Лабораторная работа №3

1. Постановка задачи

Квадрат разбит на 4^k равновеликих квадратных клеток. Квадрат перегибается поочередно относительно вертикальной (правая половина подкладывается под левую) и горизонтальной (нижняя половина подкладывается под верхнюю) оси симметрии до тех пор, пока все клетки не будут расположены друг под другом. Требуется занумеровать клетки исходного квадрата таким образом, чтобы в результате выполнения операций перегиба номера клеток, расположенных друг под другом, образовали числовую последовательность $1, 2, 3, ..., 4^k$, начиная с верхней клетки.

Задача требует использовать некоторую структуру данных: стек, очередь и т.д.

Следует реализовать структуру данных 3 способами:

1) Через массив
2) Через связный список
3) С использованием стандартной библиотеки (STL C++)

2. Описание решения.

Программа написана на языке C++ и скомпилирована в Visual Studio 2022, настройки проекта Visual Studio следующие:

Версия пакета SDK для Windows	**10.0 (последняя установленная версия)**
Набор инструментов платформы	**Visual Studio 2022 (v143)**
Стандарт языка C++	**Стандарт ISO C++14 (/std:c++14)**

Рисунок 1 - настройки проекта

Решение задачи происходит следующим образом: Реализован шаблонный класс *CustomStack*. Реализация методов зависит от параметра шаблона. Допускаются три параметра шаблона:

1) *int** – стек реализован через массив
2) *list* – стек реализован через односвязный список (свой, не из STL)
3) *stack<int>* – стек реализован через stack из STL

Реализации для каждого параметра шаблона описаны в файле "*CustomStack.cpp*"

Решение происходит следующим образом: создаётся трёхмерный массив, первые два измерения - контейнер *vector* из STL, а третье – объект собственного класса *CustomStack*. Будем считать, что первые два измерения – плоскость, на которой лежат «стопки» (стеки), которые растут вглубь. Изначально заполнен только стек в (0, 0), в нём содержатся все числа $1, 2, 3, ..., 4^k$. Пусть эта «стопка» будет результатом сгибания квадрата в соответствии с условием задачи. Начнём разворачивать квадрат в обратном порядке (сначала развернём вниз, потом вправо) до тех пор, пока «глубина» всех стеков не станет равной 1, тем самым найдём необходимое исходное расположение чисел в квадрате.

3. Тесты

Результаты работы программы для $k = 1, k = 2, k = 3$:

```
1 2
4 3

Kuibarov Vyacheslav Nikolaevich 090304-RPIa-o21
```

Рисунок 2 - результат при k=1

```
1 8 7 2
16 9 10 15
13 12 11 14
4 5 6 3

Kuibarov Vyacheslav Nikolaevich 090304-RPIa-o21
```

Рисунок 3 - Результат при k=2

```
1 32 25 8 7 26 31 2
64 33 40 57 58 39 34 63
49 48 41 56 55 42 47 50
16 17 24 9 10 23 18 15
13 20 21 12 11 22 19 14
52 45 44 53 54 43 46 51
61 36 37 60 59 38 35 62
4 29 28 5 6 27 30 3

Kuibarov Vyacheslav Nikolaevich 090304-RPIa-o21
```

Рисунок 4 - результат при k=3

Для проверки правильности результата использовались обычные квадратные листы бумаги: лист делился на необходимое количество клеточек, заполнялся значениями и сворачивался по заданным правилам. К сожалению, нельзя согнуть лист пополам более 7 раз, поэтому проверить правильность решения для $k > 3$ таким простым способом не представляется возможным. Результаты одинаковы для всех реализаций.

4. Листинг кода

Файл lab3.cpp

```
#include <iostream>
#include <vector>
#include <stack>

#include "CustomStack.h"

using namespace std;

/*
* Чтобы изменить реализацию стека нужно передать сюда одно из трёх значений:
* int* - реализация через обычный массив
* list - реализация через односвязный список (описан в CustomStack.h)
* stack<int> - реализация через стек из STL (да, стек реализован с помощью стека)
*/
//using Stack = CustomStack<int*>;
using Stack = CustomStack<list>;
//using Stack = CustomStack<stack<int>>;
using CubeRow = vector<Stack*>;
using Cube = vector<CubeRow>;

/*
    Берём половину элементов из стека "from"
    и запихиваем в стек "to" в обратном порядке

    Было:
    from: 1, 2, 3, 4
    to:   -

    Станет:
    from: 1, 2
    to:   4, 3
*/
void reverseHalf(Stack* from, Stack* to, int depth) {
    for (int i = 0; i < depth / 2; i++) {
        to->push(from->pop());
    }
}

/*
    4 4 0 0
    4 4 0 0
    0 0 0 0
    0 0 0 0

    Разворачивается и становится:

    2 2 0 0
    2 2 0 0
    2 2 0 0
```

```
    2 2 0 0
*/
void expandToBottom(Cube cube, int cubeSide, int bottomCount, int depth) {
    int subside = 1 << bottomCount; // 1, 2, 4, 8, 16, ...

    for (int i = 0; i < subside; i++) {
        for (int j = 0; j < subside; j++) {
            reverseHalf(cube[subside - 1 - i][j], cube[subside + i][j], depth);
        }
    }
}

/*
    2 2 0 0
    2 2 0 0
    2 2 0 0
    2 2 0 0

    Разворачивается и становится:

    1 1 1 1
    1 1 1 1
    1 1 1 1
    1 1 1 1
*/
void expandToRight(Cube cube, int cubeSide, int rightCount, int depth) {
    int height = 2 << rightCount; // 2, 4, 8, 16, ...
    int width = 1 << rightCount; // 1, 2, 4, 8, ...

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            reverseHalf(cube[i][width - 1 - j], cube[i][width + j], depth);
        }
    }
}

void printCubeIn2d(Cube cube, int cubeSide) {
    for (int i = 0; i < cubeSide; i++) {
        for (int j = 0; j < cubeSide; j++) {
            cout << cube[i][j]->getTop() << " ";
        }

        cout << endl;
    }
}

int main()
{
    const int k = 3;
    const int cubeSide = 1 << k;
    const int total = cubeSide * cubeSide; // сколько всего клеточек

    // трёхмерный массив
    // x - ширина, y - высота, z - стэк, глубина
    // По x,y находятся "стопки" чисел, изначально она одна в (0, 0)
    // и снаружи вглубь идёт как 1, 2, 3, 4, ..., total

    Cube cube(cubeSide);

    for (int i = 0; i < cubeSide; i++) {
        cube[i] = *new CubeRow(cubeSide);

        // Я пытался создавать стек перед тем как вызывать reverseHalf,
        // передавая реально необходимый размер, но по какой-то причине
        // код переставал работать (грешу на волшебные "оптимизации" Visual Studio)
        // Поэтому в случае реализации через массив приходится тратить много памяти
        for (int j = 0; j < cubeSide; j++) {
            cube[i][j] = new Stack(total);
        }
    }
```

```
    // заполняем исходную стопку (0, 0) от 1 до total
    for (int k = 0; k < total; k++) {
        cube[0][0]->push(k + 1);
    }

    // depth - количество слоёв у "стопок"
    int depth = total;
    // сколько раз развернули вниз
    int bottomCount = 0;
    // сколько раз развернули вправо
    int rightCount = 0;

    while (depth != 1) {
        expandToBottom(cube, cubeSide, bottomCount, depth);
        depth /= 2;

        expandToRight(cube, cubeSide, rightCount, depth);
        depth /= 2;

        bottomCount++;
        rightCount++;
    }

    //printCubeIn2d(cube, cubeSide);

    cout << endl << "Kuibarov Vyacheslav Nikolaevich 090304-RPIa-o21" << endl;

    return 0;
}
```

Файл CustomStack.h

```
#pragma once

#include<stack>

struct list {
	int data;
	list* next;
};

template<typename T>
class CustomStack
{
	// для реализации через массив
	int nextIndex;
	int* arr;

	// для реализации через связный список
	list* top;

	// для реализации через стек STL
	std::stack<int> s;
public:
	CustomStack(int n);

	bool is_empty();
	void push(int a);
	int pop();
	int getTop();
};
```

Файл CustomStach.cpp

```
#include <iostream>
#include <stack>
#include "CustomStack.h"
```

```
using namespace std;

/*
	Для разных шаблонных типов разные реализации методов
*/

CustomStack<list>::CustomStack(int n) {
	top = nullptr;
	nextIndex = 0;
	arr = nullptr;
	top = nullptr;
}
CustomStack<int*>::CustomStack(int n) {
	top = nullptr;
	nextIndex = 0;
	arr = new int[n];
	top = nullptr;
}
CustomStack<stack<int>>::CustomStack(int n) {
	top = nullptr;
	nextIndex = 0;
	arr = nullptr;
	top = nullptr;
}

bool CustomStack<list>::is_empty() {
	return top == nullptr;
}
bool CustomStack<int*>::is_empty() {
	return nextIndex == 0;
}
bool CustomStack<stack<int>>::is_empty() {
	return s.empty();
}


void CustomStack<int*>::push(int a) {
	arr[nextIndex++] = a;
}
void CustomStack<list>::push(int a) {
	list* temp = new list{ a, top };

	top = temp;
}
void CustomStack<stack<int>>::push(int a) {
	s.push(a);
}

int CustomStack<list>::pop() {
	if (is_empty()) {
		cout << "stack underflow" << endl;

		return 0;
	}

	int a = top->data;
	list* temp = top->next;

	delete top;

	top = temp;

	return a;
}
int CustomStack<int*>::pop() {
	if (is_empty()) {
		cout << "stack underflow" << endl;

		return 0;
	}
```

```
	int a = arr[--nextIndex];

	return a;
}
int CustomStack<stack<int>>::pop() {
	if (is_empty()) {
		cout << "stack underflow" << endl;

		return 0;
	}

	int a = s.top();
	s.pop();

	return a;
}

int CustomStack<list>::getTop() {
	return top->data;
}
int CustomStack<int*>::getTop() {
	return arr[nextIndex - 1];
}
int CustomStack<stack<int>>::getTop() {
	return s.top();
}
```