question – 1:

Count the number of characters and digits:

AIM:

- To count the number of characters in string, which is given as input, using lex.

PROGRAM:

```
%{
#include<stdio.h>
int numdig = 0, numc = 0;
%}

%%

[0-9] { ++numdig; }
.  { ++numc; }

%%

int yywrap(void) {
     return 1;
}

int main(int argc, char* argv[]) {
          printf("Enter the expression: " );
```

```
        yylex();
        printf("\n Number of digits: %d\n Number of characters: %d",
numdig, numc);
         printf("\n\n");
        return 0;
}
```

OUTPUT:

```
[s2020103016@centos8-linux Sat Dec 10 09:51 AM week01]$ ./a.out
Enter the expression: kathir163
Number of digits: 3
Number of characters: 6
[s2020103016@centos8-linux Sat Dec 10 09:51 AM week01]$
```

Question – 2:

Count the number of lines

AIM:

- To count the number of lines in string, which is given as input, using lex.

PROGRAM:

```
%{

#include <stdio.h>

int num_lines = 0, num_chars = 0;

%}

%%

\n { ++num_lines; ++ num_chars; }
. { ++num_chars; }

%%

int yywrap (void) {
        return 1;
}

int main (int argc, char * argv[]){
        yylex();
```

```
        printf("# of lines = %d, # of chars = %d\n", num_lines,
num_chars);
        return 0;
}
```

OUTPUT:

```
 }
 [s2020103016@centos8-linux Sat Dec 10 09:52 AM week01]$ gcc lex.yy.c
 [s2020103016@centos8-linux Sat Dec 10 09:52 AM week01]$ ./a.out
 helloWorld
 Goodbye
 # of lines = 2 , #of chars = 19
 [s2020103016@centos8-linux Sat Dec 10 09:53 AM week01]$
```

Simple word handling using Lex

EX NO: 2 **NAME: Kathir R M**

DATE: 30.08.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Find and print five consecutive same alphabets:

AIM:

- To find and print five consecutive same alphabets in string, which is given as input, using lex.

PROGRAM:

```
%{


%}

%%

a{5,5} |
b{5,5} |
c{5,5} |
d{5,5} |
e{5,5} |
f{5,5} |
g{5,5} |
h{5,5} |
i{5,5} |
j{5,5} |
k{5,5} |
l{5,5} |
m{5,5} |
n{5,5} |
o{5,5} |
p{5,5} |
q{5,5} |
r{5,5} |
s{5,5} |
t{5,5} |
```

```
u{5,5} |
v{5,5} |
w{5,5} |
x{5,5} |
y{5,5} |
z{5,5} { printf("Matched 5 consecutive letters: %s\n", yytext); }
\n|. {}

%%

int yywrap(void){
        return 1;
}

void main (){

        yylex();
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 02:38 PM week03]$ cd ../week02/
[s2020103016@centos8-linux Mon Dec 12 02:38 PM week02]$ ./a.out
kattttthir
 5 successive found , they are ttttt
```

Question – 2:

Find vowels and print

AIM:

To find the vowels in string, which is given as input, and print them using lex.

PROGRAM:

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

%%

[aeiouAEIOU] { printf(" %s \n", yytext); }
\n|. { }

%%
```

```
int yywrap(void) { }

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

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 02:40 PM week02]$ ./a.out
kathir
Vowel found is aVowel found is i
```

Question – 3: (SPOT)

Group up the declaration of variables based on datatypes:

AIM:

- To group up integer and floating point variables from their declarations statements, which is given as input, and print the grouped declaration statements using lex.

PROGRAM:

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node{
char datatype[100];
char identifier[100];
struct node* next;
};
struct node* head = NULL;
struct node* tail = NULL;
%}
%%
^int[ ]+[a-zA-Z_]+[0-9]*("["[0-9]*"]")*; {
struct node* curr = (struct node*) malloc ( 1 * sizeof(struct node));
strcpy(curr -> datatype , "int");
int i = 3;
while ( yytext[i] == ' ')
```

```
        i++;
char varname[100];
int j = 0;
while(yytext[i] != ';'){
varname[j++] = yytext[i++];
}
varname[j] = '\0';
strcpy(curr -> identifier , varname);
curr -> next = NULL;
if (!head) {
head = tail = curr;
}
else{
tail -> next = curr;
tail = curr;
}
}
^float[ ]+[a-zA-Z_]+[0-9]*("["[0-9]*"]")*; {
struct node* curr = (struct node*) malloc ( 1 * sizeof(struct node));
strcpy(curr -> datatype , "float");
int i = 5;
while ( yytext[i] == ' ')
        i++;
char varname[100];
int j = 0;
while(yytext[i] != ';'){
varname[j++] = yytext[i++];
}
varname[j] = '\0';
strcpy(curr -> identifier , varname);
curr -> next = NULL;
if (!head) {
head = tail = curr;
}
else{
tail -> next = curr;
tail = curr;
}
}
^char[ ]+[a-zA-Z_]+[0-9]*("["[0-9]*"]")*; {
struct node* curr = (struct node*) malloc ( 1 * sizeof(struct node));
strcpy(curr -> datatype , "char");
int i = 4;
while ( yytext[i] == ' ')
        i++;
char varname[100];
int j = 0;
while(yytext[i] != ';'){
varname[j++] = yytext[i++];
}
varname[j] = '\0';
strcpy(curr -> identifier , varname);
curr -> next = NULL;
if (!head) {
```

```
head = tail = curr;
}
else{
tail -> next = curr;
tail = curr;
}
}
%%
int yywrap(void){}
int main()
{
        yylex();
        struct node* p = head;
        while(p) {
        printf("%s %s\n" , p-> datatype , p ->identifier);
        p = p ->next;
}
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 02:41 PM CD]$ lex symboltable.l
[s2020103016@centos8-linux Mon Dec 12 02:46 PM CD]$ gcc lex.yy.c
[s2020103016@centos8-linux Mon Dec 12 02:46 PM CD]$ ./a.out
int a;

float myfloat;

int myint;

float f2;

int a
float myfloat
int myint
float f2
[s2020103016@centos8-linux Mon Dec 12 02:46 PM CD]$
```

Conversion infix to postfix expression

EX NO: 3 **NAME: Kathir R M**

DATE: 06.09.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Write a program to convert infix expression of binary operators to post fix expression using lex

AIM:

- To translate the infix expression of an arithmetic expression into postfix expression.

PROGRAM:

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

typedef struct node {
      int val;
      struct node *next;
}node_t;

int pop();
void push(int val);
int priority (int op);
int peek();
bool isEmpty();

%}

op [+/*-]
exp "**"

%%

[a-zA-Z] { printf("%s", yytext); }
```

```
"(" {push(yytext[0]);}
{op} {
      while ( priority(peek()) >= priority(yytext[0])){
            printf("%c",pop());
      }
      push(yytext[0]);
}
{exp}  {
      while (peek() == '^')
            printf("**");
      push('^');
}
")" {
      while ( peek() != '(') {
            printf("%c",pop());
      }
      pop();
}
\n {
      while (!isEmpty()){
            printf("%c",pop());
      }
      printf("\n");
}
exit { return 0; }
. { }

%%

node_t *stack_head = NULL;

int priority (int op) {
      switch (op) {
            case '+':
            case '-':
                  return 1;
            case '*':
            case '/':
                  return 2;
      }
      return -1;
}
```

```
int peek() {
      if (stack_head == NULL) return -1;
      return stack_head->val;
}

bool isEmpty() {
      if (stack_head == NULL) return true;
      return false;
}

int pop() {
      if (stack_head == NULL) return -1;
      int val = stack_head->val;
      node_t *temp = stack_head;
      stack_head = stack_head->next;
      free(temp);
      return val;
}

void push (int val) {
      node_t *temp = (node_t *) malloc (sizeof(node_t));
      temp->val = val;
      temp->next = stack_head;
      stack_head = temp;
}

int yywrap() {
      return 1;
}

void main () {
      yylex();
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 02:51 PM week03]$ lex intopost.l
[s2020103016@centos8-linux Mon Dec 12 02:51 PM week03]$ gcc lex.yy.c
[s2020103016@centos8-linux Mon Dec 12 02:51 PM week03]$ ./a.out
a+b/c*(d+e)
abc/de+*+
```

Question – 2: (SPOT)

Write a program to convert infix expression of unary operators to post fix expression using lex

AIM:

- To translate the infix expression of an arithmetic expression into postfix expression.

```
%{
#include<stdio.h>
char stack[200];
int top = -1;
char prevRead = '\0';
%}

%%

[a-zA-Z0-9] { printf("%s", yytext);prevRead = '\0'; }

"^" { stack[++top] = yytext[0]; }
"(" { stack[++top] = yytext[0]; }
"*" |
"/" |
"%" {
        if(top==-1 || stack[top]=='+' || stack[top] == '-')
                stack[++top]= yytext[0];
        else {
                while(top>=0  &&  stack[top]!='+'  &&  stack[top]!='-'  &&
stack[top]!='(')
                        printf("%c",stack[top--]);
                stack[++top] = yytext[0];
        }
}
"+" |
"-" {
        if ( prevRead == '-' || prevRead == '+'){
                stack[++top] = '~';
                prevRead = yytext[0];
}
        else if(top==-1 || stack[top]=='(')

                stack[++top]= yytext[0];
        else {
                while(top>=0 && stack[top]!='(')
printf("%c",stack[top--]);
                stack[++top] = yytext[0];
        }
}
")" {
        while(top>=0 && stack[top]!='(')
                printf("%c", stack[top--]);
        top--;
}
. { }
```

```
\n { return 0; }

%%

int yywrap(void) { }
int main() {
        yylex();
        while ( top >= 0)
        printf("%c" ,stack[top--]);
        printf("\n\n");
        return 0;
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 02:53 PM week03]$ lex mdfintopost.l
[s2020103016@centos8-linux Mon Dec 12 02:54 PM week03]$ gcc lex.yy.c
[s2020103016@centos8-linux Mon Dec 12 02:54 PM week03]$ ./a.out
a+-b
a+b-
```

Conversion of for to while syntax

EX NO: 4 **NAME: Kathir R M**

DATE: 13.09.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Convert while loop to for loop and vice versa

AIM:

- Match the while loop and print them in the for loop
- Match the for loop and print them in the while loop

```
%{

#include<stdio.h>

#include<string.h>

int cond = 0, step = 0, buff_flag = 0;

int sccount = 0, paranflag = 1, closeflag = 0;

char cond_buffer[200], step_buffer[200], buffer[200];

%}

%%

"while(" {

        printf("while loop found!\n");

        cond = 1;

        buffer[0] = '\0';

}

")" {

        if(paranflag==1) {

                cond = 0;

                buff_flag = 1;

                paranflag = 0;
```

```
        }
        else {
                REJECT;
        }
}
"{" {
        closeflag++;
        if(closeflag>1)
                REJECT;
}
(([a-zA-Z0-9])[+\-=][ ])*[;][\n][}] {
        closeflag--;
        if(closeflag==0) {
                int i=0;
                while(yytext[i]!=';') {
                        step_buffer[i] = yytext[i];
                        i++;
                }
        }
        else {
                REJECT;
        }
}
"\n" { }
. {
        if(cond)
                strcat(cond_buffer, yytext);
        else if(buff_flag)
                strcat(buffer, yytext);
```

```
        else {
                printf(" error! \n");
                return -1;
        }
}


%%


int yywrap(void) { }


int main() {
        char *filename = malloc(100*sizeof(char));
        printf("Enter filename : ");
        scanf("%s", filename);
        FILE *fp1 = fopen(filename, "r");
        if(fp1)
                yyin = fp1;
        yylex();
        FILE *fp2 = fopen("output2.txt", "w");
        char incrementCond[20];
printf("%s" , buffer);
        int i = 0;
        int j = 0;
        int len = strlen(buffer);
        i = len - 1;
        while (i >= 0 && buffer[i] != ';')
                i--;
        i--;
        while ( i >= 0 && buffer[i] != ';')
                i--;
        i++;
        int pos = i;
        while ( i < len && buffer[i] != ';'){
                incrementCond[j++] = buffer[i++];
}
        incrementCond[j] = '\0';
        buffer[pos] = '\0';
        fprintf(fp2, "\nfor(; %s; %s) {\n%s\n}\n", cond_buffer,incrementCond ,
buffer);
```

```
        fclose(fp2);

        return 0;

}
```

INPUT:

```
[s2020103016@centos8-linux Mon Dec 12 03:17 PM week04]$ cat while.txt
while(i<10){
printf("Hello world");
i--;
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 03:17 PM week04]$ cat output2.txt

for(; i<10; i--;) {
printf("Hello world")
}
[s2020103016@centos8-linux Mon Dec 12 03:17 PM week04]$
```

For to while

```
%{
#include<stdio.h>
#include<stdlib.h>
int foraFlag = 0;
int firstCond = 0;
int secondCond = 0;
int thirdCond = 0;
int foraBody = 0;
char forbuffer[100];
char fbuffer[100];
char sbuffer[100];
char tbuffer[100];
char bbuffer[100];
char forbodybuffer[100];
char beforeforbuffer[100];
%}

%%
"for(" {
foraFlag = 1;
firstCond = 1;
}
";" {
if(foraFlag){
if (!secondCond){
secondCond = 1;
firstCond = 0;
}
else {
```

```
thirdCond = 1;
secondCond = 0;
}
}
}
")" {
if(foraFlag){
foraFlag=firstCond=secondCond=thirdCond=0;
foraBody = 1;
}
else{
strcat(forbodybuffer,yytext);
}
}
. {
if(foraBody)
strcat(forbodybuffer,yytext);
if(!foraFlag &&!foraBody)
strcat(beforeforbuffer , yytext);
else if(firstCond && foraFlag && !secondCond && !thirdCond)
strcat(fbuffer,yytext);
else if(secondCond)
strcat(sbuffer,yytext);
else if (thirdCond)
strcat(tbuffer , yytext);
else
strcat(bbuffer,yytext);
}

%%
int yywrap(void){}
int main()
{
char* filename = malloc(100 * sizeof(char));
printf("Enter the filename: ");
scanf("%s" , filename );
FILE* fp = fopen(filename,"r");
if(fp)
yyin=fp;
yylex();
FILE* fp2 = fopen("output.txt","w");
int i =strlen(forbodybuffer);
forbodybuffer[i-1] = '\0';
fprintf(fp2,"%s\n %s \n while(%s) %s\n %s \n   }\n", beforeforbuffer ,
fbuffer , sbuffer , forbodybuffer , tbuffer);
}
```

Output:

```
[s2020103016@centos8-linux Mon Dec 12 07:17 PM week04]$ cat for.txt
for(i=0;i<10 && i > 0;i++){
printf("Hi");
}
[s2020103016@centos8-linux Mon Dec 12 07:19 PM week04]$ cat output.txt
[s2020103016@centos8-linux Mon Dec 12 07:19 PM week04]$ lex fortowhile.l
[s2020103016@centos8-linux Mon Dec 12 07:19 PM week04]$ gcc lex.yy.c
[s2020103016@centos8-linux Mon Dec 12 07:19 PM week04]$ ./a.out for.txt
Enter the filename: for.txt



[s2020103016@centos8-linux Mon Dec 12 07:19 PM week04]$ cat output.txt

 i=0
 while(i<10 && i > 0) {printf("Hi")
 i++
  }
[s2020103016@centos8-linux Mon Dec 12 07:19 PM week04]$
```

Question – 2: (SPOT)

Convert the nested for loop to equivalent while loop to for loop.

AIM:

- Match the nested for loop and convert them into while loop

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int firstfor = 0;
int secondfor = 0;
char firstF[100];
char secondF[100];
char bodyBuffer[200];
int substr(char* a , char c , int b);
%}

%%
^("for").* {
        if(!firstfor){
strcpy(firstF , yytext);
firstfor = 1;
}
else if (!secondfor){
strcpy(secondF , yytext);
secondfor = 1;
}
}
. {
if(secondfor){
strcat(bodyBuffer , yytext);
}
}
%%
int yywrap(void) {}
int main() {
char firstCond[20];
char secondCond[20];
```

```
char thirdCond[20];
char firstCond2[20];
char secondCond2[20];
char thirdCond2[20];
char *filename = malloc(100*sizeof(char));

int flen = strlen(firstF);
int slen = strlen(secondF);
firstF[flen-2] = '\0';
secondF[flen-2] = '\0';
printf("Enter filename : ");

scanf("%s", filename);

FILE *fp1 = fopen(filename, "r");

        if(fp1)

                yyin = fp1;

        yylex();

        FILE *fp2 = fopen("output.txt", "w");

int i = 0;
while(firstF[i] != '(')
i++;
i++;
int j = 0;
while(firstF[i] != ';'){
firstCond[j++] = firstF[i++]; //first condition 1
}
firstCond[j] = '\0';
j=0;
i++;
while(firstF[i] != ';'){
secondCond[j++] = firstF[i++]; //second condition 1
}
secondCond[j] = '\0';
j=0;
i++;
while(firstF[i] != ')'){
thirdCond[j++] = firstF[i++]; // third condition 1
}
thirdCond[j] = '\0';
j = 0;
i = 0;
while(secondF[i] != '(')
i++;
i++;
while(secondF[i] != ';'){
firstCond2[j++] = secondF[i++]; //first condition 2
}
firstCond2[j] = '\0';
```

```
j=0;
i++;
while(secondF[i] != ';'){
secondCond2[j++] = secondF[i++]; //second condition 2
}
secondCond2[j] = '\0';
j=0;
i++;
while(secondF[i] != ')'){
thirdCond2[j++] = secondF[i++]; // third condition 2
}
thirdCond2[j] = '\0';
j = 0;
        fprintf(fp2, "\n%s \n%s", firstF , secondF);
        fclose(fp2);

int k = strlen(bodyBuffer);
while ( k >= 0 && bodyBuffer[k] != ';'){
k--;
}
bodyBuffer[k+1] = '\0';
printf("%s;\n%s;\nwhile(%s  ||  %s){\n%s\n%s;\n%s;\n}\n"  ,  firstCond  ,
firstCond2 , secondCond , secondCond2 ,bodyBuffer, thirdCond , thirdCon
d2);
//printf("%s is body\n",bodyBuffer);
        return 0;

}
```

INPUT:

```
for.txt  output2.txt  output.txt  twofor.txt  while.txt
[s2020103016@centos8-linux Mon Dec 12 03:19 PM week04]$ cat twofor.txt
for(int i = 0 ; i < 10 ; i++){
for(int j = 0 ; j < 200 ; j++){
printf("why");
}
}
```

OUTPUT:

```
int i = 0 ;
int j = 0 ;
while( i < 10  ||  j < 200 ){
printf("why");
 i++;
 j++;
}
```

Conversion of struct to class syntax

EX NO: 5 **NAME: Kathir R M**

DATE: 20.09.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Convert a struct definition into a class definition using lex

AIM:

- To match the struct definition and extract the attributes and form the class

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char buffer[1000];
void func(char* s , char* t){
strcat( s , t);
}
%}
%%
^struct[" "]+[a-zA-Z_]+[0-9]*[" "]*"{" {
        char temp[1000];
        temp[0] = '\0';
        func(temp , "class ");
        int len = yyleng;
        int i = 6;
        while (yytext[i] == ' ')
        i++;
        int j = 6;
        while(yytext[i] != '{')
        temp[j++] = yytext[i++];
        temp[j++] = '{';
        temp[j] = '\0';
        func ( buffer , temp);
        func(buffer , "\nprivate:");
}

^struct[" "]+[a-zA-Z_*]+[0-9]*[" "]+[a-zA-Z_*]+[0-9]*[ ]*";" {
        char temp[100];
        int j = 6;
        while ( yytext[j] == ' ')
        j++;
```

```
                int i = 0;
        while(yytext[j]){
                temp[i++] = yytext[j++];
        }
        func(buffer , temp);
}

.|\n {
        func(buffer , yytext);
}
%%
int yywrap(void){};
int main()
{
        yylex();
        FILE* fp = fopen("output.txt" , "w");
        fprintf(fp , "%s" , buffer);
}
```

INPUT:

```
[s2020103016@centos8-linux Mon Dec 12 04:53 PM week05]$ ./a.out
struct a {
int b;
}
```

OUTPUT:

```
class a {
private:
int b;
}
```

Question – 2: (SPOT)

Convert a struct definition into a class definition along with the constructor using lex

AIM:

- To match the struct definition and extract the attributes and form the class

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node{
char datatype[100];
```

```
char identifier[100];
struct node* next;
};
struct node* head = NULL;
struct node* tail = NULL;
char buffer[10000];
char className[100];
char* constructConstructor() {
        char* cons = (char*) malloc ( 1000 * sizeof(char));
        strcat ( cons , "public ");
        strcat ( cons , className);
        strcat ( cons , " ( ");
        struct node* p = head;
        while (p) {
                strcat( cons , p -> datatype);
strcat(cons , " ");
strcat(cons , p -> identifier);
if(p -> next)
strcat(cons , ",");
p = p -> next;
}
strcat (cons , ") {\n");
p = head;
while (p) {
strcat(cons , "this.");
strcat(cons , p -> identifier);
strcat (cons , "=");
strcat(cons,p->identifier);
strcat (cons , ";\n");
p = p -> next;
}
strcat(cons , "}");
return cons;
}
void yycat ( char** f , char** g) {
        char* a = *f;
        char* b = *g;
        int len = strlen(a);
        char* s = (char*) malloc ( (strlen(a) + strlen(b)) * sizeof(char));
int i , j , k;
        for ( int i = 0 ; i < len ; i++) {
                s[i] = a[i];
}
        for (j = 0 ; j < strlen(b) ; j++ , i++) {
        s[i] = b[j];
}
s[i] = '\0';
*f = s;
}
%}

%%
^int[ ]+[a-zA-Z_*]+[0-9]*("["[0-9]*"]")*; {
struct node* curr = (struct node*) malloc ( 1 * sizeof(struct node));
```

```
strcpy(curr -> datatype , "int");
int i = 3;
while ( yytext[i] == ' ')
        i++;
char varname[100];
int j = 0;
while(yytext[i] != ';'){
varname[j++] = yytext[i++];
}
varname[j] = '\0';
strcpy(curr -> identifier , varname);
curr -> next = NULL;
if (!head) {
head = tail = curr;
}
else{
tail -> next = curr;
tail = curr;
}
strcat(buffer ,yytext);
}
^float[ ]+[a-zA-Z_*]+[0-9]*("["[0-9]*"]")*; {
struct node* curr = (struct node*) malloc ( 1 * sizeof(struct node));
strcpy(curr -> datatype , "float");
int i = 5;
while ( yytext[i] == ' ')
        i++;
char varname[100];
int j = 0;
while(yytext[i] != ';'){
varname[j++] = yytext[i++];
}
varname[j] = '\0';
strcpy(curr -> identifier , varname);
curr -> next = NULL;
if (!head) {
head = tail = curr;
}
else{
tail -> next = curr;
tail = curr;
}
strcat(buffer ,yytext);
}
^char[ ]+[a-zA-Z_*]+[0-9]*("["[0-9]*"]")*; {
struct node* curr = (struct node*) malloc ( 1 * sizeof(struct node));
strcpy(curr -> datatype , "char");
int i = 4;
while ( yytext[i] == ' ')
        i++;
char varname[100];
int j = 0;
while(yytext[i] != ';'){
varname[j++] = yytext[i++];
```

```
}
varname[j] = '\0';
strcpy(curr -> identifier , varname);
curr -> next = NULL;
if (!head) {
head = tail = curr;
}
else{
tail -> next = curr;
tail = curr;
}
strcat(buffer ,yytext);
}
^struct[" "]+[a-zA-Z_*]+[0-9]*[" "]*"{" {
char temp[1000];
temp[0] = '\0';
strcat(temp , "class ");
int i = 6;//struct
while (yytext[i] == ' ')
i++;
int j = 6;
int k = 0;
while(yytext[i]){
char c = yytext[i];
temp[j++] = c;
className[k++] = c;
i++;
}
className[k-1] = '\0';
temp[j] = '\0';
strcat ( buffer , temp);
}
^struct[" "]+[a-zA-Z_*]+[0-9]*[" "]+[a-zA-Z_*]+[0-9]*[ ]*";" {
char temp[100];
int j = 6;
while ( yytext[j] == ' ')
j++;
int i = 0;
while(yytext[j]){
temp[i++] = yytext[j++];
}
strcat(buffer , temp);
}
\n {
strcat(buffer , yytext);
}
%%
int yywrap(void){}
int main()
{
        yylex();
        struct node* p = head;
        FILE* fp = fopen("output.txt" , "w");
        int last = strlen(buffer);
```

```
        char* ans = constructConstructor();
        strcat(buffer , ans);
        strcat(buffer , "\n}");
        fprintf(fp , "%s" , buffer);

}
```

INPUT:

```
}[s2020103016@centos8-linux Mon Dec 12 05:04 PM spot]$ ./a.out

Kathir
struct student{
int marks;
float cgpa;
};
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 05:05 PM spot]$ cat output.txt
class student{
int marks;
float cgpa;


public student ( int marks,float cgpa) {
this.marks=marks;
this.cgpa=cgpa;
}
```

Study of YACC and sample programs

EX NO: 6 **NAME: Kathir R M**

DATE: 27.09.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Simple Calculator:

AIM:

- To evaluate a simple arithmetic expression

PROGRAM:

LEX

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

void yyerror(char*);
extern int yylval;

%}

%%

[\t ]+ ;
[0-9]+ {
        yylval = atoi(yytext);
        return INTEGER;
}
[-+*/] { return *yytext;}
"(" { return *yytext; }
")" { return *yytext; }
\n { return *yytext; }
. {
        char msg[25];
        sprintf(msg, "%s <%s>", "invalid character" , yytext);
        yyerror(msg);
}
```

```
%%

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

YACC

```
%{

#include <stdlib.h>
#include <stdio.h>
#include "y.tab.h"

int yylex(void);

%}

%token INTEGER

%%

program:
        line program
        | line

line:
        expr '\n' {     printf("%d\n",$1); }
        | '\n'
expr:
        expr '+' mulex { $$ = $1 + $3; }
        | expr '-' mulex { $$ = $1 - $3; }
        | mulex { $$ = $1; }
mulex:
        mulex '*' term { $$ = $1 * $3; }
        | mulex '/' term { $$ = $1 / $3; }
        | term { $$ = $1; }
term:
        '(' expr ')' { $$ = $2; }
        | INTEGER { $$ = $1; }

%%

void yyerror (char *s){
        fprintf(stderr, "%s\n", s);
        return;
}

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

```
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 05:06 PM calc]$ ./calculator
1+2*3
Matched a * bMatched a + b7
2+3
Matched a + b5
```

Question – 2: (SPOT)

Validate a C declaration

AIM:

- To validate a C declaration

PROGRAM:

LEX

```
%{

#include "y.tab.h"
#include <stdio.h>
#include <string.h>
extern char* yylval;

%}

char [A-Za-z]
num [0-9]
eq [=]
name {char}+
data {num}+
fdatatype (float|double)
idatatype (int)
cdatatype (char)
chardata (['](.)['])
floatdata ([0-9]+[.][0-9]+)
sign ("-"|"+"){0,1}
intdata {sign}([0-9]+)
expr ([0-9]+(([+*][0-9]+)|([/%][1-9]+)|("-"[0-9]+))+)
fdata {sign}({intdata}|{floatdata}|{expr})
comma [,]

%%
```

```
{fdatatype} { yylval = strdup(yytext); return FDT; }
{idatatype} { yylval = strdup(yytext); return IDT; }
{cdatatype}     { yylval = strdup(yytext); return CDT; }
{chardata}      { yylval = strdup(yytext); return CDATA; }
{intdata}       { yylval = strdup(yytext); return IDATA; }
{fdata} { yylval = strdup(yytext); return FDATA; }
{name} { yylval = strdup(yytext); return NAME; }
{eq} { yylval = strdup(yytext); return EQ; }
{data} { yylval = strdup(yytext); return DATA; }
[;]     { yylval = strdup(yytext); return SEMICOLON; }
{comma} { yylval = strdup(yytext); return COMMA; }

%%

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

YACC

```
%{

typedef char* string;
#include <stdio.h>
#define YYSTYPE string

%}

%token DATATYPE NAME EQ DATA SEMICOLON IDT FDT CDT COMMA

%%

file : record file
| record
;

record : IDT NAME EQ IDATA SEMICOLON  { printf("%s %s %s %s %s is a
valid C statement", $1, $2, $3, $4, $5); }
        | FDT NAME EQ FDATA SEMICOLON { printf("%s %s %s %s %s is a
valid C statement", $1, $2, $3, $4, $5); }
        | FDT NAME EQ IDATA SEMICOLON { printf("%s %s %s %s %s is a
valid C statement", $1, $2, $3, $4, $5); }
        | CDT NAME EQ CDATA SEMICOLON { printf("%s %s %s %s %s is a
valid C statement", $1, $2, $3, $4, $5); }
        | IDT declare { printf("It is a valid C statement", $1, $2);
}
        | FDT declare  { printf("It is a valid C statement", $1,
$2); }
```

```
declare:
        NAME COMMA declare
        | NAME SEMICOLON

;

%%

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

int yyerror(char *msg)
{
        printf("Not a valid C statement : %s \n", msg);
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 05:07 PM calc]$ ./parser
int a;
A valid int declint 1a;
Error: syntax error
[s2020103016@centos8-linux Mon Dec 12 05:07 PM calc]$
```

Conversion infix to prefix expression using Yacc

EX NO: 7 **NAME: Kathir R M**

DATE: 11.10.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Convert infix to prefix expression using lex and yacc

AIM:

- To read the simple infix arithmetic expression and convert into prefix expression using lex and yacc

PROGRAM:

LEX

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
%}
%option noyywrap
%%
[a-z] {
yylval.val = yytext[0];
return ID;
}
[-*+/=()] {
return yytext[0];
}
\n {}
\t {}
[ ] {}
%%
```

YACC

```
%{
#include<stdio.h>
#include"y.tab.h"
int yylex(void);
char M = 'A';
```

```
extern FILE* yyin;
%}
%token ID
%start S
%left '+' '-'
%left '*' '/'
%union {
char val;
}
%%
S : E ;
E : E '+' E { printf("%c = %c %c +\n" ,M , $1.val , $3.val); $$.val = M;
M++;}
 | E '-' E { printf("%c = %c %c -\n" , M , $1.val , $3.val); $$.val = M;
M++;}
 | E '*' E { printf("%c = %c %c *\n" , M , $1.val , $3.val); $$.val = M;
M++;}
 | E '/' E { printf("%c = %c %c /\n" , M , $1.val , $3.val); $$.val = M;
M++;}
 | ID { $$.val  = $1.val; }
;
%%
int yyerror(char* msg) {
return 0;
}
int main() {
yyin = fopen("intopost.txt","r");
yyparse();
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 05:08 PM prc]$ cat intopost.txt
a+b*c-d/e
[s2020103016@centos8-linux Mon Dec 12 05:16 PM prc]$ ./parser
A = b c *
B = a A +
C = d e /
D = B C -
[s2020103016@centos8-linux Mon Dec 12 05:16 PM prc]$
```

Question – 2: (SPOT)

Convert infix to prefix expression using lex and yacc and show the grammar

AIM:

- To print the parsing action for infix to prefix conversion, using lex and yacc.

PROGRAM:

LEX:

```
%{

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

#include "y.tab.h"

extern char* yylval;

%}

INTEGER    [0-9]+
IDENTIFIER [_a-zA-Z][_a-zA-Z0-9]*

%%

exit.*          { return EXIT; }
quit.*          { return EXIT; }
{INTEGER}       { yylval = strdup(yytext); return INTEGER; }
{IDENTIFIER}    { yylval = strdup(yytext); return IDENTIFIER; }
[+-]            { yylval = strdup(yytext); return OPR1; }
[*/]            { yylval = strdup(yytext); return OPR2; }
"^"             { yylval = strdup(yytext); return OPR3; }
[()]            { return yytext[0]; }
\n              { return NEWLINE; }
.               { }

%%
int yywrap(void)        { }
```

YACC:

```
%{

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

#define YYSTYPE char*

extern int yylex();

int yyerror(const char *p) { printf("%s\n",p); return 1; }
```

```
%}

%token INTEGER IDENTIFIER OPR1 OPR2 OPR3 NEWLINE EXIT

%right OPR1
%right OPR2
%right OPR3

%start lines

%%

lines:
     | lines exp NEWLINE { printf("lines -> lines exp NEWLINE\n");
}
     ;

exp: exp OPR1 exp { printf("exp -> exp\nexp -> %s\nexp -> exp\n",
$2); }
   | exp OPR2 exp { printf("exp -> exp\nexp -> %s\nexp -> exp\n",
$2); }
   | exp OPR3 exp { printf("exp -> exp\nexp -> %s\nexp -> exp\n",
$2); }
   | '(' exp ')'  { printf("exp -> (\nexp -> exp\nexp -> )\n"); }
   | INTEGER      { printf("exp -> %s\n", $1); }
   | IDENTIFIER   { printf("exp -> %s\n", $1); }
   | EXIT { exit(0); }
   ;

%%

int yywrap()
{
    return 1;
}

int main()
{
    printf(">> ");
    yyparse();
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 06:21 PM ~]$ ./a.out
infix:5+2*3
exp->5
exp->2
exp->3
exp->exp*exp
exp->exp+exp
lines->lines exp NEWLINE
```

While syntax validation using Yacc

EX NO: 8 **NAME: Kathir R M**

DATE: 15.10.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Validate while syntax using lex and yacc:

AIM:

- To Validate while syntax using lex and yacc:

Lex file

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
%}
%option noyywrap
%%
while {
printf("Returned WHILE");
return WHILE;
}
[(] {
printf("Returned OPPAR");
return OPPAR;
}
[)] {
printf("Returned CLPAR");
return CLPAR;
}
[{] {
printf("Returned opcurl");
return OPCURL;
}
[}] {
printf("Returned clcurl");
return CLCURL;
}
[=] {
printf("Returned EQ");
```

```
return EQ;
}
[;] {
printf("Returned SEMC");
return SEMC;
}
[<] |
[>] |
[<][=] |
[>][=] {
printf("Returned relop");
return RELOP;
}
"&&" |
"||" {
printf("Returned logop");
return LOGOP;
}
[_a-zA-Z]([_a-zA-Z0-9])* {
printf("Returned id");
return ID;
}
[0-9]+ {
printf("Returned num");
return NUM;
}
\t {}
\n {}
[ ] {}
. {
printf("Error at %s\n" , yytext);
}
%%
```

Yacc file

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
extern FILE* yyin;
int yylex(void);
%}
%token WHILE OPPAR CLPAR OPCURL CLCURL RELOP LOGOP ID NUM SEMC EQ
%left '+' '-'
%left '*' '/'
%start S
%%
S : stmts {printf ("Valid syntax");};
stmts : stmts stmts
      |
      | WHILE OPPAR cond CLPAR OPCURL stmts CLCURL { printf("While line
found");}
      | assign
      | WHILE OPPAR cond CLPAR
```

```
;
cond : singcond LOGOP singcond
     | singcond;
singcond: ID RELOP NUM{printf("singcond fofound");}
        | ID RELOP ID{printf("singcond fofound");}
        | NUM RELOP NUM{printf("singcond fofound");}
        | NUM RELOP ID{printf("singcond fofound");}
assign: ID EQ NUM SEMC {printf("Assignment fofound");}
      | ID EQ ID SEMC
     ;
%%
void yyerror(const char* msg) {
printf("%s\n",msg);
}
int main()
{
yyin = fopen("while.txt","r");
yyparse();
}
```

INPUT:

```
yntax[s2020103016@centos8-linux Mon Dec 12 05:55 PM whileVal]$ cat while.txt
while ( i < 20 && k > 20) {
a = 10;
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 05:55 PM whileVal]$ make
lex while.l
yacc -d while.y
yacc: 13 shift/reduce conflicts, 3 reduce/reduce conflicts.
gcc *.c
./a.out
Returned WHILEReturned OPPARReturned idReturned relopReturned numsingcond fofoundReturned logopReturned idReturned relopReturned numsingcond
 fofoundReturned CLPARReturned opcurlReturned idReturned EQReturned numReturned SEMCAssignment fofoundReturned clcurlWhile line foundValid s
yntax[s2020103016@centos8-linux Mon Dec 12 05:55 PM whileVal]$
```

Question – 2: (SPOT)

Validate nested while syntax using lex and yacc:

AIM:

- To Validate nested while syntax using lex and yacc:

Lex file

```
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
%}
%option noyywrap
```

```
%%
while {
printf("Returned WHILE");
return WHILE;
}
[(] {
printf("Returned OPPAR");
return OPPAR;
}
[)] {
printf("Returned CLPAR");
return CLPAR;
}
[{] {
printf("Returned opcurl");
return OPCURL;
}
[}] {
printf("Returned clcurl");
return CLCURL;
}
[=] {
printf("Returned EQ");
return EQ;
}
[;] {
printf("Returned SEMC");
return SEMC;
}
[<] |
[>] |
[<][=] |
[>][=] {
printf("Returned relop");
return RELOP;
}
"&&" |
"||" {
printf("Returned logop");
return LOGOP;
}
[_a-zA-Z]([_a-zA-Z0-9])* {
printf("Returned id");
return ID;
}
[0-9]+ {
printf("Returned num");
return NUM;
}
\t {}
\n {}
[ ] {}
. {
printf("Error at %s\n" , yytext);
```

```
}
%%

Yacc file
%{
#include<stdio.h>
#include<stdlib.h>
#include "y.tab.h"
extern FILE* yyin;
int yylex(void);
%}
%token WHILE OPPAR CLPAR OPCURL CLCURL RELOP LOGOP ID NUM SEMC EQ
%left '+' '-'
%left '*' '/'
%start S
%%
S : stmts {printf ("Valid syntax");};
stmts : stmts stmts
      |
      | WHILE OPPAR cond CLPAR OPCURL stmts CLCURL { printf("While line
found");}
      | assign
      | WHILE OPPAR cond CLPAR
;
cond : singcond LOGOP singcond
     | singcond;
singcond: ID RELOP NUM{printf("singcond fofound");}
        | ID RELOP ID{printf("singcond fofound");}
        | NUM RELOP NUM{printf("singcond fofound");}
        | NUM RELOP ID{printf("singcond fofound");}
assign: ID EQ NUM SEMC {printf("Assignment fofound");}
      | ID EQ ID SEMC
     ;
%%
void yyerror(const char* msg) {
printf("%s\n",msg);
}
int main()
{
yyin = fopen("while.txt","r");
yyparse();
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 05:58 PM whileVal]$ make
lex while.l
yacc -d while.y
yacc: 13 shift/reduce conflicts, 3 reduce/reduce conflicts.
gcc *.c
./a.out
Returned WHILEReturned OPPARReturned idReturned relopReturned numsingcond fofoundReturned CLPARReturned opcurlReturned WHILEReturned OPPARRe
turned idReturned relopReturned numsingcond fofoundReturned CLPARReturned opcurlReturned idReturned EQReturned numReturned SEMCAssignment fo
foundReturned clcurlWhile line foundReturned idReturned EQReturned numReturned SEMCAssignment fofoundReturned clcurlWhile line foundValid sy
ntax[s2020103016@centos8-linux Mon Dec 12 05:58 PM whileVal]$ cat while.txt
while ( i < 20) {
while(j<40){
k=20;
}
a = 10;
}
```

Three Address Code for mathematical expressions

EX NO: 9 **NAME: Kathir R M**

DATE: 05.11.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Using lex and yacc convert simple mathematical expressions to 3 address code

AIM:
To convert simple mathematical expressions to 3 address code using lex and yacc.

PROGRAM

LEX

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

%%

[0-9]+ { yylval.dval = atoi(yytext);
          return NUM;
        }
[\t ] ;
\n { return 0; }
. { return yytext[0]; }

%%
```

YACC

```
%{
        #include <stdio.h>
        #include "y.tab.h"
        int yylex(void);
        void yyerror(char *msg);
        char p='A'-1;
%}

%union{
        char dval;
}
%token NUM
%left '+' '-'
%left '*' '/'
%nonassoc UMINUS
%start S

%%

S : E  {printf("\nx = %c\n",$<dval>$);}
;
E : NUM { p++; printf("\n%c = %d", p,$<dval>1); $<dval>$=p; }
|     E     '+'     E     {p++;printf("\n%c     =     %c     +     %c
",p,$<dval>1,$<dval>3);$<dval>$=p;}
|     E     '-'     E     {p++;printf("\n%c     =     %c     -     %c
",p,$<dval>1,$<dval>3);$<dval>$=p;}
|     E     '*'     E     {p++;printf("\n%c     =     %c     *     %c
",p,$<dval>1,$<dval>3);$<dval>$=p;}
```

```
|     E     '/'     E     {p++;printf("\n%c     =     %c     /     %c
",p,$<dval>1,$<dval>3);$<dval>$=p;}

| '('E')' { $<dval>$ = p; }

| '-' E %prec UMINUS { p++; printf("\n%c = - %c ",p, $<dval>2);
$<dval>$ = p;}

;


%%


void yyerror(char *msg) {

        fprintf(stderr,"error %s", msg);

}


int main () {

        printf("Enter expr: ");

        yyparse();

}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 05:59 PM week09]$ ./parser
Enter expr: 43+23*12

A = 43
B = 23
C = 12
D = B * C
E = A + D
x = E
[s2020103016@centos8-linux Mon Dec 12 06:00 PM week09]$
```

Question – 2: (SPOT)

Using lex and yacc convert array addressing to 3 address code:

AIM:

- To convert array addressing to 3 address code using lex and yacc.

PROGRAM

LEX

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

%option noyywrap

%%
[0-9]+ {yylval.dval = atoi(yytext); return NUM; }
[a-zA-Z]+ {yylval.dval = yytext[0]; return ID; }
[\t];
\n return 0;
. {return yytext[0];}
%%
```

YACC

```
%{
#include <stdio.h>
int yylex(void);
void yyerror(char *msg);
char p = 'A' - 1;
%}

%union { char dval; }
%token NUM, ID, TYPE
%left '+' '-'
%left '*' '/'
%nonassoc UMINUS
%start S
%%

S: K '+' K { printf("\n addr = %c + %c\n", $<dval>1, $<dval>3); }
 | K '=' K '+' K { printf("\n %c = %c + %c\n", $<dval>1, $<dval>3,
$<dval>5); }
 ;
K: ID '[' F ']' { p++; printf("\n %c = %c + 4 * %c\n", p, $<dval>1,
$<dval>3); $<dval>$ = p;}
 ;
F: E { p++; printf("\n%c = %c", p, $<dval>1); $<dval>$ = p;}
 | F ']' '[' E { p++; printf("\n%c = %c + %c * 25", p, $<dval>4,
$<dval>1); $<dval>$ = p;}
;
E: NUM {p++; printf("\n%c = %d", p, $<dval>1); $<dval>$ = p;}
 | E '+' E {p++; printf("\n %c = %c + %c", p, $<dval>1, $<dval>3);
$<dval>$ = p;}
 | E '-' E {p++; printf("\n %c = %c - %c", p, $<dval>1, $<dval>3);
```

```
$<dval>$ = p;}
 | E '*' E {p++; printf("\n %c = %c * %c", p, $<dval>1, $<dval>3);
$<dval>$ = p;}
 | E '/' E {p++; printf("\n %c = %c / %c", p, $<dval>1, $<dval>3);
$<dval>$ = p;}
 | '(' E ')' { $<dval>$ = p; }
 | '-' E %prec UMINUS {p++; printf("\n %c = -%c", p, $<dval>2);
$<dval>$ = p;}
 ;
%%

void yyerror(char *msg){
        fprintf(stderr, "error %s", msg);
}

int main() {
printf("Enter expr X = : ");
yyparse();
}
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 06:03 PM spot]$ ./a.out
Enter expr: x = a[23][1] + b[2]
A=23
B=A
C=1
D=C+B*25
E=a+4*d
F=2
G=F
H=b+4*G
x=E+H
[s2020103016@centos8-linux Mon Dec 12 06:03 PM spot]$
```

Three Address Code for control statements

EX NO: 10 **NAME: Kathir R M**

DATE: 08.11.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Generate three address code for if-else and while:

AIM:

- To generate and display three address code for if-else and while constructs in C, using lex and yacc.

PROGRAM:

LEX:

```
%{

#include <stdio.h>

#include "y.tab.h"

%}
%option noyywrap

%%

"if"    { return IF; }

"else"  { return ELSE; }

"while" { return WHILE; }

[A-Za-z] { yylval.var = yytext[0]; return ID; }

[\t ] ;

[\n]  ;

. { return yytext[0]; }

%%
```

YACC:

```
%{

#include <stdio.h>

int yylex(void);

void yyerror(char *msg);

char p = 'A' - 1;

int lc = 0;

char lb[3] = "L1";

%}

%union {
        struct LEinfo {
                char addrbase;
                char arrbase;
                char addr;
                char arr;
        } le;
        struct Sinfo {
                char label;
                int labcount;
                char code[1000];
        } s;
        struct Binfo {
                char tl;
                int tlcount;
                char fl;
                int flcount;
                char code[1000];
        } b;
        char var;
};

%token <var> ID

%token IF ELSE WHILE

%type <le> L

%type <le> E
```

```
%type <s> S

%type <b> B

%left '+' '-'

%left '*' '/'

%nonassoc UMINUS

%start P

%%

P : S { ++lc;
        $1.label = 'L';
        $1.labcount = lc;
        //printf("\n  %s  \n  %c  %d  :",  $1.code,  $1.label,
$1.labcount);
        }

S : IF '(' B ')' { ++lc; $3.tl = 'L'; $3.tlcount = lc++; $3.fl =
'L'; $3.flcount = lc; printf("if %c goto %c%d\n", $3.code[0], $3.fl,
$3.flcount); } S { printf("goto %c%d\n%c%d : \n",  $3.tl, $3.tlcount,
$3.fl,  $3.flcount);  }  ELSE  S  {  printf("%c%d  :  \n",  $3.tl,
$3.tlcount); }

  | WHILE '(' B ')' { ++lc; $3.tl = 'L'; $3.tlcount = lc++; $3.fl =
'L';  $3.flcount  =  lc;  printf("%c%d:\n",  $3.tl,  $3.tlcount);
printf("if %c goto %c%d\n", $3.code[0], $3.fl, $3.flcount); } S {
printf("goto %c%d\n", $3.tl, $3.tlcount); printf("%c%d:\n", $3.fl,
$3.flcount); }

  | ID '=' E ';' { printf("\n %c = %c\n\n", $1, $3.addr); }

  | L '=' E ';' { printf("\n %c[%c] = %c\n", $1.arrbase, $1.addr,
$3.addr); }

  ;

B : ID { $$.code[0] = $1; }

  ;

E : E '+' E { ++p; $$.addr = p; printf("\n %c = %c + %c", $$.addr,
$1.addr, $3.addr); }

  | ID { $$.addr = $1; }
```

```
  | L { ++p; $$.addr = p; printf("\n %c = %c[%c]", $$.addr, $1.arrbase, $1.addr); }

  ;

L : ID '[' E ']' { $$.arr = $1; $$.arrbase = $$.arr; ++p; $$.addr = p; printf("\n %c = %c * 40", $$.addr, $3); }

  | L '[' E ']' { $$.arr = $1.arr; char t = ++p; $$.addr = ++p; printf("\n %c = %c * 4", t, $3); printf("\n %c = %c + %c", $$.addr, $1.addr, t); }

  ;

%%

void yyerror(char *msg) {

    fprintf(stderr,"error %s", msg);

}

int main () {

    yyin = fopen(“input.txt”,”r”);

    yyparse();

}
```

INPUT:

```
 L 3 :[s2020103016@centos8-linux Mon Dec 12 06:07 PM exe]$ cat input.txt
if(a)
       a = x[i][j] + c;
else
       a = x[i][j];
[s2020103016@centos8-linux Mon Dec 12 06:07 PM exe]$
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 06:07 PM exe]$ ./a.out
Enter expr: if t1 goto L2

 A = i * 40
 B = j * 4
 C = A + B
 D = x[C]
 E = D + c
 a = E

goto L1
L2 :

 F = i * 40
 G = j * 4
 H = F + G
 I = x[H]
 a = I

L1 :


 L 3 :[s2020103016@centos8-linux Mon Dec 12 06:07 PM exe]$ cat
```

Question – 2: (SPOT)

Generate three address code for switch-case:

AIM:

•To generate and display three address code for switch-case construct in C, using lex and yacc.

PROGRAM:

LEX:

```
%{

#include <stdio.h>
#include "y.tab.h"

%}

%option noyywrap

%%

"switch" { return SWITCH; }

"if"    { return IF; }

"else"  { return ELSE; }

"while" { return WHILE; }

"case"  { return CASE; }

"break" { return BREAK; }
```

```
"default" { return DEFAULT; }

[A-Za-z]        { yylval.var = yytext[0]; return ID; }

"'"[A-Za-z]"'"  { yylval.var = yytext[1]; return ID; }
[0-9]+  { yylval.val = atoi(yytext); return NUM; }

[\t ] ;

[\n]  ;

. { return yytext[0]; }

%%
```

YACC:

```
%{

#include <stdio.h>

int yylex(void);
void yyerror(char *msg);

char p = 'A' - 1;
int lc = 0;
char lb[3] = "L1";
char switchid;
char switchlabel;
int switchlabcount;

extern FILE* yyin;

%}

%union {
        struct LEinfo {
                char addrbase;
                char arrbase;
                char addr;
                char arr;
                int size;
        } le;
        struct Sinfo {
                char label;
                int labcount;
                char code[1000];
        } s;
```

```
        struct Binfo {
                char tl;
                int tlcount;
                char fl;
                int flcount;
                char code[1000];
        } b;
        char var;
        struct LSinfo {
                char tl;
                int tlcount;
                char fl;
                int flcount;
                char id;
                char label;
                int labcount;
        } ls;
        int val;
};

%token <var> ID

%token <val> NUM

%token IF ELSE WHILE SWITCH CASE BREAK DEFAULT

%type <le> L

%type <le> E

%type <s> S

%type <b> B

%type <ls> LABST

%left '+' '-'

%left '*' '/'

%nonassoc UMINUS

%start P

%%

P :  S

S : IF '(' B ')' { ++lc; $3.tl = 'L'; $3.tlcount = lc++; $3.fl =
```

```
'L'; $3.flcount = lc; $$.label = 'L'; ++lc; $$.labcount = lc;
printf("if t1 goto %c%d\ngoto %c%d\n%c%d:\n", $3.tl, $3.tlcount,
$3.fl, $3.flcount, $3.tl, $3.tlcount); } S { printf("goto %c%d\n%c%d
: \n",   $$.label, $$.labcount, $3.fl, $3.flcount); } ELSE S {
printf("%c%d:\n", $$.label, $$.labcount); }

  | WHILE '(' B ')' { ++lc; $3.tl = 'L'; $3.tlcount = lc++; $3.fl =
'L'; $3.flcount = lc; $$.label = 'L'; $$.labcount = ++lc;
printf("%c%d:\nif t1 goto %c%d\ngoto %c%d\n%c%d:\n", $$.label,
$$.labcount,  $3.tl,  $3.tlcount,  $3.fl,  $3.flcount,  $3.tl,
$3.tlcount); } S { printf("goto %c%d \n %c%d: \n", $$.label,
$$.labcount, $3.fl, $3.flcount);  }

  | SWITCH '(' ID ')' '{' { switchid = $3; lc++; } LABST '}' { }

  | ID '=' E ';' { printf("%c = %c\n\n", $1, $3.addr); }

  | L '=' E ';' { ++p; printf("%c[%c] = %c\n",  $1.arrbase, $1.addr,
$3.addr); }

  ;

B : ID { }

  ;

LABST : CASE ID ':' { $$.tl = 'L'; $$.tlcount = ++lc; $$.fl = 'L';
$$.flcount = ++lc; printf("if %c == %c goto %c%d\ngoto %c%d\n",
switchid,   $2,   $$.tl,   $$.tlcount,   $$.fl,   $$.flcount);
printf("%c%d:\n",  $$.tl,  $$.tlcount);  switchlabel  =  $$.fl;
switchlabcount  =  $$.flcount;}  S  BREAK  ';'  {  printf("goto
L1\n%c%d:\n", switchlabel, switchlabcount);  } LABST  { }

  | CASE NUM ':' { $$.tl = 'L'; $$.tlcount = ++lc; $$.fl = 'L';
$$.flcount = ++lc; printf("if %c == %d goto %c%d\ngoto %c%d\n",
switchid,   $2,   $$.tl,   $$.tlcount,   $$.fl,   $$.flcount);
printf("%c%d:\n",  $$.tl,  $$.tlcount);  switchlabel  =  $$.fl;
switchlabcount  =  $$.flcount;}  S  BREAK  ';'  {  printf("goto
L1\n%c%d:\n", switchlabel, switchlabcount);  } LABST  { }

  | CASE ID ':' { $$.tl = 'L'; $$.tlcount = ++lc; $$.fl = 'L';
$$.flcount = ++lc; printf("if %c == %c goto %c%d\ngoto %c%d\n",
switchid,   $2,   $$.tl,   $$.tlcount,   $$.fl,   $$.flcount);
printf("%c%d:\n",  $$.tl,  $$.tlcount);  switchlabel  =  $$.fl;
switchlabcount = $$.flcount;} S { printf("%c%d:\n", switchlabel,
switchlabcount);  } LABST  { }

  | CASE NUM ':' { $$.tl = 'L'; $$.tlcount = ++lc; $$.fl = 'L';
$$.flcount = ++lc; printf("if %c == %d goto %c%d\ngoto %c%d\n",
```

```
switchid, $2, $$.tl, $$.tlcount, $$.fl, $$.flcount);
printf("%c%d:\n", $$.tl, $$.tlcount); switchlabel = $$.fl;
switchlabcount = $$.flcount;} S { printf("%c%d:\n", switchlabel,
switchlabcount); } LABST { }

  | DEFAULT ':' S { printf("L1:\n"); }

  ;

E : E '+' E { ++p; $$.addr = p; printf("%c = %c + %c\n", $$.addr,
$1.addr, $3.addr); }

  | ID { $$.addr = $1; }

  | L { ++p; $$.addr = p; printf("%c = %c[%c]\n", $$.addr,
$1.arrbase, $1.addr); }

  ;

L : ID '[' E ']' { $$.arr = $1; $$.arrbase = $$.arr; ++p; $$.addr =
p; printf("%c = %c * 40\n", $$.addr, $3); }

  | L '[' E ']' { $$.arr = $1.arr; char t = ++p; $$.addr = ++p;
printf("\%c = %c * 4\n", t, $3); printf("%c = %c + %c\n", $$.addr,
$1.addr, t); }

  ;

%%

void yyerror(char *msg) {

    fprintf(stderr,"error %s", msg);

}

int main (int argc, char* argv[]) {

    yyin = fopen(argv[1], "r");
    yyparse();

}
```

INPUT:

```
[s2020103016@centos8-linux Mon Dec 12 06:10 PM week11]$ cat input.txt
switch (a) {
        case 1:
                b = b * c;
                break;
        case 2:
                x[i][j] =  5 * a;
                break;
}
[s2020103016@centos8-linux Mon Dec 12 06:10 PM week11]$
```

Output:

```
centos8-linux Mon Dec 12 06:09 PM week11]$ ./parser
if a != 1 goto L1

 A = b * c
 B = b
goto Lf

L1 :
if a != 2 goto L2

 C = i * 40
 D = j * 4
 E = C + D
 F = 5 * a
 G = x[E]
G = F
goto Lf

L2 :

Lf:
```

Three Address Code to Assembly Code

EX NO: 11 **NAME: Kathir R M**

DATE: 06.12.2022 **ROLLNO: 2020103016**

MARKS:

Normal exercise (15)	Spot exercise (5)	Document (5)	Total (25)

Question – 1:

Convert three address code of an arithmetic expression to assembly code

AIM:

- To convert three address code of an arithmetic expression to assembly code

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

#include<stdlib.h>

#include<string.h>

int yylex(void);

void yyerror(char *msg);

extern FILE *yyin;

int l = 0;

%}

%union {

        char exp[25];

        int val;

}

%token TMP VAR NUM

%start pgm

%left '+' '-'

%left '*' '/'
```

```
%%

pgm: S { printf("\n COMPLETED!!! \n\n"); }

      ;

S : S S

    | TMP '=' E '+' E ';' { printf("\n LD $R%d, %s", l+1, $3.exp);

                           printf("\n LD $R%d, %s", l+2, $5.exp);

                           printf("\n  ADD  $%s,  $R%d,  $R%d",$1.exp,  l+1,
l+2); l+=2; }

  | TMP '=' TMP '+' E ';' { printf("\n LD $R%d, %s", l+1, $5.exp);

                             printf("\n ADD $%s, $%s, $R%d",$1.exp, $3.exp,
l+1); l+=1; }

  | TMP '=' E '+' TMP ';' { printf("\n LD $R%d, %s", l+1, $3.exp);

                             printf("\n  ADD  $%s,  $R%d,  $%s",$1.exp,  l+1,
$5.exp); l+=1; }

  | TMP '=' TMP '+' TMP ';' { printf("\n ADD $%s, $%s, $%s",$1.exp, $3.exp,
$5.exp); }

  | TMP '=' E '-' E ';' { printf("\n LD $R%d, %s", l+1, $3.exp);

                          printf("\n LD $R%d, %s", l+2, $5.exp);

                          printf("\n  SUB  $%s,  $R%d,  $R%d",$1.exp,  l+1,
l+2); l+=2; }

  | TMP '=' TMP '-' E ';' { printf("\n LD $R%d, %s", l+1, $5.exp);

                             printf("\n SUB $%s, $%s, $R%d",$1.exp, $3.exp,
l+1); l+=1; }

  | TMP '=' E '-' TMP ';' { printf("\n LD $R%d, %s", l+1, $3.exp);

                             printf("\n  SUB  $%s,  $R%d,  $%s",$1.exp,  l+1,
$5.exp); l+=1; }

  | TMP '=' TMP '-' TMP ';' { printf("\n SUB $%s, $%s, $%s",$1.exp, $3.exp,
$5.exp); }

   | TMP '=' E '/' E ';' { printf("\n LD $R%d, %s", l+1, $3.exp);

                          printf("\n LD $R%d, %s", l+2, $5.exp);

                          printf("\n  DIV  $%s,  $R%d,  $R%d",$1.exp,  l+1,
```

```
l+2); l+=2; }

  | TMP '=' TMP '/' E ';' { printf("\n LD $R%d, %s", l+1, $5.exp);

                            printf("\n DIV $%s, $%s, $R%d",$1.exp, $3.exp,
l+1); l+=1; }

  | TMP '=' E '/' TMP ';' { printf("\n LD $R%d, %s", l+1, $3.exp);

                            printf("\n DIV $%s, $R%d, $%s",$1.exp, l+1,
$5.exp); l+=1; }

  | TMP '=' TMP '/' TMP ';' { printf("\n DIV $%s, $%s, $%s",$1.exp, $3.exp,
$5.exp); }

   | TMP '=' E '*' E ';' { printf("\n LD $R%d, %s", l+1, $3.exp);

                           printf("\n LD $R%d, %s", l+2, $5.exp);

                           printf("\n MULT $%s, $R%d, $R%d",$1.exp, l+1,
l+2); l+=2; }

  | TMP '=' TMP '*' E ';' { printf("\n LD $R%d, %s", l+1, $5.exp);

                            printf("\n  MULT  $%s,  $%s,  $R%d",$1.exp,
$3.exp, l+1); l+=1; }

  | TMP '=' E '*' TMP ';' { printf("\n LD $R%d, %s", l+1, $3.exp);

                            printf("\n MULT $%s, $R%d, $%s",$1.exp, l+1,
$5.exp); l+=1; }

  | TMP '=' TMP '*' TMP ';' { printf("\n MULT $%s, $%s, $%s",$1.exp,
$3.exp, $5.exp); }

  | TMP '=' TMP ';' {printf("\n MOVE $%s , $%s" , $1.exp , $3.exp);}
  | TMP '=' E ';' { printf("\n LI $%s , #%s.val" , $1.exp , $3.exp);}
  |  VAR  '='  NUM  ';'  {  printf("\n  LD  $R%d  ,  %s",  l+1  ,
$3.exp);l+=1;sprintf($1.exp,"$R%d",l-1);}
  | ;

E : VAR { strcpy($$.exp, $1.exp); }

    ;

%%

void yyerror(char *msg) {

        fprintf(stderr, "Error in parsing: %s\n", msg);

}

int main(int argc, char *argv[]) {
```

```
        yyin = fopen(argv[1], "r");

        yyparse();

        return 1;

}
```

INPUT:

```
[s2020103016@centos8-linux Mon Dec 12 06:13 PM week12]$ cat input.txt
t1 = c/d;
t2 = t1 * e;
t1 = b + t2;
t2 = a + t1;
t1 = f * g;
t3 = h * i;
t1 = t1 - t3;
t3 = t2/t1;
t1 = t2;
a=12;
t2 = q / r;
t3 = t2 / t1;
t1 = a + b;
```

OUTPUT:

```
[s2020103016@centos8-linux Mon Dec 12 06:13 PM week12]$ ./a.out input.txt

 LD $R1, c
 LD $R2, d
 DIV $t1, $R1, $R2
 LD $R3, e
 MULT $t2, $t1, $R3
 LD $R4, b
 ADD $t1, $R4, $t2
 LD $R5, a
 ADD $t2, $R5, $t1
 LD $R6, f
 LD $R7, g
 MULT $t1, $R6, $R7
 LD $R8, h
 LD $R9, i
 MULT $t3, $R8, $R9
 SUB $t1, $t1, $t3
 DIV $t3, $t2, $t1
 MOVE $t1 , $t2
 LI $R10 , 12
 LD $R11, q
 LD $R12, r
 DIV $t2, $R11, $R12
 DIV $t3, $t2, $t1
 LD $R13, b
 ADD $t1 , $R10 , $R13
 COMPLETED!!!
```

Question – 2: (SPOT)

Convert three address code of an control statement to assembly code

AIM:

- To convert three address code of an control statement to assembly code

PROGRAM

LEX

```
%{
        #include "y.tab.h"
%}

%option noyywrap

%%

[a-z] { yylval.val = *yytext; return VAR; }
"=" { return ASSIGN; }
[+*/-] { yylval.val = *yytext; return OP; }
[-+]?[0-9]+ {
        int sign = 1;
        if (yytext[0] == '-' ) {
                sign = -1;
                yytext[0] = '0';
        }
        yylval.val = atoi(yytext) * sign; return VAL;
}
("=="|"<="|"!="|">="|"<"|">") { yylval.str = strdup(yytext); return
RELOP; }
"if" { return IF; }
"goto" {return GOTO; }
[A-Z] { yylval.val = *yytext; return A; }
[A-Z]":" { yylval.str = strdup(yytext); return LABEL; }
"//".* {}
[\n\t ] { }

--

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

%option noyywrap
```

```
var [a-zA-Z]+
num [0-9]+
condnOp ("=="|"!="|"<="|">="|"<"|">")
stepOp ("++"|"--")
op [+*/-]
assignOp {op}?"="
dtype ("int"|"float"|"char")

%%

"if"  { return IF; }
"else" { return ELSE; }
"while" { return WHILE; }
"for" { return FOR; }
{op} { yylval.val = strdup(yytext); return OP; }
{dtype} {yylval.val = strdup(yytext); return DT; }
{condnOp} { yylval.val = strdup(yytext); return OP; }
{stepOp} { yylval.val = strdup(yytext); return OP; }
{assignOp} { yylval.val = strdup(yytext); return ASSIGN; }
{var} { yylval.val = strdup(yytext); return VAR; }
{num} { yylval.val = strdup(yytext); return VAL; }
[](;}{] { return *yytext; }
[\n\t ] ;

%%
```

YACC

```
%{
        #include <stdio.h>
        #include <string.h>
        #include "y.tab.h"
        int yylex(void);
        void yyerror(char *msg);
        int var = 0;
        int map[26] = { [0 ... 25] = -1 };
        extern FILE *yyin;
%}

%token OP VAL VAR ASSIGN IF GOTO A RELOP LABEL
%union {
        int val;
        char * str;
}
%type <val> VAL OP VAR A
%type <str> RELOP LABEL

%%
```

```
stmts: stmt stmts
| stmt ;
stmt: VAR ASSIGN VAL OP VAL {
        printf("li $a0, %d\n", $3);
        if ($4 == '+') {
                printf("addi  $t%d,  $a0,  %d\n",var,  $5);  map[$1  -
'a'] = var++;
        }
        else if ($4 == '-') {
                printf("subi $t%d, $a0, %d\n",var, $3, $5); map[$1
- 'a'] = var++;
        }
        else if ($4 == '/') {
                printf("div $a0, %d\n",var, $5);
                printf("mflo $t%d\n", var);  map[$1 - 'a'] = var++;
        }
        else if ($4 == '*'){
                printf("mult $a0, %d\n",var, $5);
                printf("mflo $t%d\n", var);  map[$1 - 'a'] = var++;
        }
}
|  VAR ASSIGN VAL OP VAR {
        if (map[$5 -'a'] == -1) {
                printf("lw ($t%d), %c\n", var, $5);
                map[$5 - 'a'] = var++;
        }
        if ($4 == '+') {
                printf("addi  $t%d,  $t%d,  %d\n",var,  map[$5  -'a'],
$3); map[$1 - 'a'] = var++;
        }
        else if ($4 == '-') {
                printf("subi  $t%d,  $t%d,  %d\n",var,  map[$5  -'a'],
$3); map[$1 - 'a'] = var++;
        }
        else if ($4 == '/') {
                printf("div $t%d, %d\n", map[$5 - 'a'], $3);
                printf("mflo $t%d\n", var);  map[$1 - 'a'] = var++;
        }
        else if ($4 == '*'){
                printf("mult $t%d, %d\n", $3, map[$5 - 'a'], $3);
                printf("mflo $t%d\n", var);  map[$1 - 'a'] = var++;
        }
}
| VAR ASSIGN VAR OP VAL {
        if (map[$3 -'a'] == -1) {
                printf("lw ($t%d), %c\n", var, $3);
                map[$3 - 'a'] = var++;
        }
        if ($4 == '+') {
```

```
                printf("addi $t%d, $t%d, %d\n",var, map[$3 - 'a'],
$5); map[$1 - 'a'] = var++;
        }
        else if ($4 == '-') {
                printf("subi $t%d, $t%d, %d\n",var, map[$3 -'a'],
$5); map[$1 - 'a'] = var++;
        }
        else if ($4 == '/') {
                printf("div $t%d, %d\n", map[$3 - 'a'], $5);
                printf("mflo $t%d\n", var);  map[$1 - 'a'] = var++;
        }
        else if ($4 == '*'){
                printf("mult $t%d, %d\n", map[$3 -'a'], $5);
                printf("mflo $t%d\n", var);  map[$1 - 'a'] = var++;
        }
}
| VAR ASSIGN VAR OP VAR {
        if (map[$5 -'a'] == -1) {
                printf("lw ($t%d), %c\n", var, $5);
                map[$5 - 'a'] = var++;
        }
        if (map[$3 -'a'] == -1) {
                printf("lw ($t%d), %c\n", var, $3);
                map[$3 - 'a'] = var++;
        }
        if ($4 == '+') {
                printf("add $t%d, $t%d, $t%d\n",var, map[$3 - 'a'],
map[$5 - 'a']); map[$1 - 'a'] = var++;
        }
        else if ($4 == '-') {
                printf("sub $t%d, $t%d, $t%d\n",var, map[$3 - 'a'],
map[$5 - 'a']); map[$1 - 'a'] = var++;
        }
        else if ($4 == '/') {
                printf("div $t%d, $t%d\n", map[$3 - 'a'], map[$5 -
'a']);
                printf("mflo $t%d\n", var);  map[$1 - 'a'] = var++;
        }
        else if ($4 == '*'){
                printf("mult $t%d, $t%d\n", map[$3 - 'a'], map[$5 -
'a']);
                printf("mflo $t%d\n", var);  map[$1 - 'a'] = var++;
        }
}
| VAR ASSIGN VAL { printf("li $t%d, %d\n", var, $3); map[$1 - 'a']
= var++; }
| VAR ASSIGN VAR {
        if (map[$3 -'a'] == -1) {
                printf("lw ($t%d), %c\n", var, $3);
```

```
                map[$3 - 'a'] = var++;
        }
        printf("move $t%d, $t%d\n", var, map[$3 - 'a']); map[$1 -
'a'] = var++;
}
| IF VAR RELOP VAR GOTO A {
        if (map[$2 -'a'] == -1) {
                printf("lw ($t%d), %c\n", var, $2);
                map[$2 - 'a'] = var++;
        }
        if (map[$4 -'a'] == -1) {
                printf("lw ($t%d), %c\n", var, $4);
                map[$4 - 'a'] = var++;
        }
        if (strcmp($3,"==") == 0) {
                printf("beq $t%d, %t%d, %c\n", map[$2 -'a'],map[$4-
'a'],$6);
        }
        else if (strcmp($3,">=") == 0) {
                printf("bge $t%d, %t%d, %c\n", map[$2 -'a'],map[$4-
'a'],$6);
        }
        else if (strcmp($3,"<=") == 0) {
                printf("ble $t%d, %t%d, %c\n", map[$2 -'a'],map[$4-
'a'],$6);
        }
        else if (strcmp($3,">") == 0) {
                printf("bgt $t%d, %t%d, %c\n", map[$2 -'a'],map[$4-
'a'],$6);
        }
        else if (strcmp($3,"<") == 0) {
                printf("blt $t%d, %t%d, %c\n", map[$2 -'a'],map[$4-
'a'],$6);
        }
        else if (strcmp($3,"!=") == 0) {
                printf("bne $t%d, %t%d, %c\n", map[$2 -'a'],map[$4-
'a'],$6);
        }
}
| IF VAR RELOP VAL GOTO A {
        if (map[$2 -'a'] == -1) {
                printf("lw ($t%d), %c\n", var, $2);
                map[$2 - 'a'] = var++;
        }
        if (strcmp($3,"==") == 0) {
                printf("beq $t%d, %d, %c\n", map[$2 -'a'],$4,$6);
        }
        else if (strcmp($3,">=") == 0) {
                printf("bge $t%d, %d, %c\n", map[$2 -'a'],$4,$6);
```

```
        }
        else if (strcmp($3,"<=") == 0) {
                printf("ble $t%d, %d, %c\n", map[$2 -'a'],$4,$6);
        }
        else if (strcmp($3,">") == 0) {
                printf("bgt $t%d, %d, %c\n", map[$2 -'a'],$4,$6);
        }
        else if (strcmp($3,"<") == 0) {
                printf("blt $t%d, %d, %c\n", map[$2 -'a'],$4,$6);
        }
        else if (strcmp($3,"!=") == 0) {
                printf("bne $t%d, %d, %c\n", map[$2 -'a'],$4,$6);
        }
}
| LABEL {
        printf("%s\n",$1);
}
| GOTO A {
        printf("j %c\n", $2);
}
;


%%

void yyerror(char *msg) {
        printf("Wrong format %s\n",msg);
}

int main (int argc, char *argv[]) {
        yyin = fopen("intermediate.txt","r");
        yyparse();
}

----

%{
    #include <stdio.h>
    #include "y.tab.h"
    int yylex(void);
    void yyerror(char *s);
        extern FILE *yyin;
        extern FILE *yyout;
        int addr = 0;
        typedef struct {
                char if_true;
                char if_false;
        }ctrl;
        char var = 'A';
```

```
        char map[26] = { [0 ... 25] = '\0' };
        ctrl m;
%}

%token IF ELSE WHILE FOR DT OP ASSIGN VAR VAL
%union {
    char *val;
}
%start start
%type <val> OP VAR VAL ASSIGN E

%%

start: IF '(' B ')' S  { fprintf(yyout,"%c:\n",m.if_false); }
| IF '(' B ')' S e  S { fprintf(yyout,"%c:\n",m.if_false); }
| WHILE '(' B ')'  S   {
        fprintf(yyout,"goto %c\n", map[m.if_true]);
        addr++;
        fprintf(yyout,"%c:\n", m.if_false);
}
| FOR '(' init ';' B  ';' step ')'  S  {
        fprintf(yyout,"goto %c\n",map[m.if_true]);
        addr++;
        fprintf(yyout,"%c:\n",m.if_false);
}
;
e: ELSE {
        fprintf(yyout,"goto %c\n",var++);
        addr++;
        fprintf(yyout,"%c:\n",m.if_false);
        m.if_false = var - 1;
}
B: VAR OP VAL {
        map[addr] = var;
        fprintf(yyout,"%c:\n",var++);
        fprintf(yyout,"if %s %s %s goto %c\n", $1, $2, $3, var);
        map[addr + 2] = var++;
        m.if_true = addr;
        m.if_false = var;
        addr++;
        fprintf(yyout,"goto %c\n", var);
        map[addr] = var++;
        addr++;
        fprintf(yyout,"%c:\n",map[addr]);
}
;
init: DT VAR ASSIGN VAL {
        fprintf(yyout,"%s %s %s\n", $2, $3, $4);
        addr++;
```

```
}
| VAR ASSIGN VAL {
        fprintf(yyout,"%s %s %s\n", $1, $2, $3);
        addr++;
};
step: VAR OP {
        fprintf(yyout,"%s = %s %c 1\n", $1, $1, $2[0]);
        addr++;
};
S : VAR ASSIGN VAL ';' S {
        fprintf(yyout,"%s %s %s\n", $1, $2, $3);
        addr++;
}
| DT VAR ASSIGN VAL ';' S {
        fprintf(yyout,"%s %s %s\n", $2, $3, $4);
        addr++;
}
| VAR ASSIGN E OP E ';' S {
        fprintf(yyout,"%s = %s %s %s\n", $1, $3, $4, $5);
        addr++;
}
| '{' S '}'
| ;
E : VAR {
        strcpy($$, $1);
}
| VAL {
        strcpy($$, $1);
}
;

%%

void yyerror (char *s) {
    fprintf(yyout,"ERR: %s", s);
}

int main (int argc, char *argv[]) {
        yyin = fopen(argv[1],"r");
        yyout = fopen("intermediate.txt","w");
    yyparse();
    return 1;
}
```

OUTPUT:

s2020103016@centos8-linux:~/Sem5/CD/week12/spot/it2

```
[s2020103016@centos8-linux Mon Dec 12 07:06 PM it2]$ cat intermediate.txt
i = 1
A:
if i < 10 goto B
goto C
B:
i = i + 1
q = a + b
goto A
C:
[s2020103016@centos8-linux Mon Dec 12 07:06 PM it2]$ lex assem.l
[s2020103016@centos8-linux Mon Dec 12 07:06 PM it2]$ yacc -d assem.y
[s2020103016@centos8-linux Mon Dec 12 07:06 PM it2]$ gcc *.c
[s2020103016@centos8-linux Mon Dec 12 07:06 PM it2]$ ./a.out
li $t0, 1
A:
blt $t0, 10, B
j C
B:
addi $t1, $t0, 1
lw ($t2), b
lw ($t3), a
add $t4, $t3, $t2
j A
C:
[s2020103016@centos8-linux Mon Dec 12 07:06 PM it2]$ cat input.txt
for ( i = 1 ; i < 10 ; i++) {
        q = a + b;
}
[s2020103016@centos8-linux Mon Dec 12 07:07 PM it2]$
```

```
[s2020103016@centos8-linux Mon Dec 12 07:08 PM it2]$ ./a.out
A:
lw ($t0), a
blt $t0, 5, B
j C
B:
lw ($t1), c
lw ($t2), r
add $t3, $t2, $t1
j A
C:
[s2020103016@centos8-linux Mon Dec 12 07:08 PM it2]$ cat intermediate.txt
A:
if a < 5 goto B
goto C
B:
q = r + c
goto A
C:
[s2020103016@centos8-linux Mon Dec 12 07:08 PM it2]$ cat input.txt
while ( a < 5 ) {
 q = r + c;
}
```