**Постановка задачи**

Напишите функцию, выполняющую проверку соответствия строки, заданной параметром,  
правилу, указанному в Вашем варианте.  
Функция должна возвращать значение TRUE, если строка соответствует правилу, и FALSE, если  
строка НЕ соответствует правилу.  
Замечания:  
• Реализация каждого определения должно быть оформлено в виде отдельной функции.  
• Для повторяющихся действий использовать только рекурсию.  
• Тестовые наборы данных можно определить в функции main.

Понятие выражение определено следующим образом:  
<выражение> ::= <терм> | <терм> +<выражение> | <терм> – <выражение>  
<терм> ::= <множитель> | (<множитель> \* <терм> ) |( <множитель> / <терм>)  
<множитель> ::= <целое без знака> | <идентификатор> | (<выражение>)  
<целое без знака> ::= <цифра> | <цифра> <целое без знака>  
<идентификатор> ::= <буква>  
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  
<буква> ::= a | b | c | d | e | f | x | y | z

**Задача 1. Последовательность чисел**

1. Напишите функцию, выполняющую проверку соответствия строки, заданной параметром,  
   правилу, указанному в Вашем варианте.  
   Функция должна возвращать значение TRUE, если строка соответствует правилу, и FALSE, если  
   строка НЕ соответствует правилу.  
   Пример:

Дана строка (2 + 3 \* (2 \* 2)) . Тогда в консоль выведется: TRUE.

1. Детальные требования
   1. Вводимое выражение должно быть корректным.
      1. Если вводимое выражение не соответствует понятию “выражение”, то вывести сообщение «FALSE»
      2. Если вводимое выражение соответствует понятию “выражение”, то вывести сообщение «TRUE»
2. Тест-план

|  |  |  |  |
| --- | --- | --- | --- |
| Требование | Детальные требования | Данные | Ожидаемый результат |
| 3.1. Вводимые данные должны быть корректны. | | | |
| * 1. Вводимое выражение должно быть корректным. | 2.1.1 Если вводимое выражение не соответствует понятию “выражение”, то вывести сообщение «FALSE» | (2 + 3 \* (2 \* 2)) | Сообщение: «TRUE » |
|  | * + 1. Если вводимое выражение соответствует понятию “выражение”, то вывести сообщение «TRUE» | ( + 3 \* (2 \* 2) | Сообщение: «FALSE» |

**Программа:**

#include <iostream>

#include <string>

bool IsExpression(std::string line);

bool IsTerm(std::string line);

bool IsFactor(std::string line);

bool IsUnsignedInteger(std::string line, int count, bool ist);

bool IsSymbol(std::string line);

bool IsNumber(char a);

bool IsLetter(std::string line);

bool HasBrackets(std::string line);

bool IsMultiOrDiv(std::string line);

bool IsPlusOrMinus(std::string line);

std::string RemoveBrackets(std::string line);

bool ChekOpClBrackets(std::string line);

int main()

{

std::string expression;

std::cout << "Enter expression: ";

std::getline(std::cin, expression);

if (IsExpression(expression))

std::cout << "TRUE";

else

std::cout << "FALSE";

return 0;

}

bool IsExpression(std::string line)

{

int ist = 0, countS1 = 0, countS2 = 0, i = 0;

std::string line1 = "", factor = "", factorTemp = "";

if (IsPlusOrMinus(line))

{

if (line[0] == '(')

{

if (!ChekOpClBrackets(line))

return 0;

for (i = 0; i < line.size(); i++)

{

factor = factor + line[i];

if (line[i] == '(')

countS1++;

if (line[i] == ')')

countS2++;

if (countS1 == countS2)

break;

}

if (countS1 != countS2)

return 0;

i = i + 4;

int k;

for (k = i; k < line.size(); k++)

{

factorTemp = factorTemp + line[k];

}

}

else

{

while (line[i] != ' ')

{

factor = factor + line[i];

i++;

}

i = i + 3;

int k;

for (k = i; k < line.size(); k++)

{

factorTemp = factorTemp + line[k];

}

}

if (IsTerm(factor) and (IsExpression(factorTemp)))

{

return 1;

}

else

{

return 0;

}

}

else

{

return IsTerm(line);

}

}

bool IsTerm(std::string line)

{

int ist = 0;

std::string line1 = "", factor = "", factorTemp = "";

if (HasBrackets(line) and IsMultiOrDiv(line))

{

line1 = RemoveBrackets(line);

int i = 0;

while (line1[i] != ' ')

{

factor = factor + line1[i];

i++;

}

i = i + 3;

int k;

for (k = i; k < line1.size(); k++)

{

factorTemp = factorTemp + line1[k];

}

if (IsFactor(factor) and (IsTerm(factorTemp)))

{

return 1;

}

else

{

return 0;

}

}

else

{

return IsFactor(line);

}

}

bool IsFactor(std::string line)

{

if (IsUnsignedInteger(line, line.size(), 1) or IsSymbol(line) or (HasBrackets(line) and IsExpression(RemoveBrackets(line))))

{

return 1;

}

else

{

return 0;

}

}

bool IsUnsignedInteger(std::string line, int count, bool ist)

{

int i = 0, size = 0;

if ((ist == 1) and (count != 1)) //

{ //

if (line[0] == '0')

{

std::cout << "Number cannot start with zero!\n";//

return 0;

} //

}

if (count == 1)

{

return IsNumber(line[0]);

}

size = line.size();

if (IsNumber(line[0]))

{

line.erase(line.begin());

return IsUnsignedInteger(line, size - 1, 0);

}

else

{

return 0;

}

}

bool IsSymbol(std::string a)

{

return IsLetter(a);

}

bool IsNumber(char a)

{

if ((a >= '0') and (a <= '9')) return 1;

else return 0;

}

bool IsLetter(std::string a)

{

if ((a == "a") or (a == "b") or (a == "c") or (a == "d") or (a == "e") or (a == "f") or (a == "x") or (a == "y") or (a == "z")) return 1;

else return 0;

}

bool HasBrackets(std::string line)

{

if (line[0] == '(' && line[line.size() - 1] == ')')

return 1;

}

bool IsMultiOrDiv(std::string line)

{

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

{

if ((i > 0) and (i < line.size() - 1) and (line[i - 1] == ' ') and (line[i + 1] == ' ') and ((line[i] == '\*') or (line[i] == '/')))

return 1;

}

return 0;

}

bool IsPlusOrMinus(std::string line)

{

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

{

if ((i > 0) and (i < line.size() - 1) and (line[i - 1] == ' ') and (line[i + 1] == ' ') and ((line[i] == '-') or (line[i] == '+')))

return 1;

}

return 0;

}

std::string RemoveBrackets(std::string line)

{

std::string line1 = "";

for (int i = 1; i < line.size() - 1; i++)

{

line1 = line1 + line[i];

}

return line1;

}

bool ChekOpClBrackets(std::string line)

{

int k = 0;

for (int i = 0; line[i]; ++i) {

if (line[i] == '(')

k++;

if (line[i] == ')')

k--;

}

if (k == 0)

return 1;

else

return 0;

}