Ex. No. :1
Date:

Token Separation in C

Aim:

To write a simple c program to perform token separation in c

Algorithm:

1. Start the program
2. Get the string from the user
3. Scan each character one by one
4. If the character is a letter look for identifiers and print it
5. If the character is a digit look for constants and print it
6. If the character is operator print it
7. Stop the program.

Code:

```
#include<stdio.h>
#include<string.h>
int  main()
{

            char a[20];
            int i=0,j=0;
            int id1=0,id2=0,id3=0;
            char constant[10],op[10],identifier[10];
            printf("Enter the string\n");
            fgets(a,20,stdin);
            for(i=0;i<strlen(a);i++)
            {
                        if(a[i]>='a' && a[i]<='z')
                        {
                                    identifier[id1]=a[i];
                                    id1++;
                                    i++;
                                    while(a[i]>='a' && a[i]<='z' || a[i]>='0' && a[i]<='9')
                                    {
                                                identifier[id1]=a[i];
                                                id1++;
                                                i++;
                                    }
                  identifier[id1]='\0';
                                    if (strcmp(identifier,"if")==0)
                   {
                     printf("Keyword::");
                     for(j=0;j<id1;j++)
                    {
                         printf("%c",identifier[j]);
                    }

                   }
```

```
                else                    {
                                printf("identifier::");
                                for(j=0;j<id1;j++)
                                {
                                        printf("%c",identifier[j]);
                                }
                }
                                printf("\n");
                                id1=0;
                        }
                        if(a[i]=='+'||a[i]=='-
'||a[i]=='*'||a[i]=='/'||a[i]=='%'||a[i]=='='||a[i]=='>'||a[i]=='<')                    {
                                //op[id2]=a[i];
                                //id2++;
                                printf("operator::%c\n",a[i]);
                        }
                        if(a[i]>='0' && a[i]<='9')              {
                                constant[id3]=a[i];
                                i++;
                                id3++;
                                while(a[i]>='0' && a[i]<='9')
                                {
                                        constant[id3]=a[i];
                                        id3++;
                                        i++;
                                }
                                printf("Constant::");
                                for(j=0;j<id3;j++)
                                {
                                        printf("%c",constant[j]);
                                }
                                printf("\n");
                                id3=0;

                        }

        }
}
```

Sample Input-Ouput:

```
Enter the string
if(a>b)
Keyword::if
identifier::a
operator::>
identifier::b
```

Result:

Thus C program to perform token separation is successfully executed.

Ex.No. :2
Date:

Symbol Table in C

Aim:

To write a simple c program to simulate symbol table generation

Algorithm:

1. Start processing.
2. Declare structure for input and output files
3. Declare File pointers for input and output files
4. Open Input File(s) in Read mode and Open Output File(s) in write Mode. 5. Read the Intermediate File until EOF occurs.

5.1If Symbol is not equal to NULL then 5.2Write the Symbol Name and its address into Symbol table.

6. Close all the Files.
7. Print Symbol Table is created.
8. Stop processing.

Code:

```
# include<stdio.h>
# include<string.h>
struct syt{
 char v[10],t[10];
 int s;
} st[50];
char *tt[] = {"int","float","long","double","short"};
int vt[5]= {2,4,6,8,1};
FILE *fp;
int main(){
 char *fn,u[20],t[20],s[100],*p;
 int i=0,j,k,l,y,sp=0;
 fp=fopen("out.c","r");
 printf("\nSymbol table maintenance");
 printf("\n\tVariable\tType\t\tsize\n");
 while(!feof(fp)) {
 fscanf(fp,"%s",s);
 for(i=0;i<5;i++)
 if(strcmp(s,tt[i])==0) {
 fscanf(fp,"%s",s);
 p=strchr(s,',');
 if(p) {
 j=0;
 while(s[j]!=',')
 j++;
 for(k=0;k<j;k++)
 t[k]=s[k];
 t[k]='\0';
 printf("\n%10s\t%10s\t%10d",t,tt[i],vt[i]);
 strcpy(st[sp].v,t);
 strcpy(st[sp].t,tt[i]);
 st[sp].s=vt[i];
 sp++;
 kjk:
 y=j;
 j++;
 while(s[j]!='\0'&&s[j]!=',')
 j++;
 for(l=y;l<j;l++)
 t[l-2]='\0';
 printf("\n%10s\t%10s\t%10d",t,tt[i],vt[i]);
 strcpy(st[sp].t,tt[i]);
 st[sp].s=vt[i];
 sp++;
 if(s[j]==',')
 goto kjk;
 }
 else{
 printf("\n%10s\t%10s\t%10d",s,tt[i],vt[i]);
 strcpy(st[sp].v,t);
 strcpy(st[sp].t,tt[i]);
 st[sp].s=vt[i];
 }
 }
 }
 fclose(fp);
 for(i=0;i<sp;i++)
 strcpy(t,st[i].v);
 k=0;
 for(j=0;j<strlen(t);j++) {
 if(t[i]!=',') {
 u[k]=u[j];
 k++;
 }
 u[k]='\0';
 strcpy(st[i].v,u);
 }
 for(i=0;i<sp-2;i++)
 for(j=i+1;j<sp;j++) {
 if(strcmp(st[i].v,st[j].v)==0)
 printf("\n\nMultiple Declaration for %s",st[i].v);
 }
 }
```

Sample Input-Ouput:

```
Symbol table maintenance
       Variable       Type          size
        a            int            2
        b          float            4
```

Result:

Thus the program simulating symbol table is successfully executed.

Ex. No.: 3
Date:

Lex Programs

Aim:

To use Lex tools to perform simple addition, counting vowels, and letters and token in a given piece of code.

Algorithm:

1. Start the program.
2. Lex program consists of three parts.
a. Declaration %%
b. Translation rules %%
c. Auxilary procedure.
3. The declaration section includes declaration of variables, main test, constants and regular definitions.
4. Translation rule of lex program are statements of the forma. P1 {action}b. P2 {action}c. …d. …e. Pn {action}
5. Write a program in the vi editor and save it with .l extension.
6. Compile the lex program with lex compiler to produce output file as lex.yy.c. eg $ lex filename.l$ cc lex.yy.c –ll
7. Compile that file with C compiler and verify the output

Code:

Simple addition:

```
%{
#include<stdio.h>
int a,b,c;
%}
%%

"a" printf("enter the value of a:"); scanf("%d",&a);
"b" printf("enter the value of b:"); scanf("%d",&b);
"c" printf("the addition of %d,%d is %d:",a,b,c=a+b);

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

Counting Vowels

```
%{
#include<stdio.h>
int vow=0,num=0,let=0;
%}
%%
```

```
[aeiouAEIOU] vow++;
[0-9] num++;
[A-Za-z] let++;
";" printf("\nVOWELS=%d,LETTERS=%d,DIGITS=%d",vow,let,num);
%%
int main()
{
yylex();
return 0;
}
int yywrap()
{
return 0;
}
```

Counting tokens:

```
%{
#include<stdio.h>
#include<string.h>
char str[20];
int opc=0,dc=0,lc=0,varc=0,idc=0,key=0;
%}
ke (if|else|for|while|break|switch|do|default)
digit [0-9]
op [%|*|+|-|=]
id [_a-zA-Z][_a-zA-Z0-9]*
%%
{digit} dc++;
{op} opc++;
{ke} key++;
{id} idc++;
"\n" printf("digit %d\noperator%d,\nidentifier%d,keyword=%d",dc,opc,idc,key);
%%
int main()
{
yylex();
return 0;
}
int yywrap()
{
return 0;
}
```

Sample Input-Ouput:

1.

```
a
enter the value of a:5
b
enter the value of b:4
c
The sum of 5 and 4 is 9
```

2.

```
hello;
vowels=2, consonants=3, digits=0
```

3.

```
2+r=p
Digit=1
Operator=2
Identifier=2
```

Result:

Thus the lex programs are successfully executes.

Ex. No. : 4(a)
Date:

VALIDATE ARITHMETIC EXPRESSION

AIM:

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

ALGORITHM:

```
1)  Start the YACC Program.
2)  Define the Rules, user-defined subroutines and definitions.
        {definitions}
        %%
        {rules}
        %%
        {user-defined subroutines}
3)  yyparse() – implies parsing status
        if( yyparse()==0 )
            Parsing successful
        elseif( yyparse()==1 )
            Parsing failed due to invalid input
        else( yyparse()==2 )
            Parsing failed due to memory exhaustion
4)  yylex() – implies the entry point for the lex and reads input to generate tokens.
        if( yylex()==0)
            End of input stream
5)  yyerror() – it is called when YACC encounters invalid syntax.
6)  Stop the Program.
```

PROGRAM:

```
%{
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#define YYSTYPE double
%}
%token num
%left '+' '-'
%left '*' '/'
%%
st: st expr '\n' {printf("The given expression is Valid");}
|st '\n'
|
|error '\n' {printf("The given expression is Invalid");}
;
```

```
expr: num
|expr '+' expr
|expr '/' expr
%%
main()
{
printf(" Enter the expression to be validated:");
yyparse();
}
yylex()
{
int ch;
while((ch=getchar())==' ');
if(isdigit(ch)|ch=='.')
{
ungetc(ch,stdin);
scanf("%lf",&yylval);
return num;
}
return ch;
}
yyerror(char *s)
{
printf("%S",s);
}
```

OUTPUT:

```
vi arithex.y
yacc -d arithex.y
cc y.tab.c
./a.out
 Enter the expression to be validated:4+5
The given expression is Valid^C
./a.out
 Enter the expression to be validated:d++
The given expression is Invalid^C
```

RESULT:

Thus a YACC Program to recognize a valid arithmetic expression that uses operator +,-,* and / is implemented.

Ex. No. : 4(b)
Date:

VALIDATE VARIABLE

AIM:

To write a YACC program to recognize that a valid variable which starts with a letter followed by any number of letters or digits.

ALGORITHM:

```
1) Start the YACC Program.
2) Define the Rules, user-defined subroutines and definitions.
       {definitions}
       %%
       {rules}
       %%
       {user-defined subroutines}
3) yyparse() – implies parsing status
       if( yyparse()==0 )
              Parsing successful
       elseif( yyparse()==1 )
              Parsing failed due to invalid input
       else( yyparse()==2 )
              Parsing failed due to memory exhaustion
4) yylex() – implies the entry point for the lex and reads input to generate tokens.
       if( yylex()==0)
              End of input stream
5) yyerror() – it is called when YACC encounters invalid syntax.
6) Stop the Program.
```

PROGRAM:

```
%{
#include<stdio.h>
#include<ctype.h>
%}
%token let dig
%%
sad: let recld '\n' {printf("accepted\n"); exit(0);}
| let '\n' {printf("accepted\n"); exit(0);}
|
|error {yyerror(" rejected\n");}
;
recld: let recld
| dig recld
| let
| dig
;
%%
yylex(){
```

```
char ch;
while((ch=getchar())==' ');
if(isalpha(ch))
return let;
if(isdigit(ch))
return dig;
return ch;
}
yyerror(char *s){
printf("%s",s);
}
main(){
printf("ENTER A variable : ");
yyparse();
}
```

OUTPUT:

```
 vi valid_vari.y
 yacc -d valid_vari.y
cc y.tab.c
./a.out
ENTER A variable : 57jkfg
syntax error rejected
./a.out
ENTER A variable : S754kh
accepted
./a.out
ENTER A variable : f8gl
Accepted
```

RESULT: Thus a YACC Program to recognize that a valid variable which starts with a letter followed by any number of letters or digits is implemented.

Ex. No. : 4(c)
Date:

IMPLEMENTATION OF CALCULATOR USING LEX & YACC

AIM:

To write a Program to implement Calculator using LEX and YACC.

ALGORITHM:

```
1) Start the Program.
2) Define the Rules, user-defined subroutines and definitions.
       {definitions}
       %%
       {rules}
       %%
       {user-defined subroutines}
3) yyparse() – implies parsing status
       if( yyparse()==0 )
           Parsing successful
       elseif( yyparse()==1 )
           Parsing failed due to invalid input
       else( yyparse()==2 )
           Parsing failed due to memory exhaustion
4) yylex() – implies the entry point for the lex and reads input to generate tokens.
       if( yylex()==0)
           End of input stream
5) yyerror() – it is called when YACC encounters invalid syntax.
6) yywrap() – implies the end of file.
       if( yywrap()==1 )
           End of file
7) yylval – values associated with the token are returned by lex in this variable.
8) yytext – points to first character of the return token.
6) Stop the Program.
```

PROGRAM:

cal.l

```
%{
#include <stdlib.h>
#include <stdio.h>
#include "y.tab.h"
void yyerror(char*);
extern int yylval;
%}
%%
[ \t]+ ;
[0-9]+ {yylval = atoi(yytext);
return INTEGER;}
[-+*/] {return *yytext;}
```

```
"(" {return *yytext;}
")" {return *yytext;}
\n {return *yytext;}
. {char msg[25];
sprintf(msg,"%s <%s>","invalid character",yytext);
yyerror(msg);
}
```

cal.y

```
%{
#include <stdlib.h>
#include <stdio.h>
int yylex(void);
#include "y.tab.h"
%}
%token INTEGER
%%
program:
line program
| line
line:
expr '\n' { printf("%d\n",$1); }
| 'n'
expr:
expr '+' mulex { $$ = $1 + $3; }
| expr '-' mulex { $$ = $1 - $3; }
| mulex { $$ = $1; }
mulex:
mulex '*' term { $$ = $1 * $3; }
| mulex '/' term { $$ = $1 / $3; }
| term { $$ = $1; }
term:
'(' expr ')' { $$ = $2; }
| INTEGER { $$ = $1; }
%%
void yyerror(char *s)
{
fprintf(stderr,"%s\n",s);
return;
}
yywrap()
{
return(1);
}
int main(void)
{
yyparse();
return 0;
}
```

OUTPUT:

```
vi cal.l
flex cal.l
yacc -d cal.y
gcc y.tab.c lex.yy.c
./a.out
1+2
3
76-67
9
4%3
invalid character <%>
syntax error
```

RESULT: Thus a Program for Calculator is implemented using LEX and YACC.

Ex. No. :5
Date: **ABSTRACT SYNTAX TREE**

AIM:

To write a program to Convert the BNF rules into Yacc form and write code to generate Abstract Syntax Tree.

ALGORITHM:

```
1. Start the program.
2. Declare the declarations as a header file. {include}
3. Token digit
4. Define the translations rule like line,expr,term,factor.
        Line: exp"\n"{print"\n%d\n",$1)}
        Expr: expr"+"term($$=$1=$3}
        Term: term"+"factor($$=$1*$3}
        Factor: FactorΛ"enter"),{$$=$2)
        %%
5. Define the supporting C routines.
6. Execute and verify it.
7. Stop the program.
```

TEST.C FILE:

```
main()
{
int a,b,c;
if(a<b)
{
a=a+b;}
while(a<b)
{
a=a+b;
}
if(a<=b)
{
c=a-b;
}
else
{
c=a+b;
}
}
```

CODE:

LEX FILE :

```
%{
#include"y.tab.h"
#include<stdio.h>
#include<string.h>
int LineNo=1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
return VAR;}
{number} {strcpy(yylval.var,yytext);
return NUM;}
\< |
\> |
\>= |
\<= |
== {strcpy(yylval.var,yytext);
return RELOP;}
[ \t] ;
\n LineNo++;
. return yytext[0];
%%
```

YACC FILE:

```
%{
#include<string.h>
#include<stdio.h>
void yyerror();
struct quad
{
char op[5];
char arg1[10];
char arg2[10];
char result[10];
}QUAD[30];
struct stack
{
int items[100];
int top;
}stk;
int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
%}
%union
{
char var[10];
}
```

```
%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE%type <var> EXPR ASSIGNMENT
CONDITION IFST ELSEST
WHILELOOP
%left '-' '+'
%left '*' '/'
%%
PROGRAM : MAIN BLOCK
;
BLOCK: '{' CODE '}'
;
CODE: BLOCK
| STATEMENT CODE
| STATEMENT
;
STATEMENT: DESCT ';'
| ASSIGNMENT ';'
| CONDST
| WHILEST
;
DESCT: TYPE VARLIST
;
VARLIST: VAR ',' VARLIST
| VAR
;
ASSIGNMENT: VAR '=' EXPR{
strcpy(QUAD[Index].op,"=");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,$1);
strcpy($$,QUAD[Index++].result);}
;
EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}
| EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);}
| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}
| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}
| '(' EXPR ')' {strcpy($$,$2);}
| VAR
| NUM
;
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
| IFST ELSEST
;
IFST: IF '(' CONDITION ')' {
```

```
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
};
ELSEST: ELSE{
tInd=pop();
Ind=pop();
push(tInd);
sprintf(QUAD[Ind].result,"%d",Index);
}
BLOCK{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
StNo=Index-1;
}
| VAR
| NUM
;
WHILEST: WHILELOOP{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",StNo);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
;WHILELOOP: WHILE '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
```

```
Index++;
}
;
%%
extern FILE *yyin;
int main(int argc,char *argv[])
{
FILE *fp;
int i;
if(argc>1)
{
fp=fopen(argv[1],"r");
if(!fp)
{
printf("\n File not found");exit(0);
}
yyin=fp;
}
yyparse();
printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator Arg1 Arg2

Result" "\n\t\t --------------------");
for(i=0;i<Index;i++)
{
printf("\n\t\t %d\t %s\t %s\t %s\t
%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD

[i].result);
}
printf("\n\t\t -----------------------");
printf("\n\n");
return 0;
}
void push(int data)
{
stk.top++;
if(stk.top==100)
{
printf("\n Stack overflow\n");
exit(0);
}
stk.items[stk.top]=data;
}
int pop()
{int data;
if(stk.top==-1)
{
printf("\n Stack underflow\n");
exit(0);
}
```

```
data=stk.items[stk.top--];
return data;
}
void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])
{
strcpy(QUAD[Index].op,op);
strcpy(QUAD[Index].arg1,arg1);
strcpy(QUAD[Index].arg2,arg2);
sprintf(QUAD[Index].result,"t%d",tIndex++);
strcpy(result,QUAD[Index++].result);
}

void yyerror()
{
printf("\n Error on line no= :%d",LineNo);
}
```

FRONT-END OF COMPILER:

```
#include<stdlib.h>
#include<iostream>
#include<string.h>
#include<stdio.h>
using namespace std;
char *strrev(char *str)
{
   int i = strlen(str) - 1, j = 0;

   char ch;
   while (i > j)
   {
      ch = str[i];
      str[i] = str[j];
      str[j] = ch;
      i--;
      j++;
   }
   return str;
}
void reverse(char str[], int length)
{
   int start = 0;
   int end = length -1;
   while (start < end)
   {
      swap(*(str+start), *(str+end));
      start++;
      end--;
   }
}
char* itoa(int num, char* str, int base)
```

```
{
    int i = 0;
    bool isNegative = false;

    /* Handle 0 explicitely, otherwise empty string is printed for 0 */
    if (num == 0)
    {
        str[i++] = '0';
        str[i] = '\0';
        return str;
    }

    // In standard itoa(), negative numbers are handled only with
    // base 10. Otherwise numbers are considered unsigned.
    if (num < 0 && base == 10)
    {
        isNegative = true;
        num = -num;
    }

    // Process individual digits
    while (num != 0)
    {
        int rem = num % base;
        str[i++] = (rem > 9)? (rem-10) + 'a' : rem + '0';
        num = num/base;
    }

    // If number is negative, append '-'
    if (isNegative)
        str[i++] = '-';

    str[i] = '\0'; // Append string terminator

    // Reverse the string
    reverse(str, i);

    return str;
}
struct tbl
{
char arg1[10],arg2[10],op[10],res[10];
}t[10];
char s[20][20];
int l;
int resol(int x, int j)
{
//x- line no. in i/p
int k=0,n=strlen(s[x])-1;
//resolving from left, first operation
```

```
t[j].op[k]=s[x][n-1];
t[j].arg1[k]=s[x][n-2];
t[j].arg2[k]=s[x][n];
itoa(l,t[j].res,10);
strcat(t[j].res,"t");
strrev(t[j].res);
l++;
cout<<endl<<"\t"<<j<<"\t"<<t[j].op<<"\t"<<t[j].arg1<<"\t"<<t[j].arg2<<"\t"<<t[j].res
;
j++;
//resolving the rest of the expression
for(int i=strlen(s[x])-4;i>0;i-=2)
{
t[j].op[k]=s[x][i];
if(t[j].op[k]=='=')
{
t[j].op[k]='=';
itoa(l-1,t[j].arg1,10);
strcat(t[j].arg1,"t");
strrev(t[j].arg1);
strcpy(t[j].arg2,"");
t[j].res[k]=s[x][i-1];
cout<<endl<<"\t"<<j<<"\t"<<t[j].op<<"\t"<<t[j].arg1<<"\t"<<t[j].arg2<<"\t"
<<t[j].res;
j++;
}
else
{
t[j].arg1[k]=s[x][i-1];
strcpy(t[j].arg2,t[j-1].res);
itoa(l,t[j].res,10);
strcat(t[j].res,"t");
strrev(t[j].res);
l++;cout<<endl<<"\t"<<j<<"\t"<<t[j].op<<"\t"<<t[j].arg1<<"\t("<<t[j].arg2<<")\t"<<t
[j].res;
j++;
}
}
return j;
}
void iff()
{
int i,j,k,n;
i=j=k=0;
//checking syntax
char *s1="if( )";
for(i=0;i<3;i++)
if(s1[i]!=s[0][i])
{
cout<<"Format Line 1\nTry Again!";
return;
}
if(s1[4]!=s[0][6]){
cout<<"Format Line 1\nTry Again!";
return;
}
char *s2="else";
if(strcmp(s2,s[2]))
{
cout<<"Format Line 3\nTry Again!";
return;
}
cout<<"\tPos\tOp\tArg1\tArg2\tRes";
//resolving if part
t[j].op[k]=s[0][4];
t[j].arg1[k]=s[0][3];
t[j].arg2[k]=s[0][5];
strcpy(t[j].res,"t0");
cout<<endl<<"\t"<<j<<"\t"<<t[j].op<<"\t"<<t[j].arg1<<"\t"<<t[j].arg2<<"\t"
<<t[j].res;
j++;
strcpy(t[j].op,"==");
strcpy(t[j].arg1,t[j-1].res);
strcpy(t[j].arg2,"FALSE");
int z=j+(int)((strlen(s[1])-1)/2+1);
itoa(z,t[j].res,10);
cout<<endl<<"\t"<<j<<"\t"<<t[j].op<<"\t"<<t[j].arg1<<"\t"<<t[j].arg2<<"\t"<<t[j].res
;
j++;
l=1; //temp reg no.
//resolving the true stmt
j=resol(1,j);
//resolving the false stmt
j=resol(3,j);
}
int main(){
fflush(stdin);
int i,j,k,c;
i=j=k=0;
cout<<"\n\tFront End of Compiler- 3-Addr-Code Generator for Conditional If";
cout<<"\n\t\t\tQuadruple Implementation\n";
cout<<"\nEnter stmts, to terminate enter ~ in a new line\n";
do{
gets(s[i]);
i++;
}while(strcmp(s[i-1],"~"));
iff();
cout<<"\nThe End";
return 0;
}
```

SAMPLE INPUT AND OUTPUT:

```
$lex int.l
$yacc –d int.y
$gcc lex.yy.c y.tab.c –ll –lm
$./a.out test.c
                    ---------------------------
                     Pos Operator Arg1 Arg2 Result
                     -------------------
                     0     <     a     b     t0
                     1     ==    t0    FALSE       5
                     2     +     a     b     t1
                     3     =     t1          a
                     4     GOTO                    5
                     5     <     a     b     t2
                     6     ==    t2    FALSE       10
                     7     +     a     b     t3
                     8     =     t3          a
                     9     GOTO                    5
                     10    <=    a     b     t4
                     11    ==    t4    FALSE       15
                     12    -     a     b     t5
                     13    =     t5          c
                     14    GOTO                    17
                     15    +     a     b     t6
                     16    =     t6          c
                     ----------------------
```

SAMPLE INPUT AND OUTPUT:

```
          Front End of Compiler- 3-Addr-Code Generator for Conditional If
                            Quadruple Implementation
Enter stmts, to terminate enter ~ in a new line
if(a>b)
b=c/d+e*f
else
c=g+h*9/e
~
          Pos   Op    Arg1  Arg2  Res
          0     >     a     b     t0
          1     ==    t0    FALSE       6
          2     *     e     f     t1
          3     +     d     (t1)  t2
          4     /     c     (t2)  t3
          5     =     t3          b
          6     /     9     e     t4
          7     *     h     (t4)  t5
          8     +     g     (t5)  t6
          9     =     t6          c
```

RESULT: Thus, a program to convert the BNF rules into Yacc form and write code to generate Abstract Syntax Tree is written and executed.

Ex. No. :6

Date: **TYPE CHECKING**

AIM:

To write a program to implement type checking.

ALGORITHM:

1. Define symbol table with following attributes <symbol name, type>
2. Get the expression as the input from the user. It should be in the form Operand1 <operator> Operand2
3. Read the operand1 and retrieve the corresponding Type information from the symbol table.
4. Read the operand2 and retrieve the corresponding Type information from the symbol table.
5. If type of both operands are same then print the Type as output
6. Otherwise print "TYPE MISMATCH"

CODE:

LEX FILE :

```
%{
#include<stdio.h>
#include"y.tab.h"
void yyerror(char *);
int yylval;
int yylineno;
%}
num [0-9]
intg {num}{num}*
floa {num}{num}*.({num}{1,2})
doub {num}{num}*.({num}{3,8})
%%
{intg} { return INTEGER; }
{floa} { return FLOAT; }
{doub} { return DOUBLE; }
[+=\n] return *yytext;
[)( \t] {;}
%%
 int yywrap()
{
return 1;
}
```

YACC FILE:

```
%{#include<stdlib.h>
#include<stdio.h>
#include<math.h>
int yylex(void);
void yyerror(char *);
int flag=0;
%}
%left '='
%left '+'
```

```
%left '('
%token INTEGER
%token FLOAT
%token DOUBLE
%%
program:
program E '\n' {
if(flag==0)
  {
            switch($2)
            {
            case 1: printf("Integer\n"); break;
            case 2: printf("Float\n"); break;
            case 3: printf("Double\n"); break;
            case 4: printf("Boolean\n"); break;
            }
   }
}
|
;
E:
INTEGER { $$=1; }
|
FLOAT { $$=2; }
|
DOUBLE { $$=3; }
|
E '+' E {
if($1==$3)
 $$=$1;
else
       {
            if(($1==1 && $3==2) || ($1==2 && $3==1))
            $$=2;
            else if(($1==3 && $3==2) || ($1==2 && $3==3))
             $$=3;
            else
            {
            printf("ERROR: Type mismatch\n");
            flag=1;
            }
        }
}
|
E '=' '=' E { if($1==$4) $$=4;
else
{
            printf("ERROR: Type mismatch\n");
flag=1;
  }
```

```
        }
        |
        '(' E ')' { $$=$2; }
        |
        ;
        %%
        void yyerror(char *s)
                {
                        fprintf(stderr,"%s\n",s);
                }
        int main(void)
                {
                        yyparse();
                        return 0;
                }
```

<u>SAMPLE INPUT-OUTPUT:</u>

```
./o
6
Integer
8
Integer
7.9
Float
9+
ERROR: Type mismatch
```

<u>RESULT:</u>

Thus a program to implement type checking was written and executed.

Ex. No. :7
Date:

Control Flow Graph

Aim: To construct a flow graph and perform flow analysis of different blocks.

Algorithm:

A control flow graph is a directed graph in which each node represents a basic block and each edge represents the flow of control between basic blocks.
A basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end.
We can build a CFG by first building basic blocks and then by adding edges that represent control flow between basic blocks.

1 . Start
1. Determine the set of leaders, the first statements and basic blocks. The following rules can be used.
    a. The first statement in the program is the leader.
    b. Any statement that is target of a conditional or unconditional goto statement is a leader.
    c. Any statement immediately follows a conditional or unconditional goto statement is a leader.
2. Construct the basic block using the leaders
3. Create entry and exit nodes.Create edge (Entry B) for each basic block Bk that has an exit.
4. Traverse the list of basic blocks and add CFG edge from each node Bi to each node Bj in the same execution sentence,
5. Stop.

Program:

```
#include<stdio.h>
struct variable
{int count;
int num[5];
char tac[5][5];
int gen[10];
int kill[10];
int in[7];
int out[7];
int prev_out[7];
}var[10];
int main()
{
int n,i,d_no=1,k,j,l,in_no,adj[6][6],a,b,l_no,flag=0,pass=2;
printf("\nEnter the number of blocks:");
scanf("%d",&n);
for(i=0;i<n;i++)
```

```
{
printf("\nEnter the number of equations in block %d:",i+1);
scanf("%d",&var[i].count);
for(k=0;k<var[i].count;k++)
{
var[i].num[k]=d_no;
d_no++;
printf("\nEnter the tac sequence %d:",k+1);
scanf("%s",var[i].tac[k]);
}
}
for(i=0;i<n;i++)
{
for(k=0;k<d_no-1;k++)
{
var[i].gen[k]=0;
var[i].in[k]=0;
var[i].out[k]=0;
var[i].prev_out[k]=0;
}
for(k=0;k<var[i].count;k++)
{
var[i].gen[var[i].num[k]-1]=1;
}
}
printf("\nGEN SET");
for(i=0;i<n;i++)
{
printf("\n");
for(k=0;k<d_no-1;k++)
printf("%d",var[i].gen[k]);
}
for(i=0;i<n;i++)
{
for(k=0;k<d_no-1;k++)
var[i].kill[k]=0;
for(k=0;k<var[i].count;k++)
{
for(j=0;j<n;j++)
{
if(j!=i)
{for(l=0;l<var[j].count;l++){
if(var[i].tac[k][0]==var[j].tac[l][0])
{var[i].kill[var[j].num[l]-1]=1;
}}}}}
}
printf("\nKILL SET");
for(i=0;i<n;i++)
{printf("\n");
for(k=0;k<d_no-1;k++)
```

```
printf("%d",var[i].kill[k]);}
for(i=0;i<n;i++)
{for(j=0;j<n;j++)
adj[i][j]=0; }
printf("\nEnter the number of links:");
scanf("%d",&l_no);
for(i=0;i<l_no;i++)
{printf("\nEnter the link %d:",i+1);
scanf("%d%d",&a,&b);
adj[a-1][b-1]=1; }
printf("\nAdjacency Matrix");
for(i=0;i<n;i++)
{
printf("\n");
for(j=0;j<n;j++)
printf("%d",adj[i][j]);
}
for(i=0;i<n;i++)
{
if(adj[i][0]==1)
{
for(j=0;j<d_no-1;j++)
var[0].in[j]=var[i].gen[j] | var[0].in[j];
}
}
for(i=0;i<d_no-1;i++)
{
a=var[0].in[i] | var[0].gen[i];
b=var[0].in[i] & var[0].kill[i];
var[0].out[i]=a-b;
}
/*printf("\nIn of first block:");
for(i=0;i<d_no-1;i++)
{
printf("%d",var[0].in[i]);
}
printf("\nOut of first block:");
for(i=0;i<d_no-1;i++)
{
printf("%d",var[0].out[i]);
}*/
for(i=1;i<n;i++)
{
for(k=0;k<n;k++)
{
if(adj[k][i]==1)
{
for(j=0;j<d_no-1;j++)
var[i].in[j]=var[k].out[j] | var[i].in[j];
}
```

```
}
for(j=0;j<d_no-1;j++)
{
a=var[i].in[j] | var[i].gen[j];
b=var[i].in[j] & var[i].kill[j];
var[i].out[j]=a-b;
}
}
for(i=0;i<n;i++)
{
printf("\n");
for(j=0;j<d_no-1;j++)
printf("%d",var[i].in[j]);
printf("\t");
for(j=0;j<d_no-1;j++)
printf("%d",var[i].out[j]);
}
for(i=0;i<n;i++)
{
for(j=0;j<d_no-1;j++)
var[i].prev_out[j]=var[i].out[j];
}
while(flag==0)
{
for(i=0;i<n;i++)
{
for(k=0;k<n;k++)
{
if(adj[k][i]==1)
{
for(j=0;j<d_no-1;j++)
var[i].in[j]=var[k].prev_out[j] |
var[i].in[j];
}
}
for(j=0;j<d_no-1;j++)
{
a=var[i].in[j] | var[i].gen[j];
b=var[i].in[j] & var[i].kill[j];
var[i].out[j]=a-b;
}
}
printf("\nPass:%d",pass++);
for(i=0;i<n;i++)
{printf("\n");
for(j=0;j<d_no-1;j++)
printf("%d",var[i].in[j]);
printf("\t");
for(j=0;j<d_no-1;j++)
printf("%d",var[i].out[j]);
```

```
}
flag=1;
for(i=0;i<n;i++)
{
for(j=0;j<d_no-1;j++)
{
if(var[i].out[j]!=var[i].prev_out[j])
{
break; }}}
for(i=0;i<n;i++)
{ for(j=0;j<d_no-1;j++)
var[i].prev_out[j]=var[i].out[j]; }
for(i=0;i<n;i++)
{
for(j=0;j<d_no-1;j++)
{
var[i].in[j]=0;
var[i].out[j]=0; }
}}
return 0;
}
```

Sample Input-Output:

```
gcc cfg.c
./a.out

Enter the number of blocks:5
Enter the number of equations in block 1:2
Enter the tac sequence 1:i=2
Enter the tac sequence 2:i+1
Enter the number of equations in block 2:1
Enter the tac sequence 1:i=1
Enter the number of equations in block 3:1
Enter the tac sequence 1:j=j+1
Enter the number of equations in block 4:1
Enter the tac sequence 1:j=j-4
Enter the number of equations in block 5:0




GEN SET
11000
00100
00010
00001
00000
KILL SET
00100
11000
```

```
00001
00010
00000
Enter the number of links:7

Enter the link 1:1 2

Enter the link 2:2 3

Enter the link 3:2 1
Enter the link 4:3 4
Enter the link 5:3 5
Enter the link 6:4 5
Enter the link 7:5 2
Adjacency Matrix
01000
10100
00011
00001
01000
00100  11000
11000  00100
00100  00110
00110  00101
00111  00111
Pass:2
00100  11000
11111  00111
00100  00110
00110  00101
00111  001111
```

Result:
Thus a flow graph was constructed and flow analysis of different blocks was analyzed.

Ex. No. :8
Date:

Storage allocation strategies-stack

Aim:

To Implement stack storage allocation strategy.

Algorithm:

1. Start the program.
2. Get user input to push or pop.
3. If push, get the element value to be pushed.
4. Push the value in the stack and update top.
5. If pop, display top element, remove the top most element and update top.
6. End the program.

Program:

```
#include<malloc.h>
#include<stdio.h>
int top;
struct node{
 int data;
 struct node *next;
 struct node *prev;
};
int main()
{
 int choice,n;
 char choice2;
 printf("Enter the value");
 scanf("%d",&n);
 struct node *curr;
 struct node *curr2;
 struct node *new;
 struct node *top;
 new=(struct node*)malloc(sizeof(struct node));
 curr=new;
 curr->data=n;
 top=curr;
 curr->prev=NULL;
do{
 printf("Do you want to push or pop(1/2)?");
 scanf("%d",&choice);
 switch(choice) {
 case 1: printf("Enter the value");
 scanf("%d",&n);
 new=(struct node*)malloc(sizeof(struct node));
 new->data=n;
 new->prev=curr;
 curr=new;
 top=curr;
 printf("The no. pushed inside is %d\n",n);
 break;
 case 2: printf("The no. popped out is %d\n",top->data);
 top=top->prev;
 free(curr);
 curr=top;
 break;
 }
 curr2=top;
 while(curr2!=NULL) {
 printf("%d",curr2->data);
 if(curr2->prev!=NULL) {
 printf("->");
 }
 curr2=curr2->prev;
 }
 printf("Do you want to continue?(y/n)");
 scanf("%c%c",&choice2,&choice2);
 }while(choice2=='y');
return 0;
}
```

Sample Input – Output:

```
vi storage.c
cc storage.c
./a.out
Enter the value 5
Do you want to push or pop(1/2)?1
Enter the value 6
The no. Pushed inside is 6
6->5 Do you want to continue(y/n)?y
Do you want to push or pop(1/2)?1
Enter the value 4
The no. Pushed inside is 4
4->6->5 Do you want to continue?(y/n)y
Do you want to push or pop(1/2)?1
Enter the value 9
The no. Pushed inside is 9
9->4->6->5 Do you want to continue?(y/n)y
Do you want to push or pop(1/2)?2
The no. Popped out is 9
4->6->5 DO you want to continue?(y/n)n
```

Result:

To Implement stack storage allocation strategy.

Ex. No. :9
Date:

DAG construction

Aim:

To implement construction of DAG.

Algorithm:

1. If node(y) is undefined, create a leaf labeled y, and let node(y) be this node. In case (i), if node(z) is undefined, create a leaf labeled z and let that leaf be node(z).In case(i) determine if there is a node labeled op, whose left child is node(y) and whose right child is node(z).
2. If not, create such node. In either event, let n be the node found or created. In case (ii), determine whether there is node labeled op, whose lone child is node(y). If not, create such a node, and let n be the node found or created. In case (ii), let n be node(y).
3. Delete x from the list of attached identifiers for node(x). Append x to the list of attached identifiers for the node n found in (2) and set node(x) to n.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct
{
 char op;
 char lc[5],rc[5];
 char labels[5][5];
 int p1,p2,count;
}node;
int main()
{
 char ar[10][5],str[20];
 int i=-1,j=0,k,l,f1,f2,n;
 FILE *fp;
 fp=fopen("stmt.txt","r");
 node stmt[10];
 while(fgets(str,20,fp)!=NULL)
 {
 i++;
 stmt[i].count=0;
 stmt[i].p1=-1;
 stmt[i].p2=-1;
 stmt[i].op='\0';
 if(strstr(str,"+")) stmt[i].op='+';
 else if(strstr(str,"*")) stmt[i].op='*';
 else if(strstr(str,"-")) stmt[i].op='-';
 else if(strstr(str,"/")) stmt[i].op='/';
 else if(strstr(str,"[")) stmt[i].op='[';
```

```
strcpy(stmt[i].labels[0],strtok(str,"="));
stmt[i].count++;
strcpy(stmt[i].lc,strtok(NULL,"+*-/%[ \n\0"));
if(stmt[i].op!='\0') strcpy(stmt[i].rc,strtok(NULL,"]\n\0"));
else strcpy(stmt[i].rc,"\0");
f1=0;
if(strcmp(stmt[i].rc,"\0")==0)
{
for(k=i-1;k>=0;k--)
{
for(l=0;l<stmt[k].count;l++)
{
if(strcmp(stmt[i].lc,stmt[k].labels[l])==0)
{
strcpy(stmt[k].labels[stmt[k].count],stmt[i].labels[0]);
stmt[k].count++;
i--;
f1=1;
break;
}
}
if(f1==1) break;
}
}
if(f1==1) continue;
f2=0;
for(k=i-1;k>=0;k--)
{
for(l=0;l<stmt[k].count;l++)
{
if(strcmp(stmt[i].lc,stmt[k].labels[l])==0)
{
stmt[i].p1=k;
f2=1;
break;
}
}
if(f2==0)
if(strcmp(stmt[i].lc,stmt[k].lc)==0 || strcmp(stmt[i].lc,stmt[k].rc)==0)
{
stmt[i].p1=k;
break;
}
}
f2=0;
if(strcmp(stmt[i].rc,"\0")!=0)
for(k=i-1;k>=0;k--)
{
for(l=0;l<stmt[k].count;l++)
{
 if(strcmp(stmt[i].rc,stmt[k].labels[l])==0)
 {
 stmt[i].p2=k;
 f2=1;
 break;
 }
 }
 if(f2==0)
 if(strcmp(stmt[i].rc,stmt[k].lc)==0 || strcmp(stmt[i].rc,stmt[k].rc)==0)
 {
 stmt[i].p2=k;
 break;
 }
 }
 if(stmt[i].p1==stmt[i].p2)
 {
 n=stmt[i].p1;
 if(stmt[i].op==stmt[n].op)
 {
 strcpy(stmt[n].labels[stmt[n].count],stmt[i].labels[0]);
 stmt[n].count++;
 i--;
 }
 }
 }
 for(k=0;k<=i;k++)
 {
 if(stmt[k].op=='[')
 printf("\n\nAddress %d:: Node [] is created with %s",&stmt[k],stmt[k].lc);
 else
printf("\n\nAddress %d:: Node %c is created with %s",&stmt[k],stmt[k].op,stmt[k].lc);
if(stmt[k].p1!=-1)
 printf(" from address %d",&stmt[stmt[k].p1]);
 printf(" as left child and %s",stmt[k].rc);
 if(stmt[k].p2!=-1)
 printf(" from address %d",&stmt[stmt[k].p2]);
 printf(" as right child with labels");
 for(l=0;l<stmt[k].count;l++)
 printf(" %s,",stmt[k].labels[l]);
 }
 printf("\n");
}
```

<u>stmt.txt</u>

```
t1=4*i
t2=a[t1]
t3=4*i
t4=b[t3]
t5=t2*t4
t6=prod+t5
prod=t6
t7=i+1
i=t7
```

<u>Sample Input-Output</u>

```
vi dag.c
vi stmt.txt
cc dag.c
./a.out
Address -1088868848:: Node * is created with 4 as left child and i as right child
with labels t1,t3,
Address -1088868800:: Node [] is created with a as left child and t1 from address -
1088868800 as right child with labels t2,
Address -1088868752 :: Node [] is created with b as left child and t3 from address
-1088868800 as right child with labels t4,
Address -1088868704 :: Node * is created with t2 from address -10888868000 as
left child ad t4 from address -1088868752 as right child with labels t6,prod,
Address -1088868656 :: Node + is created with prod as left child and t5 from
address -1088868704 as right child with labels t6,prod,
Address -1088868608 :: Node + is created with i from address -1088868848 as left
child and 1 as right child with labels t7,i,
```

<u>Result:</u>Thus the construction of DAG was implemented.

Ex. No. :10
Date:

Back end of compiler

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 8086 assembler.

Algorithm:

For each three address code A = B op C in the given basic block, do the following

1. Invoke the function GETREG () to determine the location L where the computation B op C should be performed. L may be either register or memory location

2. Consult the address descriptor for B to determine B. Prefer register for B', if the value of B is currently in memory and in register. If the value of B is not in L, generate the instruction MOV B,L to place a copy of B in L

3. Generate the instruction OP C , L where C is the current location of C. Update the address descriptor of A to " A in L". If L is a register, update its descriptor to "L contains A"

4. If the current values of B and/or C have no next uses or not live on exit from the block and are in registers, then update the register descriptors that, those registers no longer will contain B and/or C.

GETREG function:

The GETREG function returns the location L to hold the value of A for the assignment A = B op C

1. If the name B is in a register that holds the value of no other names and B has no next use and not live after the computation A = B op C, then return the registers of B for L. Update the address descriptor of B as "B is no longer in L"

2. Failing step (1), return an empty register for L if there is one.
3. Failing step (2), if A has a next-use in the block, or op is an operator such as indexing that requires register to perform the operation, then find an occupied register R. Store the value of R into a memory location if it is not already in the proper memory location M. Update the address descriptor for M and return R.

4. If A is not used in the block, select the memory location of A as L.

Program :

```
#include<stdio.h>
#include<conio.h>
struct regis
{
char var;
}reg[10];
int noreg;
char st[10][10];
char *opcode[10]={"add","sub","mul","div","assign"};
char oper[10]={'+','-','*','/','='};
char *toopcode(char opert)
{
int i;
for(i=0;i<5;i++)
{
if(oper[i]==opert)
{
return(opcode[i]);
}}
return(0);
}
int isinregister(char var)
{
int i;
for(i=1;i<=noreg;i++)
{
if(var==reg[i].var)
{
return(i);
}}
return(0);
}
void main()
{
int i,regno2=0,regno1=1,nost;
clrscr();
printf("\nCODE GENERATION");
printf("\n");
printf("Enter the no of statements:");
scanf("%d",&nost);
printf("Enter the statements:");
printf("\n");
for(i=0;i<nost;i++)
{
scanf("%s",st[i]);
}
printf("\n\tStatements\tTarget code");
for(i=0;i<nost;i++)
{
printf("\n\n\t%s",st[i]);
if(!regno1==isinregister(st[i][2]))
{
printf("\n\t\t\tmov %c,R%d",st[i][2],++noreg);
reg[noreg].var=st[i][2];
regno1=noreg;
}
if(regno2!=isinregister(st[i][4]))
{
printf("\n\t\t\t%s R%d,R%d",toopcode(st[i][3]),regno2,regno1);
reg[regno1].var=st[i][0];
}
else
{
printf("\n\t\t\t%s %c,R%d",toopcode(st[i][3]),st[i][4],regno1);
reg[regno1].var=st[i][0];
}
if(i==nost-1)
{
printf("\n\t\t\tmov R%d,%c",regno1,st[i][0]);
}}
getch();
}
```

Sample Input - Output:

```
Enter the no of statements:2
Enter the statements:
a=b+c
d=a*k
Statements Target code
a=b+c
mov b,R1
add c,R1
d=a*k
mul k,R1
mov R1,d
```

Result:

Thus 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 8086 assembler was implemented.

Ex. No. :11
Date:

Code Optimization

Aim:
To perform code optimization of entered input code and display the optimized code.

Algorithm:

1. Start
2. Scan through the entered code to see where optimization can be done
3. In case where redundant or repeated code is noticed, perform dead code removal.
4. In compile time arithmetic operations, substitute the value for the variables.
5. In places where multiplication by 2 is performed, substitute by left shift operator.
6. In looping statements, instead of loop check if repetition is less complex and append accordingly.
7. Display the optimized code output.
8. Stop

Program:

```
#include<bits/stdc++.h>
#define ll long long
#define ld long double
#define pb push_back
#define x first
#define y second
#define fastread ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);
#define PI (atan(1)*4)
#define mp make_pair
using namespace std;
int ptr;
map<string,int> m;
int mapper(string var){
    int val=m[var];
    if(val!=0)
        return val;
    m[var]=++ptr;
    return ptr;
}
//global declarations
const int maxn=1e3;
string ip;
vector<string> lineList;
vector< vector<int>> varList;
set< int > lastsaved[maxn];
int vis[maxn];
vector<int> adjlist[maxn];
//end global declarations
```

```
bool read(){
        ip="";
        while(ip.size()==0)
                getline(cin,ip);
        return !(ip=="$");
}

bool nonVar(char c){
        if(c>='0' && c<='9')
                return false;
        if(c=='+' || c=='-' || c=='/' || c=='*' || c=='=')
                return false;
        return true;
}

vector<int> getVariables(string s){
        int lptr=0,rptr=0;
        vector<int> seq;
        while(rptr!=s.size()){
                if(nonVar(s[rptr])){
                        rptr++;
                }
                else{
                        if(lptr!=rptr){
                                string temp=s.substr(lptr,rptr-lptr);
                                int id=mapper(temp);
                                seq.pb(id);
                        }
                        rptr++;
                        lptr=rptr;
                }
        }
        if(lptr<rptr){
                string temp=s.substr(lptr,rptr-lptr);
                int id=mapper(temp);
                seq.pb(id);
        }
        return seq;
}

void dfs(int cur){
        vis[cur]=1;
        for(int i=1;i<varList[cur].size();i++){
                int var=varList[cur][i];
                set<int>::iterator it=lastsaved[var].lower_bound(cur);
                if(it==lastsaved[var].begin())
                        continue;
                it--;
                if(vis[*it]==0){
                        vis[*it]=1;
```

```
				dfs(*it);
			}
		}
}
int main()
{
		fastread;
		cout<<"DEAD CODE OPTIMISATION"<<endl<<endl;
		while(read()){
			if((ip.find("=")==string::npos) && ip.find("print(")==string::npos)
				continue;
			while(ip.find(" ")!=string::npos)
				ip.erase(std::find(ip.begin(), ip.end(), ' '));
			lineList.pb(ip);
		}
		for(int i=0;i<lineList.size();i++){
			string u=lineList[i];
			if(u.find("print(")!=string::npos){
				int en=6,st=6;
				while(u[en]!=')')
					en++;
				int prevptr=ptr;
				int id=mapper(u.substr(st,en-st));
				int curptr=ptr;
				if(prevptr!=curptr){
					cout<<u.substr(st,en-st)<<" not used"<<endl;
					return 0;
				}
				set<int>::iterator it=lastsaved[id].end();
				if(!lastsaved[id].empty()){
					it--;
					dfs(*it);
				}
				vis[i]=1;
				vector<int> temp;
				varList.pb(temp);
				continue;
			}
			varList.pb(getVariables(u));
			lastsaved[varList.back()[0]].insert(i);
			for(int j=1;j<varList.back().size();j++){
				set<int>::iterator it=lastsaved[varList.back()[j]].end();
				if(!lastsaved[varList.back()[j]].empty()){
					it--;
					adjlist[i].pb(*it);
				}
			}
		}
		cout<<"Optimised Code"<<endl;
		for(int i=0;i<lineList.size();i++){
```

```
                    if(vis[i])
                              cout<<lineList[i]<<'\n';
          }
          return 0;
}
```

Sample Input-Output:

```
a=5
b=100
c=b*b
a=c+b
b=500
d=20
d=a+a
print(d)
$

Optimised Code

b=100
c=b*b
a=c+b
d=a+a
print(d)
```

Result:

Thus the code optimization of entered input code was performed and the optimized code was displayed.