****

**Bangladesh University of Business and Technology (BUBT)**

**Course No – CSE-224**

**Course Title – Numerical Analysis Lab**

**Lab Report**

**Submitted By:**

Name : Md Mehedi Hasan

ID : 21225103334

Intake : 49

Section : 8

**Submitted To:**

Name : Anusha Aziz

Designation : Lecturer

Department : CSE

1. Write a c/c++ program to identify whether the given lexeme is a valid identifier.

|  |
| --- |
| **Code:** |
| #include<bits/stdc++.h>  using namespace std;  void state1(string s, int len, int &Count)  {  for (int i = 1; i < len; i++)  {  if (s[i] == '\_' || (s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <= 'Z') || (s[i] >= '0' && s[i] <= '9'))  {  Count += 1;  }  else  {  cout << "Invalid Identifier\n";  return;  }  }  }  void state0(string s, int len, int &Count)  {  if (s[0] == '\_' || (s[0] >= 'a' && s[0] <= 'z') || (s[0] >= 'A' && s[0] <= 'Z'))  {  Count += 1;  state1(s, len, Count);  }  else  {  cout << "Invalid Identifier\n";  }  }  int main()  {  int t;  cout << "Enter test cases: ";  cin >> t;  while (t--)  {  string s;  cout << "Enter identifier: ";  cin >> s;  int len = s.length();  int Count = 0; // Reset Count for each test case  state0(s, len, Count);  if (Count == len)  {  cout << "Valid Identifier\n";  }  }  return 0;  } |
| Output: |
|  |

2. Write a c/c++ program to identify whether the given lexeme is a valid floating point.

|  |
| --- |
| Code: |
| #include <iostream>  #include <string>  using namespace std;  int flag = 0;  int i = 0; // Start i from 0  void state3(string lex) {  for (i; i < lex.length(); i++) {  if (lex[i] >= '0' && lex[i] <= '9') {  flag = 1;  } else {  flag = 0;  break;  }  }  }  void state2(string lex) {  if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  state3(lex);  } else {  flag = 0;  }  }  void state1(string lex) {  if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  i++;  state1(lex);  } else if (i < lex.length() && lex[i] == '.') {  i++;  state2(lex);  } else {  flag = 0;  }  }  void state0(string lex) {  if (i < lex.length() && lex[i] == '.') {  i++;  state2(lex);  } else if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  state1(lex);  } else {  flag = 0;  }  }  int main() {  string lex;  int t;  cout << "Enter test case:";  cin >> t;  while (t--) {  cout << "Enter a floating point number: ";  cin >> lex;  state0(lex);  if (flag == 1 && (i == lex.length() || (i < lex.length() && lex[i] == '.'))) {  cout << "Valid.\n";  } else {  cout << "Invalid.\n";  }  // Reset i for the next iteration  i = 0;  flag = 0;  }  return 0;  } |
| Output: |
|  |

3. Write a c/c++ program to identify whether the given lexeme is a valid exponential number.

|  |
| --- |
| Code: |
| #include <iostream>  #include <cmath>  using namespace std;  int flag = 0;  int i = 1; // Start from the second character of the input string.  // Function for state 9 (Parsing digits after the decimal point in the exponent part).  void state9(string lex) {  if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  i++;  flag = 1;  state9(lex);  } else {  if (i < lex.length())  flag = 0;  }  }  // Function for state 8 (Parsing digits after the exponent sign).  void state8(string lex) {  if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  i++;  flag = 1;  state9(lex);  } else {  flag = 0;  }  }  // Function for state 7 (Parsing digits before the decimal point in the exponent part).  void state7(string lex) {  if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  i++;  flag = 1;  state7(lex);  } else if (i < lex.length() && lex[i] == '.' && i < lex.length()) {  i++;  state8(lex);  } else {  if (i < lex.length())  flag = 0;  }  }  // Function for state 6 (Parsing digits or a decimal point after the optional sign).  void state6(string lex) {  if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  i++;  state7(lex);  } else if (i < lex.length() && lex[i] == '.') {  i++;  state8(lex);  } else {  flag = 0;  }  }  // Function for state 5 (Parsing the exponent part).  void state5(string lex) {  if (i < lex.length() && (lex[i] == '+' || lex[i] == '-')) {  i++;  state6(lex);  } else if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  i++;  flag = 1;  state7(lex);  } else if (i < lex.length() && lex[i] == '.') {  i++;  state8(lex);  } else {  flag = 0;  }  }  // Function for state 4 (Parsing digits in the exponent part after the optional sign).  void state4(string lex) {  if (i < lex.length() && lex[i] == '^') {  i++;  state5(lex);  } else {  flag = 0;  }  }  // Function for state 3 (Parsing digits and transition for the exponent part).  void state3(string lex) {  if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  i++;  state3(lex);  } else if (i < lex.length() && lex[i] == 'e' || lex[i] == 'E') {  i++;  state4(lex);  } else {  flag = 0;  }  }  // Function for state 2 (Parsing digits after the decimal point).  void state2(string lex) {  if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  i++;  state3(lex);  } else {  flag = 0;  }  }  // Function for state 1 (Parsing digits before the decimal point or the exponent part).  void state1(string lex) {  if (i < lex.length() && lex[i] >= '0' && lex[i] <= '9') {  i++;  state1(lex);  } else if (i < lex.length() && lex[i] == '.') {  i++;  state2(lex);  } else if (i < lex.length() && (lex[i] == 'e' || lex[i] == 'E')) {  i++;  state4(lex);  } else {  flag = 0;  }  }  // Function for state 0 (Initial state, starting from the first character of the input string).  void state0(string lex) {  if (i < lex.length() && lex[0] == '.') {  state2(lex);  } else if (i < lex.length() && lex[0] >= '0' && lex[0] <= '9') {  state1(lex);  } else {  flag = 0;  }  }  int main() {  int t;  string lex;  cout << "Enter test cases:";  cin >> t;  while (t--) {  i = 1; // Reset i for each test case  flag = 0; // Reset flag for each test case  cout << "Enter lexeme:";  cin >> lex;  state0(lex);  if (flag == 1) {  cout << "Valid Exponential.\n";  } else {  cout << "Invalid Exponential.\n";  }  }  return 0;  } |
| Output: |
|  |

4. Write a c/c++ program to identify all the tokens from a given block of codes

|  |
| --- |
| Code: |
| #include<bits/stdc++.h>  #include<iostream>  #include<fstream> //File Stream  #include<string> // String  using namespace std;  //Token names  int nofTokens,nofKeyword,nofIdentifier,nofOperatior,nofSymbols;  //string  string sTokens,sKeyword,sIdentifier,sOperatior,sSymbols;  bool isSeparator(char ch)  {  string sep = " ,;(){}[]";  for(int i=0; i<sep.length(); i++)  {  if(ch==sep[i])  {  if(sep[i]!=' ')  {  nofSymbols++;  sSymbols+=ch;  sSymbols+=" ";  sTokens+=ch;  sTokens+=" ";  }  return true;  }  }  return false;  }  bool isOeparator(char ch)  {  string opa = "+-\*/<>=!&||";  for(int i=0; i<opa.length(); i++)  {  if(ch==opa[i])  {  return true;  }  }  return false;  }  bool isKeyword(string s)  {  string keyword[]= {"auto", "double", "int", "struct", "break", "else",  "long", "switch", "case", "enum", "register", "typedef", "char",  "extern", "return", "union", "continue", "for", "signed", "void",  "do", "if", "static", "while", "default", "goto", "sizeof", "volatile",  "const", "float", "short", "unsigned"  };  for(int i=0; i<(sizeof(keyword)/sizeof(string)); i++)  {  if(s==keyword[i])  {  return true;  }  }  return false;  }  bool isIdentifier(string s)  {  bool valid = true;  if(isKeyword(s))  {  valid = false;  }  if(valid)  {  if(s[0]>='0' && s[0]<='9')  {  valid = false;  }  }  if(valid)  {  for(int i=0; i<s.length(); i++)  {  if(((s[i]>='a' && s[i]<='z')||(s[i]>='A' && s[i]<='Z'))||(s[i]>='0' && s[i]<='9')||(s[i]=='\_'))  {  continue;  }  else  {  valid=false;  break;  }  }  }  return valid;  }  bool isNumber(string s)  {  bool valid = true;  for(int i=0; i<s.length(); i++)  {  if((s[i]>='0' && s[i]<='9')||(s[i]=='.'))  {  continue;  }  else  {  valid=false;  break;  }  }  return valid;  }  //Main work here  //parse checking  //check token  void parse(string s)  {  vector <string> tokens; //all tokens pushed here  string token="";  for(int i=0; i<s.length(); i++)  {  if(isSeparator(s[i]))  {  if(token!="")  {  tokens.push\_back(token);  }  token="";  }  else if(isOeparator(s[i]))  {  if(token!="")  {  tokens.push\_back(token);  token="";  }  token.push\_back(s[i]);  tokens.push\_back(token);  token="";  }  else  {  token.push\_back(s[i]);  }  }  if(token!="")  {  tokens.push\_back(token);  token="";  }  //Result  for(int i=0; i<tokens.size(); i++)  {  sTokens+=tokens[i];  sTokens+=" ";// for making a tab or space between token  if(isKeyword(tokens[i]))  {  nofKeyword++;  sKeyword+=tokens[i];  sKeyword+=" ";  }  if(isIdentifier(tokens[i]))  {  nofIdentifier++;  sIdentifier+=tokens[i];  sIdentifier+=" ";  }  if(isNumber(tokens[i]))  {  }  if(isOeparator(tokens[i][0]))  {  nofOperatior++;  sOperatior+=tokens[i];  sOperatior+=" ";  }  else  {  //error  }  }  }  int main()  {  ifstream file;  string line,filename;  //getline(cin,filename);  file.open("program.cpp");  while(file)  {  getline(file,line);  parse(line);  }  nofTokens=nofKeyword+nofIdentifier+nofOperatior+nofSymbols;  cout<<"No. of Token: "<<nofTokens<<" : "<<sTokens<<endl;  cout<<"No. of Keywords: "<<nofKeyword<<" : "<<sKeyword<<endl;  cout<<"No. of Identifiers: "<<nofIdentifier<<" : "<<sIdentifier<<endl;  cout<<"No. of Operators: "<<nofOperatior<<" : "<<sOperatior<<endl;  cout<<"No. of Symbols: "<<nofSymbols<<" : "<<sSymbols<<endl;  file.close();  return 0;  }  **File:**  Program.cpp  int add(int a, int b)  {  return a+b;  } |
| Output: |
|  |

5. Write a c/c++ program to identify strings under abb, a\*b+, a.

|  |
| --- |
| Code: |
| #include<iostream>  #include<string>  using namespace std;  void processString(const string& s) {  char c;  int state = 0, i = 0;  while (i < s.length()) {  switch (state) {  case 0:  c = s[i++];  if (c == 'a') {  state = 1;  } else if (c == 'b') {  state = 2;  } else {  state = 6;  }  break;  case 1:  c = s[i++];  if (c == 'a') {  state = 3;  } else if (c == 'b') {  state = 4;  } else {  state = 6;  }  break;  case 2:  c = s[i++];  if (c == 'a') {  state = 6;  } else if (c == 'b') {  state = 2;  } else {  state = 6;  }  break;  case 3:  c = s[i++];  if (c == 'a') {  state = 3;  } else if (c == 'b') {  state = 2;  } else {  state = 6;  }  break;  case 4:  c = s[i++];  if (c == 'a') {  state = 6;  } else if (c == 'b') {  state = 5;  } else {  state = 6;  }  break;  case 5:  c = s[i++];  if (c == 'a') {  state = 6;  } else if (c == 'b') {  state = 2;  } else {  state = 6;  }  break;  case 6:  cout << "\n" << s << " is not recognized.";  return;  }  }  if (state == 1) {  cout << "\n" << s << " is accepted under rule 'a'"<<endl;  } else if ((state == 2) || (state == 4)) {  cout << "\n" << s << " is accepted under rule 'a\*b+'"<<endl;  } else if (state == 5) {  cout << "\n" << s << " is accepted under rule 'abb'"<<endl;  }  }  int main() {  int t;  string input;  cout<<"Enter test case:";  cin>>t;  cin.ignore();  while(t--)  {  cout << "Enter a string: ";  getline(cin, input);  processString(input);  }  return 0;  } |
| Output: |
|  |

6. Write flex program to identify all tokens.

|  |
| --- |
| Implementation |
|  |
| Code: |
| %{  #include <stdio.h>  %}  %%  "\/\/".\* { printf("Single line comment: %s\n", yytext); }  "\/\\*".\*"\\*\/" { printf("Multiple line Comment: %s\n", yytext); }  "int"|"float"|"else"|"while"|"for"|"if"|"return" { printf("Keyword: %s\n", yytext); }  [a-zA-Z][a-zA-Z0-9\_]\* { printf("Identifier: %s\n", yytext); }  "+"|"-"|"\*"|"/" { printf("Operator: %s\n",yytext); }  [0-9]\*[.][0-9]+[eE][^][0-9][0-9]\* {printf("Exponential Notation: %s\n", yytext);}  [0-9]+[eE][^][0-9][0-9]\* {printf("Exponential Notation: %s\n", yytext);}  [0-9]+ { printf("Integer: %s\n", yytext); }  [0-9]\*[.][0-9]+ { printf("Floating-Point Number: %s\n", yytext); }  \n  .  %%  int yywrap() {      return 1;  }  int main() {      yylex();      return 0;  } |
| Output: |
|  |

7. Write a c/c++ program to construct LL(1) parsing.

8. Write a c/c++ program to construct Recursive Descent Parsing.

|  |  |  |
| --- | --- | --- |
| Code: | | |
| #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);  }  } |
| Input | Output | |
| a\*2-3 | E->TE'  T->FT'  F->id  T'->\*FT'  F->digit  T'->null  E'->-TE'  T->FT'  F->digit  T'->null  E'->null  Accepted | |