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. 9303 _____ Муратов Р.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием «рекурсия», изучить ее особенности и реализовать программу, решающую поставленную задачу с помощью рекурсии, на языке программирования C++.

Задание.

Вариант 16.

Построить синтаксический анализатор для понятия скобки.

скобки::=A | B | (скобки скобки).

Основные теоретические положения.

Реку́рсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя. Термин «рекурсия» наиболее широкое применение находит в математике и информатике.

В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия), например, функция A вызывает функцию B, а функция B — функцию A. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии. Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

Структурно рекурсивная функция на верхнем уровне всегда представляет собой команду ветвления (выбор одной из двух или более альтернатив в зависимости от условия (условий), которое в данном случае уместно назвать «условием прекращения рекурсии»), имеющую две или более альтернативные ветви, из которых хотя бы одна является рекурсивной и хотя бы одна — терминальной. Рекурсивная ветвь выполняется, когда условие прекращения рекурсии ложно, и содержит хотя бы один

рекурсивный вызов — прямой или опосредованный вызов функцией самой себя. Терминальная ветвь выполняется, когда условие прекращения рекурсии истинно; она возвращает некоторое значение, не выполняя рекурсивного вызова. Правильно написанная рекурсивная функция должна гарантировать, что через конечное число рекурсивных вызовов будет достигнуто выполнение условия прекращения рекурсии, в результате чего цепочка последовательных рекурсивных вызовов прервётся и выполнится возврат.

Выполнение работы.

Перечисление `Error` используется для более удобного вывода сообщения об ошибках, которые могут присутствовать во входной последовательности. Функция `getError(Error e, ostream& outFile)` как раз и выводит сообщение об ошибке, зависящее от переменной `e` (`outFile` — выходной файл). Элементы перечисления:

`FILE_NOT_OPEN` — невозможно открыть входной файл, `FILE_IS_EMPTY` — входной файл пуст, `INVALID_CHAR` — встречен недопустимый символ, `INCOMPLETE_BRACKETS` — встречена неполная скобка, `BRACKETS_NOT_CLOSE` — нет закрывающей скобки, `EXTRA_CHAR` — после правильной последовательности встречены лишние символы.

Функция `areBrackets(ifstream& inFile, ofstream& outFile, char curSymbol, uint16_t recDepth)` реализует алгоритм для решения поставленной задачи. Переменные `inFile` и `outFile` — входной и выходной файлы соответственно, `curSymbol` — текущий символ, `recDepth` — глубина рекурсии.

Функции `printIndent(uint16_t indent)` и `passLineTail(ifstream& inFile)` не относятся к основному алгоритму. Первая функция нужна для выполнения сдвига при выводе промежуточных данных в терминал, вторая — для пропуска ненужного

хвоста строки во входном файле (используется, если файл имеет несколько входных последовательностей, размещенных на разных строках).

В функции `areBrackets` сначала записывается в файл и выводится в терминал текущий символ. Затем, если текущий символ не является элементом множества {«A», «B», «(», «)»}, то выводится ошибка об этом (`INVALID_CHAR`). Далее находится терминальная ветвь функции: если текущий символ является A или B, то текущий элемент — «скобка». Если же текущий символ — это «(», то выполняется рекурсивная ветвь функции: два раза подряд выполняются одинаковые действия:

1. Считать символ
2. Проверить, не является ли считанный символ символом перевода строки
3. Рекурсивно проверить, является ли данный символ «скобкой».

При не выполнении условия из пункта 2, то выводится сообщение об ошибке `BRACKETS_NOT_CLOSE`, возвращается `false`, и стек вызовов функции начинает раскручиваться.

Если результатом проверки в пункте 3 является `false`, то возвращается `false` и стек продолжает раскручиваться (здесь уместно использовать глагол «продолжает», так как данные ветви выполняются, если вызов функции `areBrackets` вернул `false`, то есть стек уже в процессе раскрутки).

Далее считывается очередной символ, проверяется, является ли он символом перевода строки: если это так, то выводится ошибка `BRACKETS_NOT_CLOSE`, иначе выполнение функции продолжается. Считанный символ записывается в выходной файл и выводится в терминал с соответствующим отступом. Если символ — это «)», то возвращается `true`, то есть найдена «скобка», иначе выводится ошибка `INVALID_CHAR` и возвращается `false`.

Если при вызове функции `curChar` = «)», то выводится ошибка `INCOMPLETE_BRACKETS` и функция возвращает `false`, так как если встречена «скобка», то алгоритм дойдет до конца и вернет `true`, при этом не передав в вызовы функции `areBrackets` символ «)».

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Ознакомился с понятием «рекурсия», изучил ее особенности и реализовал программу, решающую поставленную задачу с помощью рекурсии, на языке программирования C++.

Разработана программа, которая считывает последовательности из входного файла, путь до которого вводит пользователь, и проверяет, удовлетворяет ли очередная считанная последовательность понятию «скобки». Результат работы программы сохраняется в выходном файле result.txt, промежуточные данные, характеризующие вычислительный процесс, выводятся в терминал.

Основная сложность при выполнении работы заключалась в остановке обработки входной последовательности при определенных условиях и выводе соответствующей ошибки. Именно поэтому код основной функции состоит из большого количества управляющих конструкции if-else.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

enum Error {
  FILE_NOT_OPEN,
  FILE_IS_EMPTY,
  INVALID_CHAR,
  INCOMPLETE_BRACKETS,
  BRACKETS_NOT_CLOSE,
  EXTRA_CHAR
};

void passLineTail(ifstream& inFile);
void printIndent(uint16_t indent);
void getError(Error e, ostream& outFile);
bool areBrackets(ifstream& inFile, ofstream& outFile, char curSymbol, uint16_t recDepth);

int main() {
  bool result = false;
  char curChar = '\0';

  cout << "Please, enter an input file name:\n";
  string filePath;
  cin >> filePath;
  ifstream inFile(filePath);
  if (!inFile.is_open()) {
    getError(FILE_NOT_OPEN, cout);
    return 0;
  }

   cout << "Result was placed in \'result.txt\' file of current directory\n";
   cout << "Here is intermediate data what visualize computing process\n\n";
  ofstream outFile("./result.txt");

  if (inFile.peek() == EOF) {
    inFile.close();
    outFile.close();
    getError(FILE_IS_EMPTY, cout);
    return 0;
  }

  outFile << "Brackets analyzer\n\n";
  while (1) {
    inFile >> curChar;
```

```
      if (inFile.eof()) {
        break;
      }
      result = areBrackets(inFile, outFile, curChar, 0);
      if (result) {
        if ((curChar = inFile.get()) != '\n') {
          outFile << curChar;
          cout << curChar << '\n';
          getError(EXTRA_CHAR, outFile);
          outFile << "Input sequence is NOT brackets\n\n";
          passLineTail(inFile);
        } else {
          outFile << "\nInput sequence is brackets\n\n";
        }
      } else {
        outFile << "Input sequence is NOT brackets\n\n";
        inFile.unget();
        passLineTail(inFile);
      }
      cout << '\n';
    }

    inFile.close();
    outFile.close();

    return 0;
  }

  void printIndent(uint16_t indent) {
    for (uint16_t i = 0; i < indent; ++i)
      cout << "-";
  }

  void passLineTail(ifstream& inFile) {
    while (inFile.get() != '\n') {}
  }

  void getError(Error e, ostream& outFile) {
    switch (e) {
      case FILE_NOT_OPEN:
        outFile << "\nError: Can't open a file!\n";
        break;
      case FILE_IS_EMPTY:
        outFile << "\nError: File is empty!\n";
        break;
      case INVALID_CHAR:
        outFile << "\nError: There is an invalid char in sequence!\
n";
        break;
      case INCOMPLETE_BRACKETS:
        outFile << "\nError: There is an incomplete brackets!\n";
        break;
      case BRACKETS_NOT_CLOSE:
        outFile << "\nError: Brackets isn't closed!\n";
        break;
      case EXTRA_CHAR:
```

```
            outFile << "\nError: There is extra characters after
sequence!\n";
          break;
        }
      }

     bool areBrackets(ifstream& inFile, ofstream& outFile, char
curSymbol, uint16_t recDepth) {
        outFile << curSymbol;
        printIndent(recDepth);
        cout << curSymbol << '\n';
        if ((curSymbol == '(') || (curSymbol == ')') ||
            (curSymbol == 'A') || (curSymbol == 'B')) {
          if ((curSymbol == 'A') || (curSymbol == 'B')) {
            return true;
          } else if (curSymbol == '(') {
            if ((curSymbol = inFile.get()) != '\n') {
                if (areBrackets(inFile, outFile, curSymbol, recDepth +
1)) {
                if ((curSymbol = inFile.get()) != '\n') {
                   if (areBrackets(inFile, outFile, curSymbol, recDepth
+ 1)) {
                    if ((curSymbol = inFile.get()) != '\n') {
                      outFile << curSymbol;
                      printIndent(recDepth);
                      cout << curSymbol << '\n';
                      if (curSymbol == ')') {
                        return true;
                      } else {
                        getError(INVALID_CHAR, outFile);
                        return false;
                      }
                    } else {
                      getError(BRACKETS_NOT_CLOSE, outFile);
                      return false;
                    }
                  } else {
                    return false;
                  }
                } else {
                  getError(BRACKETS_NOT_CLOSE, outFile);
                  return false;
                }
              } else {
                return false;
              }
            } else {
              getError(BRACKETS_NOT_CLOSE, outFile);
              return false;
            }
          } else {
            getError(INCOMPLETE_BRACKETS, outFile);
            return false;
          }
        } else {
          getError(INVALID_CHAR, outFile);
```

```
    return false;
  }
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	((BB)A (A (AA (((AA)(BB)	Brackets analyzer ((BB)A Error: Brackets isn't closed! Input sequence is NOT brackets (A Error: Brackets isn't closed! Input sequence is NOT brackets (AA Error: Brackets isn't closed! Input sequence is NOT brackets (Error: Brackets isn't closed! Input sequence is NOT brackets ((AA)(BB) Error: Brackets isn't closed! Input sequence is NOT brackets	Не закрытые скобки
2.	(A(B(A(B())))) (A())	Brackets analyzer	Пустые скобки или скобки, состоящие из

	(()) (A) ()	(A(B(A(B() Error: There is an incomplete brackets! Input sequence is NOT brackets (A() Error: There is an incomplete brackets! Input sequence is NOT brackets (() Error: There is an incomplete brackets! Input sequence is NOT brackets (A) Error: There is an incomplete brackets! Input sequence is NOT brackets () Error: There is an incomplete brackets! Input sequence is NOT brackets	одного элемента
3.	A~@31234123 (BB)& ((AB)((BA)((BB)A)))! @#	Brackets analyzer A~ Error: There is extra characters	Лишние символы

		after sequence! Input sequence is NOT brackets (BB)& Error: There is extra characters after sequence! Input sequence is NOT brackets ((AB)((BA)((BB)A)))! Error: There is extra characters after sequence! Input sequence is NOT brackets	
4.	((A(B~))B) ? ((A!)B) ((BB)(BB)~)	Brackets analyzer ((A(B~ Error: There is an invalid char in sequence! Input sequence is NOT brackets ? Error: There is an invalid char in sequence! Input sequence is NOT brackets ((A! Error: There is an invalid char in sequence! Input sequence is NOT	Недопустимые символы

		brackets ((BB)(BB)~ Error: There is an invalid char in sequence! Input sequence is NOT brackets	
5.	(AB) (A(AB)) (((A(AB))(BB))(BB)) A B (BB)	Brackets analyzer (AB) Input sequence is brackets (A(AB)) Input sequence is brackets (((A(AB))(BB))(BB)) Input sequence is brackets A Input sequence is brackets B Input sequence is brackets (BB) Input sequence is brackets	Корректные последовательности