**Sona College Of Technology**

**NAME:** KARTHICK. S

**REGISTER NO:** 1518102052

**COURSE:** PRINCIPLES OF COMPILER

DESIGN LAB

**BRANCH:** COMPUTER SCIENCE AND ENGINEERING

**DATE:** 29/05/2021

|  |  |  |
| --- | --- | --- |
| **EX.NO** | **TOPIC** | **Pg.No** |
| **1** | Construction Of NFA | 3 |
| **2** | Minimized DFA | 9 |
| **3** | Implementation of Lexical analyser using LEX | 12 |
| **4** | Parser Implementation Using YACC and LEX | 17 |
| **5** | Recursive Descent Parser | 22 |
| **6** | Operator Precedence Parser | 25 |
| **7** | Symbol Table | 29 |
| **8** | Shift Reduce Parsing | 39 |
| **9** | LR Parser | 44 |
| **10** | Intermediate Code Generation | 55 |
| **11** | Code Optimization | 62 |

**Ex.no:1**  **Construction Of NFA**

**­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:**

To write a program for constructing an NFA from given regular expression using C.

**ALGORITHM:**

**Step 1:** Create a menu for getting four regular expressions input as choice.

**Step 2:** To draw NFA for a, a/b , ab , a\* create a routine for each regular expression.

**Step 3:** For converting from regular expression to NFA, certain transition had been made based on choice of input at the run time.

**Step 4:** Each of the NFA will be displayed is sequential order.

**PROGRAM:**

#include<stdio.h>

#include<string.h>

void main()

{

char re[20];

int q[20][3],i,j,len,a,b;

for(a=0;a<20;a++)

{

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

{

q[a][b]=0;

}

}

printf("Enter the Regular expression:\n");

scanf("%s",re);

len=strlen(re);

i=0;j=1;

while(i<len)

{

if(re[i]=='a'&&re[i+1]!='/'&&re[i+1]!='\*')

{

q[j][0]=j+1;j++;

}

if(re[i]=='b'&&re[i+1]!='/'&&re[i+1]!='\*')

{

q[j][1]=j+1;j++;

}

if(re[i]=='e'&&re[i+1]!='/'&&re[i+1]!='\*')

{

q[j][2]=j+1;j++;

}

if(re[i]=='a'&& re[i+1]=='/'&&re[i+2]=='b')

{

q[j][2]=((j+1)\*10)+(j+3);j++;

q[j][0]=j+1;j++;

q[j][2]=j+3;j++;

q[j][2]=j+1;j++;i=i+2;

}

if(re[i]=='b'&&re[i+1]=='/'&&re[i+2]=='a')

{

q[j][2]=((j+1)\*10)+(j+3);j++;

q[j][1]=j+1;j++;

q[j][2]=j+3;j++;

q[j][0]=j+1;j++;

q[j][2]=j+1;j++;

i=i+2;

}

if(re[i]=='a'&&re[i+1]=='\*')

{

q[j][2]=((j+1)\*10)+(j+3);

j++;

q[j][0]=j+1;

j++;

q[j][2]=((j+1)\*10)+(j-1);

j++

}

if(re[i]=='b'&&re[i+1]=='\*')

{

q[j][2]=((j+1)\*10)+(j+3);

j++;

q[i][j]=j+1;

j++;

q[j][2]=((j+1)\*10)+(j-1);

j++;

}

if(re[i]==')'&&re[i+1]=='\*')

{

q[0][2]=((j+1)\*10)+1;

q[j][2]=((j+1)\*10)+1;

j++;

}

i++;

}

printf("Transition function:\n");

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

{

if(q[i][0]!=0)

printf("\n q[%d,a]-->%d",i,q[i][0]);

if(q[i][1]!=0)

printf("\n q[%d,b]-->%d",i,q[i][1]);

if(q[i][2]!=0)

{

if(q[i][2]<10)

printf("\n q[%d,e]-->%d",i,q[i][2]);

else

printf("\n q[%d,e]-->%d & %d",i,q[i][2]/10,q[i][2]%10);

}

}

}

**OUTPUT:**

Enter the Regular expression:

(a/b)\*

Transition function:

q[0,e]-->6 & 1

q[1,e]-->2 & 4

q[2,a]-->3

q[3,e]-->6

q[4,e]-->5

q[5,e]-->6 & 1

Process returned 7 (0x7) execution time : 15.909 s

Press any key to continue.

**RESULT:**

Thus the Given Program for construction of NFA from given regular expression using C is successfully Verified.

**Ex.no: 2 Minimized DFA for Regular Expression**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:**

To write a program for minimized DFA for given regular expression.

**ALGORITHM:**

**Step 1:** We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called P0.

**Step 2:** Initialize k = 1

**Step 3:** Find Pk by partitioning the different sets of Pk-1. In each set of Pk-1, we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in Pk.

**Step 4**: Stop when Pk = Pk-1 (No change in partition)

**Step 5:** All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in Pk.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<strings.h>

void main()

{

int table[2][2],i,j,l,status=0,success;

char input[100];

printf("Program for implmenting DFA of language (a+aa\*b)\*\n\n\nEnter Input String \n");

table[0][0]=1;

table[0][1]=-1;

table[1][0]=1;

table[1][1]=0;

scanf("%s",input);

l=strlen(input);

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

{

   if(input[i]!='a'&&input[i]!='b')

   {

         printf("Value entered is wrong");

         getch();

         exit(0);

    }

 if(input[i]=='a')

status=table[status][0];

else

status=table[status][1];

if(status==-1)

{

   printf("String not Accepted by this Language");

   break;

}

}

if(i==l)

printf("String Accepted");

getch();

}

**OUTPUT:**

Program for implementing DFA of language (a+aa\*b)\*

Enter Input String

ababababa

String Accepted

**RESULT:**

Thus the program for implementation of minimized DFA for regular expression is implemented successfully.

**Ex.no:3 Implementation of Lexical Analyser Using LEX Tool**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM**:

To write a lex program to implement the lexical analyzer.

**ALGORITHM:**

**Step1**: 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

**Step2**: 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.

**Step3**: 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, pre-processor directive statements etc.

**Step4**: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.

**Step5**: 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.

**Step6**: In user subroutine section, main routine calls yylex(). Yywrap() is used to get more input.

**Step7**: 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:**

lex:

%{

int COMMENT=0;

%}

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

%%

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

int |

float |

char |

double |

while |

for |

do |

if |

break |

continue |

void |

switch |

case |

long |

struct |

const |

typedef |

return |

else |

goto {printf("\n\t%s is a KEYWORD",yytext);}

"/\*" {COMMENT = 1;}

"\*/" {COMMENT = 0;}

{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}

\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}

\} {if(!COMMENT) printf("\n BLOCK ENDS");}

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

\".\*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}

[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}

\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}

\( ECHO;

= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}

\<= |

\>= |

\< |

== |

\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}

%%

int main(int argc,char \*\*argv)

{

if (argc > 1)

{

FILE \*file;

file = fopen(argv[1],"r");

if(!file)

{

printf("could not open %s \n",argv[1]);

exit(0);

}

yyin = file;

}

yylex();

printf("\n\n");

return 0;

} int yywrap()

{

return 0;

}

second.c

#include <stdio.h>

int main()

{

int number;

printf("Enter an integer: ");

scanf("%d", &number);

if(number % 2 == 0)

printf("%d is even.", number);

else

printf("%d is odd.", number);

return 0;

}

**OUTPUT:**

C:\Users\18csea52\Desktop\pcd>flex first.l

C:\Users\18csea52\Desktop\pcd>gcc lex.yy.c

C:\Users\18csea52\Desktop\pcd>a.exe second.c

#include <stdio.h> is a PREPROCESSOR DIRECTIVE

        int is a KEYWORD

FUNCTION

        main(

        )

 BLOCK BEGINS

        int is a KEYWORD

 number IDENTIFIER;

FUNCTION

        printf(

        "Enter an integer: " is a STRING

        );

FUNCTION

        scanf(

        "%d" is a STRING, &

 number IDENTIFIER

        );

FUNCTION

        if(

 number IDENTIFIER %

        2 is a NUMBER

        == is a RELATIONAL OPERATOR

        0 is a NUMBER

        )

FUNCTION

        printf(

        "%d is even." is a STRING,

 number IDENTIFIER

        );

        else is a KEYWORD

FUNCTION

        printf(

        "%d is odd." is a STRING,

 number IDENTIFIER

        );

        return is a KEYWORD

        0 is a NUMBER;

 BLOCK ENDS

**RESULT**:

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

**Ex.no:4 Implementation of Parser Using YACC and LEX**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:**

To write a program for implementing a  computation of the given expression using semantic rules of the YACC tool and LEX.

**ALGORITHM:**

**Step1**: A Yacc source program has three parts as follows:

       Declarations %%  translation rules %%  supporting C routines

**Step2**: Declarations Section: This section contains entries that:

 i. Include standard I/O header file.

 ii. Define global variables.

 iii. Define the list rule as the place to start processing.

 iv. Define the tokens used by the parser. v. Define the operators and their precedence.

**Step3**:  Rules Section: The rules section defines the rules that parse the input stream. Each rule of a grammar  production and the associated semantic action.

**Step4**:  Programs Section: The programs section contains the following subroutines. Because these subroutines areincluded in this file, it is not necessary to use the yacc library when processing this file.

**Step5**:  Main- The required main program that calls the yyparse subroutine to start the program.

**Step6**:  yyerror(s) -This error-handling subroutine only prints a syntax error message.

**Step7**:  yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file includes statements for standard input and output, as programmer file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.

**Step8**: calc.lex contains the rules to generate these tokens from the input stream.

**PROGRAM:**

LEX PART:

%{

#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:**

C:\Users\18csea52\Desktop\Ex-4>yacc -d 4c.y

C:\Users\18csea52\Desktop\Ex-4>lex 4c.l

C:\Users\18csea52\Desktop\Ex-4>gcc lex.yy.c y. tab.c -w

C:\Users\18csea52\Desktop\Ex-4>./a.out

3\*5+4

19.000000

**RESULT:**

Thus the program for implementing a  computation of the given expression using semantic rules of the YACC tool and LEX is obtained.

**Ex.no:5 Recursive Descent Parser**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:**

To Implement the recursive descent parser for the given grammar / language.

**ALGORITHM**:

**Step 1**: Start the program.

**Step 2**: Declare the prototype functions E(), Eprime(), T(), Tprime(), F()

**Step 3**: Read the input string to be parsed.

**Step 4**: Check the productions.

**Step 5**: Compare the terminals and Non-terminals.

**Step 6**: Read the parse string.

**Step 7**: Stop the production.

**PROGRAM**:

#include<stdio.h>

#include<string.h>

#include<ctype.h>

char input[10];

int i,error;

void E();

 void T();

void Eprime();

void Tprime();

void F();

main()

{

    i=0; error=0;

    printf("Enter an arithmetic expression : ");

    gets(input);

    E();

    if(strlen(input)==i&&error==0)

        printf("\nAccepted..!!!\n");

    else

        printf("\nRejected..!!!\n");

}

void E()

{

    T();

    Eprime();

}

void Eprime()

{

    if(input[i]=='+')

    {

        i++;

        T();

        Eprime()

     }

}

void T()

{

    F();

    Tprime();

}

void Tprime()

{

    if(input[i]=='\*')

    {

        i++;

        F();

        Tprime();

    }

}

void F()

{

    if(isalnum(input[i]))

        i++;

    else if(input[i]=='(')

        {

            i++;

            E();

            if(input[i]==')')

                i++;

            else

                error=1;

        }

    else

        error=1;

}

**OUTPUT**:

Enter an arithmetic expression : a+(a\*a)

Accepted..!!!

Enter an arithmetic expression : ++a

Rejected..!!!

**RESULT:**

Thus the recursive descent parser is constructed and the given string is checked.

**EX.NO: 6 Operator Precedence parser**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:**

The program to Develop an operator precedence parser for a given language.

**ALGORITHM:**

**Step 1:** Start

**Step 2:** Enter the operator precedence table

**Step 3**: Enter the input string

**Step 4**: Insert the string into the stack

**Step 5**: Take each identifier token compare with the precedence table and precedence rule, either reduce or shift

**Step 6:** The process stops until the stack becomes empty

**Step 7:** Stop

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

void main()

{

    char stack[20],ip[20],opt[10][10][1],ter[]={"i\*+$"};

    int i,j,k,n=4,top=0,col,row;

    for(i=0;i<n;i++)

    {

        stack[i]=NULL;  ip[i]=NULL;

        for(j=0;j<n;j++)

        {

            if(ter[i]=='i'&&ter[j]=='i')

            {

                opt[i][j][0]='e';

            }

            else if(ter[i]=='i')

                opt[i][j][0]='>';

            else if(ter[j]=='i')

                opt[i][j][0]='<';

            else if(ter[i]=='\*')

                opt[i][j][0]='>';

            else if(ter[j]=='\*')

                opt[i][j][0]='<';

            else if(ter[i]=='$')

                opt[i][j][0]='< ';

            else if(ter[i]=='$'&&ter[j]=='$')

                opt[i][j][0]='a';

            else if(ter[j]=='$')

                opt[i][j][0]='>';

        }

    }

    printf("\n\*\*\*\* OPERATOR PRECEDENCE TABLE \*\*\*\*\n");

    for(i=0;i<n;i++)

    {

        printf("\t%c",ter[i]);

    }

    printf("\n");

    for(i=0;i<n;i++)

    {

        printf("\n%c",ter[i]);

        for(j=0;j<n;j++)

        {

            printf("\t%c",opt[i][j][0]);

        }

    }

    stack[top]='$';

    printf("\nEnter the input string: ");

    scanf("%s",ip);

    i=0;

    printf("\nSTACK\t\t\tINPUT STRING\t\t\tAction\n");

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

    while(i<=strlen(ip)){

        for(k=0;k<n;k++)

        {

            if(stack[top]==ter[k])

                col=k;

            if(ip[i]==ter[k])

                row=k;

        }

        if((stack[top]=='$')&&(ip[i]=='$')){

            printf("String is Accepted\n");

            break;

        }

        else if((opt[col][row][0]=='<')||(opt[col][row][0]=='='))

        {

            stack[++top]=opt[col][row][0];

            stack[++top]=ip[i];

            printf("Shift %c",ip[i]);

            i++;

        }

        else{

            if(opt[col][row][0]=='>'){

                while(stack[top]!='<'){

                    --top;

                }

                top=top-1;

                printf("Reduce");

            }

            else{

                printf("\nString is not accepted");

                break;

            }

        }

        printf("\n");

        for(k=0;k<=top;k++){

            printf("%c",stack[k]);

        }

        printf("\t\t\t");

**for(k=i;k<strlen(ip);k++){**

            printf("%c",ip[k]);

        }

        printf("\t\t\t");

    }

}

**OUTPUT:**

\*\*\*\* OPERATOR PRECEDENCE TABLE \*\*\*\*

        i       \*       +       $

i       e       >       >      >

\*      <       >       >      >

+      <       <               >

$      <       <

Enter the input string: i\*i$

STACK                   INPUT STRING            ACTION

$                      i\*i$                    Shift i

$<i                     \*i$                     Reduce

$                       \*i$                     Shift \*

$<\*                     i$                      Shift i

$<\*<i                   $                       Reduce

$<\*                     $                       Reduce

$                       $                      String is Accepted

**RESULT:**

Thus the operator precedence parser of given language is obtained.

**Ex.no:7 SYMBOL TABLE**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:**

To write a c program for implementing symbol table.

**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:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

int size=0;

void Insert();

void Display();

void Delete();

int Search(char lab[]);

void Modify();

struct SymbTab

{

    char label[10],symbol[10];

    int addr;

    struct SymbTab \*next;

};

struct SymbTab \*first,\*last;

void main()

{

int op,y;

char la[10];

do

{

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

    printf("\n\t1.INSERT\n\t2.DISPLAY\n\t3.DELETE\n\t4.SEARCH\n\t5.MODIFY\n\t6.END\n");

    printf("\n\tEnter your option : ");

    scanf("%d",&op);

    switch(op)

    {

    case 1:

        Insert();

        break;

    case 2:

        Display();

        break;

    case 3:

        Delete();

        break;

    case 4:

        printf("\n\tEnter the label to be searched : ");

        scanf("%s",la);

        y=Search(la);

        printf("\n\tSearch Result:");

    if(y==1)

        printf("\n\tThe label is present in the symbol table\n");

    else

        printf("\n\tThe label is not present in the symbol table\n");

        break;

    case 5:

        Modify();

        break;

    case 6:

        exit(0);

    }

}while(op<6);

getch();

}

void Insert()

{

    int n;

    char l[10];

    printf("\n\tEnter the label : ");

    scanf("%s",l);

    n=Search(l);

    if(n==1)

        printf("\n\tThe label exists already in the symbol table\n\tDuplicate can’t be inserted");

    else

    {

        struct SymbTab \*p;

        p=malloc(sizeof(struct SymbTab));

        strcpy(p->label,l);

        printf("\n\tEnter the symbol : ");

        scanf("%s",p->symbol);

        printf("\n\tEnter the address : ");

        scanf("%d",&p->addr);

        p->next=NULL;

    if(size==0)

    {

        first=p;

        last=p;

    }

    else

    {

        last->next=p;

        last=p;

    }

    size++;

    }

    printf("\n\tLabel inserted\n");

}

void Display()

{

    int i;

    struct SymbTab \*p;

    p=first;

    printf("\n\tLABEL\t\tSYMBOL\t\tADDRESS\n");

    for(i=0;i<size;i++)

    {

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

        p=p->next;

    }

}

int Search(char lab[])

{

    int i,flag=0;

    struct SymbTab \*p;

    p=first;

    for(i=0;i<size;i++)

    {

        if(strcmp(p->label,lab)==0)

        flag=1;

        p=p->next;

    }

    return flag;

}

void Modify()

{

    char l[10],nl[10];

    int add,choice,i,s;

    struct SymbTab \*p;

    p=first;

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

    printf("\n\t1.Only the label\n\t2.Only the address\n\t3.Both the label and address\n");

    printf("\tEnter your choice : ");

    scanf("%d",&choice);

    switch(choice)

    {

    case 1:

        printf("\n\tEnter the old label : ");

        scanf("%s",l);

        s=Search(l);

    if(s==0)

        printf("\n\tLabel not found\n");

    else

    {

        printf("\n\tEnter the new label : ");

        scanf("%s",nl);

        for(i=0;i<size;i++)

        {

            if(strcmp(p->label,l)==0)

            strcpy(p->label,nl);

            p=p->next;

        }

        printf("\n\tAfter Modification:\n");

        Display();

    }

    break;

    case 2:

        printf("\n\tEnter the label where the address is to be modified : ");

        scanf("%s",l);

        s=Search(l);

        if(s==0)

            printf("\n\tLabel not found\n");

        else

        {

            printf("\n\tEnter the new address : ");

            scanf("%d",&add);

            for(i=0;i<size;i++)

            {

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

                    p->addr=add;

                    p=p->next;

            }

            printf("\n\tAfter Modification:\n");

            Display();

        }

        break;

    case 3:

        printf("\n\tEnter the old label : ");

        scanf("%s",l);

        s=Search(l);

        if(s==0)

            printf("\n\tLabel not found\n");

        else

        {

            printf("\n\tEnter the new label : ");

            scanf("%s",nl);

            printf("\n\tEnter the new address : ");

            scanf("%d",&add);

            for(i=0;i<size;i++)

            {

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

                {

                    strcpy(p->label,nl);

                    p->addr=add;

                }

                p=p->next;

            }

            printf("\n\tAfter Modification:\n");

            Display();

        }

        break;

}

}

void Delete()

{

    int a;

    char l[10];

    struct SymbTab \*p,\*q;

    p=first;

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

    scanf("%s",l);

    a=Search(l);

    if(a==0)

        printf("\n\tLabel not found\n");

    else

    {

        if(strcmp(first->label,l)==0)

            first=first->next;

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

        {

            q=p->next;

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

            {

                p=p->next;

                q=q->next;

            }

            p->next=NULL;

            last=p;

        }

        else

        {

            q=p->next;

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

            {

                p=p->next;

                q=q->next;

            }

            p->next=q->next;

        }

        size--;

        printf("\n\tAfter Deletion:\n");

        Display();

}

}

**OUTPUT :**

SYMBOL TABLE IMPLEMENTATION

        1.INSERT

        2.DISPLAY

        3.DELETE

        4.SEARCH

        5.MODIFY

        6.END

Enter your option : 1

Enter the label : str

Enter the symbol : string

Enter the address : 1234

Label inserted

SYMBOL TABLE IMPLEMENTATION

        1.INSERT

        2.DISPLAY

        3.DELETE

        4.SEARCH

        5.MODIFY

        6.END

Enter your option : 1

Enter the label : i

Enter the symbol : int

Enter the address : 4567

Label inserted

SYMBOL TABLE IMPLEMENTATION

        1.INSERT

        2.DISPLAY

        3.DELETE

        4.SEARCH

        5.MODIFY

        6.END

Enter your option : 2

LABEL     SYMBOL     ADDRESS

    str           string             1234

    i                int                 4567

SYMBOL TABLE IMPLEMENTATION

1.INSERT

2.DISPLAY

         3.DELETE

         4.SEARCH

         5.MODIFY

         6.END

Enter your option : 4

Enter the label to be searched : i

Search Result:

The label is present in the symbol table

SYMBOL TABLE IMPLEMENTATION

1.INSERT

        2.DISPLAY

         3.DELETE

        4.SEARCH

         5.MODIFY

         6.END

Enter your option : 5

 What do you want to modify?

1.Only the label

2.Only the address

  3.Both the label and address

  Enter your choice : 1

  Enter the old label : i

  Enter the new label : i1

After Modification:

LABEL       SYMBOL    ADDRESS

    str             string         1234

     i1               int             4567

SYMBOL TABLE IMPLEMENTATION

        1.INSERT

        2.DISPLAY

        3.DELETE

        4.SEARCH

        5.MODIFY

        6.END

Enter your option : 3

Enter the label to be deleted : i1

After Deletion:

LABEL    SYMBOL       ADDRESS

    str          string                1234

        SYMBOL TABLE IMPLEMENTATION

         1.INSERT

         2.DISPLAY

         3.DELETE

         4.SEARCH

        5.MODIFY

         6.END

Enter your option : 6

Process returned 0 (0x0)   execution time : 61.312 s

Press any key to continue.

**RESULT:**

Thus, The given program is executed and the results are obtained.

**Ex.no:8 Shift Reduce Parsing**

**AIM:**

To write the C program for Shift reduce Parsing.

**ALGORITHM:**

**Step1:** Get the input expression and store it in the input buffer.

**Step 2:** Read the data from the input buffer one at the time.

**Step 3:** Using stack and push & pop operation shift and reduce symbols

with respect to production rules available.

**Step 4:** Continue the process till symbol shift and production rule reduce reaches the start symbol.

**Step 5**: Display the Stack Implementation table with corresponding Stack actions with input symbols.

**PROGRAM:**

#include<stdio.h>

#include<string.h>

int k=0,z=0,i=0,j=0,c=0;

char a[16],ac[20],stk[15],act[10];

void check();

void main()

{

puts("GRAMMAR is E->E+E \n E->E\*E \n E->(E) \n E->id");

puts("enter input string ");

gets(a);

c=strlen(a);

strcpy(act,"SHIFT->");

puts("stack \t input \t action");

for(k=0,i=0; j<c; k++,i++,j++)

{

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

{

stk[i]=a[j];

stk[i+1]=a[j+1];

stk[i+2]='\0';

a[j]=' ';

a[j+1]=' ';

printf("\n$%s\t%s$\t%sid",stk,a,act);

check();

}

else

{

stk[i]=a[j];

stk[i+1]='\0';

a[j]=' ';

printf("\n$%s\t%s$\t%ssymbols",stk,a,act);

check();

}

}

getch();

}

void check()

{

strcpy(ac,"REDUCE TO E");

for(z=0; z<c; z++)

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

{

stk[z]='E';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

j++;

}

for(z=0; z<c; z++)

if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+2]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

for(z=0; z<c; z++)

if(stk[z]=='E' && stk[z+1]=='\*' && stk[z+2]=='E')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

for(z=0; z<c; z++)

if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

**OUTPUT:**

THE GRAMMAR IS AS FOLLOWS

S -> S+T

S -> T

T -> T\*F

T -> F

F -> (S)

F -> t

I0 :

Z -> .S

S -> .S+T

S -> .T

T -> .T\*F

T -> .F

F -> .(S)

F -> .t

I1 :

Z -> S.

S -> S.+T

**RESULT:**

To write the C program for Shift reduce Parsing is Successfully verified.

**Ex. No:** **9 LR Parser**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:**

To write the C program for LR parser.

**ALGORITHM:**

**Step 1:** Get the input expression and store it in the input buffer.

**Step 2:** Read the data from the input buffer one at the time and convert in to corresponding Non Terminal using production rules available.

**Step 3:** Perform push & pop operation for LR parsing table construction.

**Step 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<string.h>

int i,j,k,m,n=0,o,p,ns=0,tn=0,rr=0,ch=0;

char read[15][10],gl[15],gr[15][10],temp,templ[15],tempr[15][10],\*ptr,temp2[5],dfa[15][15];

struct states

{

char lhs[15],rhs[15][10];

int n;

}I[15];

int compstruct(struct states s1,struct states s2)

{

int t;

if(s1.n!=s2.n)

return 0;

if( strcmp(s1.lhs,s2.lhs)!=0 )

return 0;

for(t=0;t<s1.n;t++)

if( strcmp(s1.rhs[t],s2.rhs[t])!=0 )

return 0;

return 1;

}

void moreprod()

{

int r,s,t,l1=0,rr1=0;

char \*ptr1,read1[15][10];

for(r=0;r<I[ns].n;r++)

{

ptr1=strchr(I[ns].rhs[l1],'.');

t=ptr1-I[ns].rhs[l1];

if( t+1==strlen(I[ns].rhs[l1]) )

{

l1++;

continue;

}

temp=I[ns].rhs[l1][t+1];

l1++;

for(s=0;s<rr1;s++)

if( temp==read1[s][0] )

break;

if(s==rr1)

{

read1[rr1][0]=temp;

rr1++;

}

else

continue;

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

{

if(gl[s]==temp)

{

I[ns].rhs[I[ns].n][0]='.';

I[ns].rhs[I[ns].n][1]=NULL;

strcat(I[ns].rhs[I[ns].n],gr[s]);

I[ns].lhs[I[ns].n]=gl[s];

I[ns].lhs[I[ns].n+1]=NULL;

I[ns].n++;

}

}

}

}

void canonical(int l)

{

int t1;

char read1[15][10],rr1=0,\*ptr1;

for(i=0;i<I[l].n;i++)

{

temp2[0]='.';

ptr1=strchr(I[l].rhs[i],'.');

t1=ptr1-I[l].rhs[i];

if( t1+1==strlen(I[l].rhs[i]) )

continue;

temp2[1]=I[l].rhs[i][t1+1];

temp2[2]=NULL;

for(j=0;j<rr1;j++)

if( strcmp(temp2,read1[j])==0 )

break;

if(j==rr1)

{

strcpy(read1[rr1],temp2);

read1[rr1][2]=NULL;

rr1++;

}

else

continue;

for(j=0;j<I[0].n;j++)

{

ptr=strstr(I[l].rhs[j],temp2);

if( ptr )

{

templ[tn]=I[l].lhs[j];

templ[tn+1]=NULL;

strcpy(tempr[tn],I[l].rhs[j]);

tn++;

}

}

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

{

ptr=strchr(tempr[j],'.');

p=ptr-tempr[j];

tempr[j][p]=tempr[j][p+1];

tempr[j][p+1]='.';

I[ns].lhs[I[ns].n]=templ[j];

I[ns].lhs[I[ns].n+1]=NULL;

strcpy(I[ns].rhs[I[ns].n],tempr[j]);

I[ns].n++;

}

moreprod();

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

{

//if ( memcmp(&I[ns],&I[j],sizeof(struct states))==1 )

if( compstruct(I[ns],I[j])==1 )

{

I[ns].lhs[0]=NULL;

for(k=0;k<I[ns].n;k++)

I[ns].rhs[k][0]=NULL;

I[ns].n=0;

dfa[l][j]=temp2[1];

break;

}

}

if(j<ns)

{

tn=0;

for(j=0;j<15;j++)

{

templ[j]=NULL;

tempr[j][0]=NULL;

}

continue;

}

dfa[l][j]=temp2[1];

printf("\n\nI%d :",ns);

for(j=0;j<I[ns].n;j++)

printf("\n\t%c -> %s",I[ns].lhs[j],I[ns].rhs[j]);

getch();

ns++;

tn=0;

for(j=0;j<15;j++)

{

templ[j]=NULL;

tempr[j][0]=NULL;

}

}

}

void main()

{

FILE \*f;

int l;

for(i=0;i<15;i++)

{

I[i].n=0;

I[i].lhs[0]=NULL;

I[i].rhs[0][0]=NULL;

dfa[i][0]=NULL;

}

f=fopen("tab6.txt","r");

while(!feof(f))

{

fscanf(f,"%c",&gl[n]);

fscanf(f,"%s\n",gr[n]);

n++;

}

printf("THE GRAMMAR IS AS FOLLOWS\n");

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

printf("\t\t\t\t%c -> %s\n",gl[i],gr[i]);

I[0].lhs[0]='Z';

strcpy(I[0].rhs[0],".S");

I[0].n++;

l=0;

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

{

temp=I[0].rhs[l][1];

l++;

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

if( temp==read[j][0] )

break;

if(j==rr)

{

read[rr][0]=temp;

rr++;

}

else

continue;

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

{

if(gl[j]==temp)

{

I[0].rhs[I[0].n][0]='.';

strcat(I[0].rhs[I[0].n],gr[j]);

I[0].lhs[I[0].n]=gl[j];

I[0].n++;

}

}

}

ns++;

printf("\nI%d :\n",ns-1);

for(i=0;i<I[0].n;i++)

printf("\t%c -> %s\n",I[0].lhs[i],I[0].rhs[i]);

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

canonical(l);

printf("\n\n\t\tPRESS ANY KEY FOR DFA TABLE");

getch();

printf("\t\t\tDFA TABLE IS AS FOLLOWS\n\n\n");

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

{

printf("I%d : ",i);

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

if(dfa[i][j]!='\0')

printf("'%c'->I%d | ",dfa[i][j],j);

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

}

printf("\n\n\n\t\tPRESS ANY KEY TO EXIT");

getch();

}

**OUTPUT:**

THE GRAMMAR IS AS FOLLOWS

S -> S+T

S -> T

T -> T\*F

T -> F

F -> (S)

F -> t

I0 :

Z -> .S

S -> .S+T

S -> .T

T -> .T\*F

T -> .F

F -> .(S)

F -> .t

I1 :

Z -> S.

S -> S.+T

**RESULT:**

Thus the program to verify LR Parser is Successfully Verified.

**Ex.no:10 Intermediate Code Generation**

**AIM:**

To Write the C program for Intermediate Code Generator

**ALGORITHM:**

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

**PROGRAM:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int i=1,j=0,no=0,tmpch=90;

char str[100],left[15],right[15];

void findopr();

void explore();

void fleft(int);

void fright(int);

struct exp{

int pos;

char op;

}k[15];

void main()

{

printf("\t\tINTERMEDIATE CODE GENERATION\n\n");

printf("Enter the Expression");

scanf("%s",str);

printf("The intermediate code:\t\tExpression\n");

findopr();

explore();

}

void findopr()

{

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

if(str[i]==':')

{

k[j].pos=i;

k[j++].op=':';

}

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

if(str[i]=='/')

{

k[j].pos=i;

k[j++].op='/';

}

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

if(str[i]=='\*')

{

k[j].pos=i;

k[j++].op='\*';

}

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

if(str[i]=='+')

{

k[j].pos=i;

k[j++].op='+';

}

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

if(str[i]=='-')

{

k[j].pos=i;

k[j++].op='-';

}

}

void explore()

{

i=1;

while(k[i].op!='\0')

{

fleft(k[i].pos);

fright(k[i].pos);

str[k[i].pos]=tmpch--;

printf("\t%c:= %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);

for(j=0;j<strlen(str);j++)

if(str[j]!='$')

printf("%c",str[j]);

printf("\n");

i++;

}

fright(-1);

if(no==0){

fleft(strlen(str));

printf("\t%s :=%s",right,left);

exit(0);

}

printf("\t%s := %c",right,str[k[--i].pos]);

}

void fleft(int x)

{

int w=0,flag=0;

x--;

while(x!=-1 && str[x]!='+' &&str[x]!='\*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&str[x]!='/'&&str[x]!=':')

{

if(str[x]!='$'&& flag==0)

{

left[w++]=str[x];

left[w]='\0';

str[x]='$';

flag=1;

}

x--;

}

}

void fright(int x)

{

int w=0,flag=0;

x++;

while(x!=-1&&str[x]!='+'&&str[x]!='\*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-'&&str[x]!='/')

{

if(str[x]!='$'&&flag==0)

{

right[w++]=str[x];

right[w]='\0';

str[x]='$';

flag=1;

}

x++;

}

}

**OUTPUT:**

INTERMEDIATE CODE GENERATION

Enter the Expression a\*b+c/d-e/f+g\*h

The intermediate code: Expression

Z:= e/f a\*b+c/d-Z+g\*h

Y:= a\*b Y+c/d-Z+g\*h

X:= g\*h Y+c/d-Z+X

W:= Y+c W/d-Z+X

V:= Z+X W/d-V

U:= d-V W/U

W :=U

Process returned 0 (0x0) execution time : 35.975 s

Press any key to continue.

**RESULT:**

Thus the program for intermediate Code Generator is successfully verified.

**Ex.no:11 Code Optimization Technique**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Factorial of a number**

**AIM:**

To write the program for Code Optimization in C.

**ALGORITHM:**

START

**Step 1** : Take integer variable A

**Step 2** : Assign value to the variable

**Step 3 :** From value A upto 1 multiply each digit and store

**Step 4** : the final stored value is factorial of A

STOP

**Before Optimization:**

#include<stdio.h>

int main()

{

int i,n;

int fact=1;

printf("Enter a number: ");

scanf("%d",&n);

for(i=n;i>=1;i--)

{

fact=fact\*i;

}

printf("The factorial value is %d",fact);

return 0;

}

**OUTPUT:**

Enter a number: 7

The factorial value is 5040

Process returned 0 (0x0) **execution time : 5.155 s**

Press any key to continue.

**After optimization:**

#include<stdio.h>

int main()

{

int n,fact;

fact=1;

printf("Enter the number: ");

scanf("%d",&n);

do{

fact=fact\*n;

n--;

}while(n>0);

printf("The factorial value is %d",fact);

return 0;

}

**OUTPUT:**

Enter the number: 7

The factorial value is 5040

Process returned 0 (0x0) **execution time : 2.075 s**

Press any key to continue.

1. **Area of a Circle**

**Before Optimization:**

#include<stdio.h>

#define x 3.147

void main()

{

float r,A;

printf("Enter the radius for the circle: ");

scanf("%f",&r);

A=x\*r\*r;

printf("\nArea of a circle:%f",A);

}

**OUTPUT:**

Enter the radius for the circle: 7

Area of a circle: 154.203003

Process returned 27 (0x1B) **execution time : 4.883 s**

Press any key to continue.

**After Optimization:**

#include<stdio.h>

void main()

{

float r;

printf("Enter the radius for the circle: ");

scanf("%f",&r);

printf("\nArea of a circle:%f",3.147\*r\*r);

}

**OUTPUT:**

Enter the radius for the circle: 7

Area of a circle:154.203003

Process returned 27 (0x1B) **execution time : 4.800 s**

Press any key to continue.

**RESULT:**

The Program for Code Optimization is Successfully Verified.