Program 1: Lexical Analyser

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

void keyword(char str[10]) {
      if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||str
cmp("int",str
      )==0||strcmp("float",str)==0||strcmp("char",str)==0||strcmp("double",str)
==0||strcmp(
      "static",str)==0||strcmp("switch",str)==0||strcmp("case",str)==0
            printf("\n%s is a keyword",str);
      else
            printf("\n%s is an identifier",str);
}

int main() {
      FILE*f1,*f2,*f3;
      char c,str[10],st1[10];
      int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
      printf("\n Enter the C Program");/* gets(st1);*/
      f1=fopen("input","w");
      while((c=getchar())!=EOF)
            putc(c,f1);
      fclose(f1);
      f1=fopen("input","r");
      f2=fopen("identifier","w");
      f3=fopen("specialchar","w");
      while((c=getc(f1))!=EOF) {
            if(isdigit(c)) {
                  tokenvalue=c-'0';
                  c=getc(f1);
                  while(isdigit(c)) {
                        tokenvalue*=10+c-'0';
                        c=getc(f1);
                  }
            num[i++]=tokenvalue;
            ungetc(c,f1);
            } else if(isalpha(c)) {
                  putc(c,f2);
                  c=getc(f1);
                  while(isdigit(c)||isalpha(c)||c=='-'||c=='$') {
                        putc(c,f2);
                        c=getc(f1);
                  }
                  putc(' ',f2);
                  ungetc(c,f1);
```

```
        } else if(c==' '||c=='\t')
              printf(" ");
        else if(c=='\n')
              lineno++;
        else
              putc(c,f3);
  }
  fclose(f2);
  fclose(f3);
  fclose(f1);
  printf("\n The no's in the program are");
  for(j=0;j<i;j++)
  printf("%d",num[j]);
  printf("\n");
  f2=fopen("identifier","r");
  k=0;
  printf("The keywords and identifiers are");
  while((c=getc(f2))!=EOF) {
        if(c!=' ')
              str[k++]=c;
        else {
              str[k]='\0';
              keyword(str);
              k=0;
        }
  }
  fclose(f2);
  f3=fopen("specialchar","r");
  printf("\n special characters are");
  while((c=getc(f3))!=EOF)
        printf("%c",c);
  printf("\n");
  fclose(f3);
  printf("Total no.of lines are:%d",lineno);
}
```

Program 2: Implementation of Lexical Analyser using Lex tool

```
%{
      int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
struct |
typedef |
do |
if |
break |
continue |
void |
switch |
return |
else |
goto {printf("\n\t%s is a keyword",yytext);}
"/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{ {if(!COMMENT)printf("\n BLOCK BEGINS");}
\} {if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s
IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);} \)(\:)?
{if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
```

```
int main(int argc, char **argv) {
      FILE *file;
      file=fopen("var.c","r");
      if(!file) {
            printf("could not open the file"); exit(0);
      }
      yyin=file; yylex(); printf("\n"); return(0); }
int yywrap() {
      return(1);
}
```

Program 3: Lex program to count no of lines, spaces & tabs

```
%{
      #include int sc=0,wc=0,lc=0,cc=0;
%}
%%
[\n] { lc++; cc+=yyleng;}
[ \t] { sc++; cc+=yyleng;}
[^\t\n ]+ { wc++; cc+=yyleng;}
%%
int main(int argc ,char* argv[ ]) {
      printf("Enter the input:\n");
      yylex();
      printf("The number of lines=%d\n",lc);
      printf("The number of spaces=%d\n",sc);
      printf("The number of words=%d\n",wc);
      printf("The number of characters are=%d\n",cc);
}
```

Program 4: Lowercase to Uppercase

```
%{
      #include<stdio.h>
      int i;
%}

%%

[a-z A-Z]* {
      for(i=0;i<yyleng;i++) {
          if((yytext[i]=='a')&&(yytext[i+1]=='b')&&(yytext[i+2]=='c')){
                  yytext[i]='A';yytext[i+1]='B';
                  yytext[i+2]='C'; }
          }
                  printf("Converted String is:%s",yytext);
      }
[\t]*return;
.* {ECHO;}
\n {printf("%s",yytext);}
%%
main() {
      printf("Enter the string:"); yylex();
}
int yywrap() {
      return 1;
}
```

Program 5: Vowels & Consonants

```
%{
      int vow_count=0;
      int const_count=0;
%}
%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap(){
}
int main() {
      printf("Enter the string of vowels and consonants:");
      yylex();
      printf("Number of vowels are: %d\n",vow_count);
      printf("Number of consonants are: %d\n", const_count);
      return 0;
}
```

Program 6: Recognition of arithmetic expression

Lex Part

```
%{
      #include "y.tab.h"
      extern yylval;
%}
%%
[0-9]+ {
      yylval = atoi(yytext);
      return NUMBER;
}
[a-zA-Z]+ { return ID; }
[ \t]+ ;
\n { return 0; }
. { return yytext[ 0]; }
%%
```

YACC Part

```
%{
      #include<stdio.h>
%}
%token NUMBER ID
%left '+' '-'
%left '*' '/'
%%
E : T {
      printf("Result = %d\n ", $$);
      return 0 ;
}
T :
      T '+' T { $$ = $1 + $3; }
      | T '-' T { $$ = $1 - $3; }
      | T '*' T { $$ = $1 * $3; }
      | T '/' T { $$ = $1 / $3; }
      | '-' NUMBER { $$ = -$2; }
      | '-' ID { $$ = -$2; }
      | '(' T ')' { $$ = $2; }
      | NUMBER { $$ = $1; }
      | ID { $$ = $1; } ;
%%
int main() {
      printf("Enter the expression\n ");
      yyparse();
}
int yyerror(char* s) {
      printf( "\nExpression is invalid\n "); }
```

Program 7: Recognize valid variable (letter followed by numbers)

LEX part

```
%{
      #include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return letter;
[0-9] return digit;
. return yytext[0];
\n return 0;
%%
int yywrap() {
      return 1;
}
```

YACC part

```
%{
      #include int valid=1;
%}
%token digit letter
%%
start : letter s
s : letter s
      | digit s
      |
      ;
%%
int yyerror() {
      printf("\nIts not a identifier!\n"); valid=0; return 0;
}
int main() {
      printf("\nEnter a name to tested for identifier ");
      yyparse();
      if(valid) {
            printf("\nIt is a identifier!\n");
      }
}
```

Program 8: Calculator using LEX

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
      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, Division, 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;
}
```

Program 9: First & Follow

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

void followfirst(char, int, int);
void follow(char c);
void findfirst(char, int, int);
int count, n = 0;
calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;
int main(int argc, char **argv) {
      int jm = 0;
      int km = 0;
      int i, choice;
      char c, ch;
      count = 8; // The Input grammar strcpy(production[0], "E=TR");
      strcpy(production[1], "R=+TR");
      strcpy(production[2], "R=#");
      strcpy(production[3], "T=FY");
      strcpy(production[4], "Y=*FY");
      strcpy(production[5], "Y=#");
      strcpy(production[6], "F=(E)");
      strcpy(production[7], "F=i");
      int kay;
      char done[count];
      int ptr = -1;
      for(k = 0; k < count; k++) {
            for(kay = 0; kay < 100; kay++) {
                  calc_first[k][kay] = '!';
            }
      }
      int point1 = 0, point2, xxx;
      for(k = 0; k < count; k++) {
            c = production[k][0];
            point2 = 0;
            xxx = 0;
            for(kay = 0; kay <= ptr; kay++)
                  if(c == done[kay])
                        xxx = 1;
                  if (xxx == 1) continue;
                  findfirst(c, 0, 0);
```

```
    ptr += 1;
    done[ptr] = c;
    printf("\n First(%c) = { ", c); calc_first[point1][point2++]
    = c;
    for(i = 0 + jm; i < n; i++) {
        int lark = 0, chk = 0;
        for(lark = 0; lark < point2; lark++) {
            if (first[i] == calc_first[point1][lark]) {
                chk = 1;
                break;
            }
        }
        if(chk == 0) {
            printf("%c, ", first[i]);
            calc_first[point1][point2++] = first[i];
        }
    }
    printf("}\n"); jm = n; point1++;
}
printf("\n");
printf("----------------------------------------------\n\n");
char donee[count];
ptr = -1;
for(k = 0; k < count; k++) {
    for(kay = 0; kay < 100; kay++) {
        calc_follow[k][kay] = '!';
    }
}
point1 = 0;
int land = 0;
for(e = 0; e < count; e++) {
    ck = production[e][0];
    point2 = 0;
    xxx = 0;
    for(kay = 0; kay <= ptr; kay++)
        if(ck == donee[kay])
            xxx = 1;
        if (xxx == 1) continue;
        land += 1;
        follow(ck);
        ptr += 1;
        donee[ptr] = ck;
        printf(" Follow(%c) = { ", ck);
        calc_follow[point1][point2++] = ck;
        for(i = 0 + km; i < m; i++) {
            int lark = 0, chk = 0;
            for(lark = 0; lark < point2; lark++) {
                if (f[i] == calc_follow[point1][lark]){
                    chk = 1;
```

```
                                        break;
                                    }
                            }
                            if(chk == 0) {
                                printf("%c, ", f[i]);
                                calc_follow[point1][point2++] = f[i];
                            }
                    }
                    printf(" }\n\n");
                    km = m;
                    point1++;
        }
}
void follow(char c) {
        int i, j;
        if(production[0][0] == c) {
                f[m++] = '$';
        }
        for(i = 0; i < 10; i++) {
                for(j = 2;j < 10; j++) {
                        if(production[i][j] == c) {
                                if(production[i][j+1] != '\0') {
                                        followfirst(production[i][j+1], i, (j+2)); }
                                if(production[i][j+1]=='\0' && c!=production[i][0]) {
                                        follow(production[i][0]);
                                }
                        }
                }
        }
}
void findfirst(char c, int q1, int q2) {
        int j;
        if(!(isupper(c))) {
                first[n++] = c;
        }
        for(j = 0; j < count; j++) {
                if(production[j][0] == c) {
                        if(production[j][2] == '#') {
                                if(production[q1][q2] == '\0') first[n++] = '#';
                                else if(production[q1][q2] != '\0' && (q1 != 0 || q2 !=
                        0)) {
                                        findfirst(production[q1][q2], q1, (q2+1)); } else
                                first[n++] = '#';
                } else if(!isupper(production[j][2])) {
                        first[n++] = production[j][2];
                } else {
                        findfirst(production[j][2], j, 3);
                }
        }
```

```
} }
void followfirst(char c, int c1, int c2) {
      int k;
      if(!(isupper(c)))
            f[m++] = c;
      else {
            int i = 0, j = 1;
            for(i = 0; i < count; i++) {
                  if(calc_first[i][0] == c) break;
            }
            while(calc_first[i][j] != '!') {
                  if(calc_first[i][j] != '#') {
                        f[m++] = calc_first[i][j];
                  } else {
                        if(production[c1][c2] == '\0') {
                              follow(production[c1][0]);
                        } else {
                              followfirst(production[c1][c2], c1, c2+1);
                        }
                  }
                  j++;
            }
      }
}
```

Program 10: Recursive Descent Parser

```
#include<stdio.h>
#include<string.h>
#define SUCCESS 1
#define FAILED 0 i
nt E(),Edash(), T(), Tdash(), F();
const char *cursor;
char string[64];
int main() {
      puts("Enter the string");
      sscanf("i+(i+i)*i" , "%s ",string);
      cursor=string; puts("");
      puts("input Action");
      puts("--------------------");
      if (E() && *cursor == '\0' ) {
            puts("------------------");
            puts("String is successfully parsed");
            return 0 ;
      } else {
            puts("------------------");
            puts("Error in parsing String");
            return 1 ;
      }
}
int E() {
      printf( "%-16s E -> T E'\n ",cursor);
      if (T()) {
            if (Edash())
                  return SUCCESS;
            else return FAILED;
      } else return FAILED;
}
int Edash() {
      if (*cursor == '+' ) {
            printf( "%-16s E' -> + T E'\n ",cursor);
            cursor++;
            if (T()) {
                  if (Edash())
                        return SUCCESS;
                  else return FAILED;
            } else return FAILED;
      } else {
            printf( "%-16s E' -> $\n ",cursor);
            return SUCCESS;
      }
}
int T() {
      printf( "%-16s T -> F T'\n ",cursor);
```

```
    if (F()) {
        if (Tdash()) return SUCCESS;
        else return FAILED;
    } else return FAILED;
}
int Tdash() {
    if (*cursor == '*' ) {
        printf( "%-16s T' -> *F T'\n ",cursor);
        cursor++;
        if (F()) {
            if (Tdash()) return SUCCESS;
            else return FAILED;
        } else return FAILED;
    } else {
        printf( "%-16s T' -> $\n ", cursor);
        return SUCCESS;
    }
}
int F() {
    if (*cursor == '(' ) {
        printf( "%-16s F -> (E)\n ", cursor);
        cursor++;
        if (E()) {
            if (*cursor == ')' ) {
                cursor++;
                return SUCCESS;
            } else return FAILED;
        } else return FAILED;
    } else if (*cursor == 'i' ) {
        cursor++;
        printf( "%-16s F -> i\n ",cursor);
        return SUCCESS;
    } else return FAILED;
}
```

Program 11: Shift Reduce Parser

```
#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();
int 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();
            }
      }
}
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=i2;
            }
      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=i2;
            }
      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;
            }
}
```

Program 12: Constant Propogation

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
void input();
void output();
void change(int p,char *res);
void constant();
struct expr {
      char op[2],op1[5],op2[5],res[5];
      int flag;
}arr[10];
int n;
void main(){
      input();
      constant();
      output();
}
void input(){
      int i;
      printf("\n\nEnter the maximum number of expressions : ");
      scanf("%d",&n);
      printf("\nEnter the input : \n");
      for(i=0;i<n;i++){
            scanf("%s",arr[i].op);
            scanf("%s",arr[i].op1);
            scanf("%s",arr[i].op2);
            scanf("%s",arr[i].res);
            arr[i].flag=0;
      }
}
void constant(){
      int i;
      int op1,op2,res;
      char op,res1[5];
      for(i=0;i<n;i++){
            if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) ||
            strcmp(arr[i].op,"=")==0){
                  op1=atoi(arr[i].op1);
                  op2=atoi(arr[i].op2);
                  op=arr[i].op[0];
                  switch(op){
                        case '+':
                        res=op1+op2;
                        break;
                        case '-':
                        res=op1-op2;
                        break;
```

```
                    case '*':
                    res=op1*op2;
                    break;
                    case '/':
                    res=op1/op2;
                    break;
                    case '=':
                    res=op1;
                    break;
                }
                sprintf(res1,"%d",res);
                arr[i].flag=1;
                change(i,res1);
            }
        }
}
void constant(){
        int i;
        int op1,op2,res;
        char op,res1[5];
        for(i=0;i<n;i++){
            if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) ||
        strcmp(arr[i].op,"=")==0){
                op1=atoi(arr[i].op1);
                op2=atoi(arr[i].op2);
                op=arr[i].op[0];
                switch(op){
                    case '+':
                    res=op1+op2;
                    break;
                    case '-':
                    res=op1-op2;
                    break;
                    case '*':
                    res=op1*op2;
                    break;
                    case '/':
                    res=op1/op2;
                    break;
                    case '=':
                    res=op1;
                    break;
                }
                sprintf(res1,"%d",res);
                arr[i].flag=1;
                change(i,res1);
            }
        }
}
```

Program 13: Code Optimization Techniques

```
#include<stdio.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;
      printf("Enter the Number of Values:");
      scanf("%d",&n);
      for(i=0;i<n;i++){
            printf("left:");
            scanf("%s",&op[i].l);
            printf("right:");
            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<n1;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[n1].l;
      strcpy(pr[z].r,op[n1].r);
      z++;
      printf("After Dead Code Elimination\n");
      for(k=0;k<z;k++){
            printf("%c=",pr[k].l);
            printf("%s\n",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\n",a);
                                      pr[i].r[a]=pr[m].l;
                                }
                          }
                    }
              }
        }
        printf("Eliminate Common Expression\n");
        for(i=0;i<z;i++){
              printf("%c=",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';
                    }
              }
        }
        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);
              }
        }
}
```

Program 14: Intermediate Code Generation

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
int isp(char item);
void output(char item);
void push(char item);
char pop(void);
void quad(void);
char exp[20];
char res[20];
char a[20],opr[20],opd1[20],opd2[20],result[20];
int st[20],value[20];
int top=0,z=0,i=0,op1,op2,k,j,p,l;
char x,item;
void main(){
      printf("Enter the infix expression:");
      gets(exp);
      l=strlen(exp);
      push('#');
      while((item=exp[i])!='\0'){
            if(isalpha(exp[i])) output(item);
            else if(item=='+'||item=='-'||item=='*'||item=='/'||item=='^')
                  push(item);
            else if(item=='(')
                  push(item);
            else if(item==')'){
                  while((x=pop())!='(') output(x);
            }
            else if(isp(x=pop())<isp(item)){
                  push(x);
                  push(item);
            } else {
                  output(x);
                  push(item);
            }
            i++;
      }
      while((x=pop())!='#')
            output(x);
      printf("Postfix expression:");
      puts(res);
      quad();
}
int isp(char item){
      if((item=='+')||(item=='-'))
            return(1);
      else if((item=='*')||(item=='/'))
```

```
            return(2);
        else if((item=='^'))
            return(3);
        else return(0);
}
void output(char item){
        res[z++]=item;
}
void push(char item){
        a[++top]=item;
}
char pop(void){
        item=a[top--];
        return(item);
}
void quad(){
        int i,x=0;
        char m,n,p,temp,str1[5],str2[5];
        printf("\noperator\top1\top2\tresult\n");
        printf("--------------------------------------");
        for(i=0;i<l;i++){
            if(isalnum(res[i])){
                push(res[i]);
            } else {
                if(isalpha(m=pop())){
                    str1[0]=m;
                    str1[1]='\0';
                } else {
                    str1[0]='t';
                    str1[1]=m;
                    str1[2]='\0';
                }
                if(isalpha(n=pop())) {
                    str2[0]=n;
                    str2[1]='\0';
                } else {
                    str2[2]='\0';
                }
                x++;
                printf("\n%c\t\t%s\t%s\tt%d\n",res[i],str2,str1,x);
                temp = x+'0';
                push(temp);
            }
        }
}
```

Program 15: Backend Compiler

```
#include<stdio.h>
#include<string.h>
void main(){
      char data[50],res[5],OP1[5],OP2[5],val;
      int i,OP,j,k,q,n,m;
      while(1){
            n=q=j=k=OP=m=0;
            strcpy(res,"\n");
            printf("Enter the intermediate code enter'quit' to end\n");
            scanf("%s",data);
            if(strcmp(data,"quit")==0)
                  break;
            for(i=0;i<strlen(data);i++){
                  if(data[i]=='=')
                        break;
                  else
                        res[m++]=data[i];
                  for(i=m+1;i<strlen(data);i++){
                        if(data[i]=='+'||data[i]=='/'||data[i]=='*'||data[i]=='
                  -'){
                              OP=i;
                              val=data[i];
                        }
                        if((isalpha(data[i])&&OP==0)||(isdigit(data[i])&&n==1&&
                  OP==0)){
                              OP1[j++]=data[i];
                              n=1;
                        } else if
((isalpha(data[i])&&OP!=0)||(isdigit(data[i])&&q==1&&OP!=0)){
                              OP2[k+1]='\0';
                              OP2[k++]=data[i];
                              q=1;
                        } else if(isdigit(data[i])&&OP==0&&n==0){
                              OP1[0]='#';
                              OP1[++j]=data[i];
                        } else if(isdigit(data[i])&&OP!=0&&q==0){
                              OP2[0]=='#';
                              OP2[++k]=data[i];
                        }
                  }
                  printf("\n MOV %s,R1",OP1);
                  if(OP2[0]!='#'){
                        printf("\n MOV %s,R2",OP2);
                        strcpy(OP2,"R2");
                  }
```

```
                        switch(val){
                                case '+':printf("\n ADD %s,R1",OP2);
                                break;
                                case '-':printf("\n SUB %s,R1",OP2);
                                break;
                                case '*':printf("\n MUL %s,R1",OP2);
                                break;
                                case '/':printf("\n DIV %s,R1",OP2);
                                break;
                        }
                        printf("\n MOV R1,%s\n",res);
                }
        }
}
```