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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
Тема: Рекурсивные функции

Студент гр. 9382 _____ Субботин М. О.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++.

Задание

20. Построить синтаксический анализатор понятия *список_параметров.*
список_параметров::= параметр | *параметр , список_параметров*
параметр::= имя=цифра цифра | *имя=*(*список_параметров*)
имя::= буква буква буква

Основные теоретические положения.

Требуется написать синтаксический анализатор понятия список_параметров.

Буквой будет называться символ из a-z или A-Z (т.е. как строчные, так и заглавные латинские символы). Цифрой же символ из 0-9.

В этих шаблонах не будет использоваться символ пробел. Можно подумать, что к примеру, между параметр и список_параметров стоит не просто “,”, а “ , “. При желании можно решить и такую задачу, но пусть будет так.

Описание алгоритма:

Понятие в синтаксическом анализаторе представляет из себя какое-то условие. Если это условие выполняется для рассматриваемой последовательности, это значит, что последовательность подходит под понятие. В данной задаче у нас есть несколько понятий. Некоторые из этих понятий являются частью других понятий, вследствие чего где-то здесь должна быть рекурсия.

Определение, подходит ли заданная последовательность под понятие список_параметров, будет проходит следующим образом. Сначала будет определяться подходит ли строка под правое условие понятия список_параметров, т.к. левое условие является частью правого. Т.е. если бы

первым проверялось левое условие, при том, что верно правое условие, тогда верно и левое условие. В этом случае алгоритм пойдет по неверному пути. Чтобы проверить правильность условия, в случае левого условия, надо проверить правильность понятия параметр, а правого параметр,список_параметров. Чтобы проверить параметр, нужно в любом случае проверять понятие имя, имя уже проверяется однозначно. В случае понятий список_параметров и параметр их правое условие вызывает как бы рекурсию, т.е. снова надо проверять на понятие список_параметров.

Идея алгоритма заключается в том, чтобы проверять условия понятий, которые иногда включают в себя другие понятия. Чтобы проверить одно, надо проверить другое. И с помощью такого ветвления от одного понятия к другому и рекурсии и решается данная задача.

Описание основных функций:

Сигнатура: bool listOfParameters(string sequence, int& currentIdx, int& shift)

Функция определяет подходит ли подстрока под шаблон "список_параметров"

string sequence - анализируемая последовательность

int& currentIdx - индекс, с которого начнется рассмотрение шаблона

int& shift - сдвиг для корректного вывода информации на консоль

Возвращаемое значение bool - результат подошел шаблон или нет

Сигнатура: bool parameter(string sequence, int& currentIdx, int& shift)

Функция определяет подходит ли подстрока под шаблон "параметр"

string sequence - анализируемая последовательность

int& currentIdx - индекс, с которого начнется рассмотрение шаблона

int& shift - сдвиг для корректного вывода информации на консоль

Возвращаемое значение bool - результат подошел шаблон или нет

Сигнатура: bool name(string sequence, int& currentIdx, int& shift)

Функция определяет подходит ли подстрока под шаблон "имя"

string sequence - анализируемая последовательность

int& currentIdx - индекс, с которого начнется рассмотрение шаблона

int& shift - сдвиг для корректного вывода информации на консоль

Возвращаемое значение bool - результат подошел шаблон или нет

Эти функции выполняют роль паттернов. Они проверяют одно из своих условий – если оно подошло, то все хорошо. Если же оно не подошло, мы выводим информацию об этом и проверяем другое условие. При этом мы должны проверять последовательность с того же места, которое проверяли в первый раз, а после первого раза наш индекс сдвинулся, поэтому его надо вернуть.

Тестирование

№	Входные данные	Выходные данные
1	Dad=66	true
2	age=(age=64)	true
3	Age=(acb=(bca=56))	true
4	tea=12,bee=15,cow=(tea=12,bee=12)	true
5	foo=(foo=(mee=(typ=(ooo=(age=34,cro=45,ttt=12)))))	true
6	pie=12,coo=14	true
7	dot=(dot=12,dot=1,dot=12)	false
8	dot	false
9	dot=(dot=12,dot=155,dot=12)	false
10	dot=(dot=12,dot=12,dot=12	false
11	Top=11,bot=242	false

Обработка результатов тестирования.

Программа выдает ожидаемые результаты для всех тестов. Особое внимание следует уделить 10 тесту. Из-за этого теста пришлось добавлять в программу еще одно условие. Основной алгоритм находил правильной последовательность Top=11,bot=24 но забывал про то, что существуют еще символы за 24.

Теперь имея условие о том, что длина последовательности должна равняться длине получившегося шаблона, давайте посмотрим на тест 6. Если расставить условия так, как это показано в задаче, то можно получить результат false, когда на самом деле он true. Дело в том, что первое условие списка_параметров является составной частью его второго условия. Прочитав pie=12 мы увидим, что эта последовательность подходит, потом заметив что длины не сходятся выведем ответ false. Чтобы программа работала корректно, следует проверять сначала второе условие, а затем первое.

Выводы.

Были изучены основные понятия и приёмы рекурсивного программирования, получены навыки программирования рекурсивных процедур и функций на языке программирования C++. Произведено тестирование программы и выявлены некоторые случаи, которые следует учесть.

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

characterCheckers.h

```
#include <string>
#ifndef UNTITLED_CHARACTERCHECKERS_H
#define UNTITLED_CHARACTERCHECKERS_H

bool checkChar(std::string str, int &currentIdx, char ch);

bool isDigit(std::string str, int &currentIdx);

#endif //UNTITLED_CHARACTERCHECKERS_H
```

characterCheckers.cpp

```
#include "characterCheckers.h"

bool isDigit(std::string sequence, int &currentIdx)
{
    if (currentIdx >= size(sequence))
    {
        return false;
    }
    if (isdigit(sequence[currentIdx]))
    {
        currentIdx++;
        return true;
    }
    else
    {
        return false;
    }
}

bool isLetter(std::string sequence, int &currentIdx)
{
    if (currentIdx >= size(sequence))
    {
        return false;
    }
    if (isalpha(sequence[currentIdx]))
    {
        currentIdx++;
        return true;
    }
    else
    {
        return false;
    }
}

bool checkChar(std::string str, int& currentIdx, char ch)
{
    if(size(str) <= currentIdx )
    {
        return false;
    }
    if(str[currentIdx] != ch)
    {
        return false;
    }
```

```
        currentIdx++;
        return true;
    }

    shiftingOutput.h
    #include <string>
    #ifndef UNTITLED_LEFTALIGNMENT_H
    #define UNTITLED_LEFTALIGNMENT_H

    void shifting(int numberOfSpaces);

    void printPattern(std::string pattern, int currentIdx, int &shift);

    void printPatternResult(std::string pattern, std::string sequence, bool
check, int& shift);

    #endif //UNTITLED_LEFTALIGNMENT_H

    shiftingOutput.cpp
    #include <iostream>
    #include "shiftingOutput.h"

    void shifting(int numberOfSpaces)
    {
        for (int i = 0; i < numberOfSpaces; i++)
        {
            std::cout << " ";
        }
    }

    void printPattern(std::string pattern, int currentIdx, int &shift)
    {
        shifting(shift);
        std::cout << pattern << "[" << currentIdx << "]\n";
        shift += 3;
    }

    void printPatternResult(std::string pattern, std::string sequence, bool
check, int &shift){
        shift-=3;
        shifting(shift);
        std::cout << pattern << "[" << sequence << "] - " << std::boolalpha <<
check << "\n";
    }

    syntaxPatterns.h
    #include <string>
    #ifndef UNTITLED_SYNTAXPATTERNS_H
    #define UNTITLED_SYNTAXPATTERNS_H

    bool parameter(std::string sequence, int &currentIdx, int &shift);

    bool listOfParameters(std::string sequence, int &currentIdx, int &shift);

    bool name (std::string sequence, int &currentIdx, int &shift);

    #endif //UNTITLED_SYNTAXPATTERNS_H

    syntaxPatterns.cpp
    #include "../shiftingOutput/shiftingOutput.h"
    #include "../characterCheckers/characterCheckers.h"
    #include "syntaxPatterns.h"
    #include <iostream>
```

```
      bool name(std::string sequence, int &currentIdx, int &shift)
      {
          printPattern("name", currentIdx, shift);
          const int givenIdx = currentIdx;

          bool    isCorrectName    =    isLetter(sequence,    currentIdx)    &&
isLetter(sequence, currentIdx) &&
                                isLetter(sequence, currentIdx);
          if (!isCorrectName)
          {
              printPatternResult("name", sequence.substr(givenIdx, currentIdx -
givenIdx), false, shift);
              return false;
          }

          printPatternResult("name",   sequence.substr(givenIdx,   currentIdx   -
givenIdx), true, shift);
          return true;
      }

      bool parameter(std::string sequence, int &currentIdx, int &shift)
      {
          printPattern("parameter", currentIdx, shift);
          const int givenIdx = currentIdx;

          bool bracketsCondition =
              name(sequence,    currentIdx,    shift)    &&    checkChar(sequence,
currentIdx, '=') &&
              checkChar(sequence, currentIdx, '(') && listOfParameters(sequence,
currentIdx, shift) &&
              checkChar(sequence, currentIdx, ')');
          if (bracketsCondition)
          {
              printPatternResult("parameter",          sequence.substr(givenIdx,
currentIdx - givenIdx), true,
                                 shift);
              return true;
          }

          shifting(shift);
          std::cout << "not condition with brackets \n";
          //Если первое условие не подходит, то индекс будет указывать на
символ, который не подошел
          //вернемся к исходному индексу, чтобы проверить другое условие
          currentIdx = givenIdx;

          bool twoDigitsCondition = name(sequence, currentIdx, shift) &&
                                    checkChar(sequence, currentIdx, '=') &&
                                    isDigit(sequence,       currentIdx)       &&
isDigit(sequence, currentIdx);
          if (twoDigitsCondition)
          {
              printPatternResult("parameter",          sequence.substr(givenIdx,
currentIdx - givenIdx), true,
                                 shift);
              return true;
          }

          printPatternResult("parameter", sequence.substr(givenIdx, currentIdx -
givenIdx), false, shift);
          return false;
      }
```

```
bool listOfParameters(std::string sequence, int &currentIdx, int &shift)
{
    printPattern("listOfParameters", currentIdx, shift);
    const int givenIdx = currentIdx;

    bool commaCondition = parameter(sequence, currentIdx, shift) &&
                          checkChar(sequence, currentIdx, ',') &&
                          listOfParameters(sequence, currentIdx, shift);
    if(commaCondition)
    {
        printPatternResult("listOfParameters",   sequence.substr(givenIdx,
currentIdx - givenIdx), true, shift);
        return true;
    }

    shifting(shift);
    std::cout << "not condition with comma \n";
    //Если первое условие не подходит, то индекс будет указывать на
символ, который не подошел
    //вернемся к исходному индексу, чтобы проверить другое условие
    currentIdx = givenIdx;

    if(parameter(sequence, currentIdx, shift))
    {
        printPatternResult("listOfParameters",   sequence.substr(givenIdx,
currentIdx - givenIdx), true, shift);
        return true;
    }

    printPatternResult("listOfParameters",       sequence.substr(givenIdx,
currentIdx - givenIdx), false, shift);
    return false;
}
```

main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include "syntaxPatterns/syntaxPatterns.h"

void analyzeSyntax(std::string sequence)
{
    int shift = 0;
    int currentIdx = 0;
    if (listOfParameters(sequence, currentIdx, shift))
    {
        //Проверим на соответствие количества символов в получившемся
шаблоне и заданной
        //последовательности
        if (currentIdx == size(sequence))
        {
            std::cout  <<  "  \nПоследовательность  соответствует  шаблону!
\n";
        }
        else
        {
            std::cout << "\nИзвините, но ваша последовательность длиннее
чем получившийся шаблон! \n";
        }
    }
    else
    {
```

```
            std::cout << "\nИзвините, но ваша последовательность не подходит
под шаблон! \n";
        }
        std::cout << "Длина последовательности = " << size(sequence)
                  << " Длина получившегося шаблона = " << currentIdx;
    }

    int main() {
        std::string sequence;
        std::ifstream inputFile;
        inputFile.open("../in.txt");
        if(inputFile.is_open()){
            getline(inputFile, sequence);
        }
        inputFile.close();
        std::cout << "Шаблон: \n"
                     "список_параметров::=параметр                              |
параметр,список_параметров\n"
                     "параметр::=имя=цифра цифра | имя=(список_параметров)\n"
                     "имя::=буква буква буква\n\n";
        if(size(sequence) > 0){
            std::cout << "Последовательность для анализа: " << sequence <<
"\n";
            analyzeSyntax(sequence);
        }
        else {
            std::cout << "Последовательность пуста!";
        }

        return 0;
    }
```