**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: Рекурсия

Вариант 21

Студентка группы 6383 Терещенко В.Н.  
Преподаватель Шолохова О.М.

Санкт-Петербург

2017

**1. Цель работы:** ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования С++.

**2. Задание:** построить синтаксический анализатор для понятия *скобки.*

*скобки::=квадратные / круглые*

*квадратные::=[[квадратные] (круглые)] / В*

*круглые::=((круглые)[квадратные]) / А*

**3. Постановка задачи:**

1. проанализировать полученное задание, выделив рекурсивно определяемые информационные объекты и (или) действия;
2. разработать программу, использующую рекурсию;
3. сопоставить рекурсивное решение с итеративным решением задачи;
4. сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

**4. Основные теоретические сведения.**

Взаимно-рекурсивные функции и процедуры

Пусть требуется построить *синтаксический анализатор* понятия *скобки:*

*cкобки::=квадратные | круглые   
 квадратные::=*[*круглые круглые*]*| + круглые::=*(*квадратные квадратные*)*| −*

В этом рекурсивном определении последовательности символов, называемой *cкобки*, присутствуют две взаимно-рекурсивные части: *квадратные* определяются через *круглые*, и наоборот, *круглые* − через *квадратные.* В простейшем случае *квадратные* есть символ «+», а *круглые* есть символ «−»*.* Другиепримеры последовательностей, порождаемых этим рекурсивным определением:

‘[– –]’, ‘(++)’, ‘[(++)([–(++)][– –])]’, ‘(+[(++)([–(++)][(+[– –])–])])’.

Синтаксическим анализатором назовём программу, которая определяет, является ли заданная (входная) последовательность символов *скобками* или нет. В случае ответа «нет» сообщается место и причина ошибки.

Реализуем основную часть этой программы как булевскую функцию Bracket, которая вызывает две другие (парные) булевские функции Round и Square, определяющие, является ли текущая подпоследовательность частью *круглые* или *квадратные* соответственно. Каждая из функций Round и Square в свою очередь вызывает парную к себе (Squareи Roundсоответственно). По правилам языка Паскаль описания этих функций в программе должны следовать в разделе описаний, например в такой последовательности:

**function** Round : Boolean; Forward; {*опережающее* описание }

**function** Square : Boolean;

**begin**

… {тело функции}

**end**{ Square };

**function** Round : Boolean;

**begin**

… {тело функции}

**end**{ Round };

**function** Bracket: Boolean;

**begin**

… {тело функции}

**end**{ Bracket }

Пусть входная последовательность читается из файла F, а результат и вспомогательные сообщения выводятся в файл G. Оба эти файла будут глобальными для функций Bracket, Round и Square. Вспомогательные сообщения квалифицируют ошибки в записи последовательности *скобки* в том случае, когда результат функции Bracket есть False. Для формирования этих сообщений будет использована процедура Error.

Функции Round и Square реализованы так, что они читают очередной символ входной последовательности и далее действуют в прямом соответствии с рекурсивными определениями частей *круглые* и *квадратные* соответственно. При этом в функции Bracket приходится читать первый символ входной последовательности дважды. Можно было бы избежать этого, используя «заглядывание вперёд», однако такая реализация менее прозрачна.

**Program** SyntaxAnalysisOfBracket;

{Bracket = скобки, Round = кругл, Square = квадр }

{ скобки ::= квадр | кругл }

{ квадр ::= + | [кругл кругл] }

{ кругл ::= – | (квадр квадр) }

**var** F,G : Text;

b: Boolean;

**procedure** Error (k: Word);

**begin**

WriteLn(G);

**case** k **of**

1:{Bracket} WriteLn(G,'! - Лишние символы во входной строке.');

2:{Bracket} WriteLn(G,'! - Недопустимый начальный символ.');

3:{Square} WriteLn(G,'! - Отсутствует "]".');

4:{Square} WriteLn(G,'! - Отсутствует "+" или "[".');

5:{Square} WriteLn(G,'! - Очередной квадр - пуст.');

6:{Round} WriteLn(G,'! - Отсутствует ")".');

7:{Round} WriteLn(G,'! - Отсутствует "–" или "(".');

8:{Round} WriteLn(G,'! - Очередной кругл - пуст.');

**else** {?} WriteLn(G,'! - ...');

**end**{case};

**end**{Error};

**function** Round : Boolean; Forward;

**function** Square : Boolean;

{ квадр ::= + | [кругл кругл] }

**var** s: Char;

k: Boolean;

**begin** Square := False;

**if not** Eof(F) **then**

**begin** Read(F,s); Write(G,s);

**if** s='+' **then** Square := True

**else if** s='[' **then**

**begin**

{ квадр ::= [кругл кругл] }

k := Round;

**if** **k then** k:={k **and** } Round

**else** {первый кругл ошибочен};

**if** k **then** {оба кругл правильны}

**if not** Eof(F) **then**

**begin**

Read(F,s); Write(G,s);

**if** (s=']') **then** Square:=True **else** Error(3)

**end**

**else** {нет ]} Error(3)

**else** {хотя бы один кругл ошибочен};

**end** {конец анализа [кругл кругл]}

**else** { не + и не [ } Error(4)

**end**

**else** {квадр - пуст} Error(5)

**end** {Square};

**function** Round : Boolean;

{ кругл ::= – | (квадр квадр) }

**var** s: Char;

k: Boolean;

**begin** Round := False;

**if not** Eof(F) **then**

**begin** Read(F,s); Write(G,s);

**if** s='–' **then** Round := True

**else if** s='(' **then**

**begin** { кругл ::= (квадр квадр) }

k := Square;

**if** k **then** k:={k **and** }Square

**else** {первый квадр ошибочен};

**if** k **then** {оба квадр правильны}

**if not** Eof(F) **then**

**begin**

Read(F,s); Write(G,s);

**if** (s=')') **then** Round:=True **else** Error(6)

**end**

**else** {нет )} Error(6)

**else** {хотя бы один квадр ошибочен};

**end** {конец анализа (квадр квадр)}

**else** { не – и не ( } Error(7)

**end**

**else** {кругл – пуст} Error(8)

**end** {Round};

**function** Bracket : Boolean;

{ not Eof(F) }

**var** b: Boolean; c: Char;

**begin**

b:=False;

Read(F,c); Reset(F);

**if** (c='+') **or** (c='[') **then** b:=Square

**else if** (c='−') **or** (c='(') **then** b:=Round

**else** {недопустимый начальный символ} Error(2);

Bracket := b **and** Eof(F);

**if** b **and not** Eof(F) **then** {лишние символы} Error(1);

**end** {Bracket};

**begin** { SyntaxAnalysisOfBracket }

Assign(F, 'Bracket.DAT'); Assign(G, 'BracketRes.DAT');

Reset (F); {Rewrite} Append(G);

WriteLn(G,'Анализатор скобок:');

**if not** Eof(F) **then**

**begin**

b := Bracket;

WriteLn(G);

**if** b **then**  WriteLn(G,'ЭТО СКОБКИ!')

**else** WriteLn(G,'НЕТ, ЭТО НЕ СКОБКИ!');

**end**

**else** WriteLn(G,'Пусто!');

Close(G);

**end.**

Отметим взаимную симметричность текстов процедур Round и Square, что соответствует симметричности определения понятия *скобки* относительно частей *круглые* и *квадратные* соответственно.

Результаты выполнения программы на некоторых тестовых данных приведены в табл. 2.6.

Таблица 2.6

|  |  |  |
| --- | --- | --- |
| Номер теста | Исходные данные  (файл Bracket.dat) | Результат  (файл BracketRes.dat) |
| 1 | (++) | Анализатор скобок:  (++)  ЭТО СКОБКИ! |
| 2 | [(++)([–(++)][– –])] | Анализатор скобок:  [(++)([–(++)][– –])]  ЭТО СКОБКИ! |
| 3 | [(++)([–(+–)][– –])] | Анализатор скобок:  [(++)([–(+–  ! - Отсутствует "+" или "[".  НЕТ, ЭТО НЕ СКОБКИ! |
| 4 | [(++)([–(++)][– –])]( | Анализатор скобок:  [(++)([–(++)][– –])]  ! – Лишние символы во входной строке.  НЕТ, ЭТО НЕ СКОБКИ! |

Программа в процессе анализа входной последовательности выводит её либо целиком, если она правильная (см., например, тесты 1 и 2), либо до того символа, который является ошибочным (см., например, тесты 3 и 4). Сообщение об ошибке выводится с новой строки после знака «!».

**5. Описание программы.**

После открытия входного файла переходим в функцию Brackets(), в которой идёт проверка текста на соответствие определению указанного в задании. Если же файл не открыт, программа выводит сообщение об ошибке и заканчивает работу. Из функции Brackets() мы переходим в рекурсию (Round() & Square()). Эти рекурсивные алгоритмы имеют симметричное строение и разбиты на две части: первая часть («квадратные:: = [ [ квадратные ] ( круглые ) ]»/ «круглые::=((круглые)[квадратные])») состоит из if-ов, условия в которых заданны заранее или вычисляются в ходе вызова рекурсивной функции (может и той же самой), вторая проверяет условие «квадратные:: = В»/ «круглые::=А». Затем программа выводит результат и заканчивает работу.

Описание функций и глобальных переменных:

bool Brackets (ifstream &f); // булева функция, которая является синтаксическим анализатором понятия "скобки"

bool Round (ifstream &f, char c); // взаимно рекурсивная функция с функцией Square (ifstream &f, char c). Возвращает 1, если определено понятие "квадратные" и 0 в противном случае

bool Square (ifstream &f, char c); // взаимно рекурсивная функция с функцией Round (ifstream &f, char c). Возвращает 1, если определено понятие "круглые" и 0 в противном случае

void Error (short n); // в случае нахождения ошибки 0 или 1 (пустая строка или недопустимый начальный символ) выводит сообщения об ошибке

**6. Тестирование.**

|  |  |
| --- | --- |
| Исходные данные | Результат |
|  | Пустая строка |
| ]BA] | Недопустимый начальный символ |
| () | Нет, это не скобка |
| [[]A] | Нет, это не скобка |
| [BA] | Да, это скобка |
| (AB) | Да, это скобка |
| ((AB)B) | Да, это скобка |
| [[B(AB)](AB)] | Да, это скобка |

**Вывод*:*** в данной работе я знакомилась с основными понятиями и приёмами рекурсивного программирования, получила навыки программирования рекурсивных процедур и функций.

**Приложение 1. Исходный код.**

// Lr ¹1

// 1.cpp

// Tereshchenko Viktoria, group 6383

// 18.09.17

// Var 21

#include <iostream>

#include <fstream>

using namespace std ;

bool Brackets(ifstream &f);

bool Round (ifstream &f, char c);

bool Square (ifstream &f, char c);

void Error (short n);

int main ( )

{

setlocale (LC\_ALL,"");

ifstream f ("input.txt");

if (!f.is\_open()) {

cout << "File can not be opened or created";

return 1;

}

bool a = Brackets (f);

if (a) cout << "Yes, it's'" << endl;

else cout << "No, it isn't'" << endl;

system("pause");

return 0;

}

bool Round (ifstream &f, char c)

{

if (c == 'A')

return true;

else

if (c == '(') {

f >> c;

if (Round (f, c))

f >> c;

else return false;

if (Square (f, c)) {

f >> c;

return (c == ')');

}

else return false;

}

else return false;

}

bool Square (ifstream &f, char c)

{

if (c == 'B')

return true;

else

if (c == '[') {

f >> c;

if (Square (f, c))

f >> c;

else return false;

if (Round (f, c)) {

f >> c;

return (c == ']');

}

else return false;

}

else return false;

}

bool Brackets(ifstream &f)

{

char c;

bool a;

a = false;

if (f >> c) {

if ((c == 'B') || (c == '[')) a = Square (f, c);

else if ((c == 'A') || (c == '(')) a = Round (f, c);

else Error(1);

}

else Error (0);

return a;

}

void Error (short n)

{

cout << "err ¹" << n << endl;

switch (n) {

case 0: cout << "Empty line" << endl;

break;

case 1: cout << "Invalid start character" << endl;

break;

default:

break;

};

}