МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
ТЕМА: РЕКУРСИЯ

Студент гр. 9382 _____ Кодуков А.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы:

Ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций.

Задание:

10. Построить синтаксический анализатор для определяемого далее понятия *константное_выражение*.

константное_выражение::=ряд_цифр|

константное_выражение знак_операции константное_выражение

знак_операции::=+ | -| *

ряд_цифр::=цифра | цифра ряд_цифр

Выполнение работы:

Была рекурсивно реализована функция анализа числового ряда (ParseRow). Функция принимает строку, в которой содержится набор символов для проверки на соответствие определению числового ряда. Ответ возвращается в виде логического значения (bool).

Алгоритм:

- обработка тривиального случая пустой ввод
- ввод содержит 1 символ: проверка является ли символ цифрой и возвращение ответа как результата функции.
- ввод содержит несколько символов: вызов этой же функции от первого символа и строки без него и возвращение их конъюнкции как результата функции.

Затем, используя функцию определения числового ряда, была реализована функция анализа константного выражения (ParseExpr). Аргументы и возвращаемое значение сконструированы аналогично ParseRow.

Алгоритм:

- обработка тривиального случая пустой ввод

- ввод не содержит знак_операции: проверка является ли строка числовым рядом и возвращение ответа как результата функции.
- ввод содержит знак_операции: вызов этой же функции от строки справа и слева от знака операции и возвращение их конъюнкции как результата функции.

Также для обеих функций предусмотрен вывод промежуточных проверок.

Тестирование:

№	Входные данные	Результат
1	123+45-91*37	**Constant expression!**
2	123+45q-s91*37	**Not constant expression!**
3	1	**Constant expression!**
4	+	**Not constant expression!**
5	12++13	**Not constant expression!**
6	(пустой ввод)	**Not constant expression!**

Вывод:

В результате выполнения работы были изучены принципы рекурсивного программирования и понятие синтаксического анализатора. Была реализован синтаксический анализатор, работающий с рекурсивно определенным выражением.

Были выявлены как преимущества рекурсивного подхода: простота реализации, наглядность, так и недостатки: хранение всех данных прошлых вызовов в стеке приводит к увеличению времени работы и потребления памяти относительно итеративных алгоритмов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
/* Kodukov A. 9382 v.10
 *
 * Построить синтаксический анализатор для определяемого далее понятия
константное_выражение.
 * константное_выражение ::= ряд_цифр|константное_выражение знак_операции
константное_выражение
 * знак_операции ::= + | - | *
 * ряд_цифр ::= цифра | цифра ряд_цифр
 */

#include <iostream>
#include <fstream>
#include <string>
#include <conio.h>

// Flag for detalyzed row of numbers parsing output
//#define ROW_DEBUG

// Macro functions
#define IS_SIGN(c) (c == '+' || c == '-' || c == '*')
#define IS_DIGIT(c) (c <= '9' && c >= '0')

/* Row of numbers parse function
 * ARGUMENTS:
 *   - data to analyze:
 *       const std::string &input;
 *   - recursion level (for detalyzed output):
 *       int lvl;
 * RETURNS:
 *   (bool) data is row of numbers (Yes/No).
 */
bool ParseRow(const std::string &input, int lvl)
{
  if (input.empty())
    return false;
  // Quitting from recursion if only one symbol was passed
  if (input.size() == 1)
  {
#ifdef ROW_DEBUG
    for (int i = 0; i < lvl; i++)
      std::cout << "  ";
    std::cout << input[0] << '\n';
#endif
    return IS_DIGIT(input[0]);
  }
  // Parse first data symbol
  if (!ParseRow(std::string({input[0]}), lvl))
    return false;
  // Parse remaining data
  bool res =  ParseRow(input.substr(1), lvl + 1);
#ifdef ROW_DEBUG
  if (res)
  {
    for (int i = 0; i < lvl; i++)
      std::cout << "  ";
    std::cout << input << '\n';
  }
#endif
  return res;
}
```

```
/* Row of numbers parse function
 * ARGUMENTS:
 *   - data to analyze:
 *       const std::string &input;
 *   - recursion level (for detalyzed output):
 *       int lvl;
 * RETURNS:
 *   (bool) data is constexpr (Yes/No).
 */
bool ParseExpr(const std::string &input, int lvl)
{
  int ind = 0;

  if (input.empty())
    return false;
  while (size_t(ind) < input.size() && !IS_SIGN(input[ind]))
    ind++;
  if (ind == input.size())
  {
    for (int i = 0; i < lvl; i++)
      std::cout << "  ";
    std::cout << input << "\n";
#ifdef ROW_DEBUG
    std::cout << "----------\n" ;
#endif
    bool res = ParseRow(input, 0);
#ifdef ROW_DEBUG
    std::cout << "----------\n";
#endif
    return res;
  }
  if (!ParseExpr(input.substr(0, ind), lvl))
    return false;
  std::cout << input[ind] << '\n';
  bool res =  ParseExpr(input.substr(ind + 1), lvl + 1);
  if (res)
  {
    for (int i = 0; i < lvl; i++)
      std::cout << "  ";
    std::cout << input << '\n';
  }
  return res;
}

/* Main function */
int main()
{
  std::ifstream fs;
  while (1)
  {
    system("cls");
    fs.open("input.txt");
    if (fs.is_open())
    {
      std::string input;
      std::getline(fs, input);
      fs.close();
      // Erasing spaces
      for (int i = 0; size_t(i) < input.size(); i++)
        if (input[i] == ' ')
          input.erase(input.cbegin() + i);
      std::cout << "Input:" << input << '\n';
      std::cout << (ParseExpr(input, 0) ? "\nConstant expression!\n" : "\nNot constant
expression!\n");
    }
```

```
    else
      std::cout << "\nImpossible to open file\n";
    std::cout << "\nPress any key...";
    _getch();
  }
  return 0;
}
```