**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Специальность программная инженерия**

Отчёт по лабораторной работе №1

По дисциплине «Математическое программирование»

На тему «Вспомогательные функции»

Выполнил:

Студент 2 курса 8 группы

Мамонько Денис Александрович

Преподаватель: асс. Ромыш А.С.

2025, Минск

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

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

**start** – функция установки начального числа как текущего значения для генератора.

**dget** – функция возвращает действительное число.

**iget** – функция возвращает целое число.

Перед тем, как начать работу, мы должны установить предкомпилированный заголовок. Для этого нужно создать файлы pch.h и pch.cpp. По умолчанию данных файлов проекта нет. В pch.h пишем все используемые в данное проекте библиотеки. В pch.cpp пишем только подключенный предкомпилированный файл. В свойствах проекта предкомпилированный заголовок stdafx.h меняем на pch.h, а в свойствах pch.cpp устанавливаем галочку на Yc (Создать). Далее приступаем непосредственно к работе. В задании 1 у нас были прототипы функций и сама их реализация, в листинге ниже приведен пример кода.

|  |
| --- |
| #pragma once  #include <cstdlib>  namespace auxil  {  void start(); // старт генератора сл. чисел  double dget(double rmin, double rmax); // получить случайное число  int iget(int rmin, int rmax); // получить случайное число  }; |

Листинг 1.1 – Файл Auxil.h

|  |
| --- |
| #include "pch.h"  namespace auxil  {  void start() // старт генератора сл. чисел  {  srand((unsigned)time(NULL));  };  double dget(double rmin, double rmax) // получить случайное число  {  return ((double)rand() / (double)RAND\_MAX) \* (rmax - rmin) + rmin;  };  int iget(int rmin, int rmax) // получить случайное число  {  return (int)dget((double)rmin, (double)rmax);  };  } |

Листинг 1.2 – Файл Auxil.cpp

***Задание 2***

1. Реализовать пример 2.
2. Для проверки работоспособности разработанных функций и приобретения навыков замера продолжительности процесса вычисления реализуйте программу, приведенную в примере 2.

В данном примере проверяем работоспособность наших трех функций, прототипы которых мы описывали ранее.

|  |
| --- |
| #include "pch.h"  #define CYCLE 1000000 // количество циклов  int main()  {  setlocale(LC\_ALL, "rus");  double av1 = 0, av2 = 0;  clock\_t t1 = 0, t2 = 0;  auxil::start(); // старт генерации  t1 = clock(); // фиксация времени  for (int i = 0; i < CYCLE; i++)  {  av1 += (double)auxil::iget(-100, 100); // сумма случайных чисел  av2 += auxil::dget(-100, 100); // сумма случайных чисел  }  t2 = clock(); // фиксация времени  std::cout << std::endl << "количество циклов: " << CYCLE;  std::cout << std::endl << "среднее значение (int): " << av1 / CYCLE;  std::cout << std::endl << "среднее значение (double): " << av2 / CYCLE;  std::cout << std::endl << "продолжительность (у.е): " << (t2 - t1);  std::cout << std::endl << " (сек): "  << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 2.1 – Файл main.cpp

Программа работает корректно, результат программы со всеми вычислениями представлен на рисунке 1.

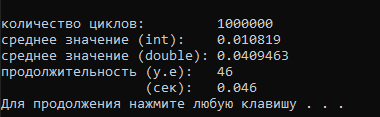


Рисунок 1 – Результат работы программы

**Задание 3**

Проведите необходимые эксперименты и постройте график зависимости (Excel) продолжительности процесса вычисления от количества циклов в примере 2. 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.

Для начала нам нужно найти зависимость продолжительности вычислений от количества циклов. Для этого просто каждый раз будем менять вручную количество циклов и проверим за какое время они будут выполняться. Диапазон циклов будет от 100000 до 1000000.

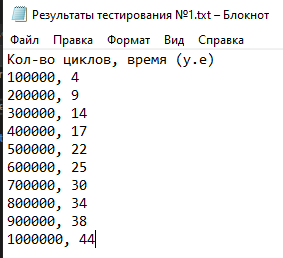


Рисунок 2 – Результаты тестирования №1



Рисунок 3 – График зависимости времени от цикла

Как мы видим, зависимость у нас линейная, так как количество операций прямо пропорционально числу циклов.

Проведем теперь исследование с числами Фибоначчи. Напишем программу, реализующую этот алгоритм.

|  |
| --- |
| #pragma once  int fibonachi(int n); |

Листинг 3.1 – Файл Fibonachi.h

|  |
| --- |
| #include "pch.h"  using namespace std;  int fibonachi(int n)  {  if (n == 0)  return 0;  if (n == 1)  return 1;  return fibonachi(n - 1) + fibonachi(n - 2);  } |

Листинг 3.2 – Файл Fibonachi.cpp

|  |
| --- |
| #include "pch.h"  #define CYCLE 1000000 // количество циклов  int main()  {  setlocale(LC\_ALL, "rus");  cout << "\t\tЧисла Фибоначчи" << endl;  int N;  clock\_t t3 = 0, t4 = 0;  cout << "Введите количество чисел N: ";  cin >> N;  t3 = clock();  int result = fibonachi(N);  cout << "Результат функции чисел Фибоначчи: " << result;  t4 = clock();  std::cout << std::endl << "продолжительность (у.е): " << (t4 - t3);  std::cout << std::endl << " (сек): "  << ((double)(t4 - t3)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 3.3 – Файл main.cpp

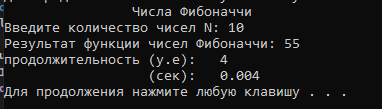


Рисунок 4 – Результат работы программы

Для того, чтобы построить график и определить зависимость, мы также должны проверить за какое время будет выполняться алгоритм чисел Фибоначчи исходя из входного параметра N. Значения N будем менять вручную в диапазоне от 10 до 40 с шагом 5.

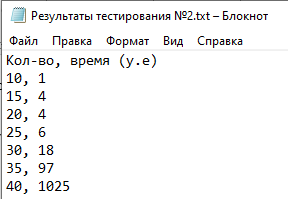


Рисунок 5 – Результаты тестирования №2

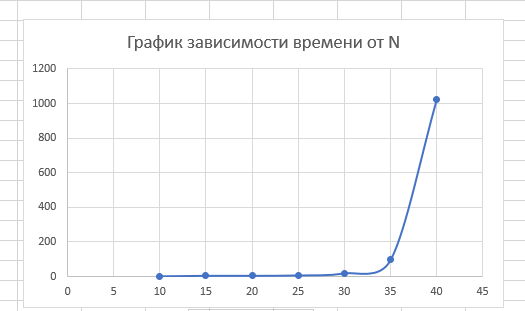


Рисунок 6 – График зависимости времени от N

На графике видно большое отклонение, так что данную зависимость можно считать экспоненциальной.

**ИТОГ**

1. В первом эксперименте зависимость времени выполнения от количества циклов линейная.
2. Во втором эксперименте зависимость времени выполнения рекурсивного алгоритма Фибоначчи экспоненциальная.