Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**МАТЕМАТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

**Отчет по лабораторной работе №1**

**Вспомогательные функции**

Выполнила:

Цывинская Алина Евгеньевна

студентка 2 курса 4 группы

факультета ФИТ

Минск 2025

**Лабораторная работа 1. Вспомогательные функции**

**ЦЕЛЬ РАБОТЫ:** приобретение навыков составления и отладки программ с использованием пользовательских функций для замера продолжительности процесса вычисления.

**ВЫПОЛНЕНИЕ РАБОТЫ:** составить и реализовать программы.

***Задание 1.*** Разработайте три функции (start, dget и iget), используя следующие спецификации:

//-- установка начального числа для генератора псевдослучайных // чисел

// # include "Auxil.h"

// namespace auxil

**void start();**

// функция устанавливает в качестве начального числа для // генератора псевдослучайных чисел текущее значение // системного времени в фомате функции time()

//-- генерация действительного псевдослучайного числа в // заданом

// # include "Auxil.h"

// namespace auxil

**double dget(**

**double rmin,** //[in]минимальное значение

**double rmax** //[in]максимальное значение

**);**

//-- функция возвращает действительное псевдослучайное число в // диапазоне оn **rmin** до **rmax**

//-- генерация целого псевдослучайного числа в заданом // диапазоне

// # include "Auxil.h"

// namespace auxil

**int iget(**

**int rmin,** //[in]минимальное значение

**int rmax** //[in]максимальное значение

**);**

//-- функция возвращает целое псевдослучайное число в // диапазоне оn **rmin** до **rmax**

**Примечание**: разработанные функции должны располагаться в файле **Auxil.cpp**,  а в файле **Auxil.h –** прототипы функций (см. пример 1).

Заголовочный файл:

#pragma once

#include <cstdlib>

namespace auxil {

void start();

double dget(double rmin, double rmax);

int iget(int rmin, int rmax);

}

Cpp. файл:

#include "Auxil.h"

#include <ctime>

namespace auxil {

void start() {

srand((unsigned)time(NULL));

}

double dget(double rmin, double rmax) {

return ((double)rand() / (double)RAND\_MAX) \* (rmax - rmin) + rmin;

}

int iget(int rmin, int rmax) {

return (int)dget((double)rmin, (double)rmax);

}

}

***Задание 2.*** Реализация mail.cpp

#include "Auxil.h"

#include <iostream>

#include <ctime>

#include <locale>

#define CYCLE 1000000

int main() {

setlocale(LC\_ALL, "rus");

double av1 = 0, av2 = 0;

clock\_t t1, t2;

auxil::start();

t1 = clock();

for (int i = 0; i < CYCLE; i++) {

av1 += (double)auxil::iget(-100, 100);

av2 += auxil::dget(-100, 100);

}

t2 = clock();

std::cout << "Количество циклов: " << CYCLE << std::endl;

std::cout << "Среднее значение (int): " << av1 / CYCLE << std::endl;

std::cout << "Среднее значение (double): " << av2 / CYCLE << std::endl;

std::cout << "Время выполнения (у.е.): " << (t2 - t1) << std::endl;

std::cout << "Время выполнения (сек): " << ((double)(t2 - t1)) / CLOCKS\_PER\_SEC << std::endl;

return 0;

}

***Задание 3.***

Проведите необходимые эксперименты и постройте график зависимости (Excel) продолжительности процесса вычисления от количества циклов в примере 2 (main.cpp). Проанализируйте характер зависимости. Проведите исследование любого другого рекурсивного алгоритма, например, вычисления факториала или генератора чисел Фибоначчи (прим. – например вычислите каким будет 100-е, 200-е, 300-е и т.д число), и включите в отчет график.

#include <iostream>

#include <ctime>

#include "Auxil.h"

#define CYCLE\_COUNT 5

#define FIB\_TESTS 7

int cycles[CYCLE\_COUNT] = { 10000, 50000, 100000, 500000, 1000000 };

int fib\_tests[FIB\_TESTS] = { 10, 20, 30, 40, 50, 100, 200 };

long long fibonacci(int n) {

if (n <= 1) return n;

return fibonacci(n - 1) + fibonacci(n - 2);

}

int main() {

setlocale(LC\_ALL, "rus");

std::cout << "Зависимость времени от количества циклов\n";

for (int i = 0; i < CYCLE\_COUNT; i++) {

double av1 = 0, av2 = 0;

clock\_t t1 = clock();

for (int j = 0; j < cycles[i]; j++) {

av1 += auxil::iget(-100, 100);

av2 += auxil::dget(-100, 100);

}

clock\_t t2 = clock();

double elapsed\_time = (double)(t2 - t1) / CLOCKS\_PER\_SEC;

std::cout << "Циклы: " << cycles[i] << ", Время (сек): " << elapsed\_time << std::endl;

}

std::cout << "\nВремя вычисления чисел Фибоначчи\n";

for (int i = 0; i < FIB\_TESTS; i++) {

clock\_t t1 = clock();

long long fib = fibonacci(fib\_tests[i]);

clock\_t t2 = clock();

double elapsed\_time = (double)(t2 - t1) / CLOCKS\_PER\_SEC;

std::cout << "F(" << fib\_tests[i] << ") = " << fib << ", Время (сек): " << elapsed\_time << std::endl;

}

system("pause");

return 0;

}

**Примечание**: продолжительность вычисления измерять в условных единицах процессорного времени (функция **clock**).



Время выполнения алгоритма рекурсивного вычисления чисел Фибоначчи зависит от n экспоненциально. Это связано с тем, что каждая функция вызывает себя дважды, что приводит к сложности O(2^n).  
Экспоненциальный рост подтверждается графиком, на котором видно, что время сначала растёт медленно, но затем увеличивается **в геометрической прогрессии**.  
При больших n (40+) время выполнения **становится неподходящим для реального использования**, что доказывает неэффективность простого рекурсивного метода.