**MANAV RACHNA INTERNATIONAL INSTITUTE OF RESEARCH AND STUDIES**

****

**COMPILER DESIGN LAB FILE**

**SUBMITTED TO: SUBMITTED BY:**

**Ms. Poonam Katyal Sapna Sinha**

**Assistant Professor 1/18/FET/BCS/079**

**Department Of CSE**

**FET MRIIRS**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S No.** | **Practical** | **Date** |
| **1** | Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Simulate the same in C language. |  |
| **2** | Write a C program to identify whether a given line is a comment or not. |  |
| **3** | Write a C program to recognize strings under 'a', 'a\*b+',' abb'. |  |
| **4** | Write a C program to test whether a given identifier is valid or not. |  |
| **5** | Write a C program to simulate lexical analyzer for validating operators. |  |
| **6** | Write a C program for construction of LL(1) parsing. |  |
| **7** | Write a C program to implement LALR parsing. |  |
| **8** | Write a C program to implement Program semantic rules to calculate the expression that takes an expression with digits, +and \* and computes the value. |  |
| **9** | Implementation of symbol table. |  |
| **10** | Implement any one storage allocation strategies (Heap,Stack,Static). |  |
| **11** | Implementation of simple code optimization techniques (constant folding. etc.) |  |

**PROGRAM – 01**

**AIM:** Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Simulate the same in C language.

**LOGIC:**  
1. Read the input Expression  
2. Check whether input is alphabet or digits then store it as identifier  
3. If the input is “is” operator store it as symbol  
4. Check the input for keywords

**SOURCE CODE:**

#include<string.h>

#include<ctype.h>

#include<stdio.h>

void keyword(char str[10])

{

if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0|| strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||strcmp("double",str)==0||

strcmp("static",str)==0||strcmp("switch",str)==0||strcmp("case",str)==0)

printf("\n%s is a keyword",str);

else

printf("\n%s is an identifier",str);

}

main()

{

FILE \*f1,\*f2,\*f3;

char c,str[10],st1[10];

int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;

printf("\nEnter the c program");/\*gets(st1);\*/

f1=fopen("input","w");

while((c=getchar())!=EOF)

putc(c,f1);

fclose(f1);

f1=fopen("input","r");

f2=fopen("identifier","w");

f3=fopen("specialchar","w");

while((c=getc(f1))!=EOF){

if(isdigit(c))

{

tokenvalue=c-'0';

c=getc(f1);

while(isdigit(c)){

tokenvalue\*=10+c-'0';

c=getc(f1);

}

num[i++]=tokenvalue;

ungetc(c,f1);

}

else if(isalpha(c))

{

putc(c,f2);

c=getc(f1);

while(isdigit(c)||isalpha(c)||c=='\_'||c=='$')

{

putc(c,f2);

c=getc(f1);

}

putc(' ',f2);

ungetc(c,f1);

}

else if(c==' '||c=='\t')

printf(" ");

else

if(c=='\n')

lineno++;

else

putc(c,f3);

}

fclose(f2);

fclose(f3);

fclose(f1);

printf("\nThe no's in the program are");

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

printf("%d",num[j]);

printf("\n");

f2=fopen("identifier","r");

k=0;

printf("The keywords and identifiersare:");

while((c=getc(f2))!=EOF){

if(c!=' ')

str[k++]=c;

else

{

str[k]='\0';

keyword(str);

k=0;

}

}

fclose(f2);

f3=fopen("specialchar","r");

printf("\nSpecial characters are");

while((c=getc(f3))!=EOF)

printf("%c",c);

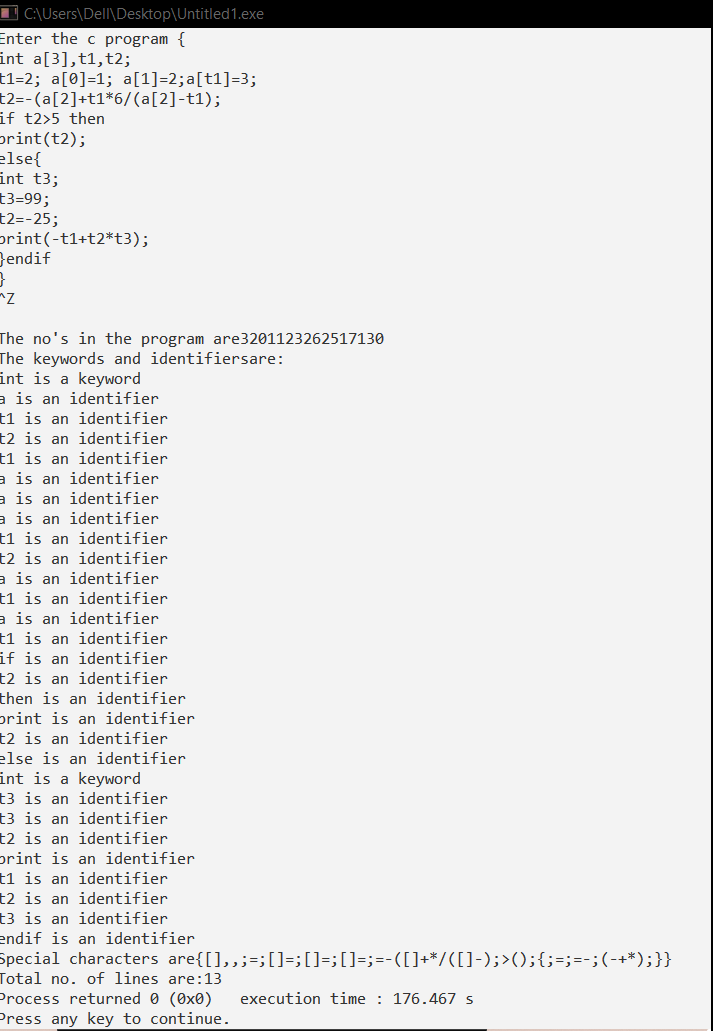
printf("\n");

fclose(f3);

printf("Total no. of lines are:%d",lineno);

}

**OUTPUT:**

****

**PROGRAM – 02**

**AIM:** Write a C program to identify whether a given line is a comment or not.

**LOGIC:**  
Read the input string. Check whether the string is starting with „/‟ and check next character is „/‟ or‟\*‟. If condition satisfies print comment. Else not a comment.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

void main() {

char com[30];

int i=2,a=0;

printf("\n Enter comment:");

gets(com);

if(com[0]=='/') {

if(com[1]=='/')

printf("\n It is a comment");

else if(com[1]=='\*') {

for(i=2;i<=30;i++){

if(com[i]=='\*'&&com[i+1]=='/'){

printf("\n It is a comment");

a=1; break; }

else

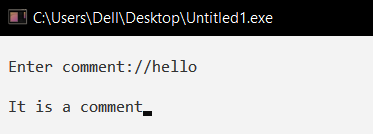
continue; }

if(a==0) printf("\n It is not a comment");}

else printf("\n It is not a comment");}

else printf("\n It is not a comment");

getch();}

**OUTPUT:  **

**PROGRAM – 03**

**AIM:** Write a C program to recognize strings under 'a', 'a\*b+',' abb'.

**LOGIC:**  
By using transition diagram we verify input of the state.  
If the state recognize the given pattern rule.  
Then print string is accepted under a\*/ a\*b+/ abb.  
Else print string not accepted.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

void main(){

char s[20],c;

int state=0,i=0;

clrscr();

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

gets(s);

while(s[i]!='\0'){

switch(state){

case 0: c=s[i++];

if(c=='a')

state=1;

else if(c=='b')

state=2;

else

state=6;

break;

case 1: c=s[i++];

if(c=='a')

state=3;

else if(c=='b')

state=4;

else

state=6;

break;

case 2: c=s[i++];

if(c=='a')

state=6;

else if(c=='b')

state=2;

else

state=6;

break;

case 3: c=s[i++];

if(c=='a')

state=3;

else if(c=='b')

state=2;

else

state=6;

break;

case 4: c=s[i++];

if(c=='a')

state=6;

else if(c=='b')

state=5;

else

state=6;

break;

case 5: c=s[i++];

if(c=='a')

state=6;

else if(c=='b')

state=2;

else

state=6;

break;

case 6: printf("\n %s is not recognised.",s);

exit(0);}}

if(state==1)

printf("\n %s is accepted under rule 'a'",s);

else if((state==2)||(state==4))

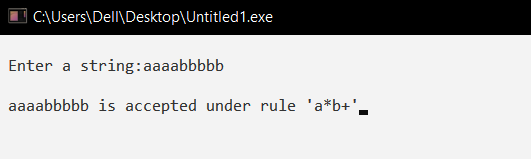
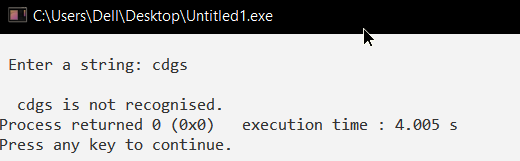
printf("\n %s is accepted under rule 'a\*b+'",s);

else if(state==5)

printf("\n %s is accepted under rule 'abb'",s);

getch();}

**OUTPUT:**

** **

**PROGRAM – 04**

**AIM:** Write a C program to test whether a given identifier is valid or not.

**LOGIC:**

Read the given input string.

Check the initial character of the string is numerical or any special character except ‘\_’ then print it is not a valid identifier.

Otherwise print it as valid identifier if remaining characters of string doesn’t contains any special characters except ‘\_’.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

void main(){

char a[10];

int flag, i=1;

clrscr();

printf("\n Enter an identifier:");

gets(a);

if(isalpha(a[0]))

flag=1;

else

printf("\n Not a valid identifier");

while(a[i]!='\0'){

if(!isdigit(a[i])&&!isalpha(a[i])){

flag=0;

break;}

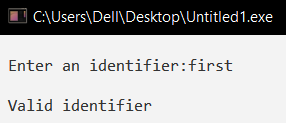
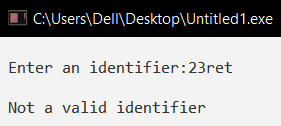
i++;}

if(flag==1)

printf("\n Valid identifier");

getch();}

**OUTPUT:**

** **

**PROGRAM – 05**

**AIM:** Write a C program to simulate lexical analyzer for validating operators.

**LOGIC :**  
Read the given input.  
If the given input matches with any operator symbol.  
Then display in terms of words of the particular symbol.  
Else print not a operator.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

void main()

{

char s[5];

clrscr();

printf("\n Enter any operator:");

gets(s);

switch(s[0])

{

case'>': if(s[1]=='=')

printf("\n Greater than or equal");

else

printf("\n Greater than");

break;

case'<': if(s[1]=='=')

printf("\n Less than or equal");

else

printf("\nLess than");

break;

case'=': if(s[1]=='=')

printf("\nEqual to");

else

printf("\nAssignment");

break;

case'!': if(s[1]=='=')

printf("\nNot Equal");

else

printf("\n Bit Not");

break;

case'&': if(s[1]=='&')

printf("\nLogical AND");

else

printf("\n Bitwise AND");

break;

case'|': if(s[1]=='|')

printf("\nLogical OR");

else

printf("\nBitwise OR");

break;

case'+': printf("\n Addition");

break;

case'-': printf("\nSubstraction");

break;

case'\*': printf("\nMultiplication");

break;

case'/': printf("\nDivision");

break;

case'%': printf("Modulus");

break;

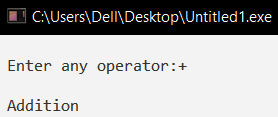
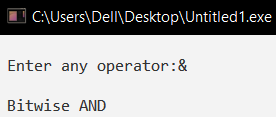
default: printf("\n Not a operator");

}

getch();

}

**OUTPUT:**

** **

**PROGRAM – 06**

**AIM:** Write a C program for construction of LL(1) parsing.

**LOGIC:**

Read the input string.

Using predictive parsing table parse the given input using stack .

If stack [i] matches with token input string pop the token else shift  it repeat the process  until it reaches to $.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

char s[20],stack[20];

void main()

{

char m[5][6][3]={"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","fc"," "," "," ","n","\*fc"," a ","n","n","i"," "," ","(e)"," "," "};

int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};

int i,j,k,n,str1,str2;

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

scanf("%s",s);

strcat(s,"$");

n=strlen(s);

stack[0]='$';

stack[1]='e';

i=1;

j=0;

printf("\nStack Input\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

while((stack[i]!='$')&&(s[j]!='$')){

if(stack[i]==s[j]){

i--;

j++;}

switch(stack[i]){

case 'e': str1=0; break;

case 'b': str1=1; break;

case 't': str1=2; break;

case 'c': str1=3; break;

case 'f': str1=4; break;}

switch(s[j]){

case 'i': str2=0; break;

case '+': str2=1; break;

case '\*': str2=2; break;

case '(': str2=3; break;

case ')': str2=4; break;

case '$': str2=5; break;}

if(m[str1][str2][0]=='\0'){

printf("\nERROR");

exit(0);}

else if(m[str1][str2][0]=='n')

i--;

else if(m[str1][str2][0]=='i')

stack[i]='i';

else{

for(k=size[str1][str2]-1;k>=0;k--){

stack[i]=m[str1][str2][k];

i++;

}

i--;}

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

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

printf(" ");

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

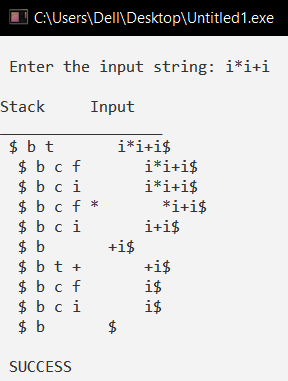
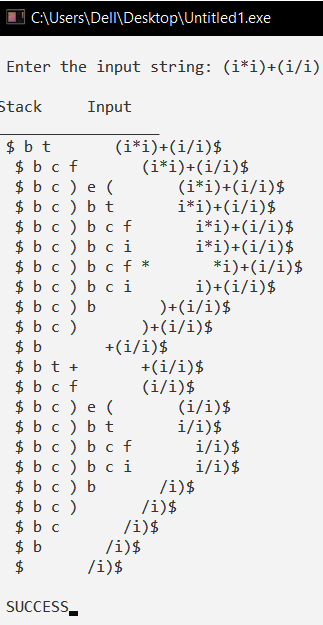
printf("%c",s[k]);

printf(" \n ");}

printf("\n SUCCESS");

getch(); }

**OUTPUT:**

** **

**PROGRAM – 07**

**AIM:** Write a C program to implement LALR parsing.

**LOGIC:**

Read the input string.

Push the input symbol with its state symbols in to the stack by referring lookaheads

We perform shift and reduce actions to parse the grammar.

Parsing is completed when we reach $ symbol.

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

void push(char \*,int \*,char);

char stacktop(char \*);

void isproduct(char,char);

int ister(char);

int isnter(char);

int isstate(char);

void error();

void isreduce(char,char);

char pop(char \*,int \*);

void printt(char \*,int \*,char [],int);

void rep(char [],int);

struct action{

char row[6][5];

};

const struct action A[12]={

{"sf","emp","emp","se","emp","emp"},

{"emp","sg","emp","emp","emp","acc"},

{"emp","rc","sh","emp","rc","rc"},

{"emp","re","re","emp","re","re"},

{"sf","emp","emp","se","emp","emp"},

{"emp","rg","rg","emp","rg","rg"},

{"sf","emp","emp","se","emp","emp"},

{"sf","emp","emp","se","emp","emp"},

{"emp","sg","emp","emp","sl","emp"},

{"emp","rb","sh","emp","rb","rb"},

{"emp","rb","rd","emp","rd","rd"},

{"emp","rf","rf","emp","rf","rf"}

};

struct gotol{

char r[3][4];

};

const struct gotol G[12]={

{"b","c","d"},

{"emp","emp","emp"},

{"emp","emp","emp"},

{"emp","emp","emp"},

{"i","c","d"},

{"emp","emp","emp"},

{"emp","j","d"},

{"emp","emp","k"},

{"emp","emp","emp"},

{"emp","emp","emp"},

};

char ter[6]={'i','+','\*',')','(','$'};

char nter[3]={'E','T','F'};

char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};

char stack[100];

int top=-1;

char temp[10];

struct grammar{

char left;

char right[5];};

const struct grammar rl[6]={

{'E',"e+T"},

{'E',"T"},

{'T',"T\*F"},

{'T',"F"},

{'F',"(E)"},

{'F',"i"},};

void main(){

char inp[80],x,p,dl[80],y,bl='a';

int i=0,j,k,l,n,m,c,len;

printf(" Enter the input :");

scanf("%s",inp);

len=strlen(inp);

inp[len]='$';

inp[len+1]='\0';

push(stack,&top,bl);

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

printt(stack,&top,inp,i);

do{

x=inp[i];

p=stacktop(stack);

isproduct(x,p);

if(strcmp(temp,"emp")==0)

error();

if(strcmp(temp,"acc")==0)

break;

else{

if(temp[0]=='s')

{

push(stack,&top,inp[i]);

push(stack,&top,temp[1]);

i++;

}

else

{

if(temp[0]=='r')

{

j=isstate(temp[1]);

strcpy(temp,rl[j-2].right);

dl[0]=rl[j-2].left;

dl[1]='\0';

n=strlen(temp);

for(k=0;k<2\*n;k++)

pop(stack,&top);

for(m=0;dl[m]!='\0';m++)

push(stack,&top,dl[m]);

l=top;

y=stack[l-1];

isreduce(y,dl[0]);

for(m=0;temp[m]!='\0';m++)

push(stack,&top,temp[m]);

}

}

}

printt(stack,&top,inp,i);

}while(inp[i]!='\0');

if(strcmp(temp,"acc")==0)

printf(" \n accept the input ");

else

printf(" \n do not accept the input ");

getch();}

void push(char \*s,int \*sp,char item){

if(\*sp==100)

printf(" stack is full ");

else{

\*sp=\*sp+1;

s[\*sp]=item;

}}

char stacktop(char \*s){

char i;

i=s[top];

return i;}

void isproduct(char x,char p){

int k,l;

k=ister(x);

l=isstate(p);

strcpy(temp,A[l-1].row[k-1]);}

int ister(char x){

int i;

for(i=0;i<6;i++)

if(x==ter[i])

return i+1;

return 0;}

int isnter(char x){

int i;

for(i=0;i<3;i++)

if(x==nter[i])

return i+1;

return 0;}

int isstate(char p){

int i;

for(i=0;i<12;i++)

if(p==states[i])

return i+1;

return 0;}

void error(){

printf(" error in the input ");

exit(0);}

void isreduce(char x,char p){

int k,l;

k=isstate(x);

l=isnter(p);

strcpy(temp,G[k-1].r[l-1]);}

char pop(char \*s,int \*sp){

char item;

if(\*sp==-1)

printf(" stack is empty ");

else{

item=s[\*sp];

\*sp=\*sp-1;}

return item;}

void printt(char \*t,int \*p,char inp[],int i){

int r;

printf("\n");

for(r=0;r<=\*p;r++)

rep(t,r);

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

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

printf("%c",inp[r]);}

void rep(char t[],int r){

char c;

c=t[r];

switch(c){

case 'a': printf("0");

break;

case 'b': printf("1");

break;

case 'c': printf("2");

break;

case 'd': printf("3");

break;

case 'e': printf("4");

break;

case 'f': printf("5");

break;

case 'g': printf("6");

break;

case 'h': printf("7");

break;

case 'm': printf("8");

break;

case 'j': printf("9");

break;

case 'k': printf("10");

break;

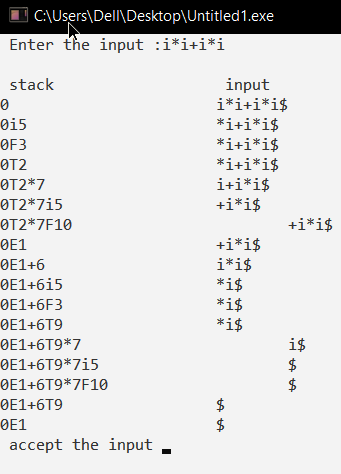
case 'l': printf("11");

break;

default :printf("%c",t[r]);

break;}}

**OUTPUT:**

****

**PROGRAM – 08**

**AIM:** Write a C program to implement Program semantic rules to calculate the expression that takes an expression with digits, +and \* and computes the value.

**SOURCE CODE:**

**PROGRAM – 09**

**AIM:** Implementation of symbol table.

**SOURCE CODE:**

# include <stdio.h>

# include <conio.h>

# include <string.h>

# define null 0

int size=0;

void insert();

void del();

int search(char lab[]);

void modify();

void display();

struct symbtab{

char label[10];

int addr;

struct symtab \*next;

};

struct symbtab \*first,\*last;

void main(){

int op;

int y;

char la[10];

do{

printf("\nSYMBOL TABLE IMPLEMENTATION\n");

printf("1. INSERT\n");

printf("2. DISPLAY\n");

printf("3. DELETE\n");

printf("4. SEARCH\n");

printf("5. MODIFY\n");

printf("6. END\n");

printf("Enter your option : ");

scanf("%d",&op);

switch(op){

case 1:

insert();

display();

break;

case 2:

display();

break;

case 3:

del();

display();

break;

case 4:

printf("Enter the label to be searched : ");

scanf("%s",la);

y=search(la);

if(y==1)

{

printf("The label is already in the symbol Table");

}

else

{

printf("The label is not found in the symbol table");

}

break;

case 5:

modify();

display();

break;

case 6:

break;}}

while(op<6);

getch();

}

void insert(){

int n;

char l[10];

printf("Enter the label : ");

scanf("%s",l);

n=search(l);

if(n==1){

printf("The label already exists. Duplicate cant be inserted\n");

}

else{

struct symbtab \*p;

p=malloc(sizeof(struct symbtab));

strcpy(p->label,l);

printf("Enter the address : ");

scanf("%d",&p->addr);

p->next=null;

if(size==0){

first=p;

last=p;

}

else{

last->next=p;

last=p;

}

size++;}}

void display(){

int i;

struct symbtab \*p;

p=first;

printf("LABEL\tADDRESS\n");

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

printf("%s\t%d\n",p->label,p->addr);

p=p->next;}}

int search(char lab[]){

int i,flag=0;

struct symbtab \*p;

p=first;

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

if(strcmp(p->label,lab)==0){

flag=1;}

p=p->next;}

return flag;}

void modify(){

char l[10],nl[10];

int add, choice, i, s;

struct symbtab \*p;

p=first;

printf("What do you want to modify?\n");

printf("1. Only the label\n");

printf("2. Only the address of a particular label\n");

printf("3. Both the label and address\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice){

case 1:

printf("Enter the old label\n");

scanf("%s",l);

printf("Enter the new label\n");

scanf("%s",nl);

s=search(l);

if(s==0){

printf("NO such label");}

else{

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

if(strcmp(p->label,l)==0){

strcpy(p->label,nl);}

p=p->next;}}

break;

case 2:

printf("Enter the label whose address is to modified\n");

scanf("%s",l);

printf("Enter the new address\n");

scanf("%d",&add);

s=search(l);

if(s==0){

printf("NO such label");}

else{

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

if(strcmp(p->label,l)==0){

p->addr=add;}

p=p->next;}}

break;

case 3:

printf("Enter the old label : ");

scanf("%s",l);

printf("Enter the new label : ");

scanf("%s",nl);

printf("Enter the new address : ");

scanf("%d",&add);

s=search(l);

if(s==0){

printf("NO such label");}

else{

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

if(strcmp(p->label,l)==0){

strcpy(p->label,nl);

p->addr=add;}

p=p->next;}}

break;}}

void del(){

int a;

char l[10];

struct symbtab \*p,\*q;

p=first;

printf("Enter the label to be deleted\n");

scanf("%s",l);

a=search(l);

if(a==0){

printf("Label not found\n");}

else{

if(strcmp(first->label,l)==0){

first=first->next;}

else if(strcmp(last->label,l)==0){

q=p->next;

while(strcmp(q->label,l)!=0){

p=p->next;

q=q->next;}

p->next=null;

last=p;}

else{

q=p->next;

while(strcmp(q->label,l)!=0){

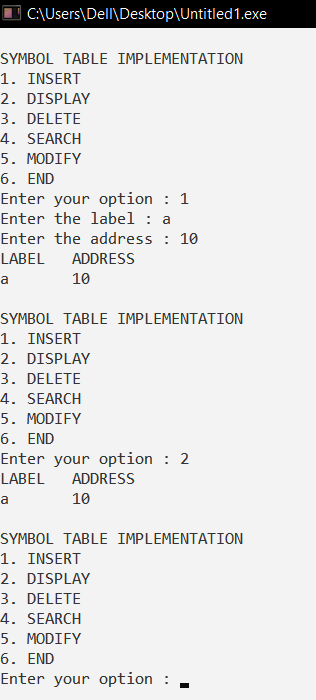
p=p->next;

q=q->next;}

p->next=q->next;}

size--;}}

**OUTPUT:**



**PROGRAM – 10**

**AIM:** Implement any one storage allocation strategies (Heap, Stack, Static).

**SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

typedef struct Heap

{ int data;

struct Heap \*next;

}node;

node \*create();

void main()

{ /\*local declarations\*/

int choice,val;

char ans;

node \*head;

void display(node \*);

node \*search(node \*,int);

node \*insert(node \*);

void dele(node \*\*);

head=NULL;

do

{

printf("\n Program to perform various operations on heapusing dynamic memory management");

printf("\n1.Create");

printf("\n2.Display");

printf("\n3.Insert an element in a list");

printf("\n4.Delete an element from list");

printf("\n5.Quit");

printf("\n Enter Your Choice(1-5)");

scanf("%d",&choice);

switch(choice)

{ case 1:head=create();

break;

case 2:display(head);

break;

case 3:head=insert(head);

break;

case 4:dele(&head);

break;

case 5:exit(0);

default:

printf("Invalid Choice,Try again");

getch();

}}

while(choice!=5);

}

node \*create()

{

node \*temp,\*new1,\*head;

int val,flag;

char ans='y';

node \*get\_node();

temp=NULL;

flag=TRUE;

/\*flag to indicate whether a new node is created for the first time or not\*/

do

{

printf("\n Enter the Element");

scanf("%d",&val);

/\*allocate new node\*/

new1=get\_node();

if(new1==NULL)

printf("\n Memory is not allocated");

new1-> data=val;

if (flag==TRUE)/\* Executed only for the first time\*/

{

head=new1;

temp=head; /\*head is the first node in the heap\*/

flag=FALSE;

} else

{ /\*temp keeps track of the most recently created node\*/

temp->next=new1;

temp=new1;

}

printf("\nDo you want to enter more elements?(y/n)");

ans=getch();

}while(ans=='y');

printf("\nThe list is created");

getch();

return head;

}

node \*get\_node()

{

node \*temp;

temp=(node\*)malloc(sizeof(node));

//using the mem. Allocation function

temp->next=NULL;

return temp;

}

void display(node\*head)

{

node \*temp;

temp=head;

if(temp==NULL)

{

printf("\n The list is empty\n");

getch();

return;

}

while(temp!= NULL)

{

printf("%d->",temp-> data);

temp=temp->next;

}

printf("NULL");

getch();

}

node \*search(node \*head,int key)

{

node \*temp;

int found;

temp=head;

if (temp==NULL)

{

printf("The linked list is empty\n");

getch();

return NULL;

}

found=FALSE;

while((temp!=NULL)&&(found==FALSE))

{ if(temp->data != key)

temp = temp->next;

else

found = TRUE;

} if(found == TRUE)

{

printf("\n The Elements is present in the list\n");

getch();

return temp;

} else

printf("\n The Element is not present in the list\n");

getch();

return NULL;

}

node \*insert(node \*head)

{ int choice;

node \*insert\_head(node\*);

void insert\_after(node\*);

void insert\_last(node\*);

printf("\nInsert a node as a head node");

printf("\nInsert a node as a last node");

printf("\nInsert a node as at the intermediate position in the list ");

printf("\nEnter your choice for insertion of node ");

scanf("%d",&choice);

switch(choice)

{ case 1:head =

insert\_head(head);

break;

case 2:insert\_last(head);

break;

case 3:insert\_after (head);

break;

}

return head;

} /\*Insertion of node at first position\*/

node \*insert\_head(node\*head)

{

node \*New,\*temp;

New = get\_node();

printf ("\n Enter the element which you want to insert ");

scanf("%d",&New->data);

if(head == NULL)

head = New;

else

{ temp=head;

New->next = temp;

head= New;

}

return head;

} /\*Insertion of node at last position\*/

void insert\_last(node \*head)

{

node \*New,\*temp;

New = get\_node();

printf ("\n Enter the element which you want to insert ");

scanf("%d",&New->data);

if(head == NULL)

{

head = New;

} else

{ temp=head;

while(temp->next!=NULL)

temp=temp->next;

temp->next=New;

New->next=NULL;

}} /\*Insertion of node at intermediate position\*/

void insert\_after(node \*head)

{ int key;

node \*New,\*temp;

New = get\_node();

printf("Enter the element after which you want to insert ");

scanf("%d",&key);

temp=head;

do

{ if(temp->data==key)

{

printf ("Enter element which you want to insert ");

scanf("%d",&New->data);

New->next=temp->next;

temp->next=New;

return;

} else

temp=temp->next;

}while(temp!=NULL);

}

node \*get\_prev(node \*head,int val)

{

node \*temp,\*prev;

int flag;

temp = head;

if(temp == NULL)

return NULL;

flag = FALSE;

prev = NULL;

while(temp!=NULL && !flag)

{ if(temp->data!=val)

{

prev = temp;

temp = temp->next;

} else

flag = TRUE;

} if(flag) /\*if Flag is true\*/

return prev;

else

return NULL;

}

void dele(node \*\*head)

{ int key;

node \*New,\*temp, \*prev;

temp=\*head;

if (temp== NULL)

{

printf ("\n The list is empty\n ");

getch();

return;

}

printf("\nENTER the Element you want to delete:");

scanf("%d",&key);

temp= search(\*head,key);

if(temp !=NULL)

{

prev = get\_prev(\*head,key);

if(prev != NULL)

{

prev ->next = temp-> next;

free(temp);

} else

{

\*head = temp->next;

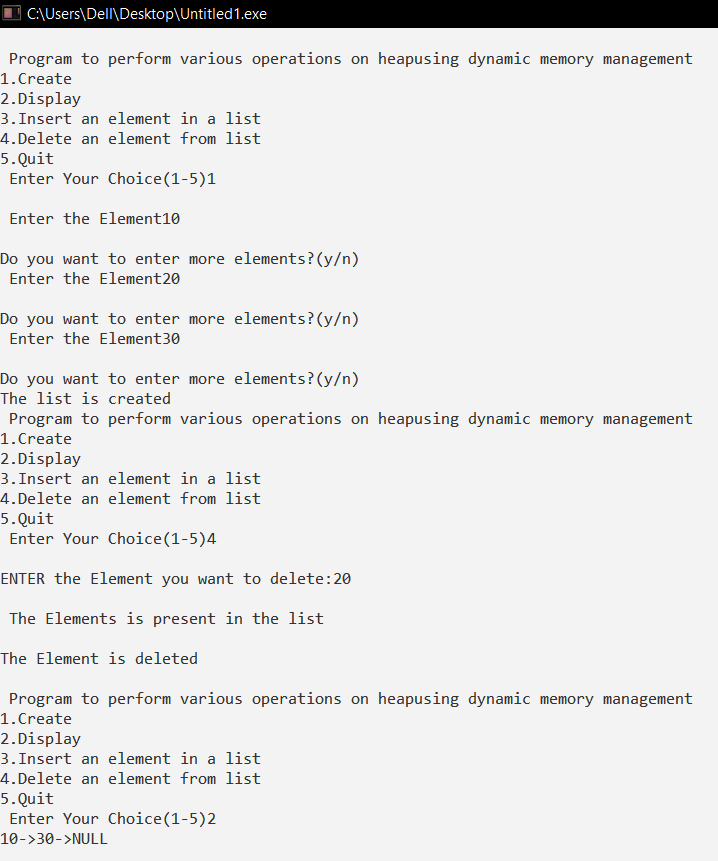
free(temp); // using the mem. Dellocation function

}

printf("\nThe Element is deleted\n");

getch();}}

**OUTPUT:**



**PROGRAM – 11**

**AIM:** Implementation of simple code optimization techniques (constant folding. etc.)

**ALGORITHM:**

1. Start
2. Create an input file which contains three address code.
3. Open the file in read mode.
4. If the file pointer returns NULL, exit the program else to go 5.
5. Scan the input symbol from left to right. (Common Sub expression elimination)
6. Store the first expression in the string.
7. Compare the string with the other expressions in the file.
8. If there is a match, remove the expressions from the file.
9. Perform these steps 5 to 8 for all the input symbols in the file. (dead code elimination)
10. Scan the input symbols from the file from left to right.
11. Get the operands before the operator from the three address code.
12. Check whether the operands is used in any other expression in the three address code.
13. If the operand is not used, then eliminate the complete expression from the three address code else go to 14.
14. Perform steps 11 to 13 for all operands in the address code till end of file is reached.
15. Stop.

**SOURCE CODE:**

#include<stdio.h>

#include<string.h>

struct op

{

char l;

char r[20];

}

op[10],pr[10];

void main()

{

int a,i,k,j,n,z=0,m,q;

char \*p,\*l;

char temp,t;

char \*tem;

printf("Enter the Number of Values:");

scanf("%d",&n);

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

{

printf("left: ");

scanf(" %c",&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("\nAfter 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);

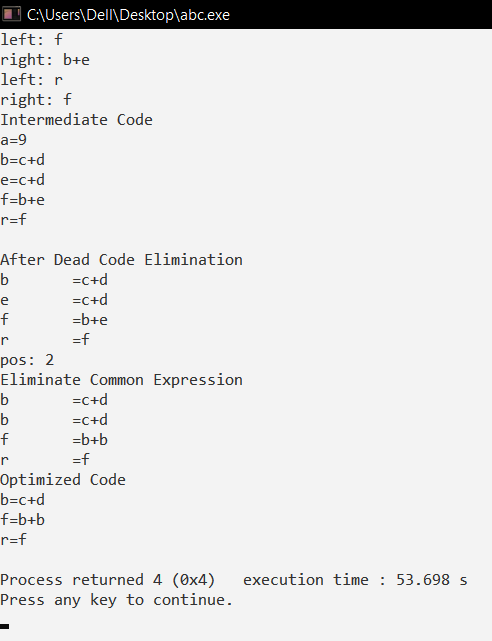
printf("%s\n",pr[i].r);

}

}

}

**OUTPUT:**

****