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: Программирование рекурсивных алгоритмов

Студент гр. 9303 _____ Максимов Е.А.

Преподаватель _____ Филатов Ар. Ю.

Санкт-Петербург
2020

Цель работы.

Создание рекурсивного алгоритма для анализа понятия «скобки».

Задание.

Вариант №12 лабораторной работы.

Построить синтаксический анализатор для понятия скобки.

скобки::=квадратные | круглые | фигурные

квадратные::=[круглые фигурные] | +

круглые::=(фигурные квадратные) | −

фигурные::={квадратные круглые} | 0

Основные теоретические положения.

Рекурсивным называется объект, содержащий сам себя или определенный с помощью самого себя. Мощность рекурсии связана с тем, что она позволяет определить бесконечное множество объектов с помощью конечного высказывания. Бесконечные вычисления можно описать с помощью конечной рекурсивной программы. Рекурсивные алгоритмы лучше всего использовать, когда решаемая задача, вычисляемая функция или обрабатываемая структура данных определены с помощью рекурсии.

Если процедура (или функция) P содержит явное обращение к самой себе, она называется прямо рекурсивной. Если P содержит обращение к процедуре (функции) Q, которая содержит (прямо или косвенно) обращение к P, то P называется косвенно рекурсивной. Многие известные функции могут быть определены рекурсивно. Например факториал, который присутствует практически во всех учебниках по программированию, а также наибольший общий делитель, числа Фибоначчи, степенная функция и др.

Выполнение работы.

В программе реализованы следующие функции:

1. Функция *bool bracket(string line)* принимает на вход переменную *line*

типа *string*, представляющую собой набор символов. Функция является входной точкой рекурсивного алгоритма программы. В зависимости от первого символа, функция вызывает одну из трёх других функций, перечисленных ниже, которые обрабатывают набор символов и проверяют, соответствует ли входные данные заданным правилам построения скобок. Функция возвращает значения типа *bool* - результат обработки программы (является ли входная строка скобками или нет).

2. Функция *bool round(string line, int* i, int depth)* принимает на вход обрабатываемую строку *line*, указатель на целую переменную *i,* определяемую в функции *bool bracket(string line)*, и переменную *depth* типа *int*, обозначающая глубину рекурсии. Функция проверяет символ строки *line* по индексу *i* на соответствие заранее определённому синтаксису скобок,

Если символом круглой скобки является «-», то возвращается значение *true.*

Если символом круглой скобки является «(», то по правилам построения круглой скобки вызываются сначала *bool figure(string line, int* i, int depth)* для последующего символа, после, в случае успешной обработки вложенных скобок, - функция *bool square(string line, int* i, int depth)* так же для последующего символа, после - проверяется символ «)». В случае, если все вложенные функции вернули *true*, и последнее условие также выполнилось, то функция возвращает *true*, иначе функция выводит на экран ошибку и возвращает *false.*

3. Функции *bool figure(string line, int* i, int depth)* и *bool square(string line, int* i, int depth)* работают аналогичным способом, но с другим набором проверяемых символов и вызываемых функций.

4. Функция *bool error(int n)* принимает на вход переменную *n* типа *int* - номер ошибки, из-за которой не выполнилось одно из условий функций, описанных выше. Функция печатает на экран сообщение об ошибке.

Функция всегда возвращает *false.* (Функция реализована для сокращения исходного кода программы)

5. Функция *void recursionDepth(char s, int n)* печатает на экран сообщение, содержащее обрабатываемый символ и глубину рекурсии работающей программы, которая определяется переменной *depth,* определённой в функциях выше.

6. Функция *int main(int argc, char *argv[])* вызывает функцию *bool bracket(string line)* и обрабатывает входные данные. После запуска программы пользователю предлагается ввести выражение, и программа напечатает на экран шаги алгоритма и результат работы.

Пользователь может при запуске исполняемого файла ввести в качестве аргумента файл, в котором содержатся скобочные выражения, разделённые символом переноса строки. Если файл не пуст, то программа обработает каждую строку и запишет все шаги программы в файл «*BracketsOut.txt*».

Исходный код программы представлен в приложении А.

Результаты тестирования программы представлены в приложении Б.

Выводы.

Была реализована программа, для синтаксического анализа выражения «скобки» согласно заранее определённым правилам. Программа включает в себя рекурсивные функции, благодаря которым осуществляется анализ.

Были выполнены следующие требования: возможность ввода данных из файла или с консоли, вывод сообщений о глубине рекурсии, вывод информации о работе программы в отдельный файл или консоль.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp.

```
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc, char *argv[]){
  if(argc==1){
    cout << "Write input bracket: " << endl;
    string line;
    cin >> line;
    if(bracket(line)) cout << line << " is a bracket." << endl; else cout << line
 << " is NOT a bracket." << endl;
  }else if(argc==2){
    string line(argv[1]);
    ifstream in(line);
    if(!in){
      cout << "File not found!" << endl;
      return 0;
    }

    ofstream out("BracketsOut.txt");
    streambuf *coutbuf = cout.rdbuf();
    cout.rdbuf(out.rdbuf());
    int lineNum = 1;
    for(lineNum; getline(in, line); lineNum++){
      cout << "String #" << lineNum << endl;
      if(bracket(line)) cout << line << " is a bracket.\n" << endl; else cout <<
line << " is NOT a bracket.\n" << endl;
    }

    cout.rdbuf(coutbuf);
    if(lineNum == 1) cout << "Empty file!" << endl; else cout << "Results saved i
n BracketsOut.txt file." << endl;

  }else cout << "Incorrect input arguments." << endl;
  return 0;
}

bool bracket(string line){
  int i = 0;
  bool b = false;
  int depth = 0;

  if(line[0] == '\n' || line[0] == '\0') return error(0);
  if ((line[i] == '+') || (line[i] == '[')) b = square(line, &i, depth);
  else if ((line[i] == '-') || (line[i] == '(')) b = round(line, &i, depth);
  else if ((line[i] == '0') || (line[i] == '{')) b = figure(line, &i, depth);
  else return error(1);
```

```
  if(b && (i+1 < line.length())) return error(2);
  return (b && (i+1 == line.length()));
}
bool round(string line, int* i, int depth){
  bool k;
  recursionDepth(line[*i], depth);
  if(line[*i] == '-') return true;
  else if(line[*i] == '('){
    if(++(*i) < line.length()){
      k = figure(line, i, depth+1);
      if(k){
        if(++(*i) < line.length()){
          k = square(line, i, depth+1);
        }else return error(6);
      }else return false;

      if(k){
        if(++(*i) < line.length()){
          recursionDepth(line[*i], depth);
          return (line[*i] == ')');
        }else return error(5);
      }else return false;
    }else return error(7);
  }else return error(8);
}

bool square(string line, int* i, int depth){
  bool k;
  recursionDepth(line[*i], depth);
  if(line[*i] == '+') return true;
  else if(line[*i] == '['){
    if(++(*i) < line.length()){
      k = round(line, i, depth+1);
      if(k){
        if(++(*i) < line.length()){
          k = figure(line, i, depth+1);
        }else return error(7);
      }else return false;

      if(k){
        if(++(*i) < line.length()){
          recursionDepth(line[*i], depth);
          return (line[*i] == ']');
        }else return error(3);
      }else return false;
    }else return error(8);
  }else return error(6);
}

bool figure(string line, int* i, int depth){
  bool k;
  recursionDepth(line[*i], depth);
  if (line[*i] == '0') return true;
```

```
  else if(line[*i] == '{'){
    if(++(*i) < line.length()){
      k = square(line, i, depth+1);
      if(k){
        if(++(*i) < line.length()){
          k = round(line, i, depth+1);
        }else return error(8);
      }else return false;

      if(k){
        if(++(*i) < line.length()){
          recursionDepth(line[*i], depth);
          return (line[*i] == '}');
        }else return error(4);
      }else return false;
    }else return error(6);
  }else return error(7);
}

bool error(int n){
  cout << "Error!\t";
  switch (n){
    case 0: cout << "Empty string." << endl; break;
    case 1: cout << "Non-permitted symbol." << endl; break;
    case 2: cout << "Unexpected sequence." << endl; break;
    case 3: cout << "Expected ']' symbol." << endl; break;
    case 4: cout << "Expected '}' symbol." << endl; break;
    case 5: cout << "Expected ')' symbol." << endl; break;
    case 6: cout << "Expected '+' or '[' symbol." << endl; break;
    case 7: cout << "Expected '0' or '{' symbol." << endl; break;
    case 8: cout << "Expected '-' or '(' symbol." << endl; break;
  };
  return false;
}

void recursionDepth(char s, int n){
  cout << "Symbol: " << s << "\tRecursion depth is: " << n << endl;
  return;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица 1 - Тестирование программы.

№	Входные данные	Выходные данные	Комментарий
1.		Error! Empty string. is NOT a bracket.	Тест обработки пустой строки.
2.	A	Error! Non-permitted symbol. A is NOT a bracket.	Тест обработки некорректных символов.
3.	[-A]	Symbol: [Recursion depth is: 0 Symbol: - Recursion depth is: 1 Symbol: A Recursion depth is: 1 Error! Expected '0' or '{' symbol. [-A] is NOT a bracket.	Тест обработки некорректных символов.
4.	++	Symbol: + Recursion depth is: 0 Error! Unexpected sequence. ++ is NOT a bracket.	Тест обработки лишних символов.
5.	---	Symbol: - Recursion depth is: 0 Error! Unexpected sequence. --- is NOT a bracket.	Тест обработки лишних символов.
6.	{	Symbol: { Recursion depth is: 0 Error! Expected '+' or '[' symbol. { is NOT a bracket.	Тест обработки неполной конструкции скобок.
7.	{}	Symbol: { Recursion depth is: 0 Symbol: } Recursion depth is: 1 Error! Expected '+' or '[' symbol.	Тест обработки неполной

		{} is NOT a bracket.	конструкции скобок.
8.	{[}	Symbol: { Recursion depth is: 0 Symbol: [Recursion depth is: 1 Symbol: } Recursion depth is: 2 Error! Expected '-' or '(' symbol. {[} is NOT a bracket.	Тест обработки неполной конструкции скобок.
9.	(0	Symbol: (Recursion depth is: 0 Symbol: 0 Recursion depth is: 1 Error! Expected '+' or '[' symbol. (0 is NOT a bracket.	Тест обработки неполной конструкции скобок.
10.	{+-	Symbol: { Recursion depth is: 0 Symbol: + Recursion depth is: 1 Symbol: - Recursion depth is: 1 Error! Expected '}' symbol. {+- is NOT a bracket.	Тест обработки неполной конструкции скобок.
11.	0	Symbol: 0 Recursion depth is: 0 0 is a bracket.	Тест обработки примитивной конструкции скобок.
12.	[-{[-{[-{[-{[-{[-{+-}]-}]-}]-}]-}]-}]	Symbol: [Recursion depth is: 0 Symbol: - Recursion depth is: 1 Symbol: { Recursion depth is: 1 Symbol: [Recursion depth is: 2 Symbol: - Recursion depth is: 3 Symbol: { Recursion depth is: 3 Symbol: [Recursion depth is: 4 Symbol: - Recursion depth is: 5	Тест обработки конструкции скобок с большой глубиной рекурсии.

		Symbol: { Recursion depth is: 5 Symbol: [Recursion depth is: 6 Symbol: - Recursion depth is: 7 Symbol: { Recursion depth is: 7 Symbol: [Recursion depth is: 8 Symbol: - Recursion depth is: 9 Symbol: { Recursion depth is: 9 Symbol: [Recursion depth is: 10 Symbol: - Recursion depth is: 11 Symbol: { Recursion depth is: 11 Symbol: [Recursion depth is: 12 Symbol: - Recursion depth is: 13 Symbol: { Recursion depth is: 13 Symbol: + Recursion depth is: 14 Symbol: - Recursion depth is: 14 Symbol: } Recursion depth is: 13 Symbol:] Recursion depth is: 12 Symbol: - Recursion depth is: 12 Symbol: } Recursion depth is: 11 Symbol:] Recursion depth is: 10	

		Symbol: - Recursion depth is: 10 Symbol: } Recursion depth is: 9 Symbol:] Recursion depth is: 8 Symbol: - Recursion depth is: 8 Symbol: } Recursion depth is: 7 Symbol:] Recursion depth is: 6 Symbol: - Recursion depth is: 6 Symbol: } Recursion depth is: 5 Symbol:] Recursion depth is: 4 Symbol: - Recursion depth is: 4 Symbol: } Recursion depth is: 3 Symbol:] Recursion depth is: 2 Symbol: - Recursion depth is: 2 Symbol: } Recursion depth is: 1 Symbol:] Recursion depth is: 0 [-{[-{[-{[-{[-{[-{+-}]-}]-}]-}]-}]-}]-}] is a bracket.	