**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. 9304 Шуняев А.В.

Преподаватель Фиалковский М.С.

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.**

Требования и рекомендации к выполнению задания:

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.

13. Построить синтаксический анализатор для понятия скобки.

скобки::=A | скобка скобки

скобка::= ( B скобки)

**Выполнение работы.**

Описание алгоритма работы программы:

На вход программы подаются строки неограниченной длинны. Данные строки записываются в список строк. После этого вызывается рекурсивный метод, который обрабатывает строки и возвращает булевое значение. Данное булевое значение определяет ответ, который будет записан в файл вывода.

Формат входных и выходных данных:

Входные данные представлены в виде строк, каждый символ которых может быть одним из символов ‘(‘, ‘)’, ‘A’, ‘B’. Они считываются из файлов input.txt, input2.txt, input3.txt. Выходными данные являются те же строки, плюс строки-индикаторы “ – THIS IS A BRACKETS” и “ – THIS IS NOT A

BRACKETS”, которые указывают является ли строка скобками или нет. Выходные данные записываются в файлы output1.txt, output2.txt, output3.txt.

Описание основных структур данных и функций (кратко):

Реализованы 2 класса Data и Recursion. Класс Data хранит в себе список строк, полученных на вход. Также в нем реализован статический метод start, который запускается из функции main. Данный метод вызывает все необходимые методы для выполнения алгоритма, создает объекты классов и выполняет запись результат. Также в данном классе реализован метод SpaceErase, который удаляет все пробелы в строке.

В классе Recursion реализована основная логика алгоритма. В нем реализован рекурсивный метод IsBrackets. Данный метод обрабатывает строки посимвольно.

Тестирование:

| Входные данные | Выходные данные | Результат теста |
|---|---|---|
| A | A - THIS IS A BRACKETS | True! |
| ( B A) A | ( B A) A - THIS IS A BRACKETS | True! |
| (B ( B A) A) A | (B ( B A) A) A - THIS IS A BRACKETS | True! |
| AA | AA - THIS IS NOT A BRACKETS | True! |
| ((B A) A | ((B A) A - THIS IS NOT A BRACKETS | True! |
| (BBA)A | (BBA)A - THIS IS NOT A BRACKETS | True! |
| B A ) A | B A ) A - THIS IS NOT A BRACKETS | True! |
| (B (B A) (B A) A) A | (B (B A) (B A) A) A - THIS IS A BRACKETS | True! |
| (B ( B (B A) A) A) A | (B ( B (B A) A) A) A - THIS IS A BRACKETS | True! |

**Выводы.**

Сопоставляя рекурсивное решение с итеративным, очевидно, что данная задача легко решается без рекурсии. Я считаю, применение рекурсии в решение является целесообразным только в целях обучения т.к. многократный вызов рекурсивной функции требует больше ресурсов чем цикл.

## ПРИЛОЖЕНИЕ А
## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "Game.h"
//#include "SFML/Graphics.hpp"

int main() {
  /* sf::RenderWindow window(sf::VideoMode(200, 200), "SFML works!");
   sf::CircleShape shape(100.f);
   shape.setFillColor(sf::Color::Green);

   while (window.isOpen())
   {
      sf::Event event;
      while (window.pollEvent(event))
      {
         if (event.type == sf::Event::Closed)
            window.close();
      }

      window.clear();
      window.draw(shape);
      window.display();
   }*/

      Game::start();
      return 0;
}
```

Название файла: cell.h

```
#pragma once

class Cell
{
public:
	~Cell();

	void SetCell(int row, int column, bool is_passable, int tag);
	void ChangeCell(bool is_passable, int tag = 0);
	int GetRow();
	int GetColumn();
	bool IsPassable();
	int GetTag();

private:
	int  row_;
	int  column_;
	int  tag_;// 0 - just cell, 1 - enter, 2 - exit.
	bool is_passable_;

};
```

Название файла: cell.cpp

```
#include "Cell.h"


void Cell::SetCell(int row, int column, bool is_passable, int tag) {
	this->row_ = row;
	this->column_ = column;
```

```
        this->is_passable_ = is_passable;
        this->tag_ = tag;
    }

    void Cell::ChangeCell(bool is_passable, int tag)
    {
        this->is_passable_ = is_passable;
        this->tag_ = tag;
    }

    int Cell::GetRow()
    {
        return this->row_;
    }

    int Cell::GetColumn()
    {
        return this->column_;
    }

    bool Cell::IsPassable()
    {
        return this->is_passable_;
    }

    int Cell::GetTag()
    {
        return this->tag_;
}
```

Название файла: Field.h

```
#pragma once

#include "Iterator.h"
#include "Cell.h"



class Field
{
 public:
        friend class Iterator;

        static Field* GetInstance(int height = 5, int width = 5);
        static void ResetInstance();
        Iterator Begin();
        Iterator End();
        ~Field();

 private:
        int height_;
        int width_;
        Cell* head_;
        Cell* end_ ;
        Cell** field_;
        static Field* ptr_field_;

        Field(int height, int width);

        Field(Field& other) = delete;
        Field& operator=(const Field&) = delete;
```

```
        Field(Field&& other) = delete;
        Field&& operator=(const Field&&) = delete;
};
```

Название файла: Field.cpp

```
#include "Field.h"

Field* Field::ptr_field_ = nullptr;

Field* Field::GetInstance(int height, int width) {
        if (ptr_field_ == nullptr) {
                ptr_field_ = new Field(height, width);
        }
        return ptr_field_;
}

void Field::ResetInstance()
{
        delete ptr_field_;
        ptr_field_ = nullptr;
}

Iterator Field::Begin()
{
        return Iterator(this->head_);
}

Iterator Field::End()
{
        return Iterator(this->end_);
```

```
}

Field::~Field()
{
	delete[] field_[0];
	delete[] field_;
}

Field::Field(int height, int width) {
	int road_count = ((height * width) - ((height + width) * 2)) / 2;

	this->height_ = height;
	this->width_ = width;
	this->field_ = new Cell * [this->height_];
	this->field_[0] = new Cell[(this->height_ * this->width_) + 1];
	for (int i = 1; i != this->height_; i++) {
		field_[i] = field_[i - 1] + this->width_;
	}

	for (int i = 0; i < this->height_; i++) {
		for (int j = 0; j < this->width_; j++) {
			this->field_[i][j].SetCell(i, j, false, 0);
		}
	}

	this->head_ = field_[0];
	this->end_ = this->field_[this->height_ - 1]+this->width_;
	int ads = 2;
}
```

Название файла: iterator.h

```
#pragma once

#include "Cell.h"



class Iterator
{
public:

	Iterator(Cell* target);

	Cell* operator*();

	bool operator==(const Iterator& other) const;
	bool operator!=(const Iterator& other) const;
	Iterator& operator++();
	Iterator  operator++(int);
	Iterator& operator--();
	Iterator  operator--(int);
	Iterator& operator+=(const int);



private:
	Cell* target_cell_;
};
```

Название файла: iterator.cpp

```
#include "Iterator.h"
#include "Field.h"
```

```
Iterator::Iterator(Cell* target)
{
	this->target_cell_ = target;
}

Cell* Iterator::operator*()
{
	return (this->target_cell_);
}

bool Iterator::operator==(const Iterator& other) const
{
	return ((target_cell_->GetColumn() == other.target_cell_->GetColumn())
		&& (target_cell_->GetRow() == other.target_cell_->GetRow()));
}

bool Iterator::operator!=(const Iterator& other) const
{
	return !(*this == other);
}

Iterator& Iterator::operator++()
{
	if (*this != Field::GetInstance()->End()) {
		target_cell_++;
	}
	return *this;
```

```
}

Iterator Iterator::operator++(int)
{
	Iterator iter = *this;
	++(*this);
	return iter;
}

Iterator& Iterator::operator--()
{
	if (*this != Field::GetInstance()->Begin()) {
		target_cell_--;
	}
	return *this;
}

Iterator Iterator::operator--(int)
{
	Iterator iter = *this;
	--(*this);
	return iter;
}

Iterator& Iterator::operator+=(const int step)
{
	if (step > 0) {
		for (int i = 1; i <= step; i++) {
			++(*this);
		}
```

```
        }
        else {
            for (int i = 1; i <= step*(-1); i++) {
                --(*this);
            }
        }

        return *this;
    }
```

Название файла: Game.h

```
    #pragma once
    #include <iostream>
    #include "Cell.h"
    #include "Field.h"
    #include "Iterator.h"
    #include <random>

    class Game
    {
    public:
        static Game* ptr_game_;

        static int  RandomNumber(int min, int max);
        void CreateLandscape(int, int);
        void Draw(int);
        static void start();

    private:
```

```
        Game() {};
};
```

Название файла: Game.cpp

```
#include "Game.h"
#include <Windows.h>

Game* Game::ptr_game_ = nullptr;


int Game::RandomNumber(int min, int max)
{
        std::mt19937 generator;
        std::random_device device;

        generator.seed(device());

        std::uniform_int_distribution<unsigned> dist(min, max);

        return dist(generator);
}

void Game::CreateLandscape(int height, int width)
{
        int road_count = RandomNumber((height * width), (height * width) * 2);
        int rand_pos = 0;
        Cell* enter = nullptr;
        Iterator iter = Field::GetInstance()->Begin();

        rand_pos = RandomNumber(1, ((height * width) / 4));
        iter += rand_pos;
```

```
(*iter)->ChangeCell(true, 1);
enter = (*iter);

while (road_count > 0) {
	int dir = 0;
	if ((*iter)->GetRow() < 2) {
		if ((*iter)->GetColumn() < 2) {
			dir = RandomNumber(3, 4);


		}
		else if ((*iter)->GetColumn() > width - 3) {
			dir = RandomNumber(3, 4);
			if (dir == 3) dir = 1;
		}
		else {
			dir = RandomNumber(2, 4);
			if (dir == 2) dir = 1;
		}
	}
	else if ((*iter)->GetRow() > height - 3) {
		if ((*iter)->GetColumn() < 2) {
			dir = RandomNumber(2, 3);


		}
		else if ((*iter)->GetColumn() > width - 3) {
			dir = RandomNumber(2, 3);
			if (dir == 3) dir = 1;
		}
		else {
```

```
                dir = RandomNumber(1, 3);
        }
}
else {
        if ((*iter)->GetColumn() < 2) {
                dir = RandomNumber(2, 4);

        }
        else if ((*iter)->GetColumn() > width - 3) {
                dir = RandomNumber(2, 4);
                if (dir == 3) dir = 1;
        }
        else {
                dir = RandomNumber(1, 4);
        }
}

switch (dir)
{
case 1:
        --iter;
        (*iter)->ChangeCell(true, 0);
        --iter;
        (*iter)->ChangeCell(true, 0);
        break;
case 2:
        iter += -(width);
        (*iter)->ChangeCell(true, 0);
        iter += -(width);
        (*iter)->ChangeCell(true, 0);
```

```
                break;
            case 3:
                ++iter;
                (*iter)->ChangeCell(true, 0);
                ++iter;
                (*iter)->ChangeCell(true, 0);
                break;
            case 4:
                iter += (width);
                (*iter)->ChangeCell(true, 0);
                iter += (width);
                (*iter)->ChangeCell(true, 0);
                break;
            }

            road_count -= 2;
        }

        iter = Field::GetInstance()->Begin();
        Cell* out = (*iter);

        for (iter; iter != Field::GetInstance()->End(); ++iter) {
            if ((*(iter++))->IsPassable()) {
                out = (*iter);
            }
        }
        out->ChangeCell(true, 2);
        enter->ChangeCell(true, 1);
}
```

```
void Game::Draw(int width)
{
	Iterator iter = Field::GetInstance()->Begin();

	for (iter; iter != Field::GetInstance()->End(); ++iter) {
		if ((*iter)->IsPassable()) {
			if ((*iter)->GetTag() == 1 || (*iter)->GetTag() == 2) {
				std::cout << '@';
			}
			else {
				std::cout << '.';
			}

		}
		else {
			std::cout << '#';
		}
		if ((*iter)->GetColumn() == width) {
			std::cout << '\n';
		}
	}
}

void Game::start()
{
	int height;
	int width;
	std::cin >> height >> width;
	if (ptr_game_ == nullptr) {
		ptr_game_ = new Game();
```

```
        }
        Field::GetInstance(height, width);
        ptr_game_->CreateLandscape(height, width);

        //while (!GetAsyncKeyState(VK_ESCAPE)) {
        ptr_game_->Draw(width-1);
        //        system("cls");
        //}


        Field::GetInstance()->ResetInstance();
}
```