**Compiler Design Lab - LIST OF EXPERIMENTS (Scheme 2017)**

**1. Implementation of DFA to accept strings using C programming.**

**2. Implementation of DFA using JFLAP tool.**

**3. Implementation of Lexical Analyzer using C programming.**

**4. Elimination of Left Recursion using C programming.**

**5. Finding Left Factoring using C programming.**

**6. Implementation of First and Follow using C programming.**

**7. Implementation of Non-Recursive Predictive Parser using C programming.**

**8. Implementation of Shift Reduce parsing using C programming.**

**9. Implementation of Operator Precedence using C programming.**

**10. Implementation of Type Checking using C programming.**

**11. Implementation Stack Allocation Strategy using C programming.**

**12. Implementation of Intermediate Code Generation using C programming.**

**13. Implementation of Code generation using C programming.**

**EXPERIMENT 1:**

**1a:**

**Aim:**

**C program to accept string which contains even 0’s and even 1’s.**

**Source code:**

#include<stdio.h>

#include<conio.h>

int main()

{

intn,i,count=0,count1=0,a[10];

printf(“Enter size of string:”);

scanf(“%d”,&n);

printf(“Enter string:”);

for(i=0;i<n;i++)

{

scanf(“%d”,&a[i]);

if(a[i]==0)

count++;

else

count1++;

}

if((count%2)==0 && (count1%2)==0)

printf(“String accepted, Because it contains even 0’s and even 1’s”);

else if((count%2)==0 && (count1%2)!=0)

printf(“String is not accepted, because it contains even 0’s and odd 1’s”);

else if((count%2)!=0 && (count1%2)==0)

printf(“String is not accepted, because it contains odd 0’s and even 1’s”);

else

printf(“String is not accepted, because it contains odd 0’s and odd 1’s”);

getch();

return 0;

}

**Sample output:**

Enter size of string: 6

Enter string:

1

0

1

0

1

0

String accepted, Because it contains even 0’s and even 1’s.

**1b:**

**Aim:**

**C program to accept string which contains even 0’s and odd 1’s.**

**Source code:**

#include<stdio.h>

#include<conio.h>

int main()

{

intn,i,count=0,count1=0,a[10];

printf(“Enter size of string:”);

scanf(“%d”,&n);

printf(“Enter string:”);

for(i=0;i<n;i++)

{

scanf(“%d”,&a[i]);

if(a[i]==0)

count++;

else

count1++;

}

if((count%2)==0 && (count1%2)==0)

printf(“String is not accepted, Because it contains even 0’s and even 1’s”);

else if((count%2)==0 && (count1%2)!=0)

printf(“String is accepted, because it contains even 0’s and odd 1’s”);

else if((count%2)!=0 && (count1%2)==0)

printf(“String is not accepted, because it contains odd 0’s and even 1’s”);

else

printf(“String is not accepted, because it contains odd 0’s and odd 1’s”);

getch();

return 0;

}

**Sample output:**

Enter size of string: 6

Enter string:

1

0

1

0

1

0

String is not accepted, Because it contains even 0’s and even 1’s.

**1c:**

**Aim:**

**C program to accept string which contains odd 0’s and even 1’s.**

**Source code:**

#include<stdio.h>

#include<conio.h>

int main()

{

intn,i,count=0,count1=0,a[10];

printf(“Enter size of string:”);

scanf(“%d”,&n);

printf(“Enter string:”);

for(i=0;i<n;i++)

{

scanf(“%d”,&a[i]);

if(a[i]==0)

count++;

else

count1++;

}

if((count%2)==0 && (count1%2)==0)

printf(“String is not accepted, Because it contains even 0’s and even 1’s”);

else if((count%2)==0 && (count1%2)!=0)

printf(“String is not accepted, because it contains even 0’s and odd 1’s”);

else if((count%2)!=0 && (count1%2)==0)

printf(“String is accepted, because it contains odd 0’s and even 1’s”);

else

printf(“String is not accepted, because it contains odd 0’s and odd 1’s”);

getch();

return 0;

}

**Sample output:**

Enter size of string: 6

Enter string:

1

0

1

0

1

0

String is not accepted, Because it contains even 0’s and even 1’s.

**1d:**

**Aim:**

**C program to accept string which contains even 0’s and even 1’s.**

**Source code:**

#include<stdio.h>

#include<conio.h>

int main()

{

intn,i,count=0,count1=0,a[10];

printf(“Enter size of string:”);

scanf(“%d”,&n);

printf(“Enter string:”);

for(i=0;i<n;i++)

{

scanf(“%d”,&a[i]);

if(a[i]==0)

count++;

else

count1++;

}

if((count%2)==0 && (count1%2)==0)

printf(“String is not accepted, Because it contains even 0’s and even 1’s”);

else if((count%2)==0 && (count1%2)!=0)

printf(“String is not accepted, because it contains even 0’s and odd 1’s”);

else if((count%2)!=0 && (count1%2)==0)

printf(“String is not accepted, because it contains odd 0’s and even 1’s”);

else

printf(“String is accepted, because it contains odd 0’s and odd 1’s”);

getch();

return 0;

}

**Sample output:**

Enter size of string: 6

Enter string:

1

0

1

0

1

0

String is not accepted, Because it contains even 0’s and even 1’s.

**EXPERIMENT 3:**

**Aim:**

**Implementation of Lexical Analyzer using C programming.**

**Source Code:**

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

// Returns 'true' if the character is a DELIMITER.

boolisDelimiter(char ch)

{

if (ch == ' ' || ch == '+' || ch == '-' || ch == '\*' ||

ch == '/' || ch == ',' || ch == ';' || ch == '>' ||

ch == '<' || ch == '=' || ch == '(' || ch == ')' ||

ch == '[' || ch == ']' || ch == '{' || ch == '}')

return (true);

return (false);

}

// Returns 'true' if the character is an OPERATOR.

boolisOperator(char ch)

{

if (ch == '+' || ch == '-' || ch == '\*' ||

ch == '/' || ch == '>' || ch == '<' ||

ch == '=')

return (true);

return (false);

}

// Returns 'true' if the string is a VALID IDENTIFIER.

boolvalidIdentifier(char\* str)

{

if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||

str[0] == '3' || str[0] == '4' || str[0] == '5' ||

str[0] == '6' || str[0] == '7' || str[0] == '8' ||

str[0] == '9' || isDelimiter(str[0]) == true)

return (false);

return (true);

}

// Returns 'true' if the string is a KEYWORD.

boolisKeyword(char\* str)

{

if (!strcmp(str, "if") || !strcmp(str, "else") ||

!strcmp(str, "while") || !strcmp(str, "do") ||

!strcmp(str, "break") ||

!strcmp(str, "continue") || !strcmp(str, "int")

|| !strcmp(str, "double") || !strcmp(str, "float")

|| !strcmp(str, "return") || !strcmp(str, "char")

|| !strcmp(str, "case") || !strcmp(str, "char")

|| !strcmp(str, "sizeof") || !strcmp(str, "long")

|| !strcmp(str, "short") || !strcmp(str, "typedef")

|| !strcmp(str, "switch") || !strcmp(str, "unsigned")

|| !strcmp(str, "void") || !strcmp(str, "static")

|| !strcmp(str, "struct") || !strcmp(str, "goto"))

return (true);

return (false);

}

// Returns 'true' if the string is an INTEGER.

boolisInteger(char\* str)

{

int i, len = strlen(str);

if (len == 0)

return (false);

for (i = 0; i <len; i++) {

if (str[i] != '0' &&str[i] != '1' &&str[i] != '2'

&&str[i] != '3' &&str[i] != '4' &&str[i] != '5'

&&str[i] != '6' &&str[i] != '7' &&str[i] != '8'

&&str[i] != '9' || (str[i] == '-' && i > 0))

return (false);

}

return (true);

}

// Returns 'true' if the string is a REAL NUMBER.

boolisRealNumber(char\* str)

{

int i, len = strlen(str);

boolhasDecimal = false;

if (len == 0)

return (false);

for (i = 0; i <len; i++) {

if (str[i] != '0' &&str[i] != '1' &&str[i] != '2'

&&str[i] != '3' &&str[i] != '4' &&str[i] != '5'

&&str[i] != '6' &&str[i] != '7' &&str[i] != '8'

&&str[i] != '9' &&str[i] != '.' ||

(str[i] == '-' && i > 0))

return (false);

if (str[i] == '.')

hasDecimal = true;

}

return (hasDecimal);

}

// Extracts the SUBSTRING.

char\* subString(char\* str, int left, int right)

{

int i;

char\* subStr = (char\*)malloc(

sizeof(char) \* (right - left + 2));

for (i = left; i <= right; i++)

subStr[i - left] = str[i];

subStr[right - left + 1] = '\0';

return (subStr);

}

// Parsing the input STRING.

void parse(char\* str)

{

int left = 0, right = 0;

intlen = strlen(str);

while (right <= len&& left <= right) {

if (isDelimiter(str[right]) == false)

right++;

if (isDelimiter(str[right]) == true && left == right) {

if (isOperator(str[right]) == true)

printf("'%c' IS AN OPERATOR\n", str[right]);

right++;

left = right;

} else if (isDelimiter(str[right]) == true && left != right

|| (right == len&&left != right)) {

char\* subStr = subString(str, left, right - 1);

if (isKeyword(subStr) == true)

printf("'%s' IS A KEYWORD\n", subStr);

else if (isInteger(subStr) == true)

printf("'%s' IS AN INTEGER\n", subStr);

else if (isRealNumber(subStr) == true)

printf("'%s' IS A REAL NUMBER\n", subStr);

else if (validIdentifier(subStr) == true

&&isDelimiter(str[right - 1]) == false)

printf("'%s' IS A VALID IDENTIFIER\n", subStr);

else if (validIdentifier(subStr) == false

&&isDelimiter(str[right - 1]) == false)

printf("'%s' IS NOT A VALID IDENTIFIER\n", subStr);

left = right;

}

}

return;

}

// DRIVER FUNCTION

int main()

{

// maximum legth of string is 100 here

charstr[100] = "int a = b + 1c; ";

parse(str); // calling the parse function

return (0);

}

**Sample Output:**

‘int’ IS A KEYWORD

‘a’ IS A VALID IDENTIFIER

‘=’ IS AN OPERATOR

‘b’ IS A VALID IDENTIFIER  
‘+’ IS AN OPERATOR

**EXPERIMENT 4:**

**Aim:**

**Elimination of left recursion using C programming.**

**Source Code:**

#include<stdio.h>

#include<string.h>

#define SIZE 10

int main()

{

charnt;

charb,a;

intnum;

int i;

char p[10][SIZE];

int index=3;

printf("Enter Number of productions:");

scanf("%d",&num);

printf("Enter the grammar as E->E-A:\n");

for(i=0;i<num;i++)

scanf("%s",p[i]);

for(i=0;i<num;i++)

{

printf("\nGRAMMAR:%s",p[i]);

nt=p[i][0];

if(nt==p[i][index])

{

a=p[i][index];

printf("is left recursive:\n");

while(p[i][index]!=0 && p[i][index]!='|')

{

index++;

}

if(p[i][index]!=0)

{

b=p[i][index+1];

printf("Grammar without left recursion;\n");

printf("%c->%c%c\'\n",nt,b,nt);

printf("\n%c\'->%c%c\'|E\n",nt,a,nt);

}

else

printf("can't be reduced\n");

}

else

printf("is not left recursive\n");

index=3;

}

}

**Sample Output:**

Enter number of productions: 3

Enter the grammar as E->E-A:

E->EA|A

A->a

E->i

Grammar:E->EA|A is left recursive

Grammar without left recursion

E->AE’

E’->AE’|E

Grammar:A->a is not left recursive

Grammar:E->I is not left recursive

**EXPERIMENT 5:**

**Aim:**

**Finding left factoring using C programming.**

**Source Code:**

#include<stdio.h>

#include<string.h>

#define Max 20

int main()

{

char g[20],p1[20],p2[20],mg[20],ng[20],tg[20];

inti,j=0,k=0,l=0,pos;

printf("Enter production:A->");

fgets(g,Max,stdin);

for(i=0;g[i]!='|';i++,j++)

p1[j]=g[i];

p1[j]='\0';

for(j=++i,i=0;g[i]!='\0';j++,i++)

p2[i]=g[i];

p2[i]='\0';

for(i=0;i<strlen(p1)||i<strlen(p2);i++)

{

if(p1[i]==p2[i])

{

mg[k]=p1[i];

k++;

pos=i+1;

}

}

for(i=pos,j=0;p1[i]!='\0';i++,j++)

ng[j]=p1[i];

ng[j++]='1';

for(i=pos;p2[i]!='\0';i++,j++)

ng[j]=p2[i];

mg[k]='X';

mg[++k]='\0';

ng[j]='\0';

printf("\nA->%s",mg);

printf("\nX->%s\n",ng);

}

**Sample output:**

Enter production: A->CE+bcD|CE+cdT

A->CE+X

X->bcD+cdT

**EXPERIMENT 6:**

**Aim:**

**Implementation of First and Follow using C programming.**

**Source Code:**

#include<stdio.h>

#include<math.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

intn,m=0,p,i=0,j=0;

char a[10][10],f[10];

void follow(char c);

void first(char c);

int main()

{

inti,z;

charc,ch;

printf("Enter the no of productions:\n");

scanf("%d",&n);

printf("Enter the productions:\n");

for(i=0;i<n;i++)

scanf("%s%c",a[i],&ch);

do

{

m=0;

printf("Enter the elements whose first and follow is to be found:");

scanf("%c",&c);

first(c);

printf("first(%c)={",c);

for(i=0;i<m;i++)

printf("%c",f[i]);

printf("}\n");

strcpy(f," ");

m=0;

follow(c);

printf("follow(%c)={",c);

for(i=0;i<m;i++)

printf("%c",f[i]);

printf("}\n");

printf("continue(0/1)?");

scanf("%d%c",&z,&ch);

}while(z==1);

return(0);

}

void first(char c)

{

int k;

if(!isupper(c))

f[m++]=c;

for(k=0;k<n;k++)

{

if(a[k][0]==c)

{

if(a[k][2]=='$')

follow(a[k][0]);

else if(islower(a[k][2]))

f[m++]=a[k][2];

else first(a[k][2]);

}

}

}

void follow(char c)

{

if(a[0][0]==c)

f[m++]='$';

for(i=0;i<n;i++)

{

for(j=2;j<strlen(a[i]);j++)

{

if(a[i][j]==c)

{

if(a[i][j+1]!='\0')

first(a[i][j+1]);

if(a[i][j+1]=='\0' && c!=a[i][0])

follow(a[i][0]);

}

}

}

}

**Sample Output:**

Enter the no. of productions:

5

Enter the productions:

S=AbCd

A=Cf

A=a

C=gE

E=h

Enter the elements whose first & follow is to be found: S

First(S)={ga}

Follow(S)={$}

Continue(0/1)?1

Enter the elements whose first & follow is to be found: A

First(A)={ga}

Follow(A)={b}

Continue(0/1)?0

**EXPERIMENT NO:07**

**Aim: Implementation of Non-Recursive Predictive Parser using C programming.**

**Code:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

char s[30],st[30];

void main()

{

Char tab[5][6][5]={"ta","@","@","ta","@","@","@","+ta","@","@","!","!","fb","@","@","fb","@","@","@","!","\*fb","@","!","!","i","@","@","(e)","@","@"};

printf("Enter the string:\n");

scanf("%s",s);

strcat(s,"$");

st[0]='$';

st[1]='e';

int st\_i,s\_i,i,j,k,n,s1,s2;

st\_i=1;

s\_i=0;

char temp[20];

printf("\nStack Input\n");

while(st[st\_i]!='$' || s[s\_i]!='$')

{

switch(st[st\_i])

{

case 'e':s1=0;break;

case 'a':s1=1;break;

case 't':s1=2;break;

case 'b':s1=3;break;

case 'f':s1=4;break;

default:s1=-1;

}

switch(s[s\_i])

{

case 'i':s2=0;break;

case '+':s2=1;break;

case '\*':s2=2;break;

case '(':s2=3;break;

case ')':s2=4;break;

case '$':s2=5;break;

default:s2=-1;

}

if(s1==-1 || s2==-1)

{

printf("Failure");

exit(0);

}

if(tab[s1][s2]=="@")

{

printf("Failure");

exit(0);

}

if(tab[s1][s2][0]=='!')

{

st[st\_i]='\0';

st\_i--;

}

else

{

j=strlen(tab[s1][s2]);

for(k=0;tab[s1][s2][k]!='\0';k++)

{

temp[j-k-1]=tab[s1][s2][k];

}

temp[j]='\0';

st[st\_i]='\0';

strcat(st,temp);

st\_i=strlen(st)-1;

}

printf("%s ",st);

for(n=s\_i;s[n]!='\0';n++)

printf("%c",s[n]);

printf("\n");

if(st[st\_i]==s[s\_i] && s[s\_i]!='$')

{

st[st\_i]='\0';

s\_i++;

st\_i--;

}

}

printf("Success");

}

**OUTPUT:**

Enter string i\*i+i

STACK INPUT

$at i\*i+i$

$abf i\*i+i$

$abi \*i+i$

$abf\* i+i$

$abi i+i$

$a +i$

$at+ +i$

$abf i$

$abi i$

$a $

$ $

Success

**EXPERIMENT 8:**

**Aim:**

**Implementation of ShiftReduce Parsing using C programming.**

**Source Code:**

#include<stdio.h>

#include<string.h>

int k=0,z=0,i=0,j=0,c=0;

char a[16],ac[20],stk[15],act[10];

void check();

int main()

 {

    puts("GRAMMAR is E->E+E \n E->E\*E \n E->(E) \n E->id");

puts("enter input string ");

gets(a);

c=strlen(a);

strcpy(act,"SHIFT->");

puts("stack \t input \t action");

for(k=0,i=0; j<c; k++,i++,j++)

{

 if(a[j]=='i' && a[j+1]=='d')

 {

stk[i]=a[j];

stk[i+1]=a[j+1];

stk[i+2]='\0';

a[j]=' ';

a[j+1]=' ';

printf("\n$%s\t%s$\t%sid",stk,a,act);

check();

 }

 else

{

stk[i]=a[j];

stk[i+1]='\0';

a[j]=' ';

printf("\n$%s\t%s$\t%ssymbols",stk,a,act);

check();

 }

 }

 }

void check()

{

strcpy(ac,"REDUCE TO E");

for(z=0; z<c; z++)

 if(stk[z]=='i' &&stk[z+1]=='d')

{

 stk[z]='E';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

 j++;

}

for(z=0; z<c; z++)

 if(stk[z]=='E' &&stk[z+1]=='+' &&stk[z+2]=='E')

 {

stk[z]='E';

stk[z+1]='\0';

stk[z+2]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

 }

for(z=0; z<c; z++)

if(stk[z]=='E' &&stk[z+1]=='\*' &&stk[z+2]=='E')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

 for(z=0; z<c; z++)

 if(stk[z]=='(' &&stk[z+1]=='E' &&stk[z+2]==')')

{

 stk[z]='E';

 stk[z+1]='\0';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

}

**Sample Output:**

Grammar:E->E+E  
E->E\*E  
E->(E)  
E->id

Enterinputstring:  
id+id\*id+id  
stack              input                  action $              id +id\*id+id$         SHIFT->id  
$E               +id\*id+id$         REDUCE TO E  
$E+               id\*id+id$         SHIFT->symbols  
$E+id           \*    id+id$         SHIFT->id  
$E+E                \*id+id$         REDUCE TO E  
$E                     \*id+id$         REDUCE TO E  
$E\*                     id+id$         SHIFT->symbols  
$E\*id                     +id$         SHIFT->id  
$E\*E                      +id$         REDUCE TO E  
$E                           +id$          REDUCE TO E  
$E+                           id$          SHIFT->symbols  
$E+id                           $          SHIFT->id  
$E+E                            $          REDUCE TO E  
$E                                 $          REDUCE TO E

**EXPERIMENT NO: 09**

**Aim:**

**Implementation of Operator Precedence Using C programming.**

**Code:**

#include<stdio.h>

#include<string.h>

void main()

{

char stack[20],ip[20],opt[10][10][1],ter[10];

int i,j,k,n,top=0,col,row;

for(i=0;i<10;i++)

{

stack[i]=0;

ip[i]=0;

for(j=0;j<10;j++)

{

opt[i][j][1]=0;

}

}

printf("Enter the no. of terminals:");

scanf("%d",&n);

printf("\nEnter the terminals:");

scanf("%s",ter);

printf("\nEnter the table values:");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

printf("\nEnter the value for %c %c:",ter[i],ter[j]);

scanf("%s",opt[i][j]);

}

}

printf("\nOPERATOR PRECEDENCE TABLE:\n");

for(i=0;i<n;i++)

{

printf("\t%c",ter[i]);

}

printf("\n");

for(i=0;i<n;i++)

{

printf("\n%c",ter[i]);

for(j=0;j<n;j++)

printf("\t%c",opt[i][j][0]);

}

stack[top]='$';

printf("\nEnter the input string:");

scanf("%s",ip);

i=0;

printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");

printf("\n%s\t\t\t%s\t\t\t",stack,ip);

while(i<=strlen(ip))

{

for(k=0;k<n;k++)

{

if(stack[top]==ter[k])

col=k;

if(ip[i]==ter[k])

row=k;

}

if((stack[top]=='$')&&(ip[i]=='$'))

{

printf("String is accepted");

break;

}

else if((opt[col][row][0]=='<')||(opt[col][row][0]=='='))

{

stack[++top]=opt[col][row][0];

stack[++top]=ip[i];

printf("Shift %c",ip[i]);

i++;

}

else

{

if(opt[col][row][0]=='>')

{

while(stack[top]!='<')

{

--top;

}

top=top-1;

printf("Reduce");

}

else

{

printf("\nString is not accepted");

break;

}

}

printf("\n");

for(k=0;k<=top;k++)

{

printf("%c",stack[k]);

}

printf("\t\t\t");

for(k=i;k<strlen(ip);k++)

{

printf("%c",ip[k]);

}

printf("\t\t\t");

}

}

**OUTPUT:**

Enter the no. of terminals:4

Enter the terminals:+\*i$

Enter the table values:

Enter the value for + +:>

Enter the value for + \*:<

Enter the value for + i:<

Enter the value for + $:>

Enter the value for \* +:>

Enter the value for \* \*:>

Enter the value for \* i:<

Enter the value for \* $:>

Enter the value for i +:>

Enter the value for i \*:>

Enter the value for i i:=

Enter the value for i $:>

Enter the value for $ +:<

Enter the value for $ \*:<

Enter the value for $ i:<

Enter the value for $ $:A

OPERATOR PRECEDENCE TABLE:

+ \* i $

+ > < < >

\* > > < >

i > > = >

$ < < < A

Enter the input string:i+i\*i$

STACK INPUT STRING ACTION

$ i+i\*i$ Shift i

$<i +i\*i$ Reduce

$ +i\*i$ Shift +

$<+ i\*i$ Shift i

$<+<i \*i$ Reduce

$<+ \*i$ Shift \*

$<+<\* i$ Shift i

$<+<\*<i $ Reduce

$<+<\* $ Reduce

$<+ $ Reduce

$ $ String is accepted

**EXP NO:10**

**Aim: Implementation of Type checking using C programming.**

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int main()

{

intn,i,k,flag=0;

char var[15],typ[15],b[15],c,s[2];

printf("Enter the number of variables:");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("Enter the variables [%d]:\n",i);

scanf(" %c",&var[i]);

printf("Enter the variable-type [%d] (float-f,int-t):",i);

//scanf("%c %c",&var[i],&typ[i]);

scanf(" %c",&typ[i]);

/\*scanf("%s",s);

strcpy(var[i],s[0]);

strcpy(var[i],s[1]);\*/

if(typ[i]=='f')

flag=1;

}

printf("Enter the expression (end with $):");

i=0;

getchar();

while((c=getchar())!='$')

{

b[i]=c;

i++;

}

k=i;

for(i=0;i<k;i++)

{

if(b[i]=='/')

{

flag=1;

break;

}

}

for(i=0;i<n;i++)

{

if(b[0]==var[i])

{

if(flag==1)

{

if(typ[i]=='f')

{

printf("\nThedatatype is correctly defined..\n");

break;

}

else

{

printf("Identifier %c must be a float type..\n",var[i]);

break;

}

}

else

{

printf("\nThe data is correctly defined");

break;

}

}

}

return 0;

}

**OUTPUT:**

Enter the number of variables: 4

Enter the variable[0]:A

Enter the variable-type[0](float-f,int-i):i

Enter the variable[1]:B

Enter the variable-type[1](float-f,int-i):i

Enter the variable[2]:C

Enter the variable-type[2](float-f,int-i):f

Enter the variable[3]:D

Enter the variable-type[3](float-f,int-i):i

Enter the expression(end with $):

A=B\*C/D$

Identifier A must be a float type..!

**EXPT NO:11**

**Aim: Implementation of Stack Allocation Strategy using C programming.**

**Code:**

#include<stdio.h>

int stack[100],ch,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

top=-1;

printf("Enter the size of stack[MAX=100]");

scanf("%d",&n);

printf("\n\t stack opertaions:");

printf("\n\t--------------------------:");

printf("\n\t 1.push\t2.pop\t 3.display\t4.EXIT\t");

do

{

printf("\n Enter the choice");

scanf("%d",&ch);

switch(ch)

{

case 1:push();break;

case 2:pop();break;

case 3:display();break;

case 4:printf("\n\tExit");break;

default:printf("Please nter a valid choice:");

}

}while(ch!=4);

return 0;

}

void push()

{

if(top>=n-1)

{

printf("\n\n stack overflow");

}

else

{

printf("Enter a value to be pushed");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{

if(top==-1)

printf("\n\t stack underflow");

else

{

printf("\n\t the popped element is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in stack\n");

for(i=top;i>=0;i--)

printf("\n%d",stack[i]);

printf("\n Select next choice");

}

else

{

printf("\n the stack is empty");

}

}

**OUTPUT:**

Enter the size of STACK[MAX=100]:10

STACK OPERATIONS USING ARRAY

1.PUSH

2.POP

3.DISPLAY

4.EXIT

Enter the choice:1

Enter a value to be pushed:12

Enter the choice:1

Enter a value to be pushed:24

Enter the choice:1

Enter a value to be pushed:98

Enter the choice:3

The elements in STACK

98

24

12

Press Next Choice

Enter the choice:2

The popped elements is 98

Enter the choice:3

The elements in STACK

24

12

Press Next Choice

Enter the choice:4

EXIT POINT

**EXP NO: 12**

**Aim: Implementation of Intermediate Code Generation using C programming.**

**Code:**

#include<stdio.h>

#include<string.h>

int i=1,j=0,no=0,tmpch=90;

charstr[100],left[15],right[15];

voidfindopr();

void explore();

voidfleft(int);

void fright(int);

structexp

{

intpos;

char op;

}k[15];

void main()

{

printf("\t\tIntermediate code generation\n:");

printf("Enter the expression:");

scanf("%s",str);

printf("The Intermediate code:\t\t Expression\n");

findopr();

explore();

}

voidfindopr()

{

for(i=0;str[i]!='\0';i++)

if(str[i]==':')

{

k[j].pos=i;

k[j++].op=':';

}

for(i=0;str[i]!='\0';i++)

if(str[i]=='/')

{

k[j].pos=i;

k[j++].op='/';

}

for(i=0;str[i]!='\0';i++)

if(str[i]=='\*')

{

k[j].pos=i;

k[j++].op='\*';

}

for(i=0;str[i]!='\0';i++)

if(str[i]=='-')

{

k[j].pos=i;

k[j++].op='-';

}

}

void explore()

{

i=1;

while(k[i].op!='\0')

{

fleft(k[i].pos);

fright(k[i].pos);

str[k[i].pos]=tmpch--;

printf("\t %c:=%s%c%s\t\t",str[k[i].pos],left,k[i].op,right);

for(j=0;j<strlen(str);j++)

if(str[j]!='$')

printf("%c",str[j]);

printf("\n");

i++;

}

fright(-1);

if(no==0)

{

fleft(strlen(str));

printf("\t %s:=%s",right,left);

}

printf("\t %s:=%c",right,str[k[--i].pos]);

}

voidfleft(int x)

{

int w=0,flag=0;

x--;

while(x!=-1 &&str[x]!='+' &&str[x]!='\*' &&str[x]!='=' &&str[x]!='\0' &&str[x]!='-' &&str[x]!='/' &&str[x]!=':')

{

if(str[x]!='$' && flag==0)

{

left[w++]=str[x];

left[w]='\0';

str[x]='$';

flag=1;

}

x--;

}

}

void fright(int x)

{

int w=0,flag=0;

x++;

while(x!=-1 &&str[x]!='+' &&str[x]!='\*' &&str[x]!='=' &&str[x]!='\0' &&str[x]!='-' &&str[x]!='/' &&str[x]!=':')

{

if(str[x]!='$'&& flag==0)

{

right[w++]=str[x];

right[w]='\0';

str[x]='$';

flag=1;

}

x++;

}

}

**OUTPUT:**

INTERMEDIATE CODE GENERATION

Enter the expression: a\*b+c/d-e/f+g\*h

The intermediate code: Expression

Z:=c/d w:=a\*b+Z-e/f+g\*h

Y:=e/f w:=a\*b+Z-Y+g\*h

X:=a\*b w:=X+Z-Y+g\*h

W:=g\*h w:=X+Z-Y+W

U:=X+Z w:=U-Y+W

V:=Y+W w:=U-V

T:=U-V w:=T

W:=T

**EXP NO: 13**

**Aim: Implementation of Code Generation using C programming.**

**Code:**

#include<stdio.h>

#include<string.h>

struct op

{

char l;

char r[20];

}

op[10],pr[10];

void main()

{

inta,i,k,j,n,z=0,m,q;

char \*p,\*l;

chartemp,t;

char \*tem;

printf("Enter the number of values:");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("left:");

scanf("%s",&op[i].l);

printf("Right:");

scanf("%s",&op[i].r);

}

printf("Intermediate code \n");

for(i=0;i<n;i++)

{

printf("%c=",op[i].l);

printf("%s\n",op[i].r);

}

for(i=0;i<n-1;i++)

{

temp=op[i].l;

for(j=0;j<n;j++)

{

p=strchr(op[j].r,temp);

if(p)

{

pr[z].l=op[i].l;

strcpy(pr[z].r,op[i].r);

z++;

}

}

}

pr[z].l=op[n-1].l;

strcpy(pr[z].r,op[n-1].r);

z++;

printf("\n After dead code elimination\n");

for(k=0;k<z;k++)

{

printf("%c\t=",pr[k].l);

printf("%s\n",pr[k].r);

}

for(m=0;m<z;m++)

{

tem=pr[m].r;

for(j=m+1;j<z;j++)

{

p=strstr(tem,pr[j].r);

if(p)

{

t=pr[j].l;

pr[j].l=pr[m].l;

for(i=0;i<z;i++)

{

l=strchr(pr[i].r,t);

if(l)

{

a=l-pr[i].r;

printf("pos:%d\n",a);

pr[i].r[a]=pr[m].l;

}

}

}

}

}

printf("Eliminate common expression\n");

for(i=0;i<z;i++)

{

printf("%c\t=",pr[i].l);

printf("%s\n",pr[i].r);

}

for(i=0;i<z;i++)

{

for(j=i+1;j<z;j++)

{

q=strcmp(pr[i].r,pr[j].r);

if((pr[i].l==pr[j].l)&&!q)

{

pr[i].l='\0';

}

}

}

printf("Optimized code\n");

for(i=0;i<z;i++)

{

if(pr[i].l!='\0')

{

printf("%c=",pr[i].l);

printf("%s\n",pr[i].r);

}

}

}

**OUTPUT:**

Enter the number of values:5

left:a

right:9

left:b

right:c+d

left:e

right:c+d

left:f

right:b+e

left:r

right:f

Intermediate Code

a=9

b=c+d

e=c+d

f=b+e

r=f

After Dead Code Elimination

b=c+d

e=c+d

f=b+e

r=f

pos:2

Eliminate common expression

b=c+d

b=c+d

f=b+b

r=f

Optimized Code

b=c+d

f=b+b

r=f