1. **Detect whether the given mathematical expression is properly parenthesized or not:**

|  |
| --- |
| #include<stdio.h>  #include<string.h>  //incomplete program  int main()  {  int i,j,l,t=0;  char input[20];  printf("Enter the input\n");  scanf("%s",&input);  l=strlen(input);  for(i=0;i<l;i++)  {  if(input[i]=='(')  t++;  else if(input[i]==')')  t--;  else if(input[i]=='\*'||input[i]=='/'||input[i]=='+'||input[i]=='-')  {  if(input[i-1]>96 && input[i-1]<123 && input[i+1]>96 && input[i+1]<123)  {  continue;  }  else  break;  }  }  if(i==l && t==0)  printf("Valid");  else  printf("Invalid ");  } |

1. **Detect whether the given mathematical expression is valid or not:**

|  |
| --- |
| // C++ program to evaluate a given expression  #include <iostream>  using namespace std;  // A utility function to check if a given character is operand  bool isOperand(char c) { return (c >= '0' && c <= '9'); }  // utility function to find value of and operand  int value(char c) { return (c - '0'); }  // This function evaluates simple expressions. It returns -1 if the  // given expression is invalid.  int evaluate(char \*exp)  {  // Base Case: Given expression is empty  if (\*exp == '\0') return -1;  // The first character must be an operand, find its value  int res = value(exp[0]);  // Traverse the remaining characters in pairs  for (int i = 1; exp[i]; i += 2)  {  // The next character must be an operator, and  // next to next an operand  char opr = exp[i], opd = exp[i+1];  // If next to next character is not an operand  if (!isOperand(opd)) return -1;  // Update result according to the operator  if (opr == '+') res += value(opd);  else if (opr == '-') res -= value(opd);  else if (opr == '\*') res \*= value(opd);  else if (opr == '/') res /= value(opd);  // If not a valid operator  else return -1;  }  return res;  }  // Driver program to test above function  int main()  {  char expr1[] = "1+2\*5+3";  int res = evaluate(expr1);  (res == -1)? cout << expr1 << " is " << "Invalid\n":  cout << "Value of " << expr1 << " is " << res << endl;  char expr2[] = "1+2\*3";  res = evaluate(expr2);  (res == -1)? cout << expr2 << " is " << "Invalid\n":  cout << "Value of " << expr2 << " is " << res << endl;  char expr3[] = "4-2+6\*3";  res = evaluate(expr3);  (res == -1)? cout << expr3 << " is " << "Invalid\n":  cout << "Value of " << expr3 << " is " << res << endl;  char expr4[] = "1++2";  res = evaluate(expr4);  (res == -1)? cout << expr4 << " is " << "Invalid\n":  cout << "Value of " << expr4 << " is " << res << endl;  return 0;  } |

1. **Check English grammar in a sentence:**

|  |
| --- |
| // C program to validate a given sentence for a set of rules  #include<stdio.h>  #include<string.h>  #include<stdbool.h>  // Method to check a given sentence for given rules  bool checkSentence(char str[])  {  // Calculate the length of the string.  int len = strlen(str);  // Check that the first character lies in [A-Z].  // Otherwise return false.  if (str[0] < 'A' || str[0] > 'Z')  return false;  //If the last character is not a full stop(.) no  //need to check further.  if (str[len - 1] != '.')  return false;  // Maintain 2 states. Previous and current state based  // on which vertex state you are. Initialise both with  // 0 = start state.  int prev\_state = 0, curr\_state = 0;  //Keep the index to the next character in the string.  int index = 1;  //Loop to go over the string.  while (str[index])  {  // Set states according to the input characters in the  // string and the rule defined in the description.  // If current character is [A-Z]. Set current state as 0.  if (str[index] >= 'A' && str[index] <= 'Z')  curr\_state = 0;  // If current character is a space. Set current state as 1.  else if (str[index] == ' ')  curr\_state = 1;  // If current character is [a-z]. Set current state as 2.  else if (str[index] >= 'a' && str[index] <= 'z')  curr\_state = 2;  // If current state is a dot(.). Set current state as 3.  else if (str[index] == '.')  curr\_state = 3;  // Validates all current state with previous state for the  // rules in the description of the problem.  if (prev\_state == curr\_state && curr\_state != 2)  return false;  if (prev\_state == 2 && curr\_state == 0)  return false;  // If we have reached last state and previous state is not 1,  // then check next character. If next character is '\0', then  // return true, else false  if (curr\_state == 3 && prev\_state != 1)  return (str[index + 1] == '\0');  index++;  // Set previous state as current state before going over  // to the next character.  prev\_state = curr\_state;  }  return false;  }  // Driver program  int main()  {  char \*str[] = { "I love to code.",  "I am single.", "My name is NAzia.",  "I lovE cinema.", "hi. is a quiz site.",  "I love cse.",  " You are my friend.", "I love to code" };  int str\_size = sizeof(str) / sizeof(str[0]);  int i = 0;  for (i = 0; i < str\_size; i++)  checkSentence(str[i])? printf("\"%s\" is correct \n", str[i]):  printf("\"%s\" is incorrect \n", str[i]);  return 0;  } |

**Input exp: 1+2\*5+3**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **+** | **2** | **\*** | **5** | **+** | **3** | **‘\0’** |
| **Exp[0]** | **Exp[1]** | **Exp[2]** | **Exp[3]** | **Exp[4]** | **Exp[5]** | **Exp[6]** | **Exp[7]** |

**int evaluate(char \*exp)**

**{**

**if (\*exp == '\0') return -1;**

**int res = value(exp[0]); // res=1**

**for (int i = 1; exp[i]; i += 2)**

**{**

**char opr = exp[i], opd = exp[i+1]; // opr = exp[5] == ‘+’, opd = exp[6] == ‘3’**

**if (!isOperand(opd)) return -1;**

**if (opr == '+') res += value(opd); // res = 15 + 3 = 18**

**else if (opr == '-') res -= value(opd);**

**else if (opr == '\*') res \*= value(opd); // ~~res = 3\*5=15~~**

**else if (opr == '/') res /= value(opd);**

**else return -1;**

**}**

**return res;**

**}**