****

**An Autonomous Institution**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**LAB MANUAL**

**SUBJECT NAME : 191CS62A –COMPILER DESIGNLABORATORY**

**YEAR/SEM :** III / VI

**REGULATION :** 2019

**PREPARED BY : S. DEEPA (ASSISTANT PROFESSOR)**

**V.PRIYA (ASSISTANT PROFESSOR)**

**Department of Computer Science and Engineering**

**Vision**

To emerge as centre for academic excellence in the field of Computer Science and Engineering by exposure to research and industry practices.

**Mission**

**M1** - To provide good teaching and learning environment with conductive research atmosphere in the field of Computer Science and Engineering.

**M2** - To propagate lifelong learning.

**M3** - To impart the right proportion of knowledge, attitudes and ethics in students to enable them take up positions of responsibility in the society and make significant contributions.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)**

|  |  |
| --- | --- |
| **PEOs** | **PROGRAMME EDUCATIONAL OBJECTIVES** |
| **PEO1** | Ability to identify, formulate and analyze complex Computer Scienceand Engineering problems in the areas of hardware, software, theoretical Computer Science and applications to reach significant conclusions by applying Mathematics, Natural sciences, Computer Science and Engineering principles. |
| **PEO2** | Apply knowledge of mathematics, natural science, engineering fundamentals and system fundamentals, software development, networking & communication, and information security to the solution of complex engineering problems in computer science and engineering to get benefits in their professional career or higher education and research or technological entrepreneur. |
| **PEO3** | Design solutions for complex computer science and engineering problems using state of the art tools andtechniques, components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal, and environmental considerations. |

**PROGRAMME SPECIFIC OUTCOMES (PSOs)**

|  |  |
| --- | --- |
| **PSO’s** | **PROGRAMME SPECIFIC OUTCOMES** |
| **PSO1** | An ability to apply, design and development of application oriented software systems and to test and document in accordance with Computer Science and Engineering. |
| **PSO2** | The design techniques, analysis and the building, testing, operation and maintenance of networks, databases, security and computer systems (both hardware and software). |
| **PSO3** | An ability to identify, formulate and solve hardware and software problems using sound computer engineering principles. |

**PROGRAMME OUTCOMES (POs)**

|  |  |
| --- | --- |
| **PO’s** | **PROGRAMME OUTCOMES** |
| **PO1** | **Engineering Knowledge:** Apply knowledge of **mathematics, science**, **engineering** **fundamentals** and an **Engineering Specialization** to the solution of complex engineering problems. |
| **PO2** | **Problem Analysis:** **Identify**, **formulate, review research literature** and **analyze complex engineering** problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. |
| **PO3** | **Design / Development of solutions:**Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal, and environmental considerations. |
| **PO4** | **Conduct Investigations of Complex Problems**: 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. |
| **PO5** | **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations. |
| **PO6** | **The Engineer and Society**: 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. |
| **PO7** | **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable  development. |
| **PO8** | **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| **PO9** | **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings. |
| **PO10** | **Communication:**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 |
| **PO11** | **Project Management and Finance:**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. |
| **PO12** | **Life-long learning**: 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. |

**COURSE OBJECTIVES**

### The student should be made to:

### Build using different compiler writing tools.

### Diagnose how to implement the different Phases of compiler

### Express the familiarities how to use the control flow and data flow analysis

### Design the simple optimization techniques

### COURSE OUTCOMES

**At the end of the course, the student should be able to:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CO-PO & PSO Mapping** | | | | | | | | | | | | | | | |
| **CO** | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** | **PSO1** | **PSO2** | **PSO3** |
| **CO1** | 3 | 3 | 3 | 3 | 3 | 2 | 1 | - | - | - | - | 1 | 3 | 3 | 2 |
| **CO2** | 3 | 3 | 3 | 3 | 2 | 1 | - | - | - | - | - | 1 | 3 | 3 | 1 |
| **CO3** | 3 | 3 | 3 | 3 | 2 | 1 | - | - | - | - | - | 1 | 3 | 3 | 1 |
| **CO4** | 3 | 3 | 3 | 3 | 3 | 1 | - | - | 1 | - | - | 1 | 3 | 3 | 1 |
| **CO5** | 3 | 3 | 3 | 3 | 2 | 1 | 1 | - | - | - |  | 1 | 3 | 3 | 1 |
| **CO** | 3 | 3 | 3 | 3 | 2 | 1 | 1 | - | 1 | - | - | 1 | **3** | **3** | **1** |

Mapping CO's with PO's and PSO’S

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Course Outco me** | **PO's** | | | | | | | | | | | | **PSO’s** | | |
| **PO 1** | **PO 2** | **PO 3** | **PO 4** | **PO 5** | **PO 6** | **PO 7** | **PO 8** | **PO 9** | **PO10** | **PO11** | **PO12** | **PSO 1** | **PSO 2** | **PSO 3** |
| CO1 | 3 | 3 | 3 | 3 | 3 | 2 | 1 | - | - | - | - | 1 | 3 | 3 | 2 |
| CO2 | 3 | 3 | 3 | 3 | 2 | 1 | - | - | - | - | - | 1 | 3 | 3 | 1 |
| CO3 | 3 | 3 | 3 | 3 | 2 | 1 | - | - | - | - | - | 1 | 3 | 3 | 1 |
| CO4 | 3 | 3 | 3 | 3 | 3 | 1 | - | - | 1 | - | - | 1 | 3 | 3 | 1 |
| CO5 | 3 | 3 | 3 | 3 | 2 | 1 | 1 | - | - | - | - | 1 | 3 | 3 | 1 |
| **CO** | **3** | **3** | **3** | **3** | **2** | **1** | **1** | **-** | **1** | **-** | **-** | **1** | **3** | **3** | **1** |

1 – Low 2 – Medium 3 – High

**Course Code: 191CS62A**

**Course : COMPILER DESIGN LABORATORY**

**COURSE PLAN**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ex. No** | **List of Exercises** | **COs** | | **Page No** |
| **1** | Implementation of Symbol Table | 1 | | 1 |
| **2** | Develop a lexical analyzer to recognize a few patterns in C. | 1 | | 3 |
| **3** | Implementation of Lexical Analyzer using Lex Tool | 1 | | 7 |
| **4** | Generate YACC specification for a few syntactic categories | 1 | | 9 |
| **5** | Program to recognize a valid arithmetic expression that uses operator +, - , \* | 2 | | 11 |
| **6** | Program to recognize a valid variable which starts with a letter followed by any number of letters or digits | 2 | | 15 |
| **7** | Implementation of Calculator using LEX and YACC | 1 | | 18 |
| **8** | Convert the BNF rules into Yacc form and write code to generate Abstract Syntax Tree | 3 | | 20 |
| **9** | Implement type checking | 3 | | 26 |
| **10** | Implement control flow analysis and Data flow Analysis | 5 | | 28 |
| **11** | Implement any one storage allocation strategies(Heap, Stack ,Static) | 4 | | 34 |
| **12** | Construction of DAG | 5 | | 36 |
| **13** | 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. | 3 | | 46 |
| **14** | Implementation of Simple Code Optimization Techniques | 5 | | 50 |
| **15** | Implementation Of Shift-Reduced Parsing Algorithms | 2 | | 53 |
| **16** | Construction Of LR -Parsing Table. | 2 | 56 | |
| **17** | Construction of CLR –Parsing Table. | 2 | 59 | |

**Signature of Course In-charge(s)**

**i)**

**ii)**

**Course Coordinator HOD**

**Ex.no:1 Implementation of Symbol Table**

**Date:**

**AIM:**

To write a program in C for implementing Symbol Table.

**ALGORITHM**

1.Start the program.

2. Declare the variables.Get the character and check it using while Loop.

3.If n value is less than or equal to I then print the symbol address and type.Again check the n value with j.

4.The C value is changed to ASCII and check it using if statement.Store the C value in P and print the identifier.

Else check the character is equal to +, -, \*, /, (,) using if statement. Store the C value in P and print the operator.

5. Enter any symbol to find in the Symbol Table.

6.Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

#include<ctype.h>

void main()

{

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

int flag=0;

int \*p,\*add[10];

char c,ch='y',srch,b[10],d[10];

clrscr();

printf("\ nEnter an expression and it is terminated by $");

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

{

b[i]=c;

i++;

}

n=i-1;

i=0;

printf("Symbol Table \n");

printf(" \nSymbol\t\tAddress\t\ttype");

while(j<=n)

{

c=b[j];

if(isalpha(toascii(c)))

{

p=malloc(c);

add[x]=p;

d[x]=c;

printf(" \n\t%c\t\t%d\t\tidentifier",c,p);

}

if(c=='+'||c=='-'||c=='\*'||c=='/'||c=='='||c==')')

{

p=malloc(c);

add[x]=p;

d[x]=c;

printf(" \n\t%c\t\t%d\t\tOperator",c,p);

}

x++;

j++;

}

while(ch=='y')

{

flag=0;

printf(" \nEnter the symbol to search");

fflush(stdin);

srch=getchar();

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

{

if(srch==d[i])

{

printf(" \nSymbol found\t");

printf("%c \t%s%d\n",srch,"@address",add[i]);

flag=1;

}

}

if(flag==0)

{

printf("Symbol not found");

}

printf("Do you want to continue (y/n): ");

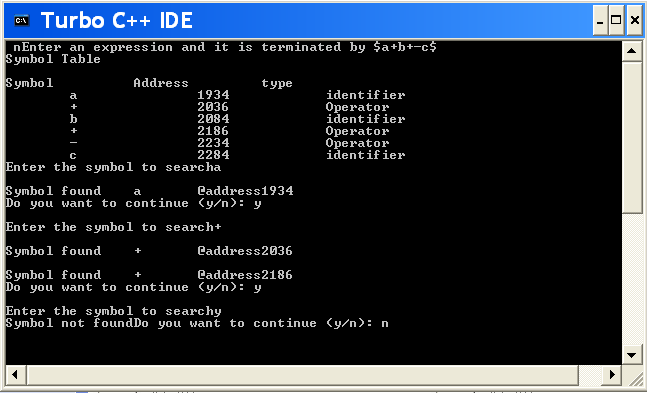
fflush(stdin);

ch=getchar();

}

}

**OUTPUT**

****

**Result:**

The above C program was successfully executed and verified

**Ex. No: 2 Develop a lexical analyzer to recognize a few patterns in C.**

**(Ex. identifiers, constants, comments, operators etc)**

**Date:**

**Aim:**

To write a C program to develop a lexical analyzer to recognize a few patterns in C. (Ex.

identifiers, constants, comments, operators etc.

**Algorithm:**

1.Start the program.

2.Intialize the symbol table with keywords.

3.Read token by token from the input string.

4.Using finite automation check for keywords, identifiers, constants & thenOperators successively.

5.If nothing matches print an error message.

6.Until all tokens are over, repeat above three steps.

7.Print token information.

8.Stop.

**Procedure to run the program:**

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, maintest, constants and regular

definitions.

4. Translation rule of lex program are statements of the form a. 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 aslex.yy.c.

eg $ lex filename.l

$ cc lex.yy.c -ll

7. Compile that file with C compiler and verify the output.

**Program:**

**Lexical.C:**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h>

void main()

{

FILE \*fi,\*fo,\*fop,\*fk;

int flag=0,i=1;

char c,t,a[15],ch[15],file[20];

clrscr();

printf("\n Enter the File Name:");

scanf("%s",&file);

fi=fopen(file,"r");

fo=fopen("inter.c","w");

fop=fopen("oper.c","r");

fk=fopen("key.c","r");

c=getc(fi);

while(!feof(fi))

{

if(isalpha(c)||isdigit(c)||(c=='['||c==']'||c=='.'==1))

fputc(c,fo);

else

{

if(c=='\n')

fprintf(fo,"\t$\t");

else

fprintf(fo,"\t%c\t",c);

}

c=getc(fi);

}

fclose(fi);

fclose(fo);

fi=fopen("inter.c","r");

printf("\n Lexical Analysis");

fscanf(fi,"%s",a);

printf("\n Line: %d\n",i++);

while(!feof(fi))

{

if(strcmp(a,"$")==0)

{

printf("\n Line: %d \n",i++);

fscanf(fi,"%s",a);

}

fscanf(fop,"%s",ch);

while(!feof(fop))

{

if(strcmp(ch,a)==0)

{

fscanf(fop,"%s",ch);

printf("\t\t%s\t:\t%s\n",a,ch);

flag=1;

}

fscanf(fop,"%s",ch);

}

rewind(fop);

fscanf(fk,"%s",ch);

while(!feof(fk))

{

if(strcmp(ch,a)==0)

{

fscanf(fk,"%k",ch);

printf("\t\t%s\t:\tKeyword\n",a);

flag=1;

}

fscanf(fk,"%s",ch);

}

rewind(fk);

if(flag==0)

{

if(isdigit(a[0]))

printf("\t\t%s\t:\tConstant\n",a);

else

printf("\t\t%s\t:\tIdentifier\n",a);

}

flag=0;

fscanf(fi,"%s",a);

}

getch();

}

**Key.C:**

int

void

main

char

if

for

while

else

printf

scanf

FILE

include

stdio.h

conio.h

iostream.h

**Oper.C:**

( open para

) closepara

{ openbrace

} closebrace

<lesser

>greater

" doublequote

' singlequote

: colon

; semicolon

# preprocessor

= equal

== asign

% percentage

^ bitwise

& reference

\* star

+ add

- sub

\ backslash

/ slash

**Input.C:**

#include "stdio.h"

#include "conio.h"

void main()

{

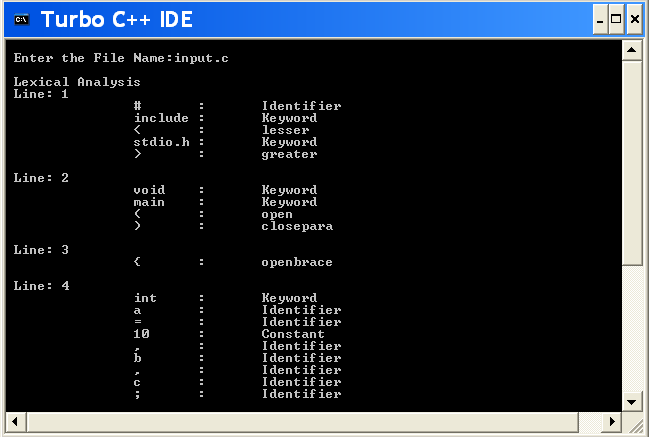
int a=10,b,c;

a=b\*c;

getch();

}

**OUTPUT**

****

**Result:**

The above C program was successfully executed and verified

**Ex. No:3 Implementation of Lexical Analyzer using Lex Tool**

**Date:**

**Aim:**To implement Lexical Analyzer using Lex Tool.

**Algorithm:**

1. Start the program

2. Open a file file.c in read and include the yylex() tool for input scanning

3. Define the alphabets and numbers.

4. Print the preprocessor, function, keyword using yytext.lex tool.

5.Print the relational, assignment and all the operator using yytext() tool.

6.Also scan and print where the loop ends and begins.

7. Use yywrap() to enter an error.

8. Stop the program.

**Program:**

**Lextool.l**

%{

%}

identifier[a-zA-Z][a-zA-Z0-9]\*

%%

#.\* printf("\n%s is PREPROCESSOR DIRECTIVE\n", yytext);

int |

float |

double |

char |

for |

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

{identifier}\( printf("\n\n FUNCTION CALL\n %s",yytext);

\{

printf("BLOCK BEGINS\n");

\}

printf("BLOCK ENDS\n");

{identifier}(\[[0-9]\*\])? printf("%s is identifier\n",yytext);

= printf("%s is a ASSIGNMENT OPERATOR\n",yytext);

[0-9]+ printf("%s is NUMBER\n",yytext);

\< |

\> |

\== |

\>= |

\<= printf("%s is a RELATIONAL OPERATOR\n",yytext);

\( printf("open para\n");

\) printf("close para\n");

\+ |

\- |

\\* printf("%s is a ARITHMETIC OPERATOR \n",yytext);

\++ printf("%s is a INCREMENTAL OPERATOR\n",yytext);

\; { ECHO; printf("\n");}

%%

main()

{

yylex();

}

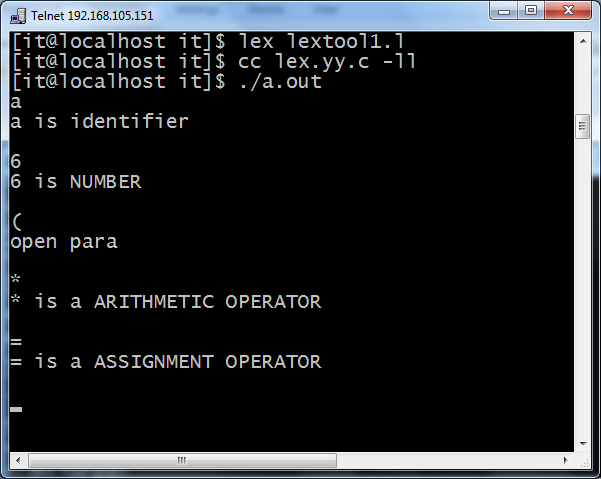
int yywrap()

{

return 1;

}

**Output**

****

**Result:**

The above C program was successfully executed and verified

**EX.NO:4 GENERATE YACC SPECIFICATION FOR A FEW SYNTACTIC CATEGORIES.**

**DATE:**

**AIM :**

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

**INTRODUCTION :**

YACC (yet another compiler) is a program designed to produce designed to compile a

LALR (1) grammar and to produce the source code of the synthetically analyses of the

language produced by the grammar.

**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");

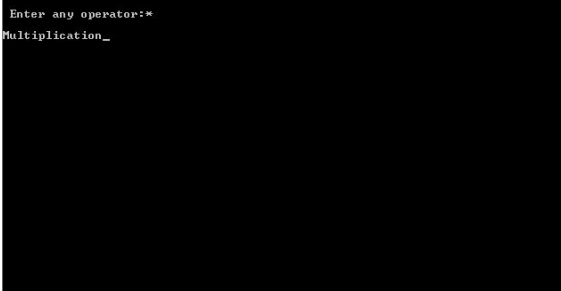
break;

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

break;

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

**Output :**



**Result:**

The above C program was successfully executed and verified.

**Ex.no: 5 Program to recognize a valid arithmetic expression that**

**Date: Uses operator +, - , \* and /.**

**Aim :**

To write a yacc and lex program to recognize a valid arithmetic expression that uses

operator+,-,\* and /.

**Algorithm:**

Input: Programming language arithmetic expression

Output: A sequence of tokens.

Tokens have to be identified and its respective attributes have to be printed.

**Lex:**

1. {Declaration and regular definition]

Define header files to include first section

2. [translation rule]

Tokens generated are used in yacc files

[a-z A-Z] alphabets are returned

0-9 one or more combinations of integers

**Yacc:**

1. Accept token generated in lex part as input

2. Specify the order of procedure

3. Define rules with end points

4. Parse input string from standard input by calling yyparse() main function.

5. Print the result of any rules defined matches as arithmetic expression as valid

6. If none of the rule defined matches print arithmetic expression is invalid.

**Program:**

**Yacc4a.y(without lex only yacc program)**

%{

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

%}

%token num let

%left '+' '-'

%left '\*' '/'

%%

stmt: stmt '\n' {printf("\n..valid Expression..\n"); exit(0);}

| expr

|

| error '\n' {printf("\n..Invalid..\n"); exit(0);}

;

expr: num

| let

| expr '+' expr

| expr '-' expr

| expr '\*' expr

| expr '/' expr

| '(' expr ')'

%%

main()

{

printf("Enter an exoression to validate :");

yyparse();

}

yylex()

{

int ch;

while((ch=getchar())==' ');

if(isdigit(ch))

return num;

if(isalpha(ch))

return let;

return ch;

}

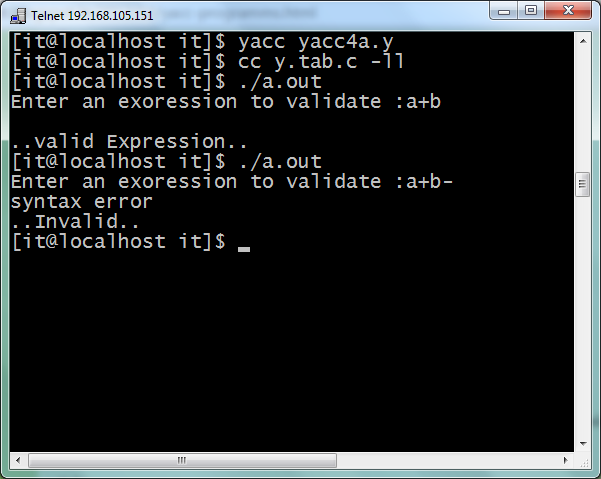
yyerror(char \*s)

{

printf("%s",s);

}

**Output :**



**Lex program (with lex and yacc)**

%{

#include"y.tab.h"

extern yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext); return NUMBER;}

[a-zA-Z]+ {return ID;}

[\t]+ ;

\n {return 0;}

. {return yytext[0];}

%%

int yywrap()

{

return(1);

}

**Yacc program**

%{

#include<stdio.h>

%}

%token NUMBER ID

%left '+' '-'

%left '\*' '/'

%%expr:

expr '+' expr

|expr '-' expr

|expr '\*' expr

|expr '/' expr

|'-'NUMBER|'-'ID

|'('expr')'

|NUMBER

|ID

;

%%

main()

{

printf("Enter the expression\n");

yyparse();

printf("\nExpression is valid\n");

exit(0);

}

int yyerror(char \*s)

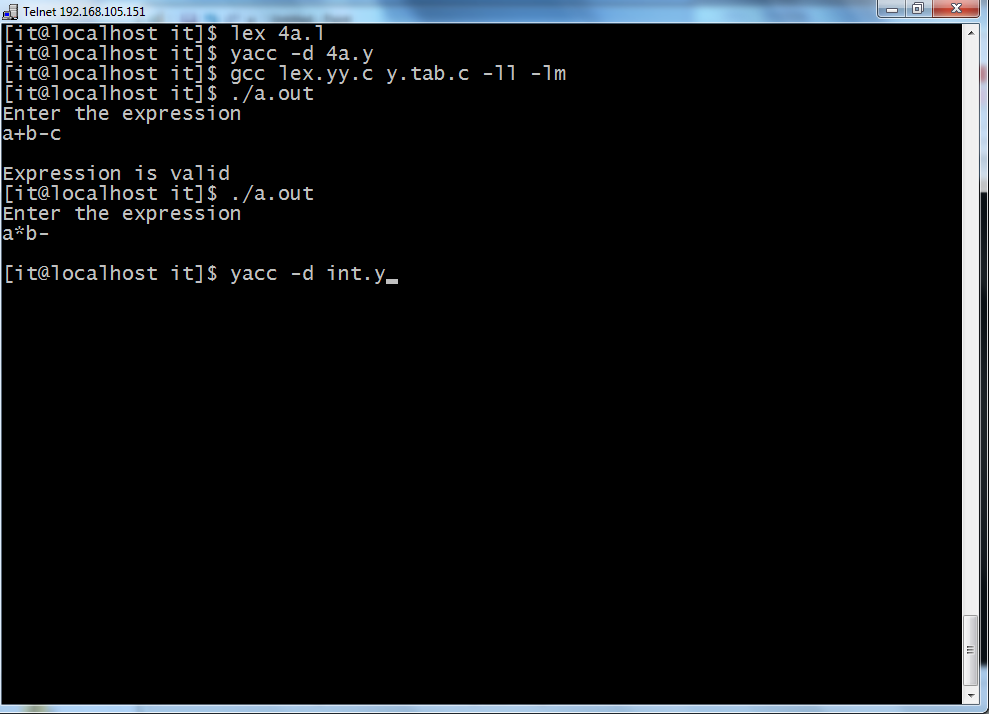
{

printf("\nExpression is invalid");

exit(0);

}

**Output**

****

**Result:**

The above C program was successfully executed and verified

**Ex.no : 6 Program to recognize a valid variable which starts with a**

**Date: Letter followed by any number of letters or digits**

**Aim:**

To write a yacc & lex program to recognize a valid variable this starts with a letter

followed by any number of letters or digits.

**Algorithm:**

Input: Programming language 'if' statement

Output: A sequence of tokens.

Tokens have to be identified and its respective attributes have to be printed

1.Include header file y.tab.h

2. Define tokens

3. Declare [a-z] as L [0-9] as D

4. Declare variable L D P

5. Call function yyerror()

6. If error exists then print “invalid” and exit

7. In main() call yyparse(), if yyparse() print “valid variable”

**Program: (without lex)**

%{

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

%

}

%token let dig

%%

TERM: XTERM '\n'

{

printf("\nAccepted\n");

exit(0);

}

|error

{

yyerror ("Rejected\n"); exit(0);

};

XTERM: XTERM let

| XTERM dig

| let

;

%%

main()

{

printf("Enter a variable:");

yyparse();

}

yylex()

{

char ch;

while((ch=getchar())==' ');

if(isalpha(ch))

return let;

if(isdigit(ch))

return dig;

return ch;

}

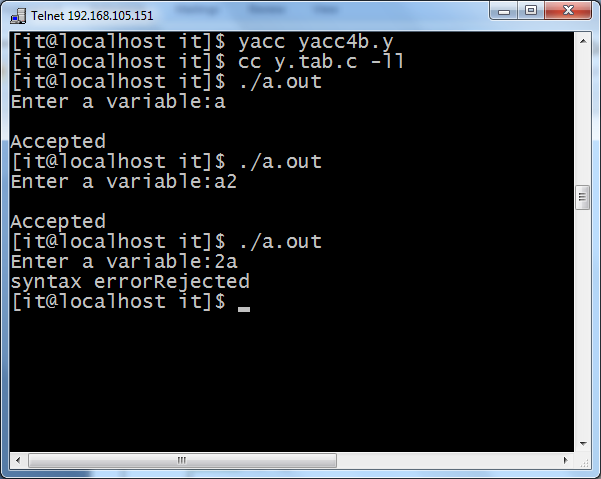
yyerror(char \*s)

{

printf("%s",s);

}

**Output**

****

**Program: with lex and yacc**

**Yacc 4b.l**

%{

#include"y.tab.h"

extern yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext); return DIGIT;}

[a-zA-Z]+ {return LETTER;}

[\t] ;

\n return 0;

. {return yytext[0];}

%%

**Yacc 4b.y**

%{

#include<stdio.h>

%}

%token LETTER DIGIT

%%

variable: LETTER|LETTER rest

;

rest: LETTER rest

|DIGIT rest

|LETTER|DIGIT;

%%

main()

{

yyparse();

printf("The string is a valid variable\n");

}

int yyerror(char \*s)

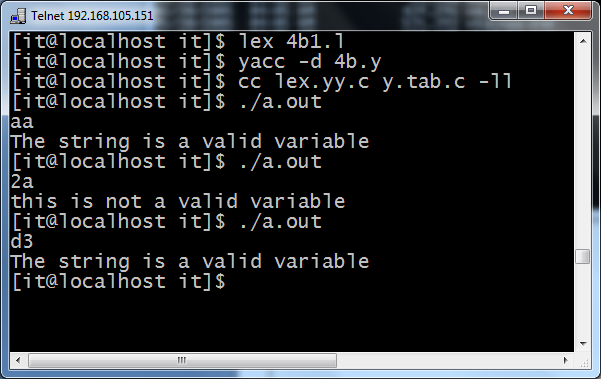
{

printf("this is not a valid variable\n");

exit(0);

}

**Output**

****

**Result:**

The above C program was successfully executed and verified

**Ex.no :7 Implementation of Calculator using LEX and YACC**

**Date :**

**Aim:**

To write a lex and yacc program to implement Calculator.

**Algorithm:**

1. Start the program.

2.Perform the calculation using both the lex and yacc.

3.In the lex tool, if the given expression contains numbers and letters then they are displayed.

4.In the same way, the digits, letters and uminus are identified and displayed using yacctool.

5.The calculation is performed and the result is displayed.

6.Stop the program

**Program: Calci.l**

%{

#include<stdio.h>

#include<math.h>

#include "y.tab.h"

%}

%%

[0-9]+ {

yylval.dval=atoi(yytext);

return NUMBER;

}

[t];

n return 0;

. {return yytext[0];}

%%

void yyerror(char \*str)

{

printf("n Invalid Character...");

}

int main()

{

printf("Enter Expression => ");

yyparse();

return(0);

}

**Program:calci.y**

%{

#include<stdio.h>

int yylex(void);

%}

%union

{

float dval;

}

%token <dval> NUMBER

%left '+' '-'

%left '\*' '/'

%nonassoc UMINUS

%type <dval> exp

%%

state : exp {printf("Answer = %fn",$1);}

;

exp : NUMBER

| exp '+' exp {$$=$1+$3;}

| exp '-' exp {$$=$1-$3;}

| exp '\*' exp {$$=$1\*$3;}

| exp '/' exp {$$=$1/$3;}

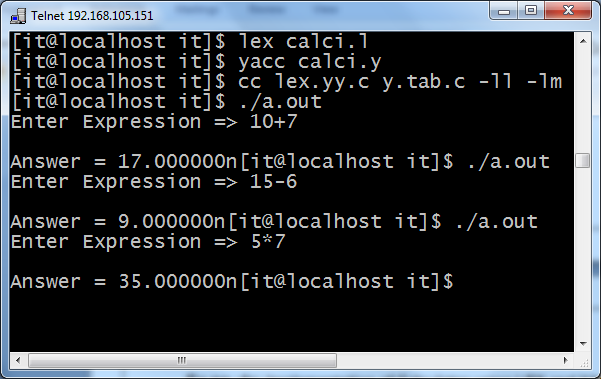
| '('exp')' {$$=$2;}

| '-' exp %prec UMINUS {$$=-$2;}

;

%%

**Output**

****

**Result:**

The above C program was successfully executed and verified

**EX.NO:8 Convert the BNF rules into YACC form and write code to**

**Date: Generate abstract syntax tree.**

**Aim:**

To convert The BNF rules into Yacc form and write code to generate abstract syntax tree

**Algorithm:**

Step1: Start the program.

Step2: Declare the declarations as a header file {include}

Step3: Declare the token digit.

Step4: Define the translations rules like line, expr, term, factor

Line: exp ‘\n’ {print(“\n %d \n”,$1)}

Expr: expr’+’ term ($$=$1=$3}

Term: term ‘+’ factor($$ =$1\*$3}

Factor

Factor:’(‘enter’) ‘{$$ =$2) % %

Step5: define the supporting C routines

Step6: Stop

**Program:**

**<int.l>**

%{

#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];

%%

<**int.y>**

%{

#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 ';'

| 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);

}

yyerror()

{

printf("\n Error on line no:%d",LineNo);

}

Input:

$vi test.c

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;

}

}

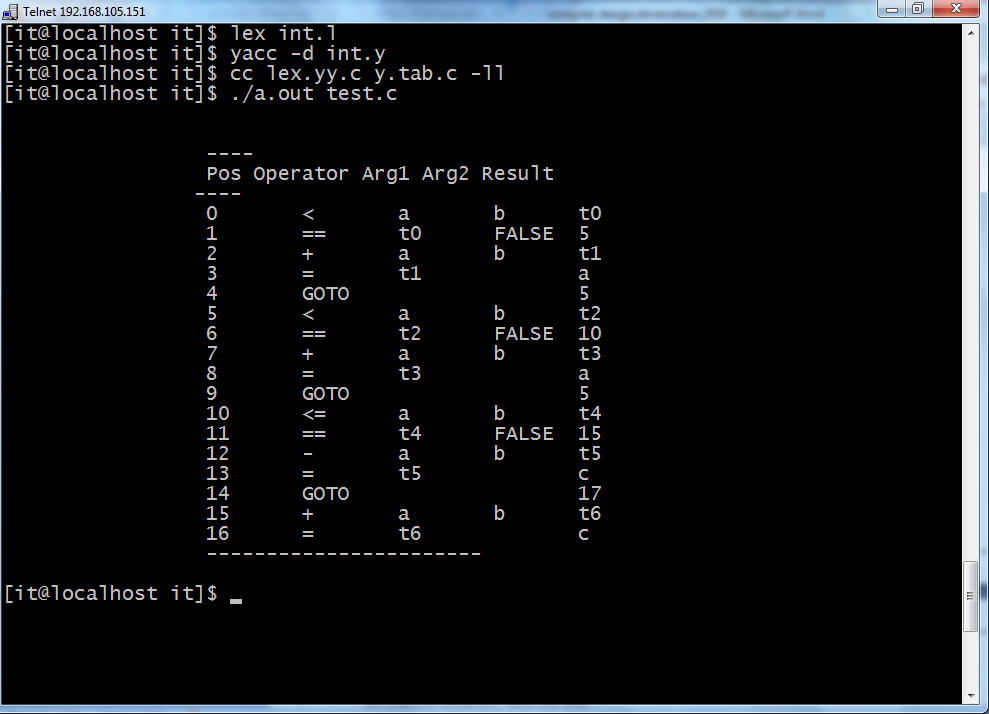
Output:

$lex int.l

$yacc –d int.y

$gcc lex.yy.c y.tab.c –ll –lm

$./a.out test.c



**Result:**

The above C program was successfully executed and verified

**Ex.no : 9 Implement type checking**

**Date:**

**AIM:**

To develop a C program to test whether a given identifier is valid or not.

**ALGORITHM:**

* Read the given input string.
* Check the initial character of the string is numerical or any special character except ‘\_’ then print it is not a valid identifier.
* Otherwise print it as valid identifier if remaining characters of string doesn’t contains any special characters except ‘\_’

PROCEDURE: Go to debug -> run or press CTRL + F9 to run the program.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

void main()

{

char a[10];

int flag, i=1;

clrscr();

printf("

\

n Enter an identifier:");

gets(a);

if(isalpha(a[0]))

flag=1;

else

printf("

\

n Not a valid identifier");

while(a[i]!='

\

0')

{

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

{

flag=0;

break;

}

i++;

}

if(flag==1)

printf("

\

n Valid identifier");

getch();

}

**OUTPUT:**

Input:

Enter an identifier: first

Output:

Valid

identifier

Enter an identifier:1aqw

Not a valid identifier

**Result:**

The above C program was successfully executed and verified

**Ex.no:10 Implement control flow analysis and Data flow Analysis**

**Date:**

**Aim:**

To implement control flow analysis and Data flow Analysis

**Algorithm:**

Step1:

**Program:**

# include<stdio.h>

# include<conio.h>

#include<alloc.h>

#include<string.h>

struct Listnode

{

char data[50];

int leader,block,u\_goto,c\_goto;

struct Listnode \*next;

char label[10],target[10];

}\*temp,\*cur,\*first=NULL,\*last=NULL,\*cur1;

FILE \*fpr;

void createnode(char code[50])

{

temp=(struct Listnode \*)malloc(sizeof(struct Listnode));

strcpy(temp->data,code);

strcpy(temp->label,'\0');

strcpy(temp->target,'\0');

temp->leader=0;

temp->block=0;

temp->u\_goto=0;

temp->c\_goto=0;

temp->next=NULL;

if(first==NULL)

{

first=temp;

last=temp;

}

else

{

last->next=temp;

last=temp;

}}

void printlist()

{

cur=first;

printf("\nMIR code is \n\n");

while(cur!=NULL)

{

printf("\ncode:%s",cur->data);

printf("\nleader:%d\t",cur->leader);

printf("block:%d\t",cur->block);

printf("u\_goto:%d\t",cur->u\_goto);

printf("c\_goto:%d\t",cur->c\_goto);

printf("label:%s\t",cur->label);

printf("target:%s\n",cur->target);

cur=cur->next;

}}

void main()

{

char codeline[50];

char c,dup[50],target[10];

char \*substring,\*token;

int i=0,block,block1;

int j=0;

fpr= fopen("input.txt","r");

clrscr();

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

{

if(c!='\n')

{

codeline[i]=c;

i++;

}

else

{

codeline[i]='\0';

createnode(codeline);

i=0;

}}

//create last node

codeline[i]='\0';

createnode(codeline);

fclose(fpr);

// printlist();

// find out leaders,conditional stmts

cur=first;

cur->leader=1;

while(cur!=NULL)

{

substring=strstr((cur->data),"if");

if(substring==NULL)

{

if((strstr((cur->data),"goto"))!=NULL)

{

cur->u\_goto=1;

(cur->next)->leader=1;

}

}

else

{

cur->c\_goto=1;

(cur->next)->leader=1;

}

substring=strstr((cur->data),":");

if(substring!=NULL)

{

cur->leader=1;

}

substring=strstr((cur->data),"call");

if(substring!=NULL)

{

cur->leader=1;

}

if(strstr(cur->data,"return")!=NULL)

{

cur->leader=1;

(cur->next)->leader=1;

}

cur=cur->next;

}

//to find labels and targets

cur=first;

while(cur!=NULL)

{

if((cur->u\_goto==1)||(cur->c\_goto==1))

{

substring=strstr(cur->data,":");

if(substring!=NULL)

{

token=strstr(substring,"L" );

if(token!=NULL)

strcpy(cur->target,token);

}

else

{

substring=strstr(cur->data,"L");

if(substring!=NULL)

strcpy(cur->target,substring);

}

}

if(strstr(cur->data,":")!=NULL)

{

strcpy(dup,cur->data);

token=strtok(dup,":");

// printf("\ntoken:%s",token);

if(token!=NULL)

strcpy(cur->label,token);

}

cur=cur->next;

}

// printlist();

//to identify blocks

cur=first;

while(cur!= NULL)

{

cur=cur->next;

if((cur->leader)==1)

{

j++;

cur->block=j;

}

else

cur->block=j;

}

// printlist();

// print basic blocks

printf("\n\n......Basic Blocks......\n");

cur=first;

j=0;

printf("\nBlock %d:",j);

while(cur!=NULL)

{

if ((cur->block)==j)

{

printf("%s",cur->data);

printf("\n\t");

cur=cur->next;

}

else

{

j++;

printf("\nBlock %d:",j);

}}

//to output the control flow from each block

printf ("\t\t.......Control Flow.......\n\n");

cur=first;

i=0;

while(cur!=NULL)

{

if((cur->block)!=(cur->next)->block)

{

block=cur->block;

if(cur->u\_goto==1)

{

strcpy(target,cur->target);

cur1=first;

while(cur1!=NULL)

{

if(strcmp(cur1->label,target)==0)

{

block1=cur1->block;

printf("\t\tBlock%d---------->Block%d\n",block,block1);

}

cur1=cur1->next;

}

}

else if(cur->c\_goto==1)

{

strcpy(target,cur->target);

cur1=first;

while(cur1!=NULL)

{

if(strcmp(cur1->label,target)==0)

{

block1=cur1->block;

printf("\t\tBlock%d---TRUE--->Block%d---FALSE--->Block%d\n",block,block1,(block+1));

}

cur1=cur1->next;

}

}

else if(strstr(cur->data,"return")==NULL)

{

printf("\t\tBlock%d---------->Block%d\n",block,(block+1));

}

else

printf("\t\tBlock%d---------->NULL\n",block);

}

cur=cur->next;

}

cur=last;

block= cur->block;

printf("\t\tBlock%d--------->NULL",block);

getch();

}

**Input file: Input.txt**

m <- 0

v <- 0

L1 : if v < n goto L5

r <- v

s <- 0

return

L2 : if r >= n goto L3

v <- v + 1

goto L1

L3 : x = a

s <- s + x

if s <= m goto L2

call P

L4 : r <- r + 1

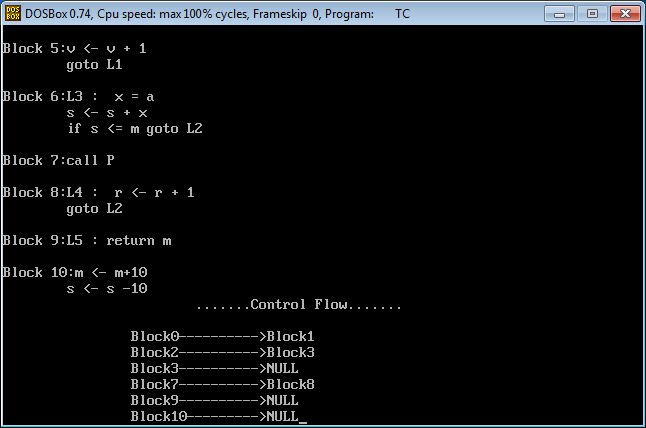
goto L2

L5 : return m

m <- m+10

s <- s -10

**Output:**



**Result:**

The above C program was successfully executed and verified

**Ex. no: 11 Implement any one storage allocation strategies(Heap, Stack, Static)**

**Date:**

**Aim:**

To implement storage allocation strategies using Static.

**Algorithm:**

Step1: Start the program

Step2: Define the pre-processor MAXNUM as 3

Step3: define the sum\_up(void) function.

Step4: Inside main function declare count and initialize it to 0.

Step5: Iterate the loop till count <MAXNUM and invoke sum\_up().

Step6: Inside sum\_up() function declare sum as static variable and get the the number and sum it.

Step7: Print the sum value.

Step8: Stop the program.

**Program:**

#include <stdio.h>

#define MAXNUM 3

void sum\_up(void);

int main()

{

int count;

printf("\n\*\*\*\*\*static storage\*\*\*\*\*\n");

printf("Key in 3 numbers to be summed ");

for(count = 0; count < MAXNUM; count++)

sum\_up();

printf("\n\*\*\*\*\*COMPLETED\*\*\*\*\*\n");

return 0;

}

void sum\_up(void)

{

/\* at compile time, sum is initialized to 0 \*/

static int sum = 0;

int num;

printf("\nEnter a number: ");

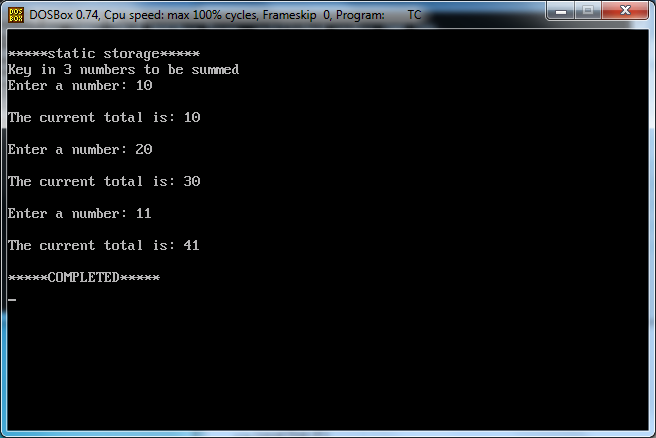
scanf("%d", &num);

sum += num;

printf("\nThe current total is: %d\n", sum);

}

Output:



**Result:**

The above C program was successfully executed and verified

**Ex.no: 12 Construction of DAG**

**Date:**

**Aim:**

To construct DAG for the given expression.

**Algorithm for labeling the nodes of tree(DAG):**

**1. For Leaf Nodes**  
           Assign label 1 to left child node and label 0 to right child node.  
**2. For Interior Nodes**  
          Case 1: If Node's children's labels are different, then   
          Node's Label = Maximum among Children's Labels.  
          Case 2: If Node's children's labels are same, then  
          Node's Label = (Left Child's Label OR Right Child's Label) + 1

**Algorithm for Generating Assembly Code:**

(Say, R is a Stack consists of registers)  
void gencode(Node)  
{  
if Node is intermediate node of tree(DAG)  
{  
    **Case 1:**if Node's left child's label == 1 && Node's right child's label == 0 && Node's left child is leaf node && Node's right child is leaf node  
     {  
       print "MOV Node's left child's data,R[top]"  
       print "op Node's right child's data,R[top]"  
     }  
  
   **Case 2:** else if Node's left child's label >= 1 && Node's right child's label == 0  
     {  
       gencode(Node's left child);  
       print "op Node's right child's data,R[top]"  
     }  
  
**Case 3:** else if Node's left child's label < Node's right child's label  
      {  
       int temp;  
       Swap Register Stack's top and second top element;  
       gencode(Node's right child);  
       temp=pop();  
       gencode(Node's left child);  
       push(temp);   
       Swap Register Stack's top and second top element;   
       print "op R[top-1],R[top]"  
     }  
  
**Case 4:** else if Node's left child's label >= Node's right child's label  
      {  
       int temp;  
       gencode(Node's left child);  
       temp=pop();  
       gencode(Node's right child);  
       push(temp);  
       print "op R[top-1],R[top]"  
     }      
  
}  
  
else if Node is leaf node and it is left child of it's immediate parent  
    {  
  print "MOV Node's data,R[top]"  
    }  
  
}

**Program:**

#include<stdlib.h>

#include<iostream>

using namespace std;

/\* We will implement DAG as Strictly Binary Tree where each node has zero or two children \*/

struct bin\_tree

{

char data;

int label;

struct bin\_tree \*right, \*left;

};

typedef bin\_tree node;

class dag

{

private:

/\* R is stack for storing registers \*/

int R[10];

int top;

/\* op will be used for opcode name w.r.t. arithmetic operator e.g. ADD for + \*/

char \*op;

public:

void initializestack(node \*root)

{

/\* value of top = index of topmost element of stack R = label of Root of tree(DAG) minus one \*/

top=root->label - 1;

**/\* Allocating Stack Registers \*/**

int temp=top;

for(int i=0;i<=top;i++)

{

R[i]=temp;

temp--;

}

}

**/\* insertnode() and insert() functions are for adding nodes to tree(DAG) \*/**

void insertnode(node \*\*tree,char val)

{

node \*temp = NULL;

if(!(\*tree))

{

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

temp->left = temp->right = NULL;

temp->data = val;

temp->label=-1;

\*tree = temp;

}

}

void insert(node \*\*tree,char val)

{

char l,r;

int numofchildren;

insertnode(tree, val);

cout<< "\nEnter number of children of " << val <<" :";

cin>> numofchildren;

if(numofchildren==2)

{

cout<< "\nEnter Left Child of " << val <<" :";

cin>> l;

insertnode(&(\*tree)->left,l);

cout<< "\nEnter Right Child of " << val <<" :";

cin>> r;

insertnode(&(\*tree)->right,r);

insert(&(\*tree)->left,l);

insert(&(\*tree)->right,r);

}

}

**/\* findleafnodelabel() will find out the label of leaf nodes of tree(DAG) \*/**

void findleafnodelabel(node \*tree,int val)

{

if(tree->left != NULL && tree->right !=NULL)

{

findleafnodelabel(tree->left,1);

findleafnodelabel(tree->right,0);

}

else

{

tree->label=val;

}

}

**/\* findinteriornodelabel() will find out the label of interior nodes of tree(DAG) \*/**

void findinteriornodelabel(node \*tree)

{

if(tree->left->label==-1)

{

findinteriornodelabel(tree->left);

}

else if(tree->right->label==-1)

{

findinteriornodelabel(tree->right);

}

else

{

if(tree->left != NULL && tree->right !=NULL)

{

if(tree->left->label == tree->right->label)

{

tree->label=(tree->left->label)+1;

}

else

{

if(tree->left->label > tree->right->label)

{

tree->label=tree->left->label;

}

else

{

tree->label=tree->right->label;

}

}

}

}

}

**/\* function print\_inorder() will print inorder of nodes. Here we are also printing label of each node of tree(DAG) \*/**

void print\_inorder(node \* tree)

{

if (tree)

{

print\_inorder(tree->left);

cout<< tree->data <<" with Label "<< tree->label << "\n";

print\_inorder(tree->right);

}

}

**/\* function swap() will swap the top and second top elements of Register stack R \*/**

void swap()

{

int temp;

temp=R[0];

R[0]=R[1];

R[1]=temp;

}

**/\* function pop() will remove and return topmost element of stack \*/**

int pop()

{

int temp=R[top];

top--;

return temp;

}

**/\* function push() will increment top by one and will insert element at top position of Register stack \*/**

void push(int temp)

{

top++;

R[top]=temp;

}

**/\* nameofoperation() will return opcode w.r.t. arithmetic operator \*/**

void nameofoperation(char temp)

{

switch(temp)

{

case '+': op =(char \*)"ADD"; break;

case '-': op =(char \*)"SUB"; break;

case '\*': op =(char \*)"MUL"; break;

case '/': op =(char \*)"DIV"; break;

}

}

/\* gencode() will generate Assembly code w.r.t. labels of tree(DAG) \*/

void gencode(node \* tree)

{

if(tree->left != NULL && tree->right != NULL)

{

if(tree->left->label == 1 && tree->right->label == 0 && tree->left->left==NULL && tree->left->right==NULL && tree->right->left==NULL && tree->right->right==NULL)

{

cout << "MOV "<< tree->left->data << "," << "R[" << R[top] << "]\n";

nameofoperation(tree->data);

cout << op << " " << tree->right->data << ",R[" << R[top] << "]\n";

}

else if(tree->left->label >= 1 && tree->right->label == 0)

{

gencode(tree->left);

nameofoperation(tree->data);

cout << op << " " << tree->right->data << ",R[" << R[top] << "]\n";

}

else if(tree->left->label < tree->right->label)

{

int temp;

swap();

gencode(tree->right);

temp=pop();

gencode(tree->left);

push(temp);

swap();

nameofoperation(tree->data);

cout<< op << " " << "R[" << R[top-1] <<"],R[" << R[top] << "]\n";

}

else if(tree->left->label >= tree->right->label)

{

int temp;

gencode(tree->left);

temp=pop();

gencode(tree->right);

push(temp);

nameofoperation(tree->data);

cout<< op << " " << "R[" << R[top-1] << "],R[" << R[top] <<"]\n";

}

}

else if(tree->left == NULL && tree->right == NULL && tree->label == 1)

{

cout << "MOV "<< tree->data << ",R[" << R[top] << "]\n";

}

}

**/\* deltree() will free the memory allocated for tree(DAG) \*/**

void deltree(node \* tree)

{

if (tree)

{

deltree(tree->left);

deltree(tree->right);

free(tree);

}

}

};

**/\* Program execution will start from main() function \*/**

int main()

{

node \*root;

root = NULL;

node \*tmp;

char val;

int i,temp;

dag d;

/\* Inserting nodes into tree(DAG) \*/

cout<< "\nEnter root of tree:";

cin>> val;

d.insert(&root,val);

/\* Finding Labels of Leaf nodes \*/

d.findleafnodelabel(root,1);

/\* Finding Labels of Interior nodes \*/

while(root->label == -1)

d.findinteriornodelabel(root);

/\* Initializing Stack contents and top variable \*/

d.initializestack(root);

/\* Printing inorder of nodes of tree(DAG) \*/

cout<< "\nInorder Display:\n";

d.print\_inorder(root);

/\* Printing assembly code w.r.t. labels of tree(DAG) \*/

cout<< "\nAssembly Code:\n";

d.gencode(root);

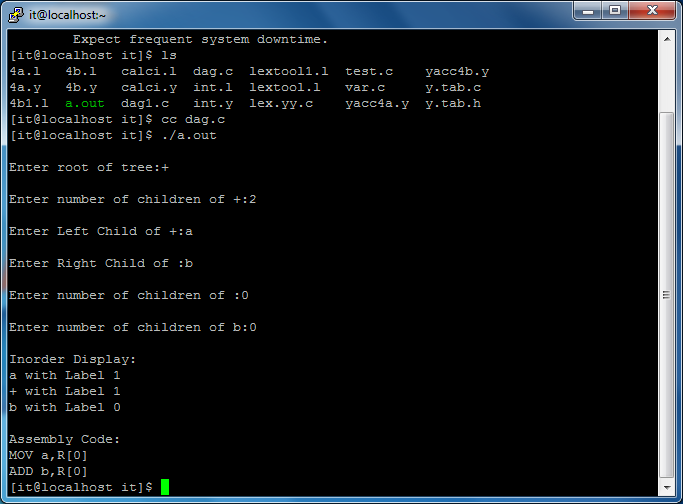
/\* Deleting all nodes of tree \*/

d.deltree(root);

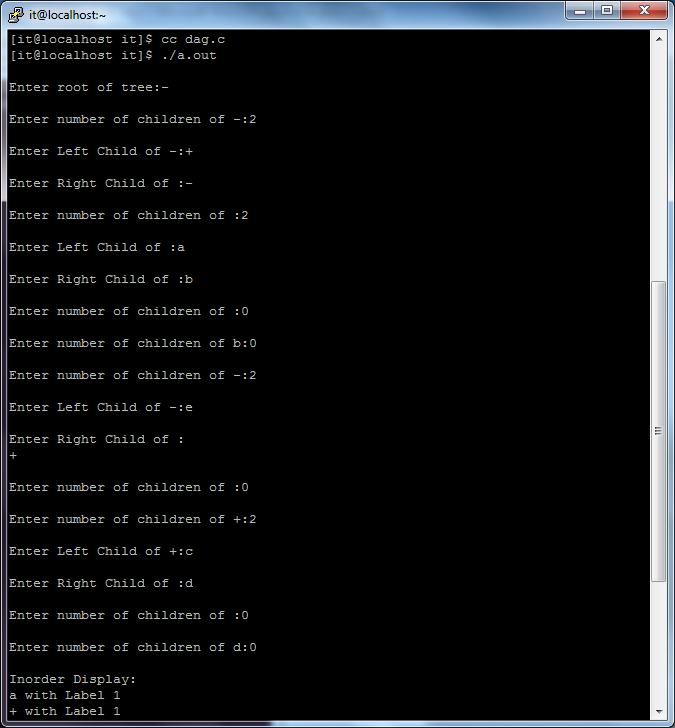
return 0;

}

**Output1:**



**OUTPUT2:**





**Result:**

The above C program was successfully executed and verified

**Ex.no: 13 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.**

**Aim:**    To write a C program to implement the code generation algorithm.

**Algorithm:**

Input: Set of three address code sequence.

Output: Assembly code sequence for three address codes (opd1=opd2, op, opd3).

Method:

1. Start the program
2. Get address code sequence.
3. Determine current location of 3 using address (for 1st operand).
4. If current location not already exist generate move (B,O).
5. Update address of A(for 2nd operand).
6. If current value of B and () is null, exist.
7. If they generate operator () A,3 ADPR.
8. Store the move instruction in memory.
9. Stop.

**Program:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

#include<graphics.h>

typedef struct

{

char var[10];

int alive;

}

regist;

regist preg[10];

void substring(char exp[],int st,int end)

{

int i,j=0;

char dup[10]="";

for(i=st;i<end;i++)

dup[j++]=exp[i];

dup[j]='0';

strcpy(exp,dup);

}

int getregister(char var[])

{

int i;

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

if(preg[i].alive==0) {

strcpy(preg[i].var,var);

break;

}}

return(i);

}

void getvar(char exp[],char v[])

{

int i,j=0;

char var[10]="";

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

if(isalpha(exp[i]))

var[j++]=exp[i];

else

break;

strcpy(v,var);

}

void main()

{

char basic[10][10],var[10][10],fstr[10],op;

int i,j,k,reg,vc,flag=0;

clrscr();

printf("\nEnter the Three Address Code:\n");

for(i=0;;i++)

{

gets(basic[i]);

if(strcmp(basic[i],"exit")==0)

break;

}

printf("\nThe Equivalent Assembly Code is:\n");

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

{

getvar(basic[j],var[vc++]);

strcpy(fstr,var[vc-1]);

substring(basic[j],strlen(var[vc-1])+1,strlen(basic[j]));

getvar(basic[j],var[vc++]);

reg=getregister(var[vc-1]);

if(preg[reg].alive==0)

{

printf("\nMov R%d,%s",reg,var[vc-1]);

preg[reg].alive=1;

}

op=basic[j][strlen(var[vc-1])];

substring(basic[j],strlen(var[vc-1])+1,strlen(basic[j]));

getvar(basic[j],var[vc++]);

switch(op)

{

case '+': printf("\nAdd"); break;

case '-': printf("\nSub"); break;

case '\*': printf("\nMul"); break;

case '/': printf("\nDiv"); break;

}

flag=1;

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

{

if(strcmp(preg[k].var,var[vc-1])==0)

{

printf("R%d, R%d",k,reg);

preg[k].alive=0;

flag=0;

break;

}

}

if(flag)

{

printf(" %s,R%d",var[vc-1],reg);

printf("\nMov %s,R%d",fstr,reg);

}

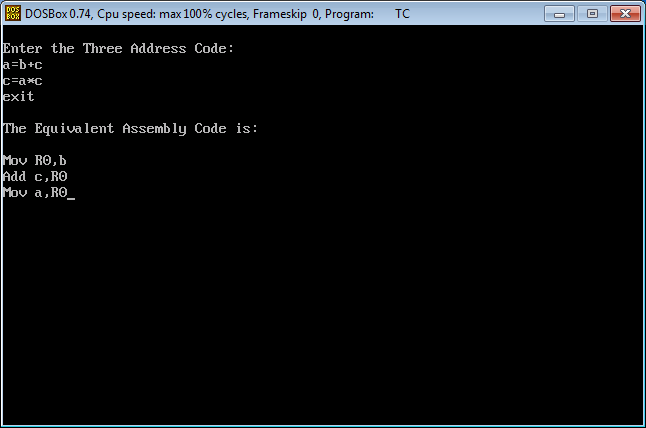
strcpy(preg[reg].var,var[vc-3]);

getch();

}

}

**Output**



**Result:**

The above C program was successfully executed and verified

**EX.NO:14                 IMPLEMENTATION OF SIMPLE CODE OPTIMIZATION TECHNIQUES  
Date:**

**AIM:**       
    To write a C program to implement the code optimization algorithm.  
 **ALGORITHM:**  
    The code generation algorithm takes as input a sequence of three – address statements constituting a basic block. For each three – address statement of the form  x := y op z we perform the following actions:  
    
1.  Invoke a function getreg to determine the location L where the result of the computation y op z should be stored. L will usually be a register, but it could also be a memory location.

2.We shall describe getreg shortly, L to place a copy of y in L.′ if the value of y is currently both in memory and a register. If the value of y is not already in L, generate the instruction MOV y′, (one of) the current location(s) of y. prefer the register for y′.

3. Consult the address descriptor for y to determine yis a current location of z. Again, prefer a register to a memory location if z is in both.

4.Update the address descriptor of x to indicate that x is in location L. If L is a register, update its descriptor to indicate that it contains the value of x, and remove x from all other register descriptors.′, L where z′.

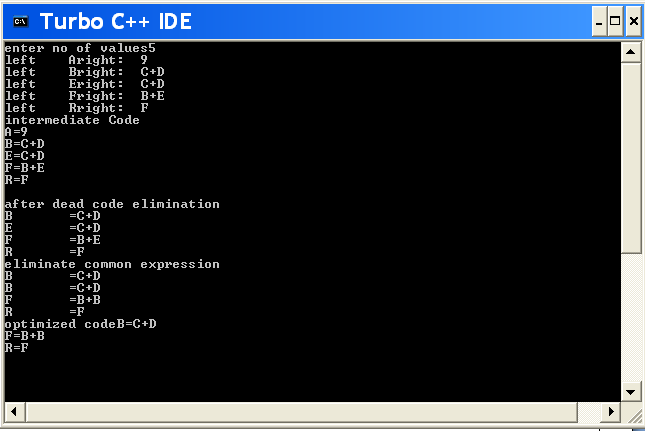
5. Generate the instruction OP z  
6. If the current values of y and/or z have no next users, are not live on exit from the block, and are in register descriptor to indicate that, after execution of x := y op z, those registers no longer will contain y and/or z, respectively.

Input: Set of ‘L’ values with corresponding ‘R’ values.

Output: Intermediate code & Optimized code after eliminating common expressions.

**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 no of values");  
scanf("%d",&n);  
for(i=0;i<n;i++)  
{  
printf("left\t");  
op[i].l=getche();  
printf("right:\t");  
scanf("%s",op[i].r);  
}  
printf("intermediate Code\n") ;  
for(i=0;i<n;i++)  
{  
printf("%c=",op[i].l);  
printf("%s\n",op[i].r);  
}  
for(i=0;i<n-1;i++)  
{  
temp=op[i].l;  
for(j=0;j<n;j++)  
{  
p=strchr(op[j].r,temp);  
if(p)  
{  
pr[z].l=op[i].l;  
strcpy(pr[z].r,op[i].r);  
z++ ;  
  
}} }  
pr[z].l=op[n-1].l;  
strcpy(pr[z].r,op[n-1].r);  
z++;  
printf("\nafter dead code elimination\n");  
for(k=0;k<z;k++)  
{  
  
printf("%c\t=",pr[k].l);  
printf("%s\n",pr[k].r);  
}  
  
//sub expression elimination  
for(m=0;m<z;m++)  
{  
tem=pr[m].r;  
for(j=m+1;j<z;j++)  
{  
p=strstr(tem,pr[j].r);  
if(p)  
{  
t=pr[j].l;  
pr[j].l=pr[m].l     ;  
for(i=0;i<z;i++)  
{  
l=strchr(pr[i].r,t) ;  
if(l)  
{  
a=l-pr[i].r;  
//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);  
}  
// duplicate production elimination  
  
for(i=0;i<z;i++)  
{  
for(j=i+1;j<z;j++)  
{  
q=strcmp(pr[i].r,pr[j].r);  
if((pr[i].l==pr[j].l)&&!q)  
  
{  
    pr[i].l='\0';  
    strcpy(pr[i].r,'\0');  
 }}  
}  
printf("optimized code");  
for(i=0;i<z;i++)  
{  
if(pr[i].l!='\0')  
{  
printf("%c=",pr[i].l);  
printf("%s\n",pr[i].r);  
}  
}  
getch();  
}

**OUTPUT:**



**Result:**

The above C program was successfully executed and verified

**Ex.No: 15 IMPLEMENTATION OF SHIFT-REDUCED PARSING ALGORITHMS**

**Date:**

**AIM:**

To write a program for implementing Shift Reduce Parsing using C.

**ALGORITHM:**

* 1. Get the input expression and store it in the input buffer.
  2. Read the data from the input buffer one at the time.
  3. Using stack and push & pop operation shift and reduce symbols with respect to production rules available.
  4. Continue the process till symbol shift and production rule reduce reaches the start symbol.
  5. Display the Stack Implementation table with corresponding Stack actions with input symbols.

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<string.h>

char ip\_sym[15],stack[15];

int ip\_ptr=0,st\_ptr=0,len,i;

char temp[2],temp2[2];

char act[15];

void check();

void main()

{

clrscr();

printf("\n\t\t SHIFT REDUCE PARSER\n");

printf("\n GRAMMER\n");

printf("\n E->E+E\n E->E/E");

printf("\n E->E\*E\n E->a/b");

printf("\n enter the input symbol:\t");

gets(ip\_sym);

printf("\n\t stack implementation table");

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

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

printf("\n $\t\t%s$\t\t\t--",ip\_sym);

strcpy(act,"shift");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

len=strlen(ip\_sym);

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

{

stack[st\_ptr]=ip\_sym[ip\_ptr];

stack[st\_ptr+1]='\0';

ip\_sym[ip\_ptr]=' ';

ip\_ptr++;

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

strcpy(act,"shift");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

check();

st\_ptr++;

}

st\_ptr++;

check(); }

void check()

{int flag=0;

temp2[0]=stack[st\_ptr];

temp2[1]='\0';

if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))

{

stack[st\_ptr]='E';

if(!strcmpi(temp2,"a"))

printf("\n $%s\t\t%s$\t\t\tE->a",stack,ip\_sym);

else

printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip\_sym);

flag=1;

}

if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"\*"))||(!strcmpi(temp2,"/")))

{flag=1;

}

if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E\*E")))

{ strcpy(stack,"E");

st\_ptr=0;

if(!strcmpi(stack,"E+E"))

printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip\_sym);

else

if(!strcmpi(stack,"E\E"))

printf("\n $%s\t\t%s$\t\t\tE->E\E",stack,ip\_sym);

else

if(!strcmpi(stack,"E\*E"))

printf("\n $%s\t\t%s$\t\t\tE->E\*E",stack,ip\_sym);

else

printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip\_sym);

flag=1;

} if(!strcmpi(stack,"E")&&ip\_ptr==len)

{

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

getch();

exit(0);

}

if(flag==0)

{

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

exit(0); }

return;

}

**OUTPUT:** SHIFT REDUCE PARSER

GRAMMER

E->E+E

E->E/E

E->E\*E

E->a/b

Enter the input symbol: a+b

Stack Implementation Table

Stack Input Symbol Action

------- ----------------- ---------

$ a+b$ --

$a +b$ shift a

$E +b$ E->a

$E+ b$ shift +

$E+b $ shift b

$E+E $ E->b

$E $ E->E+E

$E $ ACCEPT

**RESULT:**

Thus the program for implementation of Shift Reduce parsing algorithm is executed and verified

**Ex.No:16** **CONSTRUCTION OF LR-PARSING TABLE**

**Date:**

**AIM:**

To write a program for construction of LR Parsing table using C.

**ALGORITHM:**

1. Get the input expression and store it in the input buffer.
2. Read the data from the input buffer one at the time and convert in to corresponding Non Terminal using production rules available.
3. Perform push & pop operation for LR parsing table construction.
4. Display the result with conversion of corresponding input symbols to production and production reduction to start symbol. No operation performed on the operator.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

char stack[30];

int top=-1;

void push(char c)

{

top++;

stack[top]=c;

}

char pop()

{

char c;

if(top!=-1)

{

c=stack[top];

top--;

return c;

}

return'x';

}

void printstat()

{

int i;

printf("\n\t\t\t $");

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

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

}

void main()

{

int i,j,k,l;

char s1[20],s2[20],ch1,ch2,ch3;

clrscr();

printf("\n\n\t\t LR PARSING");

printf("\n\t\t ENTER THE EXPRESSION");

scanf("%s",s1);

l=strlen(s1);

j=0;

printf("\n\t\t $");

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

{

if(s1[i]=='i' && s1[i+1]=='d')

{

s1[i]=' ';

s1[i+1]='E';

printstat(); printf("id");

push('E');

printstat(); }

else if(s1[i]=='+'||s1[i]=='-'||s1[i]=='\*' ||s1[i]=='/' ||s1[i]=='d')

{

push(s1[i]);

printstat();}

}printstat();

l=strlen(s2);

while(l)

{

ch1=pop();

if(ch1=='x')

{

printf("\n\t\t\t $");

break;}

if(ch1=='+'||ch1=='/'||ch1=='\*'||ch1=='-'){

ch3=pop();

if(ch3!='E'){

printf("errror");

exit();}

else{

push('E');

printstat();

}}ch2=ch1;}getch(); }

**OUTPUT:**

LR PARSING

ENTER THE EXPRESSION

id+id\*id-id

$

$id

$E

$E+

$E+id

$E+E

$E+E\*

$E+E\*id

$E+E\*E

$E+E\*E-

$E+E\*E-id

$E+E\*E-E

$E+E\*E-E

$E+E\*E

$E

$

**RESULT:**

Thus the program for construction of LR Parsing table is executed and verified.

Ex.No:17 IMPLEMENTATION OF CLR PARSER  
Date:

AIM:

Design CLR bottom up parser for the above language

ALGORITHM:

Step1: Start

Step2: Initially the parser has s0 on the stack where s0 is the initial state and w$ is in

buffer

Step3: Set ip point to the first symbol of w$

Step4: repeat forever, begin

Step5: Let S be the state on top of the stack and a symbol pointed to by ip

Step6: If action [S, a] =shift S then begin

Push S1 on to the top of the stack

Advance ip to next input symbol

Step7: Else if action [S, a], reduce A->B then begin

Pop 2\* |B| symbols of the stack

Let S1 be the state now on the top of the stack

Step8: Output the production AB

End

Step9: else if action [S, a]=accepted, then return

Else

Error()

End

Step10: Stop

**PROGRAM:**

// \*\*\*\*\*\*\*\*\*\*\* IMPLEMENTATION OF CLR PARSING PROGRAM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

#include<stdio.h>

#include<conio.h>

#include<string.h>

#define MAX 50

void push(char item);

char pop(void);

int top=-1;

int call(char c);

char stack[MAX],input[10],str2[15],str1[8]="",c;

void prn(int j)

{

int i;

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

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

}

void prnstack(int top)

{

int i;

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

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

}

void main()

{

char str1[6],\*cmp="",c[8]="";

int i=0,cn=0, k,j;

FILE \*ptr, \*gptr;

clrscr();

printf("\n\n enter the expression :\n");

scanf("%s",input);

push('0');

printf("\n\n\t STACK \t\t COMPARISION \t\t OUTPUT \n\n");

do

{

printf("");

prnstack(top);

printf("\t\t");

prn(i);

if(strcmp(cmp,"1$")==0)

{

strcpy(str2,"accepted");

printf("\n\nthe input is accepted");

getch();

exit(0);

}

else

{

cmp[0]=stack[top];

cmp[1]=input[i];

cmp[2]='\0';

if((ptr=fopen("d:\\lrtable.doc","r"))==NULL)

printf("\n\n FILE CAN NOT BE OPEN");

else

{

while(!feof(ptr))

{

fscanf(ptr, "%s%s",str1,str2);

if(strcmp(str1,cmp)==0)

{

if(str2[0]=='s')

{

push(input[i]);

push(str2[1]);

i++;

break;

}

else if(str2[0]=='r')

62 | P a g e

{

cn=call(str2[1]);

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

pop();

c[0]=stack[top];

push(str2[0]);

c[1]=stack[top];

c[2]='

\0';

if(strcmp(c,"0E")==0)

push('1');

else if(strcmp(c,"0T")==0)

push('2');

else if(strcmp(c,"0F")==0)

push('3');

else if(strcmp(c,"0E")==0)

push('8');

else if(strcmp(c,"0T")==0)

push('2');

else if(strcmp(c,"0F")==0)

push('3');

else if(strcmp(c,"0T")==0)

push('9');

else if(strcmp(c,"0F")==0)

push('3');

else if(strcmp(c,"0F")==0)

push('t');

}

else if(strcmp(str2,"0")==0) {

printf("

\

n

\n the string is not accepted");

break; }

}

}

}

fclose(ptr);

}

printf("

\

t

\t%s

\

t

\

t

\n",cmp,str2);

}

while(input[i]!='

\0');

getch(); }

int call(char c) {

int count =0;

switch(c)

{

case 1: strcpy(str2,"E

->E+T");

count=3;

break ;

case 2: strcpy(str2,"E->T");

count=1;

break;

case 3: strcpy(str2,"T->T\*F");

count=3;

break;

case 4: strcpy(str2,"T->F");

count=1;

break;

case 5: strcpy(str2,"F->(E)");

count=3;

break;

case 6: strcpy(str2,"F->id");

count=1;

break;

}

return count ;

}

void push(char item)

{

if(top==MAX)

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

else

{

top=top+1;

stack[top]=item;

}

}

char pop(void)

{

char item;

if(top==-1)

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

else

{

item=stack[top];

top--;

}

return item;

}

OUTPUT:

CLR PARSER TABLE

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Stat es | **ACTION** | | | | | | **GOTO** | | |
| id | + | \* | ( | ) | $ | E | T | F |
| 0 | S5 |  |  | S4 |  |  | 1 | 2 | 3 |
| 1 |  |  |  |  |  | ACC |  |  |  |
| 2 |  | R2 | S7 |  | R2 | R2 |  |  |  |
| 3 |  | R4 | R4 |  | R4 | R4 |  |  |  |
| 4 | S5 |  |  | S4 |  |  | 8 | 2 | 3 |
| 5 |  | R6 | R6 |  | R6 | R6 |  |  |  |
| 6 | S5 |  |  | S4 |  |  |  | 9 | 3 |
| 7 |  | S5 |  | S4 |  |  |  |  | 10 |
| 8 |  |  | S6 |  |  |  | S11 |  |  |
| 9 |  |  | R1 | S7 |  | R1 | R1 |  |  |
| 10 |  |  | R3 | R3 |  | R3 | R3 |  |  |
| 11 |  |  | R5 | R5 |  | R5 | R5 |  |  |

OUTPUT:

id\*(id+id)$ Grammer accepted

**RESULT:**

Thus the program for construction of CLR Parsing table is executed and verified.