K.RAMAKRISHNAN COLLEGE OF ENGINEERING

(Autonomous) SAMAYAPURAM, TRICHY– 621112.



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING UCS1612 COMPILER DESIGN LABORATORY**

**REGULATION 2020 LAB MANUAL**

|  |  |  |  |
| --- | --- | --- | --- |
| **Prepared by** | |  | **Verified by,** |
| **Mr.T.JohnPeter** |  |  |
| **Mrs.N.Nithya** |  |

## VISION MISSION

**INSTITUTE VISION AND MISSION**

“To achieve a prominent position among the top technical institutions”

**M1:** To bestow standard technical education par excellence through state of the art infrastructure, competent faculty and high ethical standards.

**M2:** To nurture research and entrepreneurial skills among students in cutting edge technologies.

**M3:** To provide education for developing high-quality professionals to transform the society.

## DEPARTMENT VISION AND MISSION

**VISION**

To create eminent professionals of Computer Science and Engineering by imparting quality education.

## MISSION

**M1:** To provide technical exposure in the field of Computer Science and Engineering through state of the art infrastructure and ethical standards.

**M2:** To engage the students in research and development activities in the field of Computer Science and Engineering.

**M3:** To empower the learners to involve in industrial and multi-disciplinary projects for addressing the societal needs.

## THE PEOs OF THE PROGRAMME ARE,

Our graduates shall

**PEO1:**Analyse, design and create innovative products for addressing social needs.

**PEO2:** Equip themselves for employability, higher studies and research.

**PEO3:** Nurture the leadership qualities and entrepreneurial skills for their successful career

## PROGRAM SPECIFIC OUTCOMES (PSOs)

Students will be able to

**PSO1:** Apply the basic and advanced knowledge in developing software, hardware and firmware solutions addressing real life problems.

**PSO2:** Design, develop, test and implement product-based solutions for their career enhancement.

## PROGRAM OUTCOME (PO)

**PO1 Engineering knowledge**: Apply the 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 the specified needs with appropriate consideration for the public health and safety, and the 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 modelling 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:

1. To write test and debug simple compiler writing tools.
2. To implement the different phases of compiler.
3. To analyze the data flow and the control flow of a program.
4. To study simple optimization techniques.
5. To use assembly language program like a source language program.

## COURSE OUTCOMES:

|  |  |  |
| --- | --- | --- |
| **CO** | **DESCRIPTION** | **KNOWLEDGE**  **LEVEL** |
| C318.1 | Analyze and design solutions of compiler to test and debug compiler writing tools. | K4 |
| C318.2 | Analyze the control flow and data flow of a program. | K4 |
| C318.3 | Apply appropriate techniques to optimize a problem using LEX and YACC. | K3 |
| C318.4 | Analyze complex engineering problems to implement storage allocation strategies. | K4 |
| C318.5 | Utilize assembly language like any other source language. | K3 |

**CO PO MAPPING TABLE:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SNO** | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** | **PSO1** | **PSO2** |
| **C318.1** | 3 | 3 | 3 | - | - | - | - | 2 | 3 | 2 | - | 3 | 3 | 2 |
| **C318.2** | 3 | 3 | 3 | - | 3 | - | - | 2 | 3 | 2 | - | 3 | 3 | 2 |
| **C318.3** | 3 | 3 | 3 | - | - | - | - | 2 | 3 | 2 | - | 3 | 3 | 2 |
| **C318.4** | 3 | 3 | 3 | - | - | - | - | 2 | 3 | 2 | - | 3 | 3 | 2 |
| **C318.5** | 3 | 3 | 3 | - | 3 | - | - | 2 | 3 | 2 | - | 3 | 3 | 2 |
| **C318** | **3.00** | **3.00** | **3.00** | **0.00** | **3.00** | **0.00** | **0.00** | **2.00** | **3.00** | **2.00** | **0.00** | **3.00** | **3.00** | **2.00** |

## LIST OF EXPERIMENTS

|  |  |  |  |
| --- | --- | --- | --- |
| **Ex.No** | **Name of the Exercise** | **CO** | **PO** |
| **1.** | Implement symbol table. | CO1 | PO1-3,PO8-10,PO12,  PSO1-2 |
| **2.** | Develop a lexical analyzer to recognize a few patterns in  C. (Ex. identifiers, constants, comments, operators etc.). | CO1 | PO1-3,PO8-10,PO12,  PSO1-2 |
| **3.** | Implement a Lexical Analyzer using LEX Tool | CO3 | PO1-3,PO5, PO8-10,PO12,  PSO1-2 |
| **4.** | Generate YACC specification to recognize a valid arithmetic expression that uses operator +, - , \* and /. | CO3 | PO1-3,PO5, PO8-10,PO12,  PSO1-2 |
| **5.** | Implement an Arithmetic Calculator using LEX and  YACC. | CO3 | PO1-3,PO5, PO8-10,PO12,  PSO1-2 |
| **6.** | Generate three address code for a simple program using  C language or LEX-YACC. | CO3 | PO1-3,PO5, PO8-10,PO12,  PSO1-2 |
| **7.** | Convert the BNF rules into YACC form and write code  to generate Abstract Syntax Tree. | CO4 | PO1-3,PO8-10,PO12,  PSO1-2 |
| **8.** | Implement control flow analysis and Data flow Analysis | CO2 | PO1-3,PO8-10,PO12,  PSO1-2 |
| **9.** | Implement any one storage allocation strategies  (Heap,Stack,Static) | CO4 | PO1-3,PO8-10,PO12,  PSO1-2 |
| **10.** | Implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using a 8086 assembler. The target assembly instructions can be simple move, add, sub, jump. Also  simple addressing modes are used. | CO5 | PO1-3,PO5, PO8-10,PO12,  PSO1-2 |
| **11.** | Implement simple code optimization techniques (Constant folding, Strength reduction and Algebraic transformation) | CO5 | PO1-3,PO5, PO8-10,PO12,  PSO1-2 |
| **\*\*12** | Implementation of adding line number to a given three address code in file | CO3 | PO1-3,PO8-10,PO12,  PSO1-2 |
| **\*\*13** | Implementation of Assembly code cost calculation | CO5 | PO1-3,PO5, PO8-10,PO12,  PSO1-2 |

\*\*Content Beyond Syllabus

**EX.NO:1 IMPLEMENTATION OF SYMBOL TABLE**

**Aim**

To Write a C program to create a symbol table.

**Algorithm**

Step 1: Start the program with necessary header files

Step 2: Declare the variables, functions and structure members

Step 3: Using switch case statement to enter the options insert, display, search, modify and exit. Step 4: Insert the label, symbol and address of the values

Step 5: Display the inserted values in the format of label, symbol and address Step 6: Search the particular values in the table using strcmp() to find the label Step 7: Modify the inserted label by new label using search()

Step 8: Display the entire table Step 9: stop the program execution.

**PROGRAM:**

#include<stdio.h> #include<conio.h> #include<alloc.h> #include<string.h> #include<stdlib.h> #define NULL 0 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]; clrscr();

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;

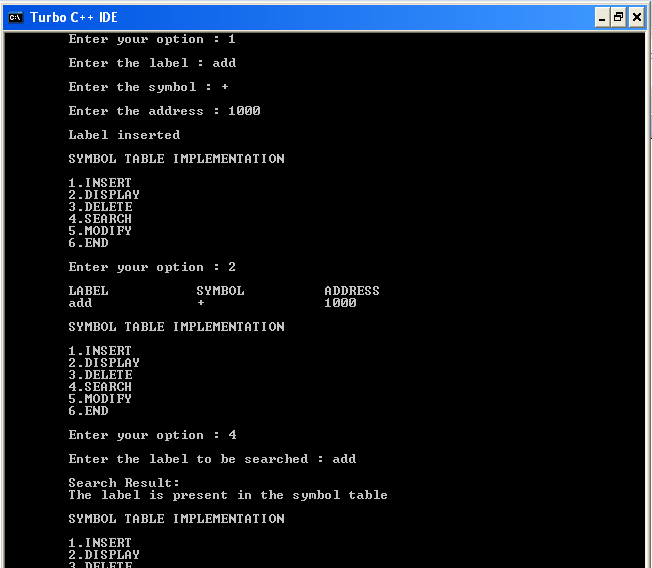
}

size--;

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

}

}

**Output:**

RESULT:

Thus the C program to implement the symbol table was executed and the output is verified.

## EX.NO:2 Develop a lexical analyzer to recognize a few patterns in C. AIM:

To write a program to implement lexical analyzer using C.

## ALGORITHM:

Step 1: Initialize four arrays to hold operators, keywords, special symbols and identifiers respectively. Step 2: Open the input file in read mode.

Step 3: Display the choices to list the tokens. Step 4: Get the choice from the user.

Step 5: Scan the characters of the input file one at a time. Step 6: Compare with the tokens in the array.

Step 7: Print the token type.

**Program:** #include<stdio.h> #include<string.h> main()

{

FILE \*fp;

char a[5]={':','-','\*','+','='};

char b[8]={'{','}','[',']','(',')'};

char q[20]={'a','b','c','d'};

char p[15][15]={"int","if","void"}; int i,j,k,n,l;

char x,ch,y[7],s[10],z[8],ch1[80],id[60]; printf("\*\*\*\*\*\*\*\*\n Choices are: \n\*\*\*\*\*\*\*\*\*"); printf("\n 1. Operators");

printf("\n 2. Special Symbols"); printf("\n 3. Keywords");

printf("\n 4. Identifiers");

printf("\n 5. Exit"); first:;

printf("\n Enter your choice:"); scanf("%d",&n);

switch(n)

{

case 1:

printf("\n 1. Operators");

for(i=0;i<strlen(a);i++)

{

fp=fopen("in6.txt","r"); do

{ch=fgetc(fp); if(ch==a[i])

{

printf("\n%c\n",ch); break;

}

}

while(!feof(fp));

}

fclose(fp); goto first; case 2:

printf("\n 2. Special Symbols"); for(j=0;j<strlen(b);j++)

{

fp=fopen("in6.txt","r"); do

{ x=fgetc(fp);

if(x==b[j])

{

printf("\n%c\n",b[j]); break;

}

}while(x!=EOF);

}

fclose(fp); goto first; break; case 3:

printf("\n 3. Keywords"); fp=fopen("in6.txt","r"); l=0;

x=getc(fp); while(x!=EOF&&x!='(')

{

id[l]=x; l++;

x=getc(fp);

}

id[l]='\0'; fclose(fp); printf("\n%s\n",id); goto first;

break; case 4:

printf("\n 4. Identifiers"); for(i=0;i<strlen(q);i++)

{

fp=fopen("in6.txt","r"); do

{

ch=fgetc(fp); if(ch==q[i])

{

printf( "\n%c\n",ch); break;

}

}

while(!feof(fp));

}

fclose(fp); goto first; break; case 5:

printf("5. You Want To Quit Give Y:"); scanf("\n%c\n",&y);

if(getchar()=='y') exit(0);

else goto first; break;

}

}

## OUTPUT:

[me23@LocalHost~]$ vi in6.txt if (a>b)a=a+2 [me23@LocalHost ~]$ cc lexip.c [me23@LocalHost ~]$ ./a.out

\*\*\*\*\*\*\*\*

Choices are:

\*\*\*\*\*\*\*\*\*

1. Operators
2. Special Symbols
3. Keywords
4. Identifiers
5. Exit

Enter your choice:1

1. Operators

+

=

Enter your choice: 2

1. Special Symbols (

)

Enter your choice:3

1. Keywords if

Enter your choice:4

1. Identifiers a

b

Enter your choice:5

1. You Want To Quit Give Y: y

## RESULT:

Thus the program to implement the lexical analyzer of a compiler was implemented in C

## Ex. No:3 IMPLEMENTATION OF LEXICAL ANALYSIS IN LEX TOOL

**AIM**

To Write a program to implement the function of Lexical Analyzer using LEX tool

## ALGORITHM

Step 1: Open the vi editor and given the name as vi lex.l Step 2: Declare the identifier [a-z A-Z][a-zA-z0-9]\*

Step 3: Read the preprocessor directive to display the keywords.

Step 4:Read the assignment operator, relational operator and display it.

Step 5: Open a file vi lex. c in read and include the yylex() tool for input scanning. Step 6: D e f i n e t h e a l p h a b e t s a n d n u m b e r s .

Step 7: Print the preprocessor, function, keyword using yytext.lex tool.

Step 8: Print the relational, assignment and all the operator using yytext() tool. Step9: Also scan and print where the loop ends and begins.

Step10: Use yywrap() to enter an error.

Step 11. Stop the program execution.

**PROGRAM**

**vi lex.l**

%{

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;}

\)(\;)? {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;

}

**vi lex1.c:**

main()

{

int a,b,c,i; a=10; b=20;

c=a+b\*10; if(a>b) printf("%d",a); else printf("%d",b); i=0;

while(i<c)

{

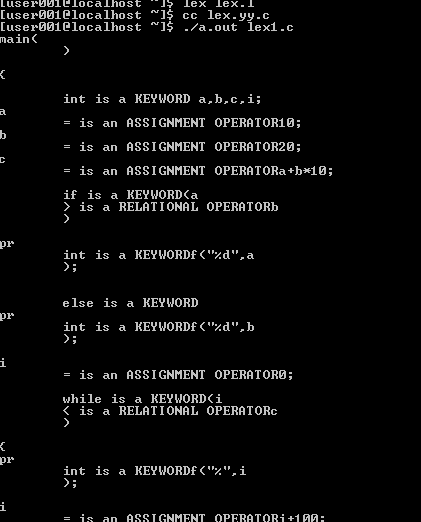
printf("%",i); i=i+100;

}

}

**Output:**

[user001@localhost ~]$ lex lex.l [user001@localhost ~]$ cc lex.yy.c [user001@localhost ~]$ ./a.out lex1.c



## RESULT

Thus the lex program was executed and the tokens of the program were classified.

## EX.NO:4 Generate YACC specification to recognize a valid arithmetic expression that uses operator +, - , \* and /.

**Aim:**

To write a C program to implement lexical analyzer for Arithmetic Expression.

## Algorithm:

Step 1: Open the vi editor and given the name as vi arith.y

Step 2: Include the header files stdio.h, ctype.h, stdlib.h and string.h in the declaration part Step 3: Read the numbers, operators and expressions in the translation rules part

Step 4: Analyze the structure of the expression to find the valid expression.

Step 5: If the given expression is valid to display the output otherwise produce the error message Step 6:In the main program read the expression using yylex function

Step 7: The getchar function is used to get the character and digit of the expression Step 8: If the given digit is valid return the number otherwise return the error message Step 9.Stop the program execution

Input: Programming language arithmetic expression Output: A sequence of tokens. Tokens have to be identified and its respective attributes have to be printed.

**PROGRAM:**

**arith.y**

%{

#include<stdio.h> #include<ctype.h> #include<stdlib.h> #include<string.h> #define YYSTYPE double

%}

%token num

%left '+' '-'

%left '\*' '/'

%%

st: st expr '\n' {printf("Valid");}

| st '\n'

|

| error '\n' {printf("INVALID");}

;

expr: num

| expr '+' expr

| expr '/' expr

%%

main()

{

printf(" ENTER AN EXPRESSION TO VALIDATE");

yyparse();

}

yylex()

{

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

if(isdigit(ch)|ch=='.')

{

ungetc(ch,stdin); scanf("%lf",&yylval); return num;

}

return ch;

}

yyerror(char \*s)

{

printf("%S",s);

}

Output:

[user041@localhost ~]$ yacc -d arith.y [user041@localhost ~]$ cc y.tab.c -ll [user041@localhost ~]$ ./a.out

ENTER AN EXPRESSION TO VALIDATE 10+6

Valid 10+

INVALID 9+^H

**RESULT**:

Thus the lexical analyzer for arithmetic expression was created and implemented.

## EX. NO. 5 IMPLEMENTATION OF AN ARITHMETIC CALCULATOR USING LEX AND YACC

**AIM:**

To write a YACC program to implement a desktop calculator

## ALGORITHM:

Step1:Place the C declaration statements inside %{ and %} Step2: Declare the tokens

Step3: Define the associativity of the operators and algebraical functions Step4: Define the expression types and action to be done.

Step5: In the main function get the input expression and start parsing. Step6: If there are no errors then print the resultant of the expression.

## Program:

%{

double memvar;

%}

%union

{

double dval;

}

%token<dval>NUMBER

%token<dval>MEM

%token LOG SINE Nlog COS TAN

%left ‘-‘ ‘+’

%left ‘\*’ ‘/’

%right ‘^’

%left LOG SINE Nlog COS TAN

%nonassoc UMINUS

%type<dval>expression

%%

start: statement ‘\n’

|start statement ‘\n’

;

statement:MEM ‘=’ expression {memvar=$3}

|expression{printf(“Answer = %g\n”,$1);}

;

expression:expression ‘+’ expression{$$ = $1 + $3;}

|expression ‘-‘ expression{$$ = $1 - $3;}

|expression ‘\*’ expression{$$ = $1 \* $3;}

| expression ‘/’ expression

{

if($3 == 0)

yyerror(“divide by zero”); else

$$=$1/$3;

}

| expression ‘^’ expression{$$ = pow($1,$3);}

;

| expression: ‘-‘ expression %prec UMINUS {$$=-$2;}

| ‘(‘ expression ‘)’ {$$ = $2;}

|LOG expression {$$ = log($2)/log(10);}

|Nlog expression{$$ = log($2);}

|SINE expression{$$=sin($2\*3.141592654/180);}

|COS expression{$$=cos($2\*3.141592654/180);}

|TAN expression{$$=tan($2\*3.141592654/180);}

|NUMBER {$$ = $1;}

|MEM {$$=memvar;}

;

%%

main()

{

printf(“Enter the Expression”); yyparse();

}

int yyerror(char \*error)

{

fprintf(stderr,”%s\n”,error);

}

## OUTPUT:

Enter the Expression 2+3

Answer 5

## RESULT:

Thus the program to design a desktop calculator using YACC tool is implemented.

## EX.NO. 6 IMPLEMENTATION OF THREE ADDRESS CODE GENERATION USING LEX AND YACC

**AIM:**

To design the front end of compiler and generate the intermediate code.

## ALGORITHM:

*LEX program:*

1. Declaration of header files specially y.tab.h which contains declaration for Letter, Digit, expr.
2. End declaration section by %%
3. Match regular expression.
4. If match found then convert it into char and store it in yylval.p where p is pointer declared in YACC
5. Return token
6. If input contains new line character (\n) then return 0
7. If input contains „.‟ then return yytext[0]
8. End rule-action section by %%
9. Declare main function
   1. open file given at command line
   2. if any error occurs then print error and exit
   3. assign file pointer fp to yyin d.call function yylex until file ends
10. End

*YACC program:*

1. Declaration of header files
2. Declare structure for three address code representation having fields of argument1, argument2, operator, result.
3. Declare pointer of char type in union.
4. Declare token expr of type pointer p.
5. Give precedence to „\*‟,‟/‟ „+‟,‟-‟.
6. End of declaration section by %%.
7. If final expression evaluates then add it to the table of three address code.
8. If input type is expression of the form.

a. exp‟+‟exp then add to table the argument1, argument2, operator. b.exp‟-‟exp then add to table the argument1, argument2, operator.

c. exp‟\*‟exp then add to table the argument1, argument2, operator. d.exp‟/‟exp then add to table the argument1, argument2, operator.

e. „(„exp‟)‟ then assign $2 to $$.

f. Digit OR Letter then assigns $1 to $$.

1. End the section by %%.
2. Declare file \*yyin externally.
3. Declare main function and call yyparse function untill yyin ends
4. Declare yyerror for if any error occurs.

## THREE ADDRESS CODE GENERATION USING LEX AND YACC

YACC PROGRAM

/\*(Yacc Program : inter.y)\*/

%token ID NUM

%right '='

%left '+' '-'

%left '\*' '/'

%left UMINUS

%%

S:ID{push();} '='{push();} E{codegen\_assign();}

;

E:E'+'{push();} T{codegen();}

|E'-'{push();} T{codegen();}

| T

;

T:T'\*'{push();} F{codegen();}

|T'/'{push();} F{codegen();}

|F

;

F:'(' E ')'

|'-'{push();} F{codegen\_umin();} %prec UMINUS

|ID{push();}

|NUM{push();}

;

%%

#include "lex.yy.c" #include<ctype.h> char st[100][10];

int top=0;

char i\_[2]="0";

char temp[2]="t"; main()

{

printf("Enter the expression : "); yyparse();

}

push()

{

strcpy(st[++top],yytext);

}

codegen()

{

strcpy(temp,"t"); strcat(temp,i\_);

printf("%s = %s %s %s\n",temp,st[top-2],st[top-1],st[top]); top-=2;

strcpy(st[top],temp); i\_[0]++;

}

codegen\_umin()

{

strcpy(temp,"t"); strcat(temp,i\_);

printf("%s = -%s\n",temp,st[top]); top--;

strcpy(st[top],temp); i\_[0]++;

}

codegen\_assign()

{

printf("%s = %s\n",st[top-2],st[top]); top-=2;

}

LEX PROGRAM

/\*(Lex Program : inter.l)\*/ ALPHA [A-Za-z]

DIGIT [0-9]

%%

{ALPHA}({ALPHA}|{DIGIT})\* return ID;

{DIGIT}+ {yylval=atoi(yytext); return NUM;} [\n\t] yyterminate();

. return yytext[0];

%%

## OUTPUT:

nn@linuxmint ~ $ lex inter.l nn@linuxmint ~ $ yacc inter.y nn@linuxmint ~ $ gcc y.tab.c -ll -ly nn@linuxmint ~ $ ./a.out

Enter the expression : a=(k+8)\*(c-s) t0 = k + 8

t1 = c - s t2 = t0 \* t1 a = t2

## RESULT:

Thus the program to design the front end of a compiler using LEX and YACC tool is implemented and the intermediate code is generated.

## EX.NO:7 CONVERT THE BNF RULES INTO YACC FORM AND WRITE CODE TO GENERATE ABSTRACT SYNTAX TREE

**AIM**

Convert The BNF rules into Yacc form and write code to generate abstract syntax tree.

## ALGORITHM

Step 1: Open the vi editor and given the name as vi innn.l

Step 2: Include the header files stdio.h, y.tab.h and string.h in the declaration part Step 3: Read the digits [0-9] and exponential part in the translation rules part Step 4: Read the identifiers [a-Za-Z] [a-Za-Z0-9] in the translation rules part

Step 5: Read the keywords IF, ELSE, WHILE and data types int, char, float and read the relational operators like >, >=, <>, <= , <

Step 6: Open the another vi editor and name it as innn.y

Step 7: Read the variables, keywords and expression for processing

Step 8: Using stack to perform the syntax tree operations like push the activation data and pop the deactivation data.

Step 9: Top pointer is used to find the current activation data. Step 10: open the c editor to get the input file

Step 11: The input file given in the c file

Step 12: Execute the program and display the abstract syntax tree. Step 13: stop the program execution.

**PROGRAM:**

**vi innn.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];

%%

vi innn1.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;

}

INPUT&OUTPUT:

$lex int.l

$yacc –d int.y

$gcc lex.yy.c y.tab.c –ll –lm

$./a.out test.c

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

**RESULT:**

Thus the Converted The BNF rules into Yacc form by using code to generate abstract syntax tree.

**EX.NO:8 IMPLEMENT CONTROL FLOW ANALYSIS AND DATA FLOW ANALYSIS**

## AIM:

To write a c program to do control flow analysis and data flow analysis in a high level language program

## ALGORITHM:

Step 1: Include the necessary library files

Step 2: Declare the variables Gen[][], Kill[][] to hold the information generated, killed in each basic block.

Step 3: Declare variables in[] and out[] to store the input and output information of a basic block. Step 4: Read the input program from the user.

Step 5: Do necessary type checking

Step 6: For every new assignment like a:= x op y , update the array Gen[]

Step 7: For every new assignments, if there are any previous definitions of a, update Kill[] Step 8: Define getnum() function to return the numeric equivalent of a character input.

Step 9:Print the information obtained in Gen[], kill[], in[] and out[] Step 10: Terminate execution

## PROGRAM

#include<stdio.h> #include<conio.h>

/\* input

S : a = E | S ; S | if E then S else S | do S while E E : a + b | a

\*/

char a[50][5]; int len=0; void main()

{

char id[26][5]={"a","b","c","d","e","f"}; static char defns[26][10];

int c,i,in=0,x,k; int line[20];

char Gen[20][30]; char Kill[20][30]; char Out[20][30];

char In[20][30]; int j=0;

int getnum(char[]); clrscr(); for(i=0;i<20;i++)

{

strcpy(In[i],"");

strcpy(Gen[i],"");

strcpy(defns[i],"");

}

printf("Enter the program code \n"); do

{

c=checktype(); switch(c)

{

case 1:

printf("1. S : a = E \n"); printf("\nThe input string is "); for(i=0;i<len;i++)

{

printf(" %s",a[i]);

}

x=getnum(a[0]); //x: line no strcpy(Gen[x],a[1]);

for(in=0;in<6;in++)

if(strcmp(id[in],a[1])==0) //a[1] : variable a

{

for(j=0;j<10;j++) if(defns[in][j]==NULL)

{

defns[in][j++]=x;

printf("\nDefinition of %s is : %d",a[1],x); break;

}

}

if(j>1)

{

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

{ if(strcmp(defns[in][k],a[0])==0)

{

strcat(Kill[x],defns[in][k]); defns[in][k]='\0';

continue;

}

}

}

else strcpy(Kill[x],"NULL"); printf("\nGen [%s] : %s",a[0],a[1]);

printf("\nkill[%s] : %s",a[0],Kill[x]);

// strcpy(Kill[a[0]],) break;

case 2:

printf("\nThe input string is "); printf("2. S | S \n");

break; case 3:

printf("3. if E then S else S \n"); printf("\nThe input string is "); for(i=0;i<len;i++)

{

printf("%s ",a[i]);

}

break; case 4:

printf("4. do S while E \n"); printf("\nThe input string is "); for(i=0;i<len;i++)

{

printf("%s ",a[i]);

}

break; default:

printf("\nNO MATCH \n "); break;

}

printf("\nTo Continue press 1 ,break 0"); scanf("%d",&in);

}while(in==1);

} //endof main int checktype()

{

char s[5]; int i=0;

printf("Enter the program with line no / label \n"); do

{

scanf("%s",s);

strcpy(a[i++],s);

if(strcmp(s,"=")==0)

{

char p; scanf("%s",p);

while(strcmp(p,";")!=0)

{

strcpy(a[i++],p);

scanf("%s",p);

}

len=i; return 1;

}

else if(strcmp(s,"if")==0)

{

char p; scanf("%s",p);

while(strcmp(p,";")!=0)

{

strcpy(a[i++],p);

scanf("%s",p);

}

len=i; return 3;

}

else if(strcmp(s,"do")==0)

{

char p; scanf("%s",p);

while(strcmp(p,";")!=0)

{

strcpy(a[i++],p);

scanf("%s",p);

}

len=i; return 4;

}

}while(1);

}

int getnum(char c[])

{

if(strcmp(c,"1")==0) return 1;

else if(strcmp(c,"2")==0) return 2;

else if(strcmp(c,"3")==0) return 3;

else if(strcmp(c,"4")==0) return 4;

else if(strcmp(c,"5")==0) return 5;

else if(strcmp(c,"6")==0) return 6;

return -1;

}

**OUTPUT:**

Enter the program code

Enter the program with line no / label 1 a = b + c ;

1. S : a = E

The input string is 1 a = b + c Definition of a is : 1

Gen [1] : a kill[1] : NULL

To Continue press 1 ,break 0 1

Enter the program with line no / label 2 if a<b then a else b ;

3. if E then S else S

The input string is 2 if a<b then a else b To Continue press 1 ,break 01

Enter the program with line no / label 3 a = d ;

1. S : a = E

The input string is 3 a = d Definition of a is : 3

Gen [3] : a

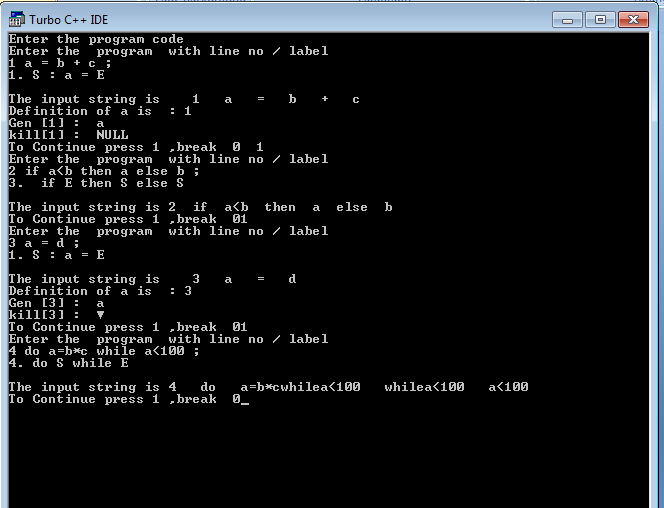
kill[3] : 1

To Continue press 1 ,break 01

Enter the program with line no / label 4 do a=b\*c while a<100 ;

4. do S while E

The input string is 4 do a=b\*cwhilea<100 whilea<100 a<100 To Continue press 1 ,break 0



RESULT:

Thus the program for control and data flow analysis was successfully executed and verified.

## Ex.No : 9. IMPLEMENTATION OF HEAP STORAGE ALLOCATION STRATEGY

**AIM:**

To write a c program to perform heap storage management at runtime for a source code

## ALGORITHM:

Step 1: Include the necessary library files

Step 2: Define a structure variable to hold the heap memory as linked list. Step 3: Define a function create() the creates a new node

Step 4 Define a function display() to print the allocated memory

Step 5: Define a function search() to find an element inside the heap memory Step 6: Define a function dele() to remove an item form the heap

Step 7: Define a function insert() to add new element in the heap and allocate meory. Step 8: If any runtime error occurs print “Memory is not allocated”

Step 9: Design a menu driven code to perform memory allocation steps Step 10: Terminate execution

**PROGRAM**

#include<stdio.h> #include<conio.h> #include<stdlib.h> #define TRUE 1

#define FALSE 0 typedef struct Heap

{

int data;

struct Heap \*next;

}node;

node \*create(); void main()

{

/\*local declarations\*/ int choice,val;

char ans; node \*head;

void display(node \*); node \*search(node \*,int); node \*insert(node \*); void dele(node \*\*); head=NULL;

do

{

clrscr();

printf("\n Program to perform various operations on heap using dynamic memory management"); printf ("\n1.Create");

printf ("\n2.Display");

printf ("\n3.Insert an element in a list"); printf ("\n4.Delete an element from list"); printf ("\n5.Quit");

printf ("\n Enter Your Choice(1-5)"); scanf("%d",&choice); switch(choice)

{

case 1:head=create(); break;

case 2:display(head);

break;

case 3:head=insert(head); break;

case 4:dele(&head); break;

case 5:exit(0); default:clrscr();

printf("Invalid Choice,Try again"); getch();

}

}while(choice!=5);

}

node \*create()

{

node \*temp,\*new,\* head; int val,flag;

char ans='y';

node \*get\_node(); temp=NULL; flag=TRUE;

/\*flag to indicate whether a new node is created for the first time or not\*/ do

{

printf("\n Enter the Element"); scanf("%d",&val);

/\*allocate new node\*/ new =get\_node(); if(new==NULL)

printf("\n Memory is not allocated"); new-> data=val;

if (flag==TRUE)/\* Executed only for the first time\*/

{

head=new;

temp=head; /\*head is the first node in the heap\*/ flag=FALSE;

}

else

{

/\*temp keeps track of the most recently created node\*/ temp->next=new;

temp=new;

}

printf("\nDo you want to enter more elements?(y/n)"); ans=getch();

}while(ans=='y');

printf("\nThe list is created"); getch();

clrscr(); return head;

}

node \*get\_node()

{

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

//using the mem. Allocation function temp->next=NULL;

return temp;

}

void display(node\*head)

{

node \*temp; temp=head; if(temp==NULL)

{

printf("\n The list is empty \n"); getch();

clrscr(); return;

}

while(temp!= NULL)

{

printf("%d->",temp-> data); temp=temp->next;

}

printf("NULL"); getch();

clrscr();

}

node \*search(node \*head,int key)

{

node\*temp; int found; temp=head;

if (temp==NULL)

{

printf("The linked list is empty\n"); getch();

clrscr(); return NULL;

}

found=FALSE;

while(temp!= NULL && found==FALSE)

{

if(temp->data != key) temp = temp->next; else

found = TRUE;

}

if(found == TRUE)

{

printf("\n The Elements is present in the list"); getch();

return temp;

}

else

printf("\n The Element is not present in the list\n"); getch();

return NULL;

}

node \*insert(node \*head)

{

int choice;

node \*insert\_head(node\*); void insert\_after(node\*); void insert\_last(node\*);

printf("\n1.Insert a node as a head node"); printf("\n2.Insert a node as a last node");

printf("\n3.Insert a node as at the intermediate position in the list "); printf("\n4.Enter your choice for insertion of node "); scanf("%d",&choice);

switch(choice)

{

case 1:head = insert\_head(head); break;

case 2:insert\_last(head); break;

case 3:insert\_after (head); break;

}

return head;

}

/\*Insertion of node at first position\*/ node \*insert\_head(node\*head)

{

node \*New,\*temp;

New = get\_node();

printf ("\n Enter the element which you want to insert "); scanf("%d",&New->data);

if(head == NULL) head = New;

else

{

temp=head;

New->next = temp; head= New;

}

return head;

}

/\*Insertion of node at last position\*/ void insert\_last(node \*head)

{

node \*New,\*temp;

New = get\_node();

printf ("\n Enter the element which you want to insert "); scanf("%d",&New->data);

if(head == NULL)

{

head = New;

}

else

{

temp=head;

while(temp->next!=NULL) temp=temp->next;

temp->next=New;

New->next=NULL;

}

}

/\*Insertion of node at intermediate position\*/ void insert\_after(node \*head)

{

int key;

node \*New,\*temp;

New = get\_node();

printf("Enter the element after which you want to insert "); scanf("%d",&key);

temp=head; do

{

if(temp->data==key)

{

printf ("Enter element which you want to insert "); scanf("%d",&New->data);

New->next=temp->next; temp->next=New; return;

}

else

temp=temp->next;

}while(temp!=NULL);

}

node \*get\_prev(node \*head,int val)

{

node \*temp, \*prev; int flag;

temp = head; if(temp == NULL) return NULL;

flag = FALSE; prev = NULL;

while(temp!=NULL && !flag)

{

if(temp->data!=val)

{

prev = temp;

temp = temp->next;

}

else

flag = TRUE;

}

if(flag) /\*if Flag is true\*/ return prev;

else

return NULL;

}

void dele(node \*\*head)

{

int key;

node \*New,\*temp,\*prev; temp=\*head;

if (temp== NULL)

{

printf ("\n The list is empty\n "); getch();

clrscr(); return;

}

clrscr();

printf("\nENTER the Element you want to delete:"); scanf("%d",&key);

temp= search(\*head,key); if(temp !=NULL)

{

prev=get\_prev(\*head,key); if(prev!= NULL)

{

prev ->next = temp-> next; free(temp);

}

else

{

\*head = temp->next;

free(temp); // using the mem. Dellocation function

}

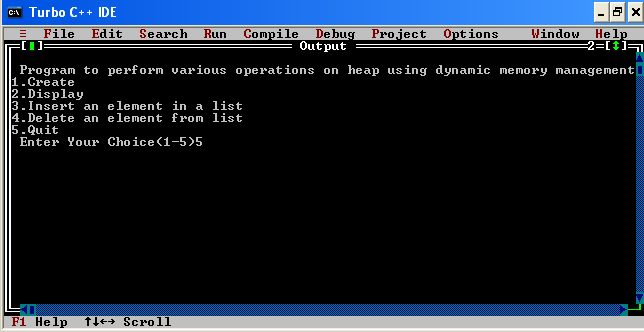
printf("\nThe Element is deleted"); getch();

clrscr();

}

}

**Output:**



**RESULT:**

Thus the program for heap management at runtime was successfully executed and verified.

## Ex. No: 10 IMPLEMENT THE BACK END OF THE COMPILER WHICH TAKES THE THREE ADDRESS CODE AND PRODUCES THE 8086 .

**AIM**

Write a C program to Implement Code Generator.

## ALGORITHM:

Step 1: Start the program with necessary header files like stdio.h, conio.h, ctype.h and ctype.h Step 2: Declare the variables, char array and files

Step 3: Enter the filename of the intermediate code to open the given file Step 4: Open the intermediate file in read only mode

Step 5: If it is not end of file read the input file and display the values like ADD, SUB, MUL Step 6: After read the intermediate file close it.

Step 7: Open one new notepad and type the intermediate code name it as k.txt Step 8: Display the intermediate code output

Step 9: stop the program execution.

## PROGRAM

#include<stdio.h> #include<conio.h> #include<ctype.h> #include<stdlib.h> void main()

{

inti=2,j=0,k=2,k1=0; charip[10],kk[10]; FILE \*fp;

clrscr();

printf("\nEnter the filename of the intermediate code"); scanf("%s",&kk);

fp=fopen(kk,"r");

if(fp==NULL)

{

printf("\nError in Opening the file"); getch();

}

clrscr(); while(!feof(fp))

{

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

}

rewind(fp);

printf("\n \n"); printf("\tStatement \t\t target code\n"); printf("\n \n");

while(!feof(fp))

{

fscanf(fp,"%s",ip);

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

printf("\t\tMOV %c,R%d\n\t",ip[i+k],j); if(ip[i+1]=='+')

printf("\t\tADD"); else printf("\t\tSUB"); if(islower(ip[i]))

printf("%c,R%d\n\n",ip[i+k1],j);

else printf("%c,%c\n",ip[i],ip[i+2]); j++;

k1=2; k=0;

}

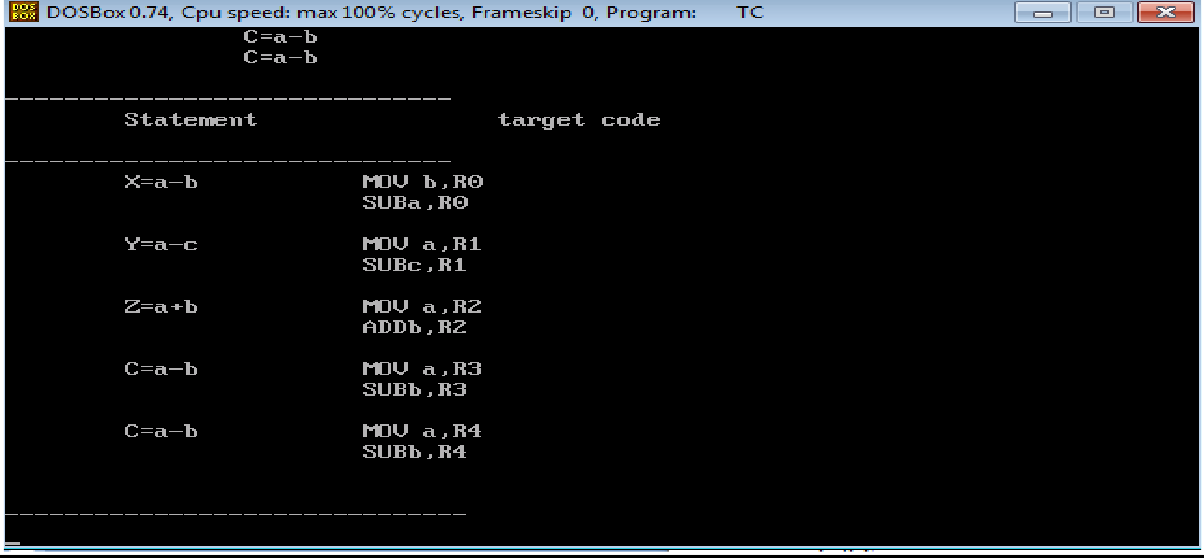
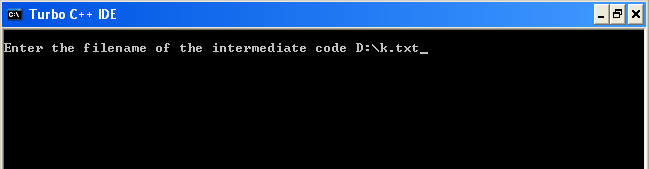
printf("\n \n"); getch();

fclose(fp);

}

**k.txt:** X=a-b Y=a-c Z=a+b C=a-b C=a-b

Output:



## RESULT

Thus the program was executed and machine code was generated.

**EX.NO:11 IMPLEMENTATION OF SIMPLE CODE OPTIMIZATION TECHNIQUES (CONSTANT FOLDING., ETC.)**

**AIM:**

To write a C program to implement the code generation 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. 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
2. Consult the address descriptor for y to determine y

is a current location of z. Again, prefer a register to a memory location if z is in both. 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

1. Generate the instruction OP z
2. 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.

**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:**

enter no of values 5 left aright: 9

left bright: c+d left eright: c+d left fright: b+e left rright: f intermediate Code a=9

b=c+d e=c+d f=b+e r=f

after dead code elimination b =c+d

e =c+d

f =b+e

r =f

eliminate common expression b =c+d

b =c+d

f =b+b

r =f

optimized codeb=c+d f=b+b

r=f

**RESULT:**

Thus the above program is compiled and executed successfully and output is verified.

# K. Ramakrishnan College of Engineering (Autonomous), Trichy

**Ex.No:12** Implementation of adding line number to a given three address code in file

**Aim:**

To write a program to adding line number to a given three address code in file

Algorithm:

Step 1: Open the vi editor and given the name as vi var.y

Step 2: Include the header files stdio.h, ctype.h in the declaration part

Step 3: Read the variable in the translation rules part

Step 4: Analyze the recognized variable, whether it begins in letter or digit.

Step 5: If the given variable is begin with letter, to display the output otherwise produce the error message

Step 6:In the main program read the expression using yylex function

Step 7: The getchar function is used to get the character and digit of the variable

Step 8: If the given variable is begin with alphabet return the accepted otherwise return the error message

Step 9.Stop the program execution

Program:

**var.y**

%{

#include<stdio.h>

#include<ctype.h>

%}

%token let dig

%%

sad: let recld '\n' {printf("accepted\n"); exit(0);}

| let '\n' {printf("accepted\n"); exit(0);}

|

|error {yyerror("rejected\n");}

;

recld: let recld

| dig recld

| let

| dig

;

%%

yylex()

{

char ch;

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

if(isalpha(ch))

return let;

if(isdigit(ch))

return dig;

return ch;

}

yyerror(char \*s)

{

printf("%s",s);

}

main()

{

printf("ENTER A variable : ");

yyparse();

}

**INPUT & OUTPUT:**

[user041@localhost ~]$ yacc -d var.y

[user041@localhost ~]$ cc y.tab.c -ll

[user041@localhost ~]$ ./a.out

ENTER A variable : adfd0102

accepted

[user041@localhost ~]$ ./a.out

ENTER A variable : 099err

syntax error rejected

**RES ULT:**

Thus the program was executed by using YACC tool.

Department of Computer Science and Engineering Page