|  |  |
| --- | --- |
| %{  %}  %%  [6-9][0-9]{9} {printf("\nMobile Number Valid\n");}  .+ {printf("\nMobile Number Invalid\n");}  %%  int yywrap(void) {}  int main() {  printf("\nEnter Mobile Number: ");  yylex();  printf("\n");  return 0;  }  %{  %}  %%  [a-z.0-9\_]+@[a-z]+".com"|".in" {printf("it is valid");}  .+ {printf("it is not valid");}  %%  int yywrap(){}  int main()  {  printf("enter the mail:");  yylex();  }  %{  %}  %%  [a-zA-Z][a-zA-Z0-9]+ {printf("\n%s is IDENTIFIER",yytext);}  .+ {printf("\n%s is NOT A IDENTIFIER",yytext);}  %%  int yywrap() {}  int main()  {  while(yylex());  }  %{  %}  %%  [0-9][0-9]\/[0-1][0-9]\/[1-2][0-9]{3} {printf("DOB is valid");}  .+ {printf("DOB is not valid");}  %%  int yywrap() {}  int main()  {  yylex();  }  5) %{  #include<stdio.h>  %}  %%  if|else|while|int|switch|for|char { printf("its a keyword");}  [a-zA-Z0-9]+ { printf("\n%s is IDENTIFIER", yytext);}  %%  int yywrap(){}  int main()  {  while(yylex());  }  6) %{  #include<stdio.h>  #include<string.h>  int i = 0;  %}  %%  ([a-zA-Z0-9])\* {i++;}  "\n" {printf("%d\n", i); i = 0;}  %%  int yywrap(){}  int main()  {  printf("Enter the Sentence:");  yylex();  return 0;  }  7) %{  int counter = 0;  %}  %%  [a-zA-Z]+ {if(yyleng>counter) counter = yyleng;}  %%  int yywrap(void)  {}  int main()  {  printf("Enter Input:");  yylex();  printf("largest %d",counter);  printf("\n");  }  8) %{  #include <stdio.h>  %}  %%  [A-Z]+ { printf("Capital Word: %s\n", yytext); }  . ; // ignore any other characters  %%  int main() {  yylex();  return 0;  }  int yywrap() {  return 1;  }  9)  %{  #include <stdio.h>  %}  %%  [0-9]+ { printf("Input is a digit: %s\n", yytext); }  [^0-9]+ { printf("Input is not a digit: %s\n", yytext); }  %%  int main() {  yylex();  return 0;  }  int yywrap() {  return 1;  }  10) %{  #include <stdio.h>  int vowel\_count = 0;  int consonant\_count = 0;  %}  %%  [aAeEiIoOuU] { vowel\_count++; }  [b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z] { consonant\_count++; }  .|\n { /\* Ignore other characters \*/ }  %%  int main() {  printf("Enter a sentence: ");  yylex();  printf("Number of vowels: %d\n", vowel\_count);  printf("Number of consonants: %d\n", consonant\_count);  return 0;  }  int yywrap() {  return 1;  }  **URL.lex**  %{  %}  %%  ((http)|(ftp))s?:\/\/[a-zA-Z0-9](.[a-z])+(.[a-zA-Z0-9+=?]\*)\* {printf("\nURL Valid\n");}  .+ {printf("\nURL Invalid\n");}  %%  void main()  {  printf("\nEnter URL : ");  yylex();  printf("\n");  }  int yywrap()  {  } | 11)  %{  #include <stdio.h>  int count\_positive = 0;  int count\_negative = 0;  %}  %%  [-+]?[0-9]+(\.[0-9]+)? { // Matches integers and floating-point numbers  double num = atof(yytext);  if (num > 0) {  count\_positive++;  } else if (num < 0) {  count\_negative++;  }  }  %%  int main() {  yylex();  printf("Count of positive numbers: %d\n", count\_positive);  printf("Count of negative numbers: %d\n", count\_negative);  return 0;  }  int yywrap() {  return 1;  }  12) %{  #include <stdio.h>  #include <string.h>  #define MAX\_KEYWORDS 32  char \*keywords[MAX\_KEYWORDS] = {  "auto", "double", "int", "struct",  "break", "else", "long", "switch",  "case", "enum", "register", "typedef",  "char", "extern", "return", "union",  "const", "float", "short", "unsigned",  "continue", "for", "signed", "void",  "default", "goto", "sizeof", "volatile",  "do", "if", "static", "while"  };  int is\_keyword(char \*word) {  for (int i = 0; i < MAX\_KEYWORDS; i++) {  if (strcmp(word, keywords[i]) == 0) {  return 1;  }  }  return 0;  }  %}  %%  [a-zA-Z\_][a-zA-Z0-9\_]\* { // Matches identifiers and keywords  if (is\_keyword(yytext)) {  printf("Keyword: %s\n", yytext);  } else {  printf("Identifier: %s\n", yytext);  }  }  [ \t\n]+ ; // Ignore whitespace  . ; // Ignore any other character  %%  int main() {  yylex();  return 0;  }  int yywrap() {  return 1;  }  13)  %{  #include <stdio.h>  #include <ctype.h>  void add\_to\_list(char \*item, FILE \*file) {  fprintf(file, "%s\n", item);  }  FILE \*numbers\_file;  FILE \*words\_file;  %}  %%  [0-9]+ { // Matches numbers  add\_to\_list(yytext, numbers\_file);  }  [a-zA-Z]+ { // Matches words  add\_to\_list(yytext, words\_file);  }  [ \t\n]+ ; // Ignore whitespace  . ; // Ignore any other character  %%  int main() {  numbers\_file = fopen("numbers.txt", "w");  words\_file = fopen("words.txt", "w");  if (!numbers\_file || !words\_file) {  fprintf(stderr, "Error opening files.\n");  return 1;  }  yylex();  fclose(numbers\_file);  fclose(words\_file);  printf("Numbers have been written to numbers.txt\n");  printf("Words have been written to words.txt\n");  return 0;  }  int yywrap() {  return 1;  }  14)  %{  #include <stdio.h>  #include <stdlib.h>  void print\_vowel\_string(const char \*text);  %}  %option noyywrap  %%  [AEIOUaeiou][a-zA-Z]\* { print\_vowel\_string(yytext); }  . ;  \n ;  %%  int main(void) {  yylex();  return 0;  }  void print\_vowel\_string(const char \*text) {  printf("String starting with a vowel: %s\n", text);  }  15)  %{  #include <stdio.h>  #include <stdlib.h>  void print\_word(const char \*text);  void print\_operator(const char \*text);  %}  %option noyywrap  %%  [a-zA-Z]+ { print\_word(yytext); }  "==" { print\_operator(yytext); }  "!=" { print\_operator(yytext); }  "<=" { print\_operator(yytext); }  ">=" { print\_operator(yytext); }  "<" { print\_operator(yytext); }  ">" { print\_operator(yytext); }  [ \t\n]+ ;  . ;  %%  int main(void) {  yylex();  return 0;  }  void print\_word(const char \*text) {  printf("Word: %s\n", text);  }  void print\_operator(const char \*text) {  printf("Relational Operator: %s\n", text);  }  16)  %{  #include<stdio.h>  #include<string.h>  char replace\_with [] = "Best";  char replace [] ="A";  %}  %%  [a-zA-Z]+ { if(strcmp(yytext, replace)==0)  fprintf(yyout, "%s", replace\_with);  else  fprintf(yyout, "%s", yytext);}  . fprintf(yyout, "%s", yytext);  %%  int yywrap()  {  return 1;  }  int main()  {  extern FILE \*yyin, \*yyout;  yyin=fopen("input.txt", "r");  yyout=fopen("output.txt", "w");  yylex();  }  17)  %{  #include<stdio.h>  int lc=0,sc=0,tc=0,ch=0,wc=0;  %}    %%  [\n] { lc++; ch+=yyleng;}  [ \t] { sc++; ch+=yyleng;}  [^\t] { tc++; ch+=yyleng;}  [^\t\n ]+ { wc++; ch+=yyleng;}  %%    int yywrap()  { return 1; }  int main(){  extern FILE \*yyin, \*yyout;  yyin=fopen("cfile.txt", "r");  yylex();  printf("Number of lines : %d\n",lc);  printf("Number of spaces : %d\n",sc);  printf("Number of tabs, words, charc : %d , %d , %d\n",tc,wc,ch);    return 0;  }  19)  %{  int macro\_count = 0;  int header\_count = 0;  %}  %%  ^#define { macro\_count++; }  ^#include { header\_count++; }  .|\n  %%  int yywrap(){}  int main() {  yylex();  printf("Number of Macros defined: %d\n", macro\_count);  printf("Number of Headers files included: %d\n", header\_count);  }  20) %{  #include <stdio.h>  %}  %%  \<[^>]\*\> fprintf(yyout,"%s\n",yytext);  .|\n;  %%  int yywrap(){}  int main() {  yyin=fopen("sample.html","r");  yyout=fopen("output.txt","w");  yylex();  return 0;  } |
| 21)  %{  int yylineno;  %}  %%  ^(.\*)\n printf("%4d\t%s", ++yylineno, yytext);  %%  int yywrap(void) {  return 1;  }  int main(){  extern FILE \*yyin, \*yyout;  yyin=fopen("line.txt", "r");  yylex();  fclose(yyin);  }  22)  %{  int com=0;  %}  %s COMMENT  %%  "/\*" {BEGIN COMMENT;}  <COMMENT>"\*/" {BEGIN 0; com++;}  <COMMENT>\n {com++;}  <COMMENT>. {;}  \/\/.\* {; com++;}  .|\n {fprintf(yyout,"%s",yytext);}  %%  void main(){  extern FILE \*yyin, \*yyout;  yyin=fopen("commentin.txt", "r");  yyout=fopen("commentout.txt", "w");  yylex();  printf("\n number of comments are = %d\n",com);  }  int yywrap()  {  return 1;  }  23)  digit [0-9]  letter [A-Za-z]  %{  int count\_id,count\_key;  %}  %%  (stdio.h|conio.h) { printf("%s is a standard library\n",yytext); }  (include|void|main|printf|int) { printf("%s is a keyword\n",yytext); count\_key++; }  {letter}({letter}|{digit})\* { printf("%s is a identifier\n", yytext); count\_id++; }  {digit}+ { printf("%s is a number\n", yytext); }  \"(\\.|[^"\\])\*\" { printf("%s is a string literal\n", yytext); }  .|\n { }  %%  int yywrap(void) {  return 1;  }  int main(){  extern FILE \*yyin, \*yyout;  yyin=fopen("to.txt", "r");  yylex();  printf("number of identifiers = %d\n", count\_id);  printf("number of keywords = %d\n", count\_key);  fclose(yyin);  }  25)  %{  int icount=0,factcount=0;  %}  %%  fact factcount++;  i icount++;  (.|\n) ;  %%  int yywrap() {}  int main()  {  printf("Enter the Input:");  yylex();  printf("Count of \"fact\"= %d \nCount of letter 'i' = %d\n",factcount,icount);  return 0;  } | 28)ARTHEMATIC OPERATION  %{  #include <stdio.h>  %}  OPERATOR "+"|"-"|"\*"|"/"  %%  {OPERATOR} {  printf("Valid operator: %s\n", yytext);  }  . {  printf("Invalid character: %s\n", yytext);  }  %%  int yywrap() {  return 1;  }  int main() {  yylex();  return 0;  }  29)WHITE SPACES AND NEW CHARECTERS  %{  #include <stdio.h>  int whitespace\_count = 0;  int newline\_count = 0;  %}  WS [ \t]  NEWLINE \n  %%  {WS} { whitespace\_count++; }  {NEWLINE} { newline\_count++; }  %%  int yywrap() {  return 1;  }  int main() {  yylex();  printf("Number of whitespace characters: %d\n", whitespace\_count);  printf("Number of newline characters: %d\n", newline\_count);  return 0;  }  30)IDENTIFIERS OR NOT  %{  #include <stdio.h>  %}  IDENTIFIER [a-zA-Z\_][a-zA-Z\_0-9]\*  %%  {IDENTIFIER} {  printf("Valid identifier: %s\n", yytext);  }  . {  printf("Invalid identifier: %s\n", yytext);  }  %%  int yywrap() {  return 1;  }  int main() {  yylex();  return 0;  } |

26) C PROGRMING FOR REDUNTANT SPACES

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

int isKeyword(char buffer[]){

char keywords[32][10] = {"main","auto","break","case","char","const","continue","default",

"do","double","else","enum","extern","float","for","goto",

"if","int","long","register","return","short","signed",

"sizeof","static","struct","switch","typedef",

"unsigned","void","printf","while"};

int i, flag = 0;

for(i = 0; i < 32; ++i){

if(strcmp(keywords[i], buffer) == 0){

flag = 1;

break;

}

}

return flag;

}

int main(){

char ch, buffer[15], operators[] = "+-\*/%=";

FILE \*fp;

int i,j=0;

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

if(fp == NULL){

printf("error while opening the file\n");

exit(0);

}

while((ch = fgetc(fp)) != EOF){

for(i = 0; i < 6; ++i){

if(ch == operators[i])

printf("%c is operator\n", ch);

}

if(isalnum(ch)){

buffer[j++] = ch;

}

else if((ch == ' ' || ch == '\n') && (j != 0)){

buffer[j] = '\0';

j = 0;

if(isKeyword(buffer) == 1)

printf("%s is keyword\n", buffer);

else

printf("%s is identifier\n", buffer);

}

}

fclose(fp);

return 0;

}

27) CPROGRAMING FOR CHECKING COMMENT LINES

#include <stdio.h>

#include <string.h>

int main() {

printf("Enter a line of text: ");

fgets(charline, sizeof(line), stdin);

if (strstr(line, "//") != NULL) {

printf("Single-line comment found.\n");

}

else if (strstr(line, "/\*") != NULL) {

printf("Start of multi-line comment found.\n");

}

else if (strstr(line, "\*/") != NULL) {

printf("End of multi-line comment found.\n");

} else {

printf("No comment found.\n");

}

return 0;

}

**Exp31.C programming for Left Recursion**

#include <stdio.h>

#include <string.h>

#define SIZE 100

void eliminateLeftRecursion(char production[SIZE]) {

char non\_terminal = production[0];

char alpha[SIZE], beta[SIZE];

int index = 3, i = 0, j = 0;

if (non\_terminal == production[index]) {

for (i = index + 1; production[i] != '|' && production[i] != '\0'; i++) {

alpha[j++] = production[i];

}

alpha[j] = '\0';

j = 0;

if (production[i] == '|') {

i++;

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

beta[j++] = production[i];

}

}

beta[j] = '\0';

printf("Grammar without left recursion:\n");

printf("%c->%s%c'\n", non\_terminal, beta, non\_terminal);

printf("%c'->%s%c'|E\n", non\_terminal, alpha, non\_terminal);

} else {

printf(" is not left recursive.\n");

}

}

int main() {

int num;

char production[SIZE][SIZE];

printf("Enter Number of Productions: ");

scanf("%d", &num);

printf("Enter the grammar as E->E-A :\n");

for (int i = 0; i < num; i++) {

scanf("%s", production[i]);

}

for (int i = 0; i < num; i++) {

printf("\nGRAMMAR: %s", production[i]);

eliminateLeftRecursion(production[i]);

}

return 0;

}\

**Exp32.C programming for left factoring**

#include<stdio.h>

#include<string.h>

int main()

{

char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];

int i,j=0,k=0,l=0,pos;

printf("Enter Production : S->");

gets(gram);

for(i=0;gram[i]!='|';i++,j++)

part1[j]=gram[i];

part1[j]='\0';

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

part2[i]=gram[j];

part2[i]='\0';

for(i=0;i<strlen(part1)||i<strlen(part2);i++)

{

if(part1[i]==part2[i])

{

modifiedGram[k]=part1[i];

k++;

pos=i+1;

}

}

for(i=pos,j=0;part1[i]!='\0';i++,j++){

newGram[j]=part1[i];

}

newGram[j++]='|';

for(i=pos;part2[i]!='\0';i++,j++){

newGram[j]=part2[i];

}

modifiedGram[k]='X';

modifiedGram[++k]='\0';

newGram[j]='\0';

printf("\n S->%s",modifiedGram);

printf("\n X->%s\n",newGram);

}

**Exp33. C programming forsymbol table**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

char name[100];

int type;

int scope;

int lineNo;

} Symbol;

typedef struct {

Symbol\* symbols;

int size;

int capacity;

} SymbolTable;

SymbolTable\* createSymbolTable(int capacity) {

SymbolTable\* symbolTable = (SymbolTable\*)malloc(sizeof(SymbolTable));

symbolTable->symbols = (Symbol\*)malloc(sizeof(Symbol) \* capacity);

symbolTable->size = 0;

symbolTable->capacity = capacity;

return symbolTable;

}

void insertSymbol(SymbolTable\* symbolTable, char\* name, int type, int scope, int lineNo) {

if (symbolTable->size == symbolTable->capacity) {

printf("Symbol table is full. Cannot insert symbol %s.\n", name);

return;

}

strcpy(symbolTable->symbols[symbolTable->size].name, name);

symbolTable->symbols[symbolTable->size].type = type;

symbolTable->symbols[symbolTable->size].scope = scope;

symbolTable->symbols[symbolTable->size].lineNo = lineNo;

symbolTable->size++;

}

int searchSymbol(SymbolTable\* symbolTable, char\* name) {

for (int i = 0; i < symbolTable->size; i++) {

if (strcmp(symbolTable->symbols[i].name, name) == 0)

return i;

}

return -1;

}

void printSymbolTable(SymbolTable\* symbolTable) {

printf("Symbol Table:\n");

for (int i = 0; i < symbolTable->size; i++) {

printf("Name: %s, Type: %d, Scope: %d, Line No: %d\n",

symbolTable->symbols[i].name,

symbolTable->symbols[i].type,

symbolTable->symbols[i].scope,

symbolTable->symbols[i].lineNo);

}

}

int main() {

SymbolTable\* symbolTable = createSymbolTable(10);

insertSymbol(symbolTable, "x", 1, 1, 5);

insertSymbol(symbolTable, "y", 1, 1, 6);

insertSymbol(symbolTable, "main", 2, 1, 1);

printSymbolTable(symbolTable);

printf("Search result for 'x': %d\n", searchSymbol(symbolTable, "x"));

printf("Search result for 'z': %d\n", searchSymbol(symbolTable, "z"));

return 0;

}

**Exp34.C programming for Grammar or not**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

enum {

NUM,

PLUS,

MUL,

LPAREN,

RPAREN,

EOF\_TOKEN

};

int check\_grammar(char\*);

int main() {

char input[100];

printf("Enter an arithmetic expression: ");

fgets(input, sizeof(input), stdin);

input[strcspn(input, "\n")] = 0;

if (check\_grammar(input)) {

printf("Input string satisfies the grammar\n");

} else {

printf("Input string does not satisfy the grammar\n");

}

return 0;

}

int check\_grammar(char\* input) {

int len = strlen(input);

int i;

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

if (input[i] >= '0' && input[i] <= '9') {

} else if (input[i] == '+') {

} else if (input[i] == '\*') {

} else if (input[i] == '(') {

} else if (input[i] == ')') {

} else {

return 0;

}

}

int paren\_count = 0;

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

if (input[i] == '(') paren\_count++;

else if (input[i] == ')') paren\_count--;

if (paren\_count < 0) return 0;

}

if (paren\_count != 0) return 0;

return 1;

}

**Exp35.C programming for Recursive Decent parsing**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

enum {

NUM,

PLUS,

MINUS,

MUL,

DIV,

LPAREN,

RPAREN,

EOF\_TOKEN

};

char\* token;

int token\_type;

void match(int);

void E();

void T();

void F();

void match(int expected) {

if (token\_type == expected) {

token = strtok(NULL, " \t\n");

if (token == NULL) token\_type = EOF\_TOKEN;

else if (strcmp(token, "+") == 0) token\_type = PLUS;

else if (strcmp(token, "-") == 0) token\_type = MINUS;

else if (strcmp(token, "\*") == 0) token\_type = MUL;

else if (strcmp(token, "/") == 0) token\_type = DIV;

else if (strcmp(token, "(") == 0) token\_type = LPAREN;

else if (strcmp(token, ")") == 0) token\_type = RPAREN;

else token\_type = NUM;

} else {

printf("Error: expected %d, got %d\n", expected, token\_type);

exit(1);

}

}

void E() {

T();

if (token\_type == PLUS || token\_type == MINUS) {

match(token\_type);

T();

}

}

void T() {

F();

if (token\_type == MUL || token\_type == DIV) {

match(token\_type);

F();

}

}

void F() {

if (token\_type == LPAREN) {

match(LPAREN);

E();

match(RPAREN);

} else {

match(NUM);

}

}

int main() {

char input[100];

printf("Enter an arithmetic expression: ");

fgets(input, sizeof(input), stdin);

token = strtok(input, " \t\n");

if (token == NULL) token\_type = EOF\_TOKEN;

else if (strcmp(token, "+") == 0) token\_type = PLUS;

else if (strcmp(token, "-") == 0) token\_type = MINUS;

else if (strcmp(token, "\*") == 0) token\_type = MUL;

else if (strcmp(token, "/") == 0) token\_type = DIV;

else if (strcmp(token, "(") == 0) token\_type = LPAREN;

else if (strcmp(token, ")") == 0) token\_type = RPAREN;

else token\_type = NUM;

E();

if (token\_type != EOF\_TOKEN) {

printf("Error: unexpected token %s\n", token);

exit(1);

}

printf("Parsing successful!\n");

return 0;

}

**Exp36.C programming for operator precedency**

#include <stdio.h>

int main() {

printf("PEMDAS: Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction\n");

printf("Example expression: 3 + 4 \* 2 - 1\n");

int result = 3 + 4 \* 2 - 1;

printf("Step 1: Multiply 4 and 2\n");

printf("4 \* 2 = %d\n", 4 \* 2);

printf("Expression becomes: 3 + 8 - 1\n");

printf("Step 2: Add 3 and 8\n");

printf("3 + 8 = %d\n", 3 + 8);

printf("Expression becomes: 11 - 1\n");

printf("Step 3: Subtract 1 from 11\n");

printf("11 - 1 = %d\n", 11 - 1);

printf("Final result: %d\n", result);

return 0;

}

**Exp37.C programming for Intermediate Code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

char op[3];

char arg1[10];

char arg2[10];

char result[10];

} Quadruple;

Quadruple quads[100];

int quadCount = 0;

void generate(char \*op, char \*arg1, char \*arg2, char \*result) {

strcpy(quads[quadCount].op, op);

strcpy(quads[quadCount].arg1, arg1);

strcpy(quads[quadCount].arg2, arg2);

strcpy(quads[quadCount].result, result);

quadCount++;

}

void printIntermediateCode() {

printf("\nIntermediate Code:\n");

for(int i = 0; i < quadCount; i++) {

printf("%s = %s %s %s\n", quads[i].result, quads[i].arg1, quads[i].op, quads[i].arg2);

}

}

int main() {

char expr[100];

printf("Enter a simple expression (e.g., a = b + c): ");

gets(expr);

char lhs[10], rhs[10], op[3], arg1[10], arg2[10];

sscanf(expr, "%s = %s %s %s", lhs, arg1, op, arg2);

generate(op, arg1, arg2, lhs);

printIntermediateCode();

return 0;

}

**Exp38.C programming for counting no of words , char and lines**

#include <stdio.h>

int main() {

char ch;

int charCount = 0, wordCount = 0, lineCount = 0;

int inWord = 0;

printf("Enter text (EOF to stop):\n");

while ((ch = getchar()) != EOF) {

charCount++;

if (ch == '\n') {

lineCount++;

}

if (ch == ' ' || ch == '\t' || ch == '\n') {

inWord = 0;

} else if (!inWord) {

wordCount++;

inWord = 1;

}

}

printf("Characters: %d\n", charCount);

printf("Words: %d\n", wordCount);

printf("Lines: %d\n", lineCount);

return 0;

}

**Exp39.C programming common Sub expression**

#include <stdio.h>

int original\_function(int a, int b) {

int temp1 = a + b;

int result1 = temp1 \* 2;

int temp2 = a + b;

int result2 = temp2 \* 3;

return result1 + result2;

}

int optimized\_function(int a, int b) {

int temp = a + b;

return temp \* 2 + temp \* 3;

}

int main() {

int a = 80, b = 10;

printf("Original function result: %d\n", original\_function(a, b));

printf("Optimized function result: %d\n", optimized\_function(a, b));

return 0;

}

**Exp40. C programming for Back end**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

char name[50];

int type;

int value;

float fvalue;

} Symbol;

typedef enum {

PUSH,

POP,

ADD,

SUB,

MUL,

DIV,

STORE,

LOAD

} Instruction;

void generateCode(Symbol\* symbols, int symbolCount) {

printf("Generated Code:\n");

for (int i = 0; i < symbolCount; i++) {

Symbol symbol = symbols[i];

if (symbol.type == 0) {

printf("PUSH %d\n", symbol.value);

} else { // float

printf("PUSH %f\n", symbol.fvalue);

}

}

printf("ADD\n");

printf("STORE\n");

printf("LOAD\n");

}

int main() {

Symbol symbols[10];

int symbolCount = 0;

strcpy(symbols[symbolCount].name, "x");

symbols[symbolCount].type = 0; // int

symbols[symbolCount].value = 10;

symbolCount++;

strcpy(symbols[symbolCount].name, "y");

symbols[symbolCount].type = 1; // float

symbols[symbolCount].fvalue = 20.5;

symbolCount++;

generateCode(symbols, symbolCount);

return 0;

}

**C programming For FOLLOW**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

int n,m=0,p,i=0,j=0;

char a[10][10],f[10];

void follow(char c);

void first(char c);

int main()

{

int i,choice;

char c,ch;

printf("Enter the no.of productions:");

scanf("%d",&n);

printf("Enter the productions(epsilon=@):\n");

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

scanf("%s%c",a[i],&ch);

do

{

m=0;

printf("Enter the element whose FOLLOW is to be found:");

scanf("%c",&c);

follow(c);

printf("FOLLOW(%c) = { ",c);

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

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

printf(" }\n");

printf("Do you want to continue(0/1)?");

scanf("%d%c",&choice,&ch);

}

while(choice==1);

}

void follow(char c)

{

if(a[0][0]==c)f[m++]='$';

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

{

for(j=2;j<strlen(a[i]);j++)

{

if(a[i][j]==c)

{

if(a[i][j+1]!='\0')first(a[i][j+1]);

if(a[i][j+1]=='\0'&&c!=a[i][0])

follow(a[i][0]);

}

}

}

}

void first(char c)

{

int k;

if(!(isupper(c)))

f[m++]=c;

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

{

if(a[k][0]==c)

{

if(a[k][2]=='@')

follow(a[i][0]);

else if(islower(a[k][2]))

f[m++]=a[k][2];

else

first(a[k][2]);

}

}

}

**C programming for FIRST**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#define MAX\_RULES 100

#define MAX\_SYMBOLS 100

typedef struct {

char non\_terminal;

char production[MAX\_SYMBOLS];

} Rule;

Rule grammar[MAX\_RULES];

int num\_rules = 0;

char first[MAX\_SYMBOLS][MAX\_SYMBOLS];

bool visited[MAX\_SYMBOLS];

int indexOf(char symbol) {

for (int i = 0; i < num\_rules; i++) {

if (grammar[i].non\_terminal == symbol) {

return i;

}

}

return -1;

}

void addFirst(char symbol, char terminal) {

int idx = indexOf(symbol);

if (idx != -1 && !strchr(first[idx], terminal)) {

strncat(first[idx], &terminal, 1);

}

}

void computeFirst(char symbol) {

if (visited[symbol]) return;

visited[symbol] = true;

int idx = indexOf(symbol);

if (idx == -1) return;

for (int i = 0; i < num\_rules; i++) {

if (grammar[i].non\_terminal == symbol) {

for (int j = 0; j < strlen(grammar[i].production); j++) {

char prod\_symbol = grammar[i].production[j];

if (prod\_symbol >= 'A' && prod\_symbol <= 'Z') { // Non-terminal

computeFirst(prod\_symbol);

int prod\_idx = indexOf(prod\_symbol);

if (prod\_idx != -1) {

for (int k = 0; k < strlen(first[prod\_idx]); k++) {

addFirst(symbol, first[prod\_idx][k]);

}

}

} else { // Terminal

addFirst(symbol, prod\_symbol);

break;

}

}

}

}

}

int main() {

printf("Enter the number of rules: ");

scanf("%d", &num\_rules);

printf("Enter the grammar rules (e.g., S->AB):\n");

for (int i = 0; i < num\_rules; i++) {

char buffer[MAX\_SYMBOLS + 3]; // To accommodate 'A->' and '\0'

scanf("%s", buffer);

grammar[i].non\_terminal = buffer[0];

strcpy(grammar[i].production, buffer + 3);

}

memset(visited, false, sizeof(visited));

for (int i = 0; i < num\_rules; i++) {

computeFirst(grammar[i].non\_terminal);

}

printf("FIRST sets:\n");

for (int i = 0; i < num\_rules; i++) {

printf("FIRST(%c) = { ", grammar[i].non\_terminal);

int idx = indexOf(grammar[i].non\_terminal);

if (idx != -1) {

for (int j = 0; j < strlen(first[idx]); j++) {

printf("%c ", first[idx][j]);

}

}

printf("}\n");

}

return 0;

}