Ex No:1 **IMPLEMENTATION OF SYMBOL TABLE**

Date:

Aim:

To write a 'C' program for the implementation of Symbol Table.

Algorithm:

```
1. Start
2. Define a structure for storing symbol table.
3. Enter the choice.
   If the choice is 1( Insert operation),
       i.   Get the symbol to be inserted.
       ii.  Check whether the current symbol is already present. If so, print it as duplicate
            symbol and go to step 4.
       iii. If not, store symbol, data and name in the symbol table.
       iv.  Increment the number of entries in the symbol table by 1.
   If the choice is 2(Delete operation),
       i.   Enter the symbol to be deleted.
       ii.  Check whether the symbol is present or not. If not, go to step 4.
       iii. If so delete the symbol from the table.
       iv.  Decrement the no. of entries in the symbol table by 1.
   If the choice is 3(Display operation),
       i.   Display the contents of symbol table.
   If the choice is 4(Search Operation),
       i.   Check whether the table is empty. If so go to step 4.
       ii.  Get the symbol to be searched.
       iii. Search the symbol if found and display it.
   If the choice is 5(Modify operation),
       i.   Enter the symbol to be modified.
       ii.  Check whether the symbol table is empty. I
       iii. If so, go to step 4.
       iv.  Otherwise, get the new value for the symbol and store.
4. Stop the program.
```

Program:-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
struct sym_tab
{
    char symbol[20];
    char type[20];
    int length;
};
struct sym_tab s[10];
int n=0;
main()
{
    int ch;
    void insert();
    void del();
    void disp();
    void search();
    void modify();
    do
    {
            printf("\n1.insert\n2.delete\n3.display\n4.search\n5.modify\n6.exit\n");
            printf("\n enter the choice\n");
            scanf("%d",&ch);
            switch(ch)
            {
                    case 1:         insert();
                                    break;
                    case 2:         del();
                                    break;
                    case 3:         disp();
                                    break;
                    case 4:         search();
                                        break;
                    case 5:         modify();
                                            break;
                    default:        break;
            }
    }while(ch<6);
}
void insert()
{
    char name[20],data[20];
    int leng,i,k,length;
    printf("enter newsymbol,datatype,length\n");
    scanf("%s%s%d",name,data,&leng);
    for(i=0;i<n;i++)
            if(strcmp(name,s[i].symbol)==0)
            {
                    printf("duplicate entry\n");
                    return;
            }
    strcpy(s[n].symbol,name);
    strcpy(s[n].type,data);
    s[n].length=leng;
    n++;
}
void del()
{
    int i,k;
    char sym[20];
    printf("Enter the symbol to be deleted\n");
    scanf("%s",sym);
    if(n==0)
    {
```

```
                printf("empty table\n");
                return;
        }
        for(i=0;i<n;i++)
                if(strcmp(sym,s[i].symbol)==0)
                {
                        for(k=i;k<n-1;k++)
                        {
                                strcpy(s[k].type,s[k+1].type);
                                s[k].length=s[k+1].length;
                        }
                        n--;
                        printf("the symbol is deleted\n");
                }
}
void modify()
{
        char name[20],data[20],old[20];
        int len,i;
        if(n==0)
        {
                printf("empty tables\n");
                return;
        }
        printf("Enter the symbol to be modified\n");
        scanf("%s",old);
        for(i=0;i<n;i++)
        {
                if(strcmp(old,s[i].symbol)==0)
                {
                        printf("symbol is found %s \t%s \t\t%d",s[i].symbol,s[i].type,s[i].length);
                        printf("\nEnter new values for datatypes,length\n");
                        scanf("%s%d",data,&len);
                        strcpy(s[i].type,data);
                        s[i].length=len;
                        printf("symbol entries modified\n");
                        return;
                }
        }
}
void search()
{
        int i;
        char name[20];
        if(n==0)
        {
                printf("empty table\n");
                return;
        }
        printf("enter the symbol to be searched\n");
        scanf("%s",name);
        for(i=0;i<n;i++)
```

```
     {
               if(strcmp(name,s[i].symbol)==0)
               {
                         printf("symbol found\n%s \t%s\t\t%d",s[i].symbol,s[i].type,s[i].length);
                         return;
               }
     }
}
void disp()
{
     int i;
     if(n==0)
     {
               printf("empty table\n");
               return;
     }
     printf("symbol\tdatatype\tlength\n");
     for(i=0;i<n;i++)
               printf("%s\t%s\t\t%d\n",s[i].symbol,s[i].type,s[i].length);
}
```

Output:

Viva answers:

1. Define compiler?
2. What are the classifications of compiler?
3. Whatare the phases of compiler?
4. Define preprocessor & what are the functions of preprocessor?
5. What are the tools available in analysis phase?

Result:

Ex No:2 **DEVELOPING A LEXICAL ANALYZER**

Date:

Aim:

To develop a lexical analyzer to recognize a few patterns in C.

Algorithm:

```
1. Start.
2. Get the input expression and store it in an array.
3. Check whether the current character is an alphabet or not.
   3.1 If, yes, extract successive character or digit and store it in a Temporary array Variable.
   3.2 Compare the words in Temporary Variable with the entire keyword list.
   3.3 If matches with anyone keyword display it as keywords.
   3.4 Otherwise display it as identifier
4. Check whether the current character is a digit or not.
   4.1 If so, extract all successive digits and store it in a temporary array, display it as constant.
5. Check the current character with all special characters.
   5.1 If match is found, display it as special character.
6. Check the current character with all operators.
   6.1 If match is found display it as operator.
   6.2 Repeat from step 3 until the end of the string is reached.
7. Stop.
```

Program:-

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
# include <ctype.h>
main()
{
        char in[50], temp[50];
        int i = 0, j = 0;
        printf("Enter the expression\n");
        gets(in);
        printf("\nKeyword \t Identifier \t constant \t operator \t sp. character \n");
        while(in[i] != '\0')
        {
                if (isalpha(in[i]))
                {
                        while (isalpha(in[i]) || isdigit(in[i]))
                                temp[j++] = in[i++];
                        temp[j] = '\0';
                if (strcmp(temp, "if") == 0 || strcmp(temp, "int") == 0 || strcmp(temp, "char") ==   0
                || strcmp(temp, "else") == 0 || strcmp(temp, "float") == 0 || strcmp(temp, "do")
                        == 0 || strcmp(temp, "for") == 0 || strcmp(temp, "while") == 0)
                        printf("\n%s", temp);
                else
                        printf("\n\t\t\t%s", temp);
                }
```

```
                    else if (isdigit(in[i]))
                    {
                              while (isdigit(in[i]))
                                        temp[j++] = in[i++];
                              temp[j] = '\0';
                              printf("\n\t\t\t\t\t%s", temp);
                    }
                    else if (in[i] =='+' || in[i] == '-' || in[i] == '*' || in[i] == '/' || in[i] == '>' || in[i] == '<'
                              || in[i] == '=' || in[i] == '!')
                              printf("\n\t\t\t\t\t\t%c", in[i++]);
else if (in[i] ==';' || in[i] == ':' || in[i] == '(' || in[i] == ')' || in[i] == '{' || in[i] == '}' ||   in[i] == '.')
                              printf("\n\t\t\t\t\t\t\t\t%c", in[i++]);
                    else
                              i++;
                    j=0;
          }
          getch ();
}
```

Output:

Viva answers:

1. Define pretty printers?
2. Define assembler and its types?
3.Give the types of a language processing system?
4. What are the functions performed in analysis phase?
5. What are the functions performed in synthesis phase?

Result:

Ex.No:3 **Implementation of Lexical analyzer using LEX tool**

Date:

Aim:

To write a program for implementing the lexical analyzer using Lex tool

Algorithm:

Step 1: Start
Step 2: Patterns for various tokens are specified in LEX language.
Step 3: Input program is given as command line argument.
Step 4: Now the LEX program identifies the various tokens in the given input program.
Step 5: Stop.

Program:

```
%{
/* program to recognize a c program */
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int |
float |
char |
double |
while |
for |
do |
if |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto {printf("\n\t%s is a KEYWORD\n",yytext);}
"/*" {COMMENT = 1;
printf("\n\t%s is a COMMENT\n",yytext);}
"*/" {COMMENT = 0;
printf("\n\t%s is a COMMENT\n",yytext);}
{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}
\} {if(!COMMENT) printf("\n BLOCK ENDS");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n\t%s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}
```

```
[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}
\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc,char **argv)
{
if (argc > 1)
{
FILE *file;
file = fopen(argv[1],"r");
if(!file)
{
printf("could not open %s \n",argv[1]);
exit(0);
}
yyin = file;
}
yylex();
printf("\n\n");
return 0;
}
int yywrap()
{
return 1;
}
```

test.c: //Save in C:\Dev-Cpp\bin>

```
#include<stdio.h>
main()
{
int a,b;
}
```

Steps to run:

```
C:\Users\merlin>cd C:\Dev-Cpp\bin\GnuWin32\bin
C:\Dev-Cpp\bin\GnuWin32\bin>flex lex1.l
C:\Dev-Cpp\bin\GnuWin32\bin>cd C:\Dev-Cpp\bin
C:\Dev-Cpp\bin>gcc lex.yy.c
C:\Dev-Cpp\bin>a.exe test.c
```

Output:

Viva answers:

1. Give the classification of processing performed by the semantic analysis?
2. Give the properties of intermediate representation?
3. What are the two different parts of compilation?
4. What is meant by lexical analysis?
5. What is meant by syntax analysis?

Result:

Ex. No. 4 **<u>Generate YACC Specifications for Few Syntactic Categories</u>**

Date:

a) YACC program to recognize a valid arithmetic expression that uses operators +,-,* and /

<u>Aim:</u>

To write YACC program to recognize a valid arithmetic expression that uses operators +,-,* and /

<u>Algorithm :</u>

```
Step 1 : Start
Step 2 : Get the input arithmetic expression.
Step 3 : Check whether the input is valid or not by using Yacc rules.
Step 4 : Print valid if the expression is correct.
Step 5 : Otherwise print it is invalid.
Step 6 : Stop
```

<u>Program:</u> **Sava in C:\Dev-Cpp\bin\GnuWin32\bin\exp.y**

```
%{ /* validate simple arithmetic expression */
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
%}
%token num let
%left '+' '-'
%left '*' '/'

%%

stmt: stmt  '\n' {printf("\n Valid \n");exit(0);}
| expr
|
|error '\n' {printf("\n Invlaid \n");exit(0);}
;
expr: num
| let
| expr '+' expr
| expr '-' expr
| expr '*' expr
| expr '/' expr
| '('expr')'

%%

main()
{
printf(" Enter an expression to validate: ");
yyparse();
}
yylex()
```

```
{
int ch;
while((ch=getchar())==' ');
if(isdigit(ch))
return num;
if(isalpha(ch))
return let;
return ch;
}

yyerror(char *s)
{
printf("%s",s);
}
```

Steps to run:

```
C:\Dev-Cpp\bin>cd C:\Dev-Cpp\bin\GnuWin32\bin
C:\Dev-Cpp\bin\GnuWin32\bin>bison exp.y
C:\Dev-Cpp\bin\GnuWin32\bin>cd C:\Dev-Cpp\bin
C:\Dev-Cpp\bin>gcc exp.tab.c
C:\Dev-Cpp\bin>a.exe
 Enter an expression to validate: c+a*5/b
 Valid
C:\Dev-Cpp\bin>
```

Output:

Viva answers:

1. Define patterns/lexeme/tokens?
2. Give the algebraic properties of regular expression?
3. What are issues available in lexical analysis?
4. Give the parts of a string?
5. What are the operations on language?

Result:

b) YACC program to recognize a valid variable, which starts with a letter, followed by any number

Aim:

To write YACC program to recognize a valid variable.

Algorithm :

```
Step 1 : Start
Step 2 : Get the input variable.
Step 3 : Check whether the input is valid or not by using Yacc rules.
Step 4 : Print valid if the variable name is correct.
Step 5 : Otherwise print it is not valid.
Step 6 : Stop
```

Program:

```
%{/* YACC pgm to recognize valid variable, which starts with a letter, followed by any number of letters or digits. */
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
%}
%token let dig
%%
TERM:XTERM'\n'{printf("\nAccepted\n");exit(0);}
|error {yyerror("\nRejected");
exit(0);
}
;
XTERM:XTERM let
|XTERM dig
|let
;
%%
main()
{
    printf("Enter a variable: ");
    yyparse();
    }
yylex()
{
     char ch;
     while((ch=getchar())==" ");
     if(isalpha(ch))
     return let;
     if(isdigit(ch))
     return dig;
     return ch;
     }
yyerror(char *s)
{
    printf("%s",s);
}
```

Steps to run:

```
C:\Dev-Cpp\bin>cd C:\Dev-Cpp\bin\GnuWin32\bin
C:\Dev-Cpp\bin\GnuWin32\bin>bison variableyacc.y
C:\Dev-Cpp\bin\GnuWin32\bin>cd C:\Dev-Cpp\bin
C:\Dev-Cpp\bin>gcc variableyacc.tab.c
variableyacc.y: In function `yylex':
variableyacc.y:26: warning: comparison between pointer and integer
C:\Dev-Cpp\bin>a.exe
```

Output:

```
C:\Dev-Cpp\bin>a.exe
Enter a variable: vari3
Accepted

C:\Dev-Cpp\bin>a.exe
Enter a variable: 3vari
syntax error
Rejected
```

Viva answers:

1. What are the implementations of lexical analyzer?
2. Define regular expression?
3. Give the types of notational shorthand’s of RE?
4. Define kleene closure or star closure and positive closure?
5. Give the error recovery strategies in lexical analyzer.

Result:

Ex. No. 5 **Implementation of Type Checking**

Date:

Aim:

To write a C program to implement type checking.

Algorithm:

1. Start
2. Enter the value for 2 variables
3. After giving input values check whether the given value matches with the type or not
4. If yes then print "No type error"
5. Else print "Type error".
6. Stop

Program:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
char* type(char[],int);
main()
{
char a[10],b[10],mess[20],mess1[20];
int i,l;
printf("\n\n int a,b;\n\n int c=a+b\n");
printf("\n\n Enter a value for a\n");
scanf("%s",a);
l=strlen(a);
printf("\n a is:");
strcpy(mess,type(a,l));
printf("%s",mess);
printf("\n\n Enter a value for b\n\n");
scanf("%s",b);
l=strlen(b);
printf("\n b is:");
strcpy(mess1,type(b,l));
printf("%s",mess1);
if(strcmp(mess,"int")==0 && strcmp(mess1,"int")==0)
{
printf("\n\n No Type Error");
}
else
{
printf("\n\n Type Error");
}
getch();
}
char* type(char x[],int m)
{
```

```
int i; char mes[20];
for(i=0;i<m;i++)
{
if(isalpha(x[i]))
{
strcpy(mes,"AlphaNumeric");
goto x; }
else if(x[i]=='.')
{ strcpy(mes,"float"); goto x; }
} strcpy(mes,"int");
x:return mes;
}
```

Output:

Viva answers:

1. What are the models of LEX compiler?
2. What are the four functions of regular expression to DFA?
3. What do u meant by parser and its types?
4.What are the different levels of syntax error handler?
5. What are error recovery strategies in parser?

Result:

Ex. No. 6 **Generate three address code for a simple program using LEX and YACC.**

Date:

Aim:

To write a C program to Generate three address code for a simple program using LEX and YACC.

Algorithm:

1. Start the program.
2. Get the choice from the user.
3. If choice is 1 enter an assignment expression then generate and display the three address code for the expression.
4. If choice is 2 enter an arithmetic expression then generate and display the three address code for the expression.
5. If choice is 3 exit the program.
6. Stop the program.

Program:

```
#include<stdio.h>
#include<string.h>
void pm();
void plus();
void div();
int i,ch,j,l;
char ex[10],ex1[10],exp1[10],ex2[10];
main()
{
while(1)
{
printf("\n 1.Assignment\n 2.Arithmatic\n 3.exit\n ENTER THE
CHOICE:");
scanf("%d",&ch);
switch(ch)
{
case 1:printf("\n enter the expression with assignment operator:");
scanf("%s",ex1);
l=strlen(ex1);
ex2[0]='\0';
i=0;
while(ex1[i]!='=')
{
i++;
}
strncat(ex2,ex1,i);
strrev(ex1);
exp1[0]='\0';
strncat(exp1,ex1,l-(i+1));
strrev(exp1);
printf("3 address code:\n temp=%s \n %s=temp\n",exp1,ex2);
break;
case 2:printf("\n enter the expression with arithmatic operator:");
scanf("%s",ex);
strcpy(ex1,ex);
l=strlen(ex1);
exp1[0]='\0';
for(i=0;i<l;i++)
{
if(ex1[i]=='+'||ex1[i]=='-')
{
if(ex1[i+2]=='/'||ex1[i+2]=='*')
{
pm();
break;
}
else
{
plus();
break;
}
}
else if(ex1[i]=='/'||ex1[i]=='*')
{
div();
break;
}
}
break;
}
}
break;
case 3:exit(0);
}
}
}
void pm()
{
strrev(exp1);
j=l-i-1;
strncat(exp1,ex1,j);
strrev(exp1);
printf("3 address code:\n temp=%s\n temp1=%c%c
temp\n",exp1,ex1[j+2],ex1[j]);
}
void div()
{
strncat(exp1,ex1,i+2);
printf("3 address code:\n temp=%s\n
temp1=temp%c%c\n",exp1,ex1[l+2],ex1[i+3]);
}
void plus()
{
```

```
strncat(exp1,ex1,i+2);
printf("3 address code:\n temp=%s\n
temp1=temp%c%c\n",exp1,ex1[l+2],ex1[i+3]);
}
```

OUTPUT:

<u>Viva answers:</u>

1.Define CFG?

2. Define ambiguity?

3. Give the several reasons for writing a grammar?

4.Define yield of the string?

5. Define left factoring?

RESULT:

Ex. No. 7 **Implementation of Heap Storage Allocation Strategy**

Date:

Aim:

To implement heap allocation strategy

Algorithm:

```
Step 1: Start
Step 2: Get the choice from user
Step 3: If the choice is 1
            i.   Allocate memory for data using dynamic memory allocation
            ii.  Make a list of data created
        If the choice is 2
            i.   Start from the first data in the list
            ii.  Display the data
            iii. Move on to the next data
        If the choice is 3
            i.   Get the data to be inserted
            ii.  Add the data into the list in the proper location
            iii. make changes in list pointers
        If the choice is 4
            i.   Get the data to be deleted
            ii.  Search for the data starting from the first data
            iii. Remove it and make changes in list pointers
        If the choice is 5
            i.   goto step 4
Step 4: Stop
```

Program:

```
#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();
main()
{
	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 heap using dynamic memory management");
		printf("\n1.create");
		printf("\n2.display");
		printf("\n3.insert an element in a list");
		printf("\n4.delete an element from a 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,*new,*head;
	int val,flag;
	char ans='y';
	node *get_node();
	temp=NULL;
	flag=TRUE;
	do
	{
		printf("\n enter the Element");
		scanf("%d",&val);
		new=get_node();
		if(new==NULL)
			printf("\n memory is not allocated");
		new->data=val;
		if(flag==TRUE)
		{
			head=new;
			temp=head;
```

```
				flag=FALSE;
			}
			else
			{
				temp->next=new;
				temp=new;
			}
			printf("\n do you want to enter more elements?(y/n)");
			ans=getch();
		}while(ans=='y');
		printf("\n the list is created");
		getch();

		return head;
}
node *get_node()
{
		node *temp;
		temp=(node*)malloc(sizeof(node));
		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("\nthe 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("\nthe 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("\n 1.insert a node as a head node");
	printf("\n 2.insert a node as a last node");
	printf("\n 3.insert a node as at the intermediate position in the list");
	printf("\n enter your choice for insertion of a 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;
}
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;
}
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;
	}
}
void insert_after(node *head)
{
	int key;
	node *New,*temp;
	New=get_node();
	printf("\n enter the element after which you want to insert");
	scanf("%d",&key);
	temp=head;
	do
	{
		if(temp->data==key)
		{
			printf("enter the 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)
		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("\n enter 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);
		}
		printf("\n element is deleted\n");
		getch();
```

```
        }
}
```

<u>Output:</u>

Viva answers:

1. Define LL (1) grammar?
2. What are the possibilities of non-recursive predictive parsing?
3. What are the actions available in shift reduce parser?
4. Define top down parsing?
5. Define handle?

Result:

Ex.No: 8 **Code Optimization Using Constant Folding**

Date:

Aim:

To perform code optimization.

Algorithm:

Step 1: Start
Step 2: Create an input file
Step 3: Read the input file to optimize the code
Step 4: Print the optimized code on to the output file
Step 5: Stop

Program:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
struct ConstFold {
char new_str[10];
char str[10];
} Opt_Data[20];
void ReadInput(char Buffer[], FILE *Out_file);
int Gen_token(char str[], char Tokens[][10]);
int New_Index = 0;
int main() {
FILE *In_file, *Out_file;
char Buffer[100], ch;
int i = 0;
In_file = fopen("code.txt", "r");
Out_file = fopen("output.txt", "w");
while(1) {
ch = fgetc(In_file);
i = 0;
while(1) {
if(ch == '\n') break;
Buffer[i++] = ch;
ch = fgetc(In_file);
if(ch == EOF) break;
}
if(ch == EOF) break;
Buffer[i] = '\0';
ReadInput(Buffer, Out_file);
}
return 0;
}
void ReadInput(char Buffer[], FILE *Out_file) {
char temp[100], Token[10][10];
int n, i, j, flag = 0;
strcpy(temp, Buffer);
n = Gen_token(temp, Token);
for(i=0; i<n; i++) {
if(!strcmp(Token[i], "=")) {
if(isdigit(Token[i+1][0]) || Token[i+1][0] == '.')
{
flag = 1;
strcpy(Opt_Data[New_Index].new_str, Token[i-1]);
strcpy(Opt_Data[New_Index++].str, Token[i+1]);
}
}
}
if(!flag) {
for(i=0; i<New_Index; i++) {
for(j=0; j<n; j++) {
if(!strcmp(Opt_Data[i].new_str, Token[j]))
strcpy(Token[j], Opt_Data[i].str);
}
}
}
fflush(Out_file);
strcpy(temp, "");
for(i=0; i<n; i++) {
strcat(temp, Token[i]);
if(Token[i+1][0]!=',' || Token[i+1][0]!=';')
strcat(temp, "");
}
strcat(temp, "\n\0");
fwrite(&temp, strlen(temp), 1, Out_file);
}
int Gen_token(char str[],char Token[][10])
{
int i=0, j=0, k=0;
while(str[k]!= '\0') {
j=0;
while(str[k]==' ' || str[k] == '\t')
k++;
while(str[k]!= ' '&&str[k]!='\0'&&str[k]!='='&&str[k]!= '/'&&str[k]!='+
'&&str[k]!='-'&&str[k]!='*'&&str[k]!=','&&str[k]!= ';')
Token[i][j++] = str[k++];
Token[i++][j] = '\0';
if(str[k] == '=' || str[k] == '/' || str[k] == '+' || str[k] == '-' ||
str[k] == '*' || str[k] == ',' || str[k] == ';')
{
Token[i][0] = str[k++];
Token[i++][1] = '\0';
}
if(str[k] == '\0')
break;
}
return i;
}
```

Input.txt

```
#include<stdio.h>
main()
{
float pi=3.14,r,a;
a=pi*r*r;
printf("a=%f",a);
return 0;
}
```

Output:

Viva answers:

1.What are the drawbacks of LR parser?
2. Define LR parser?
3. Define augmented grammar?
4. Define LR (0) items?
5. What are the two functions of LR parsing algorithm?

Result:

Ex.No: 9 **Implementation of Simple Code Optimization Techniques**

Date:

Aim:

To implement the common sub expression elimination, dead code elimination and optimization technique.

Algorithm:

```
Step 1: Start
Step 2: Get the number of values and the corresponding values for expressions
Step 3: Print the intermediate code.
Step 4: Search for dead code eliminate and display.
Step 5: Find and eliminate all common expressions and display.
Step 6: Print Optimized code.
Step 7: Stop.
```

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct op
{
char l;
char r[20];
}op[10],pr[10];

main()
{
int a,i,k,j,n,z=0,m,q;

char *p,*l;
char temp,t;
char *tem;
printf("Enter number of values: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Left: ");
op[i].l=getche();
printf("\tright: ");
scanf("%s",op[i].r);
}
printf("\nIntermediate 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);
}

//sub expression elimination
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;
pr[i].r[a]=pr[m].l;
}}}}}
printf("\nEliminate Common Expression:\n");
for(i=0;i<z;i++)
{
printf("%c\t=",pr[i].l);
printf("%s\n",pr[i].r);
}
// duplicate production elimination

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';
   strcpy(pr[i].r,'\0');
 }}
}
printf("\nOptimized code: \n");
for(i=0;i<z;i++)
{
if(pr[i].l!='\0')
{
printf("%c=",pr[i].l);
printf("%s\n",pr[i].r);
}
}
getch();
}
```

Output:

```
Enter number of values: 3
Left: a right: 8
Left: b right: c+d
Left: a right: c+d

Intermediate Code:
a=8
b=c+d
a=c+d

After Dead code Elimination:
a       =c+d

Eliminate Common Expression:
a       =c+d

Optimized code:
a=c+d
```

Viva answers:
1. Define an attribute. Give the types of an attribute?
2. Define annotated parse tree?
3. Define dependency graph?
4. What are the functio s used to create the nodes of syntax trees?
5. What are the functions for constructing syntax trees for expressions?

Result:

Ex.No: 10 **Implementation of code generator**

Date:

Aim:

To implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using a 8086 assembler. The target assembly instructions can be simple move, add, sub, jump. Also simple addressing modes are used.

Algorithm:

Step 1: Get the expression
Step 2: Move the integer identifier to register
Step 3: Based on the instruction, include mnemonic like sub,mul,mov and add with corresponding operands.
Step 4: Stop

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
        int n,i,j;
        char a[50][50];
        printf("\n Enter the number of intermediate code:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter the three address code %d:",i+1);
                for(j=0;j<6;j++)
                {
                        scanf("%c",&a[i][j]);
                }
        }
        printf("\n The Generated code:");
        for(i=0;i<n;i++)
        {
                printf("\n MOV %c,R%d",a[i][3],i);
                if(a[i][4]=='-')
                {
                        printf("\n SUB %c,R%d",a[i][5],i);
                }
                if(a[i][4]=='+')
                {
                        printf("\n ADD %c,R%d",a[i][5],i);
                }
                if(a[i][4]=='*')
                {
                        printf("\n MUL %c,R%d",a[i][5],i);
                }
                if(a[i][4]=='/')
                {
                        printf("\n DIV %c,R%d",a[i][5],i);
                }
                printf("\n MOV R%d,%c",i,a[i][1]);
                printf("\n");
        }
        getch();
}
```

Output:

```
Enter the number of intermediate code:4
Enter the three address code 1:T=A-B
Enter the three address code 2:U=A-C
Enter the three address code 3:V=T+U
Enter the three address code 4:W=V+U

 The Generated code:
 MOV A,R0
 SUB B,R0
 MOV R0,T

 MOV A,R1
 SUB C,R1
 MOV R1,U

 MOV T,R2
ADD U,R2
 MOV R2,V

 MOV V,R3
ADD U,R3
 MOV R3,W
```

Viva answers:

1. What are the two purposes of Boolean expressions?
2. Define quadruple. Give an example?
3. Define triple. Give an example?
4.Define indirect triples. Give the advantage?
5. What are the three address code for a or b and not c?

Result: