Expt.No.
Date 7/9/22
Page No.

Practical No. 1.

Aim:- Design a analyzer for a given language and the lexical analyzer should list various token present in input string.

S/W Required :- GCDB compiler.

Program :-

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{
char str[20], op[10], chr[10], num[10];
int j, i=0, k=0, m=0, l;
cout << "Enter the equation:";
cin >> str;
l = strlen(str);
for (j=0; j<1; j++)
{
if ((str[j]>='a' && str[j]='z') ||
(str[j]>='A' && str[j]<='z'))
{
chr[i] = str[j];
i++;
```

Expt.No.
Date
Page No.

```
}
else if (str[j]>='0' && str[j]<='9')
{
num[m]= str[j];
m++;
}
else
{
op[k]= str[j];
k++;
}

}
cout<<"\n OPERATOR: {";
for (j=0; j<k; j++)
{
cout<<op[j];
if (j<k-1) cout<<",";
}
cout<<"}\n CHARACTER: {";
for (j=0; j<i; j++)
{
cout<< chr[j];
if (j<i-1) cout<<",";
```

Output:-

Enter the equation : 2x+3y=0

operator : {+, =}.
character : {x, y}.
number : {2, 3, 0}.

Conclusion :- Hence, we have successfully study to design a lexical analyzer & listed out various token present in input string.

Expt.No
Date
Page No.

```
}
cout << "} \n Number ; {";
for (j=0 ; j<m ; j++)
{
cout << num [j];
if (j<m-1) cout << ", ";
}
cout << " } \n ";
return 0;
}.
```

Output :-

Enter the equation : $2x+3y=0$.

operator : { +, = }
character : { x, y }.
number : { 2, 3, 0 }.

Conclusion :- Hence, we have successfully study to design a lexical analyzer & listed out various token present in input string.

3/11

Expt.No.
Date 14/9/22
Page No.

Practical No. 2.

Aim:- Write a program to identify whether a given line is comment or not.

S/W Required:- GDB compiler.

Program:-

```
#include <bits/stdc++.h>
using namespace std;

void isComment (string line)
{
  if (line[0] == '/' && line[1] == '/'
      && line[2] != '/') {

    cout << "It is a single-line comment";
    return;
  }
  if (line[line.size() - 2] == '*'
  && line[line.size() - 1] == '/' && line[0]
  == '/' && line[1] == '*') {

  cout << "It is a multi multi-line comment";
```

Output :-

It is a multi - line comment.

Conclusion :- Thus, we have successfully run program to check whether line is comment or not.

Expt.No.
Date
Page No.

```
    return ;
}
cout << " It is not a comment ";
int main ()
{
  string line = " /* SIPNA college of engineering
                   & technology, Amravati */ "
  isComment (line);
    return 0;
}.
```

Output:-

It is a multi - line comment.

Conclusion:- Thus, we have successfully run program to check ~~wether~~ whether line comment or not.

Expt.No. _____
Date 25/9/22
Page No. _____

Practical No. 3

Aim:- Implement a program to check parenthesis of regular expression is balanced or not.

S/W Required:- Online gdb compiler.

Program:-

```
#include<bits/stdc++.h>
using namespace std;
bool arebracket(string expire){

    stack<char> s;
char x;
    for(int i=0; i<expire.length(); i++)
{
    if ( expre[i]=='(' || expre[i]=='[' ||
                                expre[i]=='{')
    {
      s.push(expre[i]);
      continue;
    }
    if (s.empty())
        return false;
```

Expt.No.
Date
Page No.

```
switch (expr[i])
{
    case ')':
    x = s.top();
    s.pop();
    if (x == '{' || x == '[')
    return false;
    break;

    case '}':
    x = s.top();
    s.pop();
    if (x == '(' || x == '[')
    return false;
    break;

    case ']';
    x = s.top();
    s.pop();
    if (x == '{' || x == '(')
    return false;
    break;
}
```

Output :-

This is balance equation

Conclusion :- Hence I have studied to check paranthesis of regular expression is balanced or not.

Expt.No. _____
Date _____
Page No. _____

```
}
return (s.empty());
}

int main()
{
    string expre = "{()}[]";
    if (arebracket(expre))
    {
        cout << "this is balanced equation";
    }
    else
    {
        cout << "This is not Balanced Equation";
    }

return 0;
}
```

Output :-

This is balance equation.

A+

3/1/

Conclusion :- Here I have studied to check parenthesis of regular expression is balanced or not.

Practical No. 4

Aim :- Write program to implement finite automata for checking given input string is accepted or not.

S/W Required :- g++ compiler in linux

Output :-

Enter the no. of states : 3
Enter state : 1 : A
Enter state : 2 : B
Enter state : 3 : C
Enter initial state : A
Enter final state : C

Enter the no. of input symbol : 2
Enter i/p symbol 1 : 1
Enter i/p symbol 2 : 0.
Enter no of transitions : 3
Enter transition 1 : A1B
Enter transition 2 : B1B.
Enter transition 3 : B0C.

A ⟶ 1B
B ⟶ 1B
C ⟶ 0C

Enter String : 110.

Expt.No. _____
Date 12/10/22
Page No. _____

Practical No. 4.

Aim:- Write a program to implement finite automata for checking a given input string is accepted or not.

S/W required:- g++ compiler in linux.

Program:-

```
#include <iostream>
using namespace std;
class Trans
{
public:
char ip_state, op_state, ip_symb;
trans()
{
  ip_state = op_state = ip_symb = '\0';
}
};
int main()
{
int nos, ips, notr, ic, i;
char state[10], ipsmb[10], str[5], istate, fstate, extra;
trans tr[20];
```

Expt.No. _____
Date _____
Page No. _____

```
cout << "Enter the no. of state: ";
cin >> nos;
for (int i=0; i<nos; i++)
{
    cout << "Enter state: " << i+1 << ": ";
    cin >> state[i];
}

cout << "enter initial state: ";
cin >> istate;
cout << "enter final state: ";
cin >> fstate;

cout << "Enter no. of i/p symbol: ";
cin >> ips;
for (i=0; i<ips; i++)
{
    cout << "Enter i/p symbol: " << i+1 << ": ";
    cin >> ipsmb[i];
}
cout << "Enter no. of transition: ";
cin >> notr;

for (i=0; i<notr; i++)
```

Expt.No. _____
Date _____
Page No. _____

```
{
cout << "Enter transition: " << i+1 << ": ";
cin >> tr[i].ip_state >> tr[i].ip_symb >> tr[i].
                                    op_state;
}
for (i=0; i<notr; i++)
cout << tr[i].ip_states << " -> " << tr[i].ip_symb <<
                    tr[i].op_state << "\n";
entra = istate;

cout << " enter string: ";
cin >> str;
int k=0
for (i=0; i<notr; i++)
{
if ((entra == tr[i].ip_state) && (str[k] == tr[i].
                                    ip_symb))
{
entra = tr[i].op_state;
k++;
i=-1;
}
}
```

Expt.No.
Date
Page No.

```
if (extra == fstate)
   cout << "String is accepted: ";
else
   cout << "String is not accepted: ";
   return 0;
}
```

Output :- Enter the no. of states : 3
Enter state : 1 : A
Enter state : 2 : B
Enter state : 3 : C
Enter initial state : A
Enter final state : C
Enter no. of i/p symbol : 2
Enter i/p symbol 1 : 2 : 1
Enter i/p symbol 1 : 2 : 0
Enter no. of transition : 3
Enter transition : 1 : A1B
Enter transition : 2 : B1B
Enter transition : 3 : B0C

A → 1B
B → 1B
C → 0C

Expt.No.
Date
Page No.

Enter String : 110

Conclusion :- Thus I have successfully executed a program for checking a given input string is accepted or not.

B

8/B
19/10/2021

Practical No. 5

Aim :- Implement symbol table contain function create(), modify(), search(), display(), Delete().

S/W Required :- C++ Compiler in linux.

Output :-

How many symbol you have to enter :- 3

Symbol	Value
A	1
B	2
C	3

1. Display
2. Modify
3. Search
4. Delete

Enter your choice : 2
Enter the symbol to be modified : A
Enter value of modification : 5

A	5
B	2
C	3

Expt.No.
Date 13/10/22
Page No.

Practical No. 5.

Aim :- Implement symbol table contain function create(), modify(), search(), display(), Delete().

S/W Required :- C++ Compiler in linux.

Program :-

```
# include <iostream.h>
# include <stdlib.h>
using namespace std;
struct symbol {
        char alpha;
        int value;
        };
void display (symbol tab[], int n);
void modify (symbol tab[], int n, char alpha);
void delete1 (symbol tab[], int n, char alpha);
void search (symbol tab[], int n, char alpha);
int main ()
{
  symbol * table ;
     int n, i ;
     char alpha ;
  cout << " How many symbol you have to enter:
```

1. Display
2. Modify
3. Search
4. Delete

Enter your choice : 3
Enter symbol to be searched : B

B 2

1. Display
2. Modify
3. Search
4. Delete

Enter your choice : 4
Enter the symbol to be deleted : C

A 5

B 2

Expt.No.
Date
Page No.

```
cin >> n;
table = new symbol [n];
cout << " Symbol || value \n ";
  for (i=0; i<n ; i++)
{
  cin >> table [i]. alpha >> table [i] value;
  }
while (1)
{
 int choice;
cout << endl;
cout << "1. Display " <<endl;
cout << "2. Modify " << endl;
cout << "3. Search " <<endl;
cout << "4. Delete " << endl;
cout << " \n Enter your choice ";
    cin >> choice;
switch (choice)
 {
 case 1: display (table, n);
break;

case 2:   modify (table n, alpha);
 break;
```

Expt.No.
Date
Page No.

```
case 3: Search (table n , alpha);
break;

case 4: delete (table n, alpha);
break;
default;
exit (0);
}
}
return 0;
void display (Symbol tab [], int n)
{
    int i;
    for (i=0; i <n ; i++)
{
 cout << tab [i]. alpha << " " < tab[i] << endl;
}
}
void modify (Symbol tab [], int n , char, alpha)
   { int temp;
cout << "Enter the symbol to modify";
   cin >> alpha;
cout << "Enter the value after modification";
   cin >> temp;
```

Expt.No.
Date
Page No.

```
{ for (int i=0; i<n; i++)
  if (tab[i].alpha == alpha)
  { tab[i].value = temp;
    cout << endl;
    display (tab, n);
  }}

void search (Symbol tab[], int n, char)
{ int i;
cout << "Enter the symbol to be search";
  cin >> alpha;
  for (i=0; i<n; i++)
  { if (tab[i].alpha == alpha)
    { cout << tab[i] alpha << i+" " <<
        tab[i].value << endl;
    }
  }}

void delete (Symbol tab[], int n, char alpha)
{
  int i;
cout << "Enter the symbol to be Delete";
      cin >> alpha;
```

Expt.No.
Date
Page No.

```
{ for (i=0; i<n; i++)
  { if (tab[i].alpha == alpha)
    {
      tab[i].alpha = tab[i+1].alpha;
      tab[i].value = tab[i+1].value;
    }
  }

    n = n - 1;
    display (tab.n);
  }
```

Conclusion:- Thus, we implement program for symbol table contain function create(), modify(), search(), display(), delete().

B
3/3
18/11/20

Expt.No.
Date 10/11/22
Page No.

Practical No. 6.

Aim:- Write program to implement Recursive descent parser.

S/W Required :- C++ compiler.

Program:-

```
#include <iostream.h>
#include <stdlib.h>
using namespace std;
/*
E  ⟶ TE'
E' ⟶ +TE' / null
T  ⟶ FT'
T' ⟶ * FT' / null
F  ⟶ id / (E)
*/
int count = 03;
void E ();
void Ed ();
void T ();
void Td ();
void F ();
```

Expt.No.
Date
Page No.

String expr:

```
int main () {
cout << "E → TE' " <<endl;
cout << "E' → +TE' " <<endl;
cout << "E' → null " << endl;
cout << "T → FT' " <<endl;
cout << "T' → *FT' " <<endl;
cout << "E → (E) " <<endl;
cout << "T' → null " << endl;
cout << "F → (E) " <<endl;
cout << "F → id " <<endl;

cout << "Enter the expression : ";
cin >> expr;
int = expr.length ();
expr += "$";
E();
if (i == count)
    cout << "Accepted " <<endl;
else
    cout << "Rejected " <<endl;
}
```

Expt.No.
Date
Page No.

```
void E() {
        T();
        Ed();
    }
void Ed() {
    if (expr[count] == '+') {
        count ++;
        T();
        Ed();
    }
else {
    // cout << "E' → null" << endl;
    }
}
void T() {
// cout << "T → FT'" << endl;
    F();
    Td();
}
void Td() {
    if (expr[count] == '*') {
        count ++;
        F();
        Td();
}
```

Output :-

$E \longrightarrow TE'$

$E' \longrightarrow +TE'$

$E' \longrightarrow$ null

$T \longrightarrow FT'$

$T' \longrightarrow *FT$

$T' \longrightarrow$ null

$F \longrightarrow$:id

$F \longrightarrow .(E)$

Conclusion :- Thus we have executed program to implement Recursive descent parser.

Expt.No.
Date
Page No.

```
    else {
      // cout << "T' → null" << endl;
    }
}
void f() {
    if (is alpha (expr [count])) {
        count ++;
    } else if (expr [count] == 'c') {
        count ++;
        exit (0);
    }
    count ++;
    } else {
        cout << "Rejected" << endl;
        exit (0);
    }
}
```

Conclusion :- Thus we have executed a program to implement Recursive Descent parser.

13

3/13

Expt. No.

Date 16/11/22

Page No.

Practical No. 7.

Aim :- Write program to implement shift Reduced parser.

S/W Required :- C++ compiler.

Program :-

```
#include <iostream.h>
int main
{
  clrscr ();
  char str [20];
  char stack [20];
  int top = 0;
  cout << "Given grammar is : \n S →
      a AB \ n A → Abc/b \nB → d" << endl;
  cout << " Enter string " << endl;
  cin >> str ;
  cout << " stack input buffer \t\t
          Action" << endl;
  cout << " - - - - -- - - - - - - - - - ."
                              << endl;
```

Expt No.
Date
Page No.

```
stack [top] = 'e' '$';
 top ++;
cout << "a\t\t bbcde$\t shift " <endl;
stack [top] = str [n];
top++;
cout << " $a\t\t bbcde$\t shift" << endl;
 stack [top] = str [2];
if (stack [top] == 'b')
{
cout << " $ab\t\t bcde$\t\t Reduce "
                                << endl;
 stack [top] = 'A';
 top ++;
cout << " $ aAb\t\t cde$ \t\t shift "
                                << endl;
stack [top] = 'b';
top ++;
cout << " $aAb\t\t cde$\t\t shift " endl;
stack [top] = 'c';
cout << " $aAbc\t\t de $\t\t Reduce" << endl;
}
if (stack [top] == 'c') && (stack [top-1] == 'b'))
{
```

S → aABe
A → Abc/b
B → a.

Enter string

abbcde

Stack	input buffer	action
$	abbcde$	Shift
$a	bbcde$	shift
$ab	bcde$	Reduce
$aA	bcde$	Shift
$aAb	cde$	Shift
$aAbc	de$	Reduce
$aA	de$	Shift
$aAd	e$	Reduce
$aAB	e$	Shift
$aABe	$	Reduce
$S	$	ACCEPTED.

String is valid

Expt.No.
Date
Page No.

```
    top = top - 2;
    cout << "$ a A \t\t de $ \t\t shift" << endl;
    top++;
    stack[top] = 'd';
    cout << "$aAd \t\t e $ \t Reduce" << endl;
}
if (stack[top] == 'd')
{
    stack[top] = 'B';
    cout << "$aAB \t\t e $ \t\t shift" << endl;
    top++;
    stack[top] = 'e';
    cout << "$aBe \t\t $ \t\t Reduce" << endl;
    top = top - 4;
    stack[top] = 'S';
    cout << "$S \t\t $ \t\t ACCEPT" << endl;
}
if (stack[top] == 'S')
{
    cout << "String is valid" << endl;
}
else
{
    cout << "string is not valid" << endl;
```

Expt.No.
Date
Page No.

```
}
    getch ();
    return (0);
}
```

Conclusion :-

In this way we implemented program ~~shift~~ reduce parser.

B

8/8
23/1/22

(Sipna Student Co-operative Consumer Store Ltd, Amravati)

Practical No. 8.

Aim :- Write program to implement DFA that accept all string which follow the language $L=(aa)^*b+$.

S/W Required :- C++ Compiler.

Output :- ACCEPTED

Expt.No.
Date 16/11/22
Page No.

Practical No. 8.

Aim:- Write program to implement DFA that accept all string which follow the language $L = (aa)^* b+$.

S/W Required:- C++ Compiler.

Program:-

```
# include <iostream.h>
# include <string.h>
using namespace std;
int dfa = 0;
void state (char c)
{
 if (c == 'a') {
     dfa = 1; }
 else if (c == 'b') {
     dfa = 3; }
   else {
       dfa = -1;
       }
     }
void state 1 (char c)
 {
```


(Sipna Student Co-operative Consumer Store Ltd, Amravati)

Expt.No.
Date
Page No.

```
if (c == 'a')
{ dfa = 2; }
else if (c == 'b')
{ dfa = 4; }
else { dfa = -1; }
}

void state 2 (char c)
{
if (c == 'b')
        dfa = 3; }
else if (c == 'a' {
        dfa = 1; }
else
    {
    dfa = -1;
    }
void state 3 (char c)
{ if (c == 'b')
    { dfa = 3; }
else if (c == 'a')
    {
    dfa = 4;
    }
```


(Sipna Student Co-operative Consumer Store Ltd, Amravati)

Expt.No.
Date
Page No.

```
else {
    dfa = -1;
}

int is_accepted (char str[])
{
  int i, len = strlen (str);
for (i=0; i<len; i++)
{
 if (dfa == 0)
     start (str[i]);
 else if (dfa == 1)
     state1 = (str[i]);
 else if (dfa == 2)
     state2 ( str[i]);
 else if (dfa == 3)
     state3 (str[i]);
 else if (dfa == 4)
     state4 ( str[i]);
 else
    return 0;
 }
if (dfa == 3)
    return 1;
```


(Sipna Student Co-operative Consumer Store Ltd, Amravati)

Expt.No.
Date
Page No.

```
    else
      return 0;
    }
int  main()
    {
    char str[] = "aaaaaa bbbbb";
      if (isAccepted (str))
          cout << "ACCEPTED";
    else
        cout << "NOT ACCEPTED";
        return 0;
      }
```

Conclusion:- Thus we write program to implement DFA that accept all string which follow language $L = (aa)^* b+$.

B

8/3
23/11/2021

(Sipna Student Co-operative Consumer Store Ltd, Amravati)

Expt.No.
Date 23/11/22
Page No.

Practical No. 9.

Aim :- Study practical on lex .yacc compiler introduction to LEX.

Lex (lexical analyzer generator) is a tool used to generate lexical analyzer. lexical analysis is a process of converting input stream with token. it takes set of regular expression given as input in file & translate it to C. implementation which is stored in lex.yy.c file.

Structure of lex program.

```
/* Declaration */
%%
/* Rule */
%%
/* Auxiliary function */
```

Declaration.

Declaration consist of two part auxiliary declaration & regular definition.

Expt.No.
Date
Page No.

Auxiliary declaration section contain optional global declaration will be copied directly to y.tab.c and are written in C and enclosed within %{ and %}.

Example:-

```
%{#include <stdio.h> /* imports */
  int YY error (char*); /* function prototype */
  int num;
  %}
```

Rule:

Rules contain two parts: Pattern to be matched and corresponding actions.

```
/* Declaration */
   %  %
  {numbers}
  [a-z  A-Z] +
      +
   %  %
   %
```

Expt.No.
Date
Page No.

When a number is sent it will return the number detected in input character. When number is sent it will return the number detected in input characters.

When text is sent, suppose world it will perform an action for printing Hello, world !.

For the rest of input, it will call yy error function.

Auxiliary Function

Lex generate C code for rules & place in function. We can also add our code to lex file.

```
/* Declaration */
    %%
/* Rule */
    %%

int main() { yy lex(); return 0; }.
```

Expt.No.
Date
Page No.

int yywrap { return 1 ; }.

Function and variable starting with yy are predefined function and variables in lex.

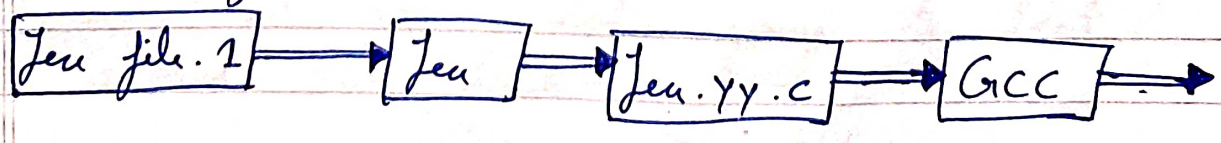
yylval is global variable that is used to pass schematic values associated with token from lexer to parser.

YY text for number is 12.

YYlex() is entry point for lex. it read input stream and generate token. It is defined in lex.YY.c. file but It needs to be manually called by user.

YYwrap() is called by yylex() when the input is exhausted. if 0 is seen yylex() will keep reading from pointer YYin. if 1 is returned then scanning process is terminated it is mandatory to define in lex file.

YY parse() it call YYlex() for lexical analysis, it return 0 is input parse according to rule else 0.



Expt.No.
Date
Page No.

⟹ Executable

Introduction to YCC.

YACC is tool used to generate Parser. A Parser is program that check whether given input meet a grammatical specification. YACC translate a given context free grammer in Y.file. & convert it to C implementation by creating y.tabc & y.tab.h file.

Structure of YACC program.

Similar to LEX files the YACC files contain three section: Declaration, Rules & Auxiliary function.

Declaration:

YACC ~~dot~~ declaration contain two parts auxiliary declaration & YACC declaration Auxiliary declaration is already defined under LEX structure so we are not going to beat around the bush.

Let's just declare it.

Expt.No.
Date
Page No.

```
%{#include <stdio.h> /* import */
  #include <stdlib.h>
  #include <ctype.h>
  int yylex(void); /* function prototype */
  int yyerror(const char *);
  %}
```

YACC declaration consist of token. We can declare token using % token <token.name>

```
% token PLUSMINUS
% token BYE
% left '+' '-'
% left '*' '/'
```

Rule

YACC rule use content-free grammar describe by Brackus Naur from (BNF). We will be only discussing enough as we need feel free to learn more here. Rules consist of two-part production consist of head & Body

for example

expr : ~~expr~~ expr '+' expr.

Expt.No.
Date
Page No.

LHS side of : i.e. expr is production head and right side expr '1' is called production body. Head is always a non-terminal meaning they are replaced by a group of terminal symbol according to production rule. Terminal symbol are token.

Auxiliary function :-

Auxiliary function are similar to LEX auxiliary function. What we define here will be added to y.tab.c. Compulsory definition contain main ().
YYlex () & YY error ().

```
int main () {
while (1) YY parse ()}
    Yacc.d    Yaccfile.}.
```

Yaccfile.1 → YACC → Y.Tab.c → GCC → executable.

Conclusion :-

Thus we have studied LEX & YACC Compiler.

B

8/3/19

Practical No. 10

Aim:- Write for three address code generation.

S/W required:- C++ compiler.

Result:-

1. assignment
2. arithmetic
3. relational
4. exit

Enter choice = 1.

Enter expression with assignment operator : a = b

Three address code : temp = b

a = temp

1. assignment
2. arithmetic
3. relational
4. exit

Enter choice : 4

Expt.No.

Date 23/11/22

Page No.

Practical No. 10

Aim:- Write a for three address code generation.

S/W:- C++ compiler.

Program:-

```
#include <stdio.h>
#include <string.h>
 void pm();
 void plus();
 void ();
int i, ch, j, l, addr = 100;
char ex[10], exp[10], exp1[10], exp2[10],
            id1[5], op[5], id2[5];
int main (){
    clrscr ();
while (1){
printf ("\n1. assignment \n 2. arithmetic
    \n 3 relational \n 4. Exit \n Enter
    choice : ");
    scanf ("%d", &ch);
    switch (ch)
    {
    case 1;
```

Expt.No.
Date
Page No.

```
Printf ("\n Enter expression with
          assignment operator : ");
Scanf ("%s" exp);
l = strlen (exp);
exp2[0] = '10';
  i = 0;
while (exp[i] != '=') { i++; }.

strcat (exp2, exp, i);
    strrev (exp);
  exp1[0] = '10';
  strncat (exp1, exp, l - (i+1));
    strrev (exp1);
printf ("Three address code: \n temp
                = %s \n %s);
    break;

case 2:
printf ("Enter the expression with arithm exp
                operator : ");
scanf ("%s", ex);
strcpy (exp, ex)
  l = strlen (exp);
```

Expt.No.
Date
Page No.

```
exp1[0] = '10';
for (i=0; i<1; i++)
{
if (exp[i] == '+' || exp[i] == '-')
{
  if (exp[i+2] == '/' || exp[i+2] == '+')
  {
    pm();
    break;
  }
else {
    plus(); break;}.
}
else if (exp[i] == '/' || exp[i] == '*')
{
      div();
      break;
    }
  }
  break;

void div cl {
    strncat (exp1, exp, i+2);
```

Expt.No. _____
Date _____
Page No. _____

```
printf ("Three address code : \n temp = %s
    temp1 = temp1. C%. \n", exp1, exp[i+2]);

void plus ()
{
    strcat (exp1, exp, i+2);
printf ("Three address code : \n temp = %s
    \n temp1 = temp%. c %. C \n", exp1, exp
                                (i+2). exp(i+3));
}
```

Conclusion :-

Thus, we write a program for three address code generation.

B

8/3

19/11/22