



JEPPIAAR INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)

“Self-Belief | Self-Discipline | Self-Respect”

Kunnam, Sunguvarchatram, Sriperumbudur – 631 604.



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS3501 – COMPILER DESIGN LABORATORY

NAME :

REG NO :

YEAR : III CSE

SEMESTER : V

**JEPPIAAR INSTITUTE OF TECHNOLOGY****(An Autonomous Institution)****“Self-Belief | Self-Discipline | Self-Respect”**

Kunnam, Sunguvarchatram, Sriperumbudur – 631 604.

**BONAFIDE CERTIFICATE**

This is a certified Bonafide Record Work of Mr./Ms. _____

Register No. _____ submitted for the End Semester Practical Examination held on _____ in CS3501- Compiler Design Laboratory during the year 2025 -2026.

Signature of the Lab In-charge

Head of the Department

Internal Examiner**External Examiner**

Date _____





INSTITUTE VISION

Jeppiaar Institute of Technology aspires to provide technical education in futuristic technologies with the perspective of innovative, industrial and social application for the betterment of humanity.

INSTITUTE MISSION

IM1: To produce competent and disciplined high-quality professionals with the practical skills necessary to excel as innovative professionals and entrepreneurs for the benefit of the society.

IM2: To improve the quality of education through excellence in teaching and learning, research, leadership and by promoting the principles of scientific analysis, and creative thinking.

IM3: To provide excellent infrastructure, serene and stimulating environment that is most conducive to learning.

IM4: To strive for productive partnership between the Industry and the Institute for research and development in the emerging fields and creating opportunities for employability.

IM5: To serve the global community by instilling ethics, values and life skills among the students needed to enrich their lives.

DEPARTMENT VISION

To impart futuristic technological education, innovation and collaborative research in the field of Computer Science and Engineering and to develop Quality Professionals for the improvement of the society and industry.

DEPARTMENT MISSION

DM1: Develop the students as professionally competent and disciplined engineers for the benefit of the development of the country.

DM2: Produce excellent infrastructure to adopt latest technologies, industry-institute interaction and encouraging research activities.

DM3: Provide multidisciplinary technical skills to pursue research activities, higher studies, entrepreneurship and perpetual learning.

DM4: Enrich students with professional integrity and ethical standards to handle social challenges successfully in their life.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates can

PEO1: Graduates can able to apply their technical competence in computer science to solve real world problems, with technical and people leadership.

PEO2: Graduates can able to conduct cutting edge research and develop solutions on problems of social relevance.

PEO3: Graduates can able to Work in a business environment, exhibiting team skills, work ethics, adaptability and lifelong learning.

PROGRAM SPECIFIC OUTCOMES

The Students will be able to

PSO1: Exhibit design and programming skills to build and automate business solutions using cutting edge technologies.

PSO2: Strong theoretical foundation leading to excellence and excitement towards research, to provide elegant solutions to complex problems.

PSO3: Ability to work effectively with various engineering fields as a team to design, build and develop system applications.

PROGRAM OUTCOMES

Engineering Graduates will be able to:

1. Engineering knowledge: (K3) Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: (K4) Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: (K4) Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: (K5) Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: (K3, K5, K6) Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: (A3) Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: (A2) Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: (A3) Apply ethical principles and commit to professional ethics and responsibilities and

norms of the engineering practice.

Individual and team work: (A3) Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

9. Communication: (A3) Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

10. Project management and finance: (A3) Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

11. Life-long learning: (A2) Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.



ANNA UNIVERSITY SYLLABUS**CS3501****COMPILER DESIGN****L T P C****3 0 2 1****COURSE OBJECTIVES:**

- To learn the various phases of compiler.
- To learn the various parsing techniques.
- To understand intermediate code generation and run-time environment.
- To learn to implement the front-end of the compiler.
- To learn to implement code generator.
- To learn to implement code optimization.

TOTAL: 45 PERIODS**COURSE OUTCOMES:**

At the end of this course, the students will be able to:

CO1: Understand the techniques in different phases of a compiler.

CO2: Design a lexical analyzer for a sample language and learn to use the LEX tool.

CO3: Apply different parsing algorithms to develop a parser and learn to use YACC tool

CO4: Understand semantics rules (SDT), intermediate code generation and run-time environment

CO5: Implement code generation and apply code optimization techniques.

CS3501 COMPILER DESIGN LABORATORY**(REGULATIONS-2021)****List of Experiments**

S.No	Name of the Experiment	Page no.
1	Implementation of symbol table.	
2	Develop a lexical analyzer to recognize a few patterns in c (ex. Identifiers, constants, comments, operators etc.)	
3	Implementation of lexical analyzer using lex tool.	
4	Generate yacc specification for a few syntactic categories. a) Program to recognize a valid arithmetic expression that uses operator +, -, * and /. b) Program to recognize a valid variable which starts with a letter followed by any number of letter or digits. c) Implementation of calculator using lex and yacc.	
5	Convert the bnf rules into yacc form and write code to generate abstract syntax tree.	
6	Implement type checking	
7	Implement control flow analysis and data flow analysis.	
8	Implement any one storage allocation strategies(heap, stack, static)	
9	Construction of DAG	
10	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.	
11	Implementation of simple code optimization techniques (constant folding. etc.)	

CS 3501-COMPILER DESIGN LABORATORY**Name of the student:****Roll no.:****INDEX**

S.No.	Date of Submission	Title of the Experiment	Marks	Signature

Signature of the Staff

COURSE OUTCOMES

Course Outcome No.	Course Outcome	Highest Cognitive Level
CO311.1	Understand the techniques in different phases of a compiler.	K3
CO311.2	Understand the techniques in different phases of a compiler.	K3
CO311.3	Understand the techniques in different phases of a compiler.	K3
CO311.4	Understand the techniques in different phases of a compiler.	K3
CO311.5	Implement code generation and apply code optimization techniques.	K3

CO's-PO's & PSO's MAPPING

CO's	PO's												PSO's		
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
1	.	2	3	.	.
2	.	1	.	.	2	2	.	2	.
3	.	2	.	.	3	3	.
4	.	.	.	1	2	3
5	.	3	2	3
AVg.	.	1	.	.	1	1	.	.	.	1	1

1 - low, 2 - medium, 3 - high, . - no correlation

Ex.No : 1**Date :****Implementation of symbol table.****Aim:**

To write a program for implementing Symbol Table using C.

Algorithm:

Step 1: Start the program for performing insert, display, delete, search and modify option in symbol table

Step 2: Define the structure of the Symbol Table

Step 3: Enter the choice for performing the operations in the symbol Table

Step 4: If the entered choice is 1, search the symbol table for the symbol to be inserted. If the symbol is already present, it displays “Duplicate Symbol”. Else, insert the symbol and the corresponding address in the symbol table.

Step 5: If the entered choice is 2, the symbols present in the symbol table are displayed.

Step 6: If the entered choice is 3, the symbol to be deleted is searched in the symbol table.

Step 7: If it is not found in the symbol table it displays “Label Not found”. Else, the symbol is deleted.

Step 8: If the entered choice is 5, the symbol to be modified is searched in the symbol table.

Program:

```
//Implementation of symbol table
```

```
#include<stdio.h>
```

```
#include<ctype.h>
```

```
#include<stdlib.h>
```

```
#include<string.h>
```

```
#include<math.h>
```

```
void main()
```

```
{
```

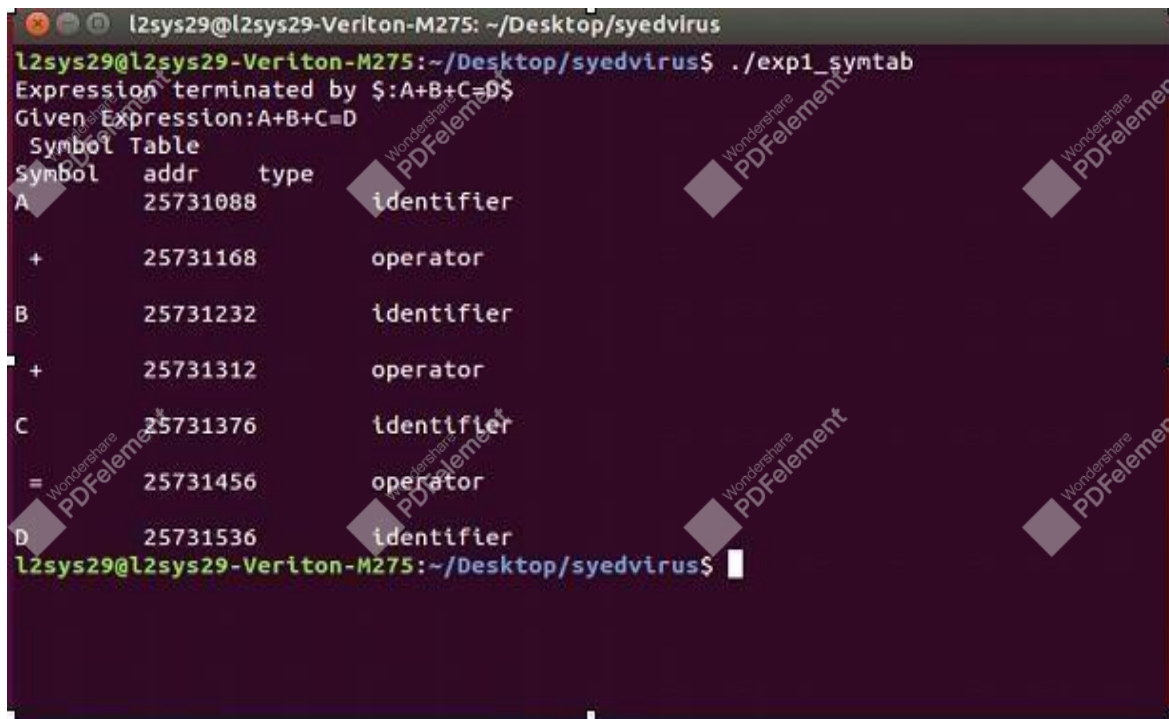
```
int i=0,j=0,x=0,n;
```

```
void *p,*add[5];
```

```
char ch,srch,b[15],d[15],c;
printf("Expression terminated by $.");
while((c=getchar())!='$')
{
b[i]=c;
i++;
}
n=i-1;
printf("Given Expression:");
i=0;
while(i<=n)
{
printf("%c",b[i]);
i++;
}
printf("\n Symbol Table\n");
printf("Symbol \t addr \t type");
while(j<=n)
{
c=b[j];
if(isalpha(toascii(c)))
{
p=malloc(c);
add[x]=p;
d[x]=c;
printf("\n%c \t %d \t identifier\n",c,p);
x++;
j++;
}
else
{
ch=c;
if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
{
p=malloc(ch);
```

```
add[x]=p;  
d[x]=ch;  
printf("\n %c \t %d \t operator\n",ch,p);  
x++;  
j++;  
}}}}
```

OUTPUT:



```
l2sys29@l2sys29-Veriton-M275: ~/Desktop/syedvirus  
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./exp1_symtab  
Expression terminated by $:A+B+C=D$  
Given Expression:A+B+C=D  
Symbol Table  
Symbol  addr    type  
A        25731088 identifier  
+        25731168 operator  
B        25731232 identifier  
+        25731312 operator  
C        25731376 identifier  
=        25731456 operator  
D        25731536 identifier  
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$
```

Result:

Thus, the program for symbol table has been executed successfully.

Ex.No : 2	Develop a lexical analyzer to recognize a few patterns in c (ex. Identifiers, constants, comments, operators etc.)
Date :	

Aim:

To develop a lexical analyzer to identify identifiers, constants, comments, operators etc using C program.

Algorithm:

Step 1: Start the program.

Step 2: Declare all the variables and file pointers.

Step 3: Display the input program.

Step 4: Separate the keyword in the program and display it.

Step 5: Display the header files of the input program

Step 6: Separate the operators of the input program and display it.

Step 7: Print the punctuation marks.

Step 8: Print the constant that are present in input program.

Step 9: Print the identifiers of the input program.

Program:

Develop a lexical analyzer to recognize a few patterns in C.

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
#include<stdlib.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("printf",s
tr)==0||strcmp("switch",str)==0||strcmp("case",str)==0)
printf("\n%s is a keyword",str);
else
printf("\n%s is an identifier",str); }
void main()
```

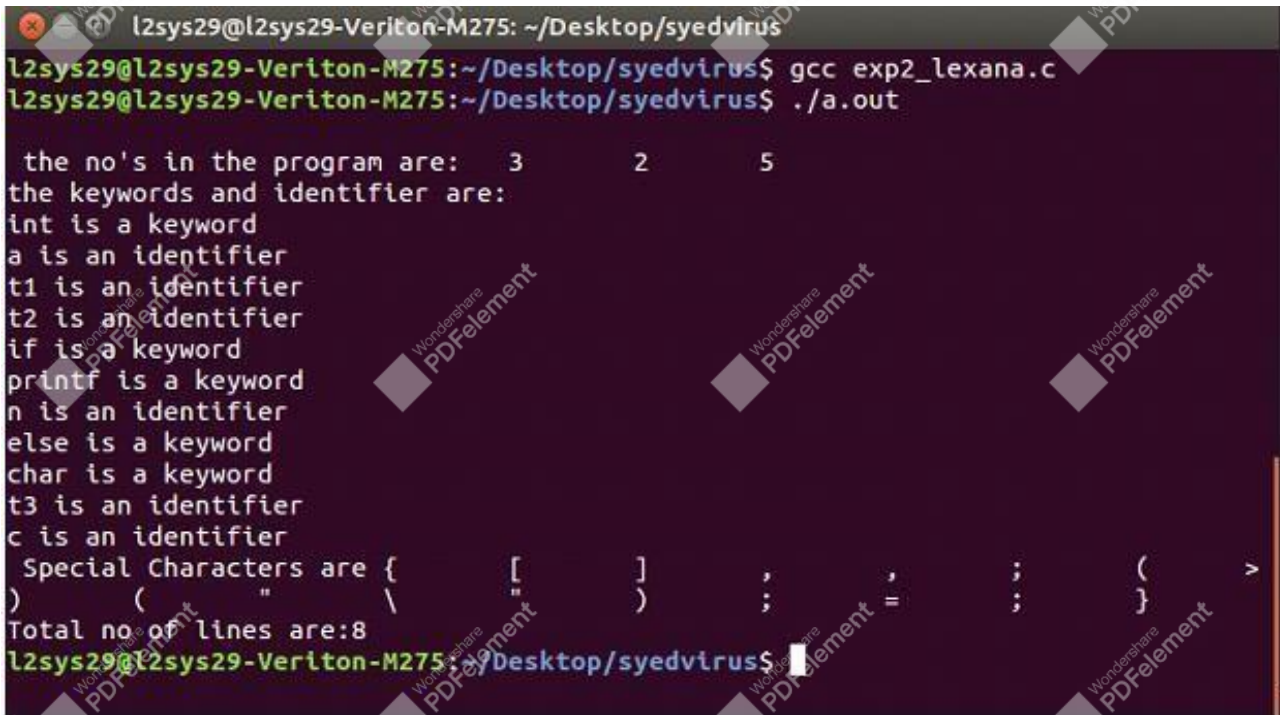
```
{  
FILE *f1,*f2,*f3;  
char c,str[10],st1[10];  
int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;  
f1=fopen("input","r");  
f2=fopen("identifier","w");  
f3=fopen("specialchar","w");  
while((c=getc(f1))!=EOF)  
{  
    if(isdigit(c))  
    {  
tokenvalue=c-'0'; c=getc(f1);  
        while(isdigit(c))  
        {  
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("\n the no's in the program are:");
for(j=0;j<i;j++)
printf("\t%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("the keywords and identifier are:");
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("\n Special Characters are");
while((c=getc(f3))!=EOF)
printf("\t%c",c);
printf("\n");
```

```
fclose(f3);  
printf("Total no of lines are:%d",lineno);  
}
```

OUTPUT



```
l2sys29@l2sys29-Veriton-M275: ~/Desktop/syedvirus  
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ gcc exp2_lexana.c  
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./a.out  
  
the no's in the program are: 3      2      5  
the keywords and identifier are:  
int is a keyword  
a is an identifier  
t1 is an identifier  
t2 is an identifier  
if is a keyword  
printf is a keyword  
n is an identifier  
else is a keyword  
char is a keyword  
t3 is an identifier  
c is an identifier  
Special Characters are { [ ] , ' ; ( )  
) ( " \ " ) ; = ; } >  
Total no of lines are:8  
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$
```

Result:

Thus, the program for developing a lexical analyzer to recognize a few patterns in C has been executed successfully.

Ex.No : 3**Date :****Implementation of lexical analyzer using lex tool.****Aim:**

To write a program for implementing a Lexical analyzer using LEX tool

Algorithm:

Step 1: Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, `%%`. The format is as follows: definitions `%%` rules `%%` user subroutines

Step 2: In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in `%{..}%`. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.

Step 3: In rules section, the left column contains the pattern to be recognized in an input file to `yylex()`. The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.

Step 4: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.

Step 5: When `yylex()` matches a string in the input stream, it copies the matched text to an external character array, `yytext`, before it executes any actions in the rules section.

Step 6: In user subroutine section, main routine calls `yylex()`. `yywrap()` is used to get more input.

Step 7: The lex command uses the rules and actions contained in file to generate a program, `lex.yy.c`, which can be compiled with the `cc` command. That program can then receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

Program:

//Implementation of Lexical Analyzer using Lex tool

```
%{  
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 |
struct |
typedef |
do |
if |
break |
continue |
void |
switch |
return |
else |
goto
{
printf("\n\t%s is a keyword",yytext);
} "/*" {COMMENT=1;}
{
printf("\n\t %s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{ {if(!COMMENT)printf("\n BLOCK BEGINS");} \}
{if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}\([ [0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %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)
{
FILE *file;
file=fopen("var.c","r");
if(!file)
{ printf("could not open the file");
exit(0);
} yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

Input:

```
//var.c
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
a=1; b=2;
c=a+b;
printf("Sum:%d",c);
}
```

OUTPUT

```
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ lex exp3_lex.l
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ cc lex.yy.c
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./a.out

#include<stdio.h> is a preprocessor directive
#include<conio.h> is a preprocessor directive
void is a keyword
FUNCTION
main(
)

BLOCK BEGINS
    int is a keyword
a IDENTIFIER,
b IDENTIFIER,
c IDENTIFIER;
a IDENTIFIER
    = is an ASSIGNMENT OPERATOR
1 is a NUMBER ;
b IDENTIFIER
    = is an ASSIGNMENT OPERATOR
2 is a NUMBER ;
c IDENTIFIER
    = is an ASSIGNMENT OPERATOR
a IDENTIFIER+
b IDENTIFIER;
FUNCTION
    printf(
        "Sum:%d" is a STRING,
c IDENTIFIER
    )
;
BLOCK ENDS
```

Result:

Thus the program for implementation of Lexical Analyzer using Lex tool has been executed successfully.

Ex.No : 4 a)**Generate yacc specification for a few syntatic categories.****Date :****a) Program to recognize a valid arithmetic expression that uses operator +, -, * and /.****AIM:**

To write a c program to do exercise on syntax analysis using YACC.

ALGORITHM:

1. Start the program.
2. Write the code for parser. l in the declaration port.
3. Write the code for the 'y' parser.
4. Also write the code for different arithmetical operations.
5. Write additional code to print the result of computation.
6. Execute and verify it.
7. Stop the program.

PROGRAM:

```
#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

```
Enter any operator: *  
Multiplication_
```

RESULT:

Thus the program for the exercise on the syntax using YACC has been executed successfully and Output is verified.

Ex.No : 4 b) Date :	Generate yacc specification for a few syntatic categories. Program to recognize a valid variable which starts with a letter followed by any number of letter or digits.
--------------------------------------	--

AIM :

To write a c program to do exercise on syntax analysis using YACC.

ALGORITHM :

1. Start the program.
2. Write the code for parser. l in the declaration port.
3. Write the code for the 'y' parser.
4. Also write the code for different arithmetical operations.
5. Write additional code to print the result of computation.
6. Execute and verify it.
7. Stop the program.

PROGRAM :

variable_test.l

```
%{  
/* This LEX program returns the tokens for the Expression */ #include  
"y.tab.h"  
%}  
%%  
"int " {return INT;} "float"  
{return FLOAT;} "double"
```

```
{return DOUBLE;} [a-zA-Z]*[0-9]* {  
printf("\nIdentifier is %s",yytext); return  
ID;  
}  
return yytext[0];  
\n return 0; int  
yywrap()  
{  
return 1;  
}
```

variable_test.y

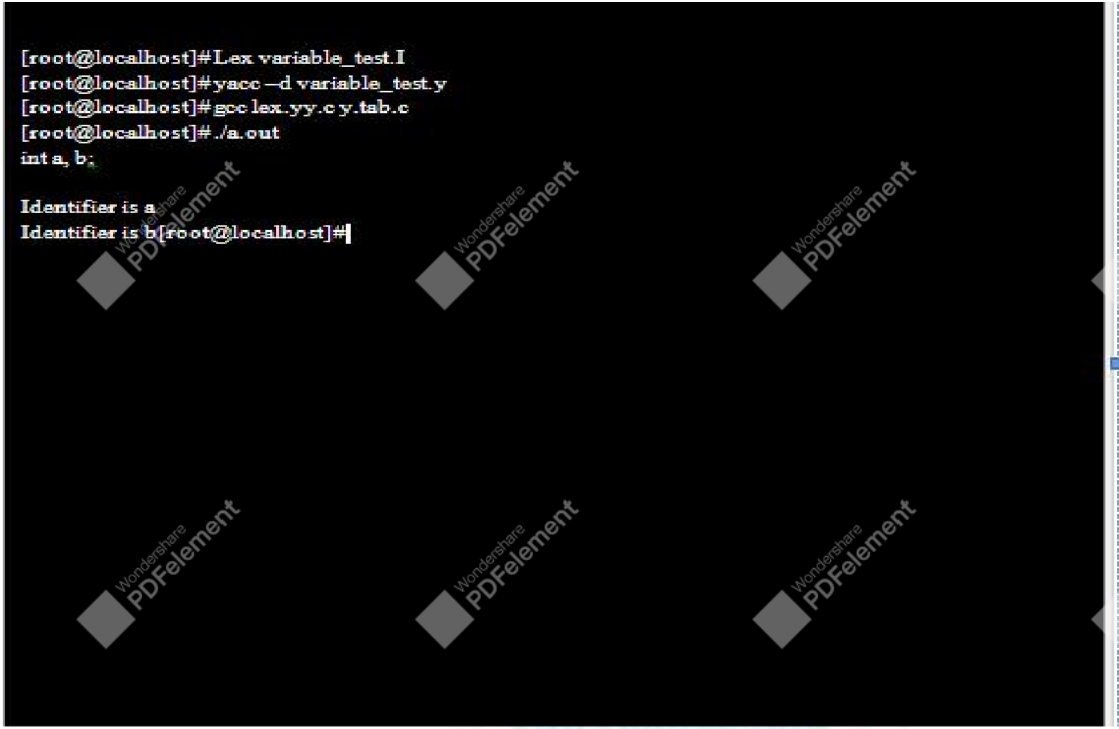
```
%{  
#include  
/* This YACC program is for recognising the Expression*/  
%}  
%token ID INT FLOAT DOUBLE  
%%  
D;T  
L  
;  
L:L,ID  
|ID  
;  
T:IN  
T  
|FLOAT  
|DOUBLE  
;
```

```
%%extern FILE *yyin; main()
```

```
{  
do  
{
```

```
yyparse();  
}while(!feof(yyin));  
}  
yyerror(char*s)  
{  
}
```

OUTPUT



```
[root@localhost]# Lex variable_test.I  
[root@localhost]# yacc -d variable_test.y  
[root@localhost]# gcc lex.yy.c y.tab.c  
[root@localhost]# ./a.out  
Identifier is a  
Identifier is b [root@localhost]#
```

RESULT:

Thus the program for the exercise on the syntax using YACC has been executed successfully and Output is verified.

Ex.No : 4 c)

Generate yacc specification for a few syntatic categories.

Date :

c). Implementation of calculator using lex and yacc.

AIM :

To write a c program to do exercise on syntax analysis using YACC.

ALGORITHM :

1. Start the program.
2. Write the code for parser. l in the declaration port.
3. Write the code for the 'y' parser.
4. Also write the code for different arithmetical operations.
5. Write additional code to print the result of computation.
6. Execute and verify it.
7. Stop the program.

PROGRAM:

```
%{
```

```
#include<stdio.h>
```

```
#include "y.tab.h"
```

```
extern int yylval;
```

```
%}
```

```
%%
```

```
[0-9]+ {
```

```
    yylval=atoi(yytext);
```

```
    return NUMBER;
}
[\t] ;
```

```
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

YACC PART:

```
%{
```

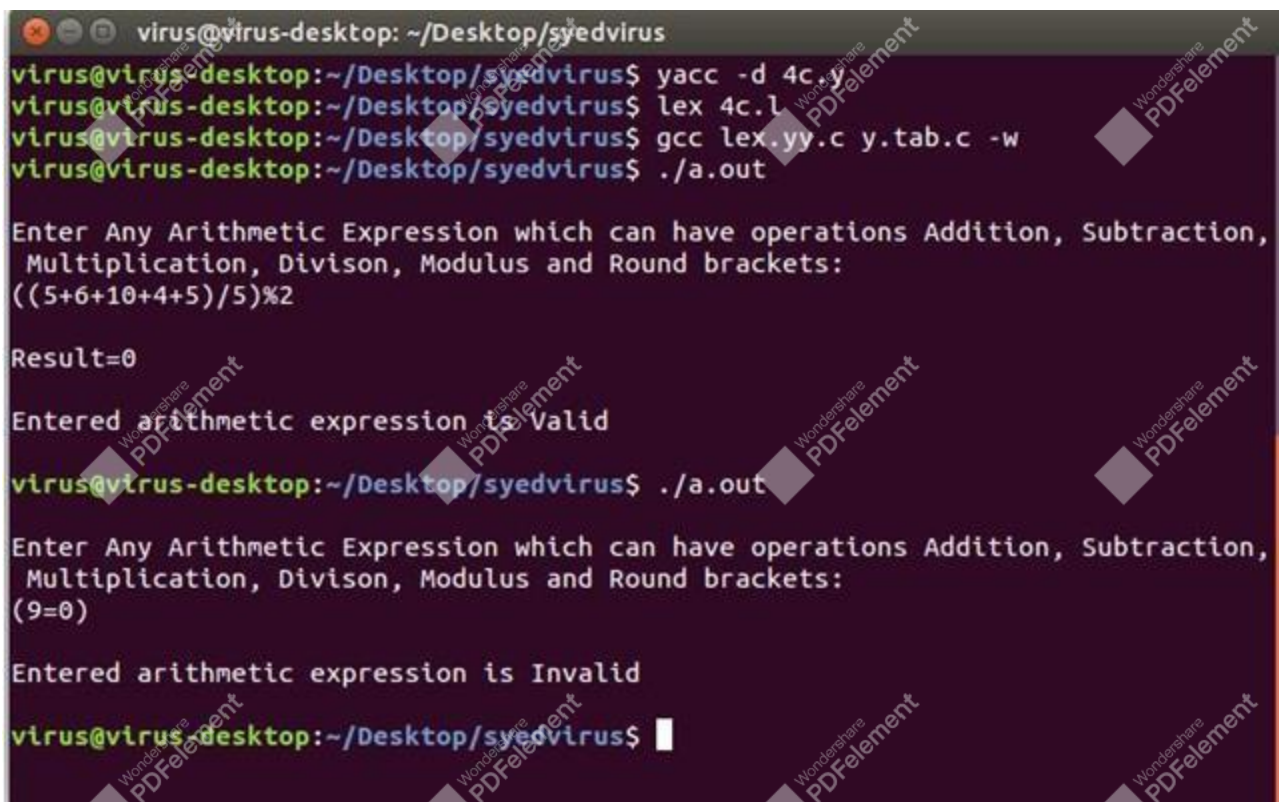
```
#include<stdio.h>
int flag=0;
%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
ArithmeticExpression: E {
    printf("\nResult=%d\n", $$);
    return 0;
};
```

```
E:E'+E {$$=$1+$3;}
|E'-E {$$=$1-$3;}
|E'*E {$$=$1*$3;}
|E'/E {$$=$1/$3;}
|E'%E {$$=$1%$3;}
|'('E)' {$$=$2;}
|NUMBER {$$=$1;}
;
%%
void main()
{
    printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");
    yyparse();
}
```

```
if(flag==0)
```

```
    printf("\nEntered arithmetic expression is Valid\n\n");  
}  
void yyerror()  
{  
    printf("\nEntered arithmetic expression is Invalid\n\n");  
    flag=1;  
}
```

OUTPUT



```
virus@virus-desktop: ~/Desktop/syedvirus  
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 4c.y  
virus@virus-desktop:~/Desktop/syedvirus$ lex 4c.l  
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -w  
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out  
  
Enter Any Arithmetic Expression which can have operations Addition, Subtraction,  
Multiplication, Divison, Modulus and Round brackets:  
((5+6+10+4+5)/5)%2  
  
Result=0  
Entered arithmetic expression is Valid  
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out  
  
Enter Any Arithmetic Expression which can have operations Addition, Subtraction,  
Multiplication, Divison, Modulus and Round brackets:  
(9=0)  
  
Entered arithmetic expression is Invalid  
virus@virus-desktop:~/Desktop/syedvirus$
```

RESULT:

Thus the program for the exercise on the syntax using YACC has been executed Successfully and Output is verified.

Ex.No : 5 Date :	Convert the bnf rules into yacc form and write code to generate abstract syntax tree.
-----------------------------------	--

Aim:

To write a YACC program to change YACC form into abstract syntax Tree

Algorithm:

Step1: Reading an expression.

Step2: Calculate the value of given expression

Step3: Display the value of the nodes based on the precedence.

Step4: Using expression rule print the result of the given values

Program:**LEX PART:**

```
%{  
#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 PART:

```
%{  
#include<string.h>  
#include<stdio.h>  
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 ';'
| CONST
| 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[Index].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Index].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 \tArg1 \tArg2 \tResult" "\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);
}
yyerror()
{
printf("\n Error on line no:%d",LineNo);
}
```

INPUT:

```
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;
}
}
```



OUTPUT

```
virus@virus-desktop: ~/Desktop/syedvirus
virus@virus-desktop:~/Desktop/syedvirus$ lex 5.l
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 5.y
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -l -lm -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out test.c
```

Pos	Operator	Arg1	Arg2	Result
0	<	a	b	t0
1	==	t0	FALSE	5
2	+	a	b	t1
3	=	t1	a	5
4	GOTO			5
5	<	a	b	t2
6	==	t2	FALSE	10
7	+	a	b	t3
8	=	t3	a	5
9	GOTO			5
10	<=	a	b	t4
11	==	t4	FALSE	15
12	-	a	b	t5
13	=	t5	c	17
14	GOTO			17
15	+	a	b	t6
16	=	t6	c	c

```
virus@virus-desktop:~/Desktop/syedvirus$
```

Result:

Thus the program for the exercise on the syntax using YACC has been executed successfully and output is verified.

Ex.No : 6**Date :**

Implement type checking

Aim:

To write a C program to implement type checking.

Algorithm:

Step1: Track the global scope type information (e.g. classes and their members)

Step2: Determine the type of expressions recursively, i.e. bottom-up, passing the resulting types upwards.

Step3: If type found correct, do the operation

Step4: Type mismatches, semantic error will be notified

Program:

//To implement type checking

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
int main()
```

```
{
```

```
int n,i,k,flag=0;
```

```
char vari[15],typ[15],b[15],c;
```

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

```
scanf(" %d",&n);
```

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

```
{
```

```
printf("Enter the variable[%d]:",i);
```

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

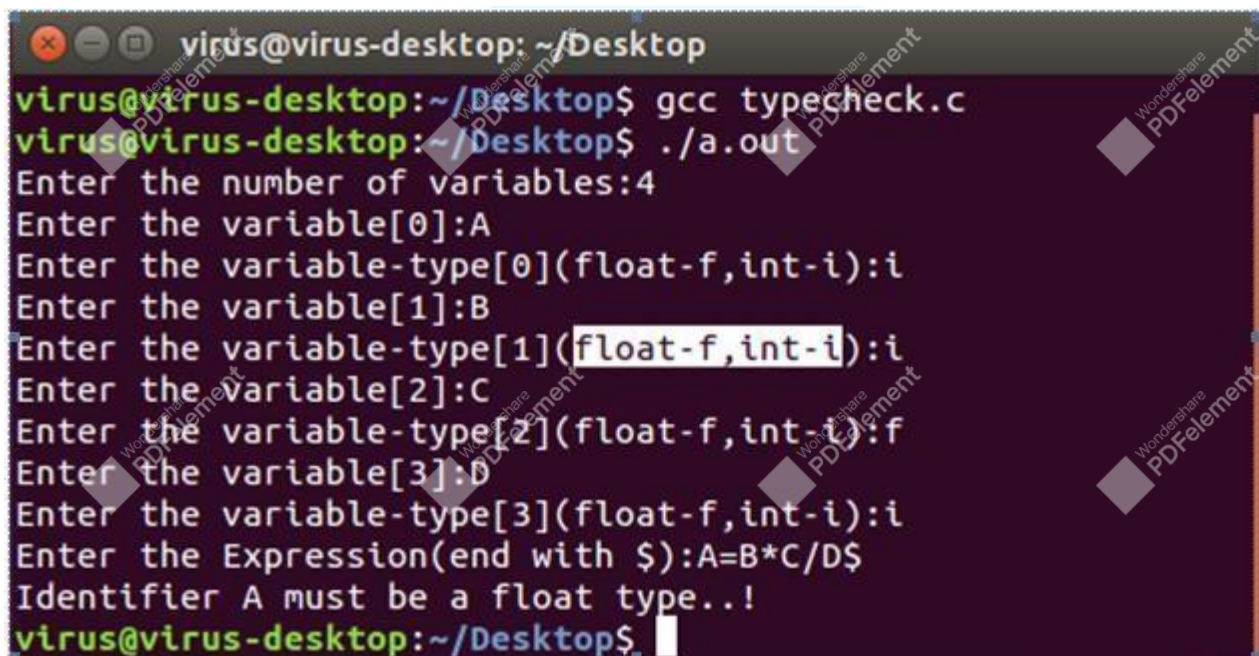
```
printf("Enter the variable-type[%d](float-f,int-i):",i);
```

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

```
if(typ[i]=='f')
flag=1;
}
printf("Enter the Expression(end with $):");
i=0;
getchar();
while((c=getchar())!='$')
{
b[i]=c;
i++; }
k=i;
for(i=0;i<k;i++)
{
if(b[i]=='/')
{
flag=1;
break; } }
for(i=0;i<n;i++)
{
if(b[0]==vari[i])
{
if(flag==1)
{
if(typ[i]=='f')
{ printf("\nthe datatype is correctly defined..!\n");
break; } }
else
{ printf("Identifier %c must be a float type..!\n",vari[i]);
break; } } }
else
```

```
{ printf("\nthe datatype is correctly defined..!\n");  
break; } }  
}  
return 0;  
}
```

OUTPUT



```
virus@virus-desktop: ~/Desktop  
virus@virus-desktop:~/Desktop$ gcc typecheck.c  
virus@virus-desktop:~/Desktop$ ./a.out  
Enter the number of variables:4  
Enter the variable[0]:A  
Enter the variable-type[0](float-f,int-i):i  
Enter the variable[1]:B  
Enter the variable-type[1](float-f,int-i):i  
Enter the variable[2]:C  
Enter the variable-type[2](float-f,int-i):f  
Enter the variable[3]:D  
Enter the variable-type[3](float-f,int-i):i  
Enter the Expression(end with $):A=B*C/D$  
Identifier A must be a float type..!  
virus@virus-desktop:~/Desktop$
```

Result:

Thus, the program for implementation of type checking has been executed successfully.

Ex.No : 7**Implement control flow analysis and data flow analysis.****Date :****Aim:**

To write a program for implementing of control flow analysis and data flow analysis

Algorithm:

1. Start the program
2. Declare the necessary variables
3. Get the choice to insert, delete and display the values in stack
4. Perform PUSH() operation
 - a. t = newnode()
 - b. Enter info to be inserted
 - c. Read n
 - d. t ->info= n
 - e. t ->next=top
 - f. top = t
 - g. Return
5. Perform POP() operation
 - a. If (top=NULL)
 - b. Print"underflow"
 - c. Return
 - d. X=top
 - e. Top=top->next
 - f. Delnode(x)
 - g. Return
6. Display the values
7. Stop the program.

Program:

(DATA FLOW AND CONTROL FLOW ANALYSIS)

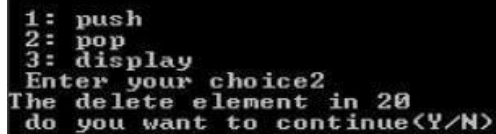
```
#include<conio.h>
```

```
struct stack
{
int no;
struct stack *next;
}
*start=NULL
typedef struct stack st;
void push();
int pop();
void display();
void main()
{
char ch;
int choice, item;
do
{
clrscr();
printf("\n1:push");
printf("\n2:pop");
printf("\n3:display");
printf("\n enter your choice");
scanf("%d",&choice);
switch(choice)
{
case 1:push();
break;
case 2:item=pop();
printf("the delete element in %d",item);
break;
case 3:display();
break;
default: printf("\nwrong choice");
};
};
```

```
printf("\n do you want to continue(y/n");
fflush(stdin);
scanf("%c",&ch);
}
while(ch=='y'||ch=='Y');
}
void push()
{
st*node;
node=(st*)malloc(sizeof(st));
printf("\n enter the number to be insert");
scanf("%d",&node->no);
node->next=start;
start=node;
}
int pop();
{
st*temp;
temp=start;
if(start==null)
{
printf("stack is already empty");
getch();
exit();
}
else
{
start=start->next;
free(temp);
}
return(temp->no);
}
```

```
void display()
{
st*temp;
temp=start;
while(temp->next!=null)
{
printf("\nno=%d",temp->no);
temp=temp->next;
}
printf("\nno=%d",temp->no);
}
```

OUTPUT



```
1: push
2: pop
3: display
Enter your choice2
The delete element in 20
do you want to continue(Y/N)
```

```
1: push
2: pop
3: display
Enter your choice3
no=20
no=10
do you want to continue(Y/N)
```

```
1: push
2: pop
3: display
Enter your choice1
Enter the number to be insert20
do you want to continue(Y/N)
```

Result:

Thus the program for implementing of control flow analysis and data flow analysis has been executed successfully.

Ex.No : 8**Implement any one storage allocation strategies(heap, stack, static)****Date :****Aim:**

To write a program for implementing storage allocation strategies using C.

Algorithm:

Step1: Initially check whether the stack is empty

Step2: Insert an element into the stack using push operation

Step3: Insert more elements onto the stack until stack becomes full

Step4: Delete an element from the stack using pop operation

Step5: Display the elements in the stack

Step6: Top the stack element will be displayed

Program:

//Implementation of heap allocation storage strategies//

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#define TRUE 1
```

```
#define FALSE 0
```

```
typedef struct Heap
```

```
{
```

```
int data;
```

```
struct Heap *next;
```

```
}
```

```
node;
```

```
node *create();
```

```
void 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("\nprogram 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 list");
printf("\n5.quit");
printf("\nenter your chioce(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");
}
}
```

```
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("\nmemory is not allocated");
New->data=val;
if(flag==TRUE)
{
head=New;
temp=head;
flag=FALSE;
}
else
{
temp->next=New;
temp=New;
}
printf("\ndo you want to enter more elements?(y/n)");
}
while(ans=='y');
printf("\nthe list is created\n");
```

```
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("\nthe list is empty\n");
return;
}
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
printf("NULL");
}
node *search(node *head,int key)
{
node *temp;
int found;
temp=head;
if(temp==NULL)
{
printf("the linked list is empty\n");
```

```
return NULL;
}
found=FALSE;
while(temp!=NULL && found==FALSE)
{
if(temp->data!=key)
temp=temp->next;
else
found=TRUE;
}
if(found==TRUE)
{
printf("\nthe element is present in the list\n");
return temp;
}
else
{
printf("the element is not present in the list\n");
return NULL;
}
}
node *insert(node *head)
{
int choice;
node *insert_head(node *);
void insert_after(node *);
void insert_last(node *);
printf("\n1.insert a node as a head node");
printf("\n2.insert a node as a head node");
printf("\n3.insert a node at intermediate position in t6he 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;
}
node *insert_head(node *head)
{
node *New,*temp;
New=get_node();
printf("\nEnter 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("\nEnter 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("\nenter the elements which you want to insert");
scanf("%d",&New->data);
if(head==NULL)
{
head=New;
}
else
{
printf("\nenter the element which you want to insert the node");
scanf("%d",&key);
temp=head;
do
{
if(temp->data==key)
{
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)
{
node *temp,*prev;
int key;
temp=*head;
if(temp==NULL)
{
printf("\nthe list is empty\n");
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);
}
printf("\nthe element is deleted\n");
}
}
```

```
l2sys23@l2sys23-Veriton-M275: ~/Desktop/dss
l2sys23@l2sys23-Veriton-M275:~$ cd Desktop
l2sys23@l2sys23-Veriton-M275:~/Desktop$ cd dss
l2sys23@l2sys23-Veriton-M275:~/Desktop/dss$ gcc heap.c
l2sys23@l2sys23-Veriton-M275:~/Desktop/dss$ ./a.out

program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)2

the list is empty

program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)5
l2sys23@l2sys23-Veriton-M275:~/Desktop/dss$
```

Result:

Thus, the program for implementing storage allocation strategies using C has been executed successfully.

Ex.No : 9**Date :**

Construction of DAG

Aim:

To write a C program to construct of DAG (**Directed Acyclic Graph**)

ALGORITHM:

1. Start the program
2. Include all the header files
3. Check for postfix expression and construct the in order DAG representation
4. Print the output
5. Stop the program

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MIN_PER_RANK 1
#define MAX_PER_RANK 5
#define MIN_RANKS 3
#define MAX_RANKS 5
#define PERCENT 30
void main()
{
int i,j,k,nodes=0;
srand(time(NULL));
int ranks=MIN_RANKS+(rand()%(MAX_RANKS-MIN_RANKS+1));
```

```

printf("DIRECTED ACYCLIC GRAPH\n");
for(i=1;i<ranks;i++)
{
int new_nodes=MIN_PER_RANK+(rand()%(MAX_PER_RANK-
MIN_PER_RANK+1));
for(j=0;j<nodes;j++)
for(k=0;k<new_nodes;k++)
if((rand()%100)<PERCENT)
printf("%d->%d;\n",j,k+nodes);
nodes+=new_nodes;
}
}

```

OUTPUT

```

l2sys29@l2sys29-Veriton-M275: ~/Desktop/syedvirus
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./a.out
DIRECTED ACYCLIC GRAPH
0->4;
0->6;
0->7;
0->9;
1->6;
1->7;
2->6;
3->7;
3->8;
4->7;
4->9;
5->6;
5->8;
1->10;
1->11;
1->12;
3->11;
4->10;
5->10;
5->11;
7->10;
8->10;
9->12;
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$

```

RESULT:

Thus the program for implementation of DAG has been successfully executed and output is verified.

Ex.No : 10 Date :	Implement the back end of the compiler which takes the three address code and produces the 8086n 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.
------------------------------------	---

Aim:

To Write a C program to Implement the back end of the compiler which takes the three address code and produces the 8086n 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:

1. Start the program.
2. Get the three variables from statements and stored in the text file k.txt.
3. Compile the program and give the path of the source file.
4. Execute the program.
5. Target code for the given statement was produced.
6. Stop the program.

PROGRAM:

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

```
int i = 0;
clrscr();
printf("\n Enter the set of intermediate code (terminated by exit):\n");
do
{
    scanf("%s", icode[i]);
} while (strcmp(icode[i++], "exit") != 0);
printf("\n target code generation");
printf("\n*****");
i = 0;
do {
    strcpy(str, icode[i]);
    switch (str[3]) {
    case '+':
        strcpy(opr, "ADD ");
        break;
    case '-':
        strcpy(opr, "SUB ");
        break;
    case '*':
        strcpy(opr, "MUL ");
        break;
    case '/':
```

```
strcpy(opr, "DIV ");  
    break;  
}  
printf("\n\tMov %c,R%d", str[2], i);  
printf("\n\t%s%c,R%d", opr, str[4], i);  
printf("\n\tMov R%d,%c", i, str[0]);  
} while (strcmp(icode[++i], "exit") != 0);  
getch();  
}
```

OUTPUT:

Enter the set of intermediate code (terminated by exit):

```
a=a*b  
c=f*h  
g=a*h  
f=Q+w  
t=q-j  
exit
```

target code generation

Mov a,R0

MUL b,R0

Mov R0,a

Mov f,R1

MUL h,R1

Mov R1,c

Mov a,R2

MUL h,R2

Mov R2,g

Mov Q,R3

ADD w,R3

Mov R3,f

Mov q,R4

SUB j,R4

Mov R4,t

Result :

Thus the C program to Implement the back end of the compiler which takes the three address code and produces the 8086n 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.

Ex.No : 11

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

Date :

Aim:

To Write a C program to Implementation of simple code optimization techniques (constant folding. etc.)

ALGORITHM:

1. Start the program
2. Declare the variables and functions.
3. Enter the expression and state it in the variable a, b, c.
4. Calculate the variables b & c with 'temp' and store it in f1 and f2.
5. If(f1=null && f2=null) then expression could not be optimized.
6. Print the results.
7. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.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;
clrscr();
printf("Enter the 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("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 Eliminationn");
for(k=0;k<z;k++)
{
printf("%ct=",pr[k].l);

```

```

printf("%sn",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",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';
strcpy(pr[i].r,'\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);
}
}
getch();
}

```

OUTPUT:

INPUT & OUTPUT:

Enter the Number of Values:5

left: a right: 9

left: b right: c+d

left: e right: c+d

left: f right: b+e

left: r right: f

Intermediate Code

a=9

b=c+d

e=c+d

f=b+e

r=f

nAfter Dead Code Eliminationnbt=c+dnet=c+dnft=b+enrt=fnpos: 2Eliminate Common Expression

b =c+d

b =c+d

f =b+b

r =f

Optimized Code

b=c+d

f=b+b

r=f

Result:

Thus the C program to Implementation of simple code optimization techniques (constant folding. etc.)