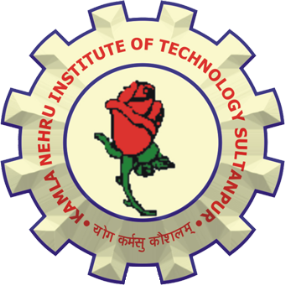
**COMPILER DESIGN LAB**

**Bachelor of Technology**

**3rd year**

**5th Semester**

**2020 - 2021**



**Submitted by :-**

**vivek verma**

**18261**

**SUBMITTED TO:**

**Prof. priyanka gautam**

ASSistant PROFESSOR

csed

**DepartMENT of Computer science & Engineering**

**Kamla Nehru Institute of Technology, Sultanpur.**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no** | **Topic** | **Page No** | **Sign** |
| 1. | Implementation of Lexical Analyzer for 'if' Statement | 3 |  |
| 2. | Implementation of Lexical Analyzer for Arithmetic Expression | 6 |  |
| 3. | C++ program for constructing an NFA from given regular expression | 9 |  |
| 4. | Construction of DFA from NFA | 13 |  |
| 5. | Implementation of Shift Reduce Parsing Algorithm | 17 |  |
| 6. | Implementation of Operator Precedence Parser | 19 |  |
| 7. | Implementation of Recursive Descent Parser | 25 |  |
| 8. | Implementation of Code Optimization Techniques | 28 |  |
| 9. | Implementation of Code Generator | 31 |  |

1. **Implementation of Lexical Analyzer for 'if' Statement**

#include<stdio.h>

#include<ctype.h>

#include<conio.h>

#include<string.h>

char vars[100][100];

int vcnt;

char input[1000],c;

char token[50],tlen;

int state=0,pos=0,i=0,id;

char\*getAddress(char str[])

{

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

if(strcmp(str,vars[i])==0)

return vars[i];

strcpy(vars[vcnt],str);

return vars[vcnt++];

}

intisrelop(char c)

{

if(c=='>'||c=='<'||c=='|'||c=='=')

return 1;

else

return 0;

}

int main(void)

{

clrscr();

printf("Enter the Input String:");

gets(input);

do

{

c=input[pos];

putchar(c);

switch(state)

{

case 0:

if(c=='i')

state=1;

break;

case 1:

if(c=='f')

{

printf("\t<1,1>\n");

state =2;

}

break;

case 2:

if(isspace(c))

printf("\b");

if(isalpha(c))

{

token[0]=c;

tlen=1;

state=3;

}

if(isdigit(c))

state=4;

if(isrelop(c))

state=5;

if(c==';')printf("\t<4,4>\n");

if(c=='(')printf("\t<5,0>\n");

if(c==')')printf("\t<5,1>\n");

if(c=='{') printf("\t<6,1>\n");

if(c=='}') printf("\t<6,2>\n");

break;

case 3:

if(!isalnum(c))

{

token[tlen]='\o';

printf("\b\t<2,%p>\n",getAddress(token));

state=2;

pos--;

}

else token[tlen+

+]=c; break;

case 4: if(!

isdigit(c))

{

printf("\b\t<3,%p>\n",&input[pos]);

state=2;

pos--;

}

break;

case 5:

id=input[pos-1];

if(c=='=')

printf("\t<%d,%d>\n",id\*10,id\*10);

else

{

printf("\b\t<%d,%d>\n",id,id);

pos--;

}

state=2;

break;

}

pos++;

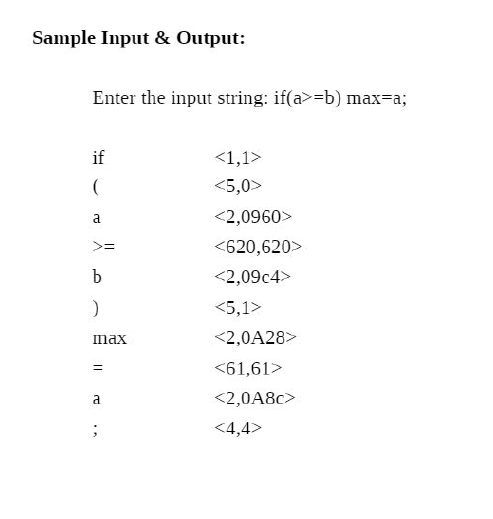
}

while(c!=0);

getch();

return 0;

};



1. **Implementation of Lexical Analyzer for Arithmetic Expression**

#include<stdio.h>

#include<ctype.h>

#include<conio.h>

#include<string.h>

char vars[100][100];

int vcnt;

char input[1000],c;

char token[50],tlen;

int state=0,pos=0,i=0,id;

char \*getAddress(char str[])

{

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

if(strcmp(str,vars[i])==0)

return vars[i];

strcpy(vars[vcnt],str);

return vars[vcnt++];

}

intisrelop(char c)

{

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

return 1;

else

return 0;

}

int main(void)

{

clrscr();

printf("Enter the Input String:");

gets(input);

do

{

c=input[pos];

putchar(c);

c);

switch(state)

{

case 0:

if(isspace(c))

printf("\b");

if(isalpha(c))

{

token[0]=c;

tlen=1;

state=1;

}

if(isdigit(c))

state=2;

if(isrelop(c))

state=3;

if(c==';')

printf("\t<3,3>\n"); if(c=='=') printf("\t<4,4>\n");

break;

case 1: if(!

isalnum(c))

{

token[tlen]='\o';

printf("\b\t<1,%p>\n",getAddress(token));

state=0;

pos--;

}

else token[tlen++]=c; break;

case 2: if(!

isdigit(c))

{

printf("\b\t<2,%p>\n",&input[pos]);

state=0;

pos--;

}

break;

case 3:

id=input[pos-1];

if(c=='=')

printf("\t<%d,%d>\n",id\*10,id\*10);

else

{

printf("\b\t<%d,%d>\n",id,id);

pos--;

}

state=0;

break;

}

pos++;

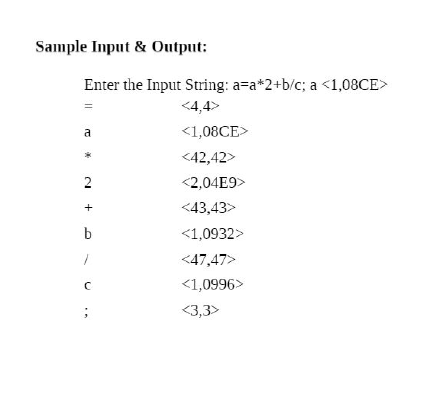
}

while(c!=0);

getch();

return 0;

}

****

1. **C++ program for constructing an NFA from given regular expression**

#include<iostream.h>

#include<conio.h>

#include<process.h>

struct node

{char start;

char alp;

node \*nstate;

}\*p,\*p1,\*p2,\*p3,\*p4,\*p5,\*p6,\*p7,\*p8;

char e='e';

void disp();

void re1()

{

p1=new(node);

p2=new(node);

p1->start='0';

p1->alp='e';

p1->nstate=p2;

p2->start='1';

p2->alp='a';

p2->nstate=NULL;

disp();

getch();

}

void re2()

{

p1=new(node);

p2=new(node);

p3=new(node);

p4=new(node);

p5=new(node);

p6=new(node);

p7=new(node);

p8=new(node);

p1->start='0';

p1->alp='e';

p1->nstate=p2;

p2->start='1';

p2->alp='a';

p2->nstate=p3;

p3->start='2';

p3->alp='e';

p3->nstate=p4;

p4->start='5';

p4->alp=' ';

p4->nstate=p5;

p5->start='0';

p5->alp='e';

p5->nstate=p6;

p6->start='3';

p6->alp='b';

p6->nstate=p7;

p7->start='4';

p7->alp='e';

p7->nstate=p8;

p8->start='5';

p8->alp=' ';

p8->nstate=NULL;

disp();

getch();

}

void re3()

{

p1=new(node);

p2=new(node);

p3=new(node);

p1->start='0';

p1->alp='a';

p1->nstate=p2;

p2->start='1';

p2->alp='b';

p2->nstate=p3;

p3->start='2';

p3->alp=' ';

p3->nstate=NULL;

disp();

getch();

}

void re4()

{

p1=new(node);

p2=new(node);

p3=new(node);

p4=new(node);

p5=new(node);

p6=new(node);

p7=new(node);

p8=new(node);

p1->start='0';

p1->alp='e';

p1->nstate=p2;

p2->start='1';

p2->alp='a';

p2->nstate=p3;

p3->start='2';

p3->alp='e';

p3->nstate=p4;

p4->start='3';

p4->alp=' ';

p4->nstate=p5;

p5->start='0';

p5->alp='e';

p5->nstate=p6;

p6->start='3';

p6->alp=' ';

p6->nstate=p7;

p7->start='2';

p7->alp='e';

p7->nstate=p8;

p8->start='1';

p8->alp=' ';

p8->nstate=NULL;

disp();

getch();}

void disp()

{p=p1;

while(p!=NULL)

{cout<<"\t"<<p->start<<"\t"<<p->alp;

p=p->nstate;}}

void main()

{p=new(node);

clrscr();

int ch=1;

while(ch!=0){

cout<<"\nMenu"<<"\n1.a"<<"\n2.a/b"<<"\n3.ab"<<"\n4.a\*";

cout<<"\n Enter the choice:";

cin>>ch;

switch(ch)

{

case 1:

{

re1();

break;

}

case 2:

{

re2();

break;

}

case 3:

{

re3();

break;

}

case 4:

{

re4();

break;

}

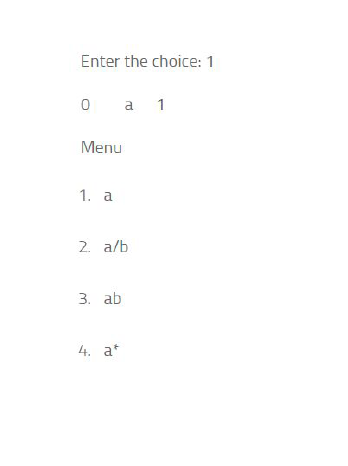
default:

{

exit(0);

}

}}}



1. **Construction of DFA from NFA.**

#include <cstdio>

#include <fstream>

#include <iostream>

#include <bitset>

#include <vector>

#include <cstring>

#include <cstdlib>

#include <algorithm>

#include <queue>

#include <set>

#define MAX\_NFA\_STATES 10

#define MAX\_ALPHABET\_SIZE 10

using namespace std;

class NFAstate

{

public:

int transitions[MAX\_ALPHABET\_SIZE][MAX\_NFA\_STATES];

NFAstate()

{

for (int i = 0; i < MAX\_ALPHABET\_SIZE; i++)

for (int j = 0; j < MAX\_NFA\_STATES; j++)

transitions[i][j] = -1;

}

}\*NFAstates;

// Representation of a DFA state

struct DFAstate

{

bool finalState;

bitset<MAX\_NFA\_STATES> constituentNFAstates;

bitset<MAX\_NFA\_STATES> transitions[MAX\_ALPHABET\_SIZE];

int symbolicTransitions[MAX\_ALPHABET\_SIZE];

};

set<int> NFA\_finalStates;

vector<int> DFA\_finalStates;

vector<DFAstate\*> DFAstates;

queue<int> incompleteDFAstates;

int N, M; // N -> No. of stattes, M -> Size of input alphabet

// finds the epsilon closure of the NFA state "state" and stores it into "closure"

void epsilonClosure(int state, bitset<MAX\_NFA\_STATES> &closure)

{

for (int i = 0; i < N && NFAstates[state].transitions[0][i] != -1; i++)

if (closure[NFAstates[state].transitions[0][i]] == 0)

{

closure[NFAstates[state].transitions[0][i]] = 1;

epsilonClosure(NFAstates[state].transitions[0][i], closure);

}

}

// finds the epsilon closure of a set of NFA states "state" and stores it into "closure"

void epsilonClosure(bitset<MAX\_NFA\_STATES> state,

bitset<MAX\_NFA\_STATES> &closure)

{

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

if (state[i] == 1)

epsilonClosure(i, closure);

}

void NFAmove(int X, int A, bitset<MAX\_NFA\_STATES> &Y)

{

for (int i = 0; i < N && NFAstates[X].transitions[A][i] != -1; i++)

Y[NFAstates[X].transitions[A][i]] = 1;

}

void NFAmove(bitset<MAX\_NFA\_STATES> X, int A, bitset<MAX\_NFA\_STATES> &Y)

{

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

if (X[i] == 1)

NFAmove(i, A, Y);

}

int main()

{

int i, j, X, Y, A, T, F, D;

// read in the underlying NFA

ifstream fin("NFA.txt");

fin >> N >> M;

NFAstates = new NFAstate[N];

fin >> F;

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

{

fin >> X;

NFA\_finalStates.insert(X);

}

fin >> T;

while (T--)

{

fin >> X >> A >> Y;

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

{

fin >> j;

NFAstates[X].transitions[A][i] = j;

}

}

fin.close();

// construct the corresponding DFA

D = 1;

DFAstates.push\_back(new DFAstate);

DFAstates[0]->constituentNFAstates[0] = 1;

epsilonClosure(0, DFAstates[0]->constituentNFAstates);

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

if (DFAstates[0]->constituentNFAstates[j] == 1 && NFA\_finalStates.find(

j) != NFA\_finalStates.end())

{

DFAstates[0]->finalState = true;

DFA\_finalStates.push\_back(0);

break;

}

incompleteDFAstates.push(0);

while (!incompleteDFAstates.empty())

{

X = incompleteDFAstates.front();

incompleteDFAstates.pop();

for (i = 1; i <= M; i++)

{

NFAmove(DFAstates[X]->constituentNFAstates, i,

DFAstates[X]->transitions[i]);

epsilonClosure(DFAstates[X]->transitions[i],

DFAstates[X]->transitions[i]);

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

if (DFAstates[X]->transitions[i]

== DFAstates[j]->constituentNFAstates)

{

DFAstates[X]->symbolicTransitions[i] = j;

break;

}

if (j == D)

{

DFAstates[X]->symbolicTransitions[i] = D;

DFAstates.push\_back(new DFAstate);

DFAstates[D]->constituentNFAstates

= DFAstates[X]->transitions[i];

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

if (DFAstates[D]->constituentNFAstates[j] == 1

&& NFA\_finalStates.find(j) != NFA\_finalStates.end())

{

DFAstates[D]->finalState = true;

DFA\_finalStates.push\_back(D);

break;

}

incompleteDFAstates.push(D);

D++;

}

}

}

ofstream fout("DFA.txt");

fout << D << " " << M << "\n" << DFA\_finalStates.size();

for (vector<int>::iterator it = DFA\_finalStates.begin(); it

!= DFA\_finalStates.end(); it++)

fout << " " << \*it;

fout << "\n";

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

{

for (j = 1; j <= M; j++)

fout << i << " " << j << " "

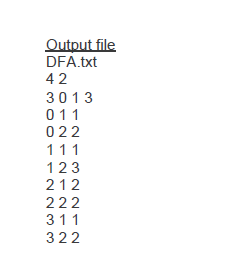
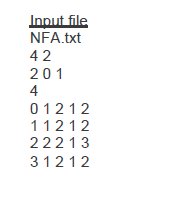
<< DFAstates[i]->symbolicTransitions[j] << "\n";

}

fout.close();

return 0;

}



1. **Implementation of Shift Reduce Parsing Algorithm**

#include <bits/stdc++.h>

using namespace std;

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

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

void check()

{ strcpy(ac,"REDUCE TO E -> ");

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

{

if(stk[z] == '4')

{

printf("%s4", ac);

stk[z] = 'E';

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

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

}

}

for(z = 0; z < c - 2; z++)

{

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

stk[z + 2] == '2')

{

printf("%s2E2", ac);

stk[z] = 'E';

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

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

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

i = i - 2;

}

}

for(z = 0; z < c - 2; z++)

{

if(stk[z] == '3' && stk[z + 1] == 'E' && stk[z + 2] == '3')

{

printf("%s3E3", ac);

stk[z]='E';

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

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

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

i = i - 2;

}

}

return ;

}

int main()

{

printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");

strcpy(a,"32423");

c=strlen(a);

strcpy(act,"SHIFT");

printf("\nstack \t input \t action");

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

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

{

printf("%s", act);

stk[i] = a[j];

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

a[j]=' ';

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

check();

}

check();

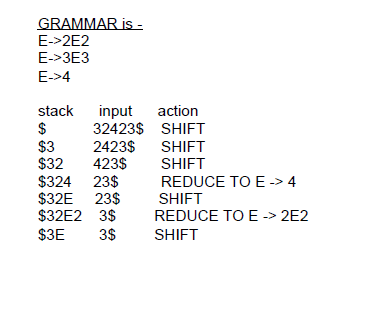
if(stk[0] == 'E' && stk[1] == '\0')

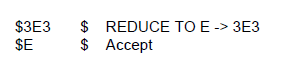
printf("Accept\n");

else

printf("Reject\n");

}





1. **Implementation of Operator Precedence Parser**

#include<iostream>

#include<string>

#include<deque>

using namespace std;

int n,n1,n2;

int getPosition(string arr[], string q, int size)

{

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

{

if(q == arr[i])

return i;

}

return -1;

}

int main()

{

string prods[10],leads[10],trails[10],nonterms[10],terms[10];

char op\_table[20][20] = {};

cout<<"Enter the number of productions : ";

cin>>n;

cin.ignore();

cout<<"Enter the productions"<<endl;

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

{

getline(cin,prods[i]);

}

cout<<"Enter the number of Terminals : ";

cin>>n2;

cin.ignore();

cout<<"Enter the Terminals"<<endl;

for(int i=0;i<n2;i++)

{

cin>>terms[i];

}

terms[n2] = "$";

n2++;

cout<<"Enter the number of Non-Terminals : ";

cin>>n1;

cin.ignore();

for(int i=0;i<n1;i++)

{

cout<<"Enter Non-Terminal : ";

getline(cin,nonterms[i]);

cout<<"Enter Leads of "<<nonterms[i]<<" : ";

getline(cin,leads[i]);

cout<<"Enter Trails of "<<nonterms[i]<<" : ";

getline(cin,trails[i]);

}

cout<<"Enter the Rules (exit to stop)"<<endl;

string rule = "";

while(rule != "exit")

{

getline(cin,rule);

if(rule[0] == '1')

{

int row = getPosition(terms,rule.substr(2,1),n2);

int column = getPosition(terms,rule.substr(4,1),n2);

op\_table[row][column] = '=';

}

if(rule[0] == '2')

{

int ntp = getPosition(nonterms,rule.substr(4,1),n1);

int row = getPosition(terms,rule.substr(2,1),n2);

for(int j=0;j<leads[ntp].size();j++)

{

int col = getPosition(terms,leads[ntp].substr(j,1),n2);

op\_table[row][col] = '<';

}

}

if(rule[0] == '3')

{

int col = getPosition(terms,rule.substr(4,1),n2);

int ntp = getPosition(nonterms,rule.substr(2,1),n1);

for(int j=0;j<trails[ntp].size();j++)

{

int row = getPosition(terms,trails[ntp].substr(j,1),n2);

op\_table[row][col] = '>';

}

}

}

for(int j=0;j<leads[0].size();j++)

{

int col = getPosition(terms,leads[0].substr(j,1),n2);

op\_table[n2-1][col] = '<';

}

for(int j=0;j<trails[0].size();j++)

{

int row = getPosition(terms,trails[0].substr(j,1),n2);

op\_table[row][n2-1] = '>';

}

cout<<endl;

cout<<"Grammar"<<endl;

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

{

cout<<prods[i]<<endl;

}

for(int j=0;j<n2;j++)

cout<<"\t"<<terms[j];

cout<<endl;

for(int i=0;i<n2;i++)

{

cout<<terms[i]<<"\t";

for(int j=0;j<n2;j++)

{

cout<<op\_table[i][j]<<"\t";

}

cout<<endl;

}

char c;

do{

string ip;

deque<string> op\_stack;

op\_stack.push\_back("$");

cout<<"Enter the string to be parsed : ";

getline(cin,ip);

ip.push\_back('$');

cout<<"Stack\ti/p Buffer\tRelation\tAction"<<endl;

while(true)

{

for(int i=0;i<op\_stack.size();i++)

cout<<op\_stack[i];

cout<<"\t";

cout<<ip<<"\t";

int row = getPosition(terms,op\_stack.back(),n2);

int column = getPosition(terms,ip.substr(0,1),n2);

if(op\_table[row][column] == '<')

{

op\_stack.push\_back("<");

op\_stack.push\_back(ip.substr(0,1));

ip = ip.substr(1);

cout<<"\t"<<"<\t\tPush";

}

else if(op\_table[row][column] == '=')

{

op\_stack.push\_back("=");

op\_stack.push\_back(ip.substr(0,1));

ip = ip.substr(1);

cout<<"\t"<<"=\t\tPush";

}

else if(op\_table[row][column] == '>')

{

string last;

do

{

op\_stack.pop\_back();

last = op\_stack.back();

op\_stack.pop\_back();

}while(last != "<");

cout<<"\t"<<">\t\tPop";

}

else

{

if(ip[0] == '$' && op\_stack.back() == "$")

{

cout<<"\t\t\tAccept\nString Parsed."<<endl;

break;

}

else

{

cout<<endl<<"String cannot be Parsed."<<endl;

break;

}

}

cout<<endl;

}

cout<<"Continue?(Y/N) ";

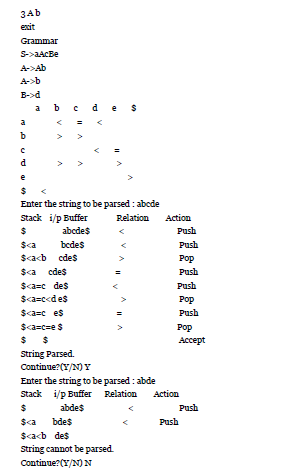
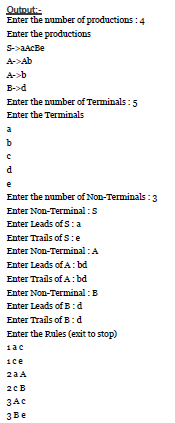
cin>>c;

cin.ignore();

}while(c=='y' || c=='Y');

return 0;

}



1. **Implementation of Recursive Descent Parser.**

#include <iostream>

#include <stdlib.h>

using namespace std;

/\*

E->TE'

E'->+TE'|-TE'|null

T-> FT'

T'->\*FT'|/FT'|null

F-> id|num|(E)

\*/

int count = 0;

void E();

void Ed();

void T();

void Td();

void F();

string expr;

int main() {

cin >> expr;

int l = expr.length();

expr += "$";

E();

if (l == count)

cout << "Accepted" << endl;

else

cout << "Rejected" << endl;

}

void E() {

cout << "E->TE'" << endl;

T();

Ed();

}

void Ed() {

if (expr[count] == '+') {

count++;

cout << "E'->+TE'" << endl;

T();

Ed();

}

else if (expr[count] == '-') {

count++;

cout << "E'->-TE'" << endl;

T();

Ed();

}

else {

cout << "E'->null" << endl;

}

}

void T() {

cout << "T->FT'" << endl;

F();

Td();

}

void Td() {

if (expr[count] == '\*') {

count++;

cout << "T'->\*FT'" << endl;

F();

Td();

}

else if (expr[count] == '/') {

count++;

cout << "T'->/FT'" << endl;

F();

Td();

}

else {

cout << "T'->null" << endl;

}

}

void F() {

if (isalpha(expr[count])) {

count++;

cout << "F->id" << endl;

} else if (isdigit(expr[count])) {

count++;

cout << "F->digit" << endl;

} else if (expr[count] == '(') {

count++;

cout << "F->(E)" << endl;

E();

if (expr[count] != ')') {

cout << "Rejected" << endl;

exit(0);

}

count++;

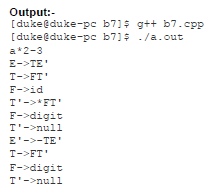
} else {

cout << "Rejected" << endl;

exit(0);

}

}





1. **Implementation of Code Optimization Techniques**

#include<stdio.h>

#include<conio.h>

#include<string.h>

struct op

{ char l;

char r[20];

}

op[10],pr[10];

void main()

{ int a,i,k,j,n,z=0,m,q;

char \*p,\*l;

char temp,t;

char \*tem;

clrscr();

printf("Enter the Number of Values:");

scanf("%d",&n);

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

{

printf("left: ");

op[i].l=getche();

printf("\tright: ");

scanf("%s",op[i].r);

}

printf("Intermediate Code\n") ;

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

{

printf("%c=",op[i].l);

printf("%s\n",op[i].r);

}

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

{ temp=op[i].l;

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

{

p=strchr(op[j].r,temp);

if(p)

{

pr[z].l=op[i].l;

strcpy(pr[z].r,op[i].r);

z++; }}}

pr[z].l=op[n-1].l;

strcpy(pr[z].r,op[n-1].r);

z++;

printf("nAfter Dead Code Eliminationn");

for(k=0;k<z;k++) {

printf("%ct=",pr[k].l);

printf("%sn",pr[k].r);

}

for(m=0;m<z;m++) {

tem=pr[m].r;

for(j=m+1;j<z;j++) {

p=strstr(tem,pr[j].r);

if(p) {

t=pr[j].l;

pr[j].l=pr[m].l;

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

l=strchr(pr[i].r,t) ;

if(l) {

a=l-pr[i].r;

printf("pos: %d",a);

pr[i].r[a]=pr[m].l; }}}}}

printf("Eliminate Common Expression\n");

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

{

printf("%c\t=",pr[i].l);

printf("%s\n",pr[i].r);

}

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

{

for(j=i+1;j<z;j++)

{

q=strcmp(pr[i].r,pr[j].r);

if((pr[i].l==pr[j].l)&&!q)

{

pr[i].l='\0';

strcpy(pr[i].r,'\0');

}}}

printf("Optimized Code\n");

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

{ if(pr[i].l!='\0')

{

printf("%c=",pr[i].l);

printf("%s\n",pr[i].r);

}

}

getch();

}

**INPUT & OUTPUT:**

Enter the Number of Values:5

left: a right: 9

left: b right: c+d

left: e right: c+d

left: f right: b+e

left: r right: f

Intermediate Code

a=9

b=c+d

e=c+d

f=b+e

r=f

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

b =c+d

b =c+d

f =b+b

r =f

Optimized Code

b=c+d

f=b+b

r=f

1. **Implementation of Code Generator**

#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:-**

Enter the Three Address Code:

a=b+c

c=a\*c

exit

The Equivalent Assembly Code is:

Mov R0,b

Add c, R0

Mov a, R0

Mov R1,a

Mul c,R1

Mov c,R1