МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ

по учебной вычислительной (ознакомительной) практике

Большакова Артёма Викторовича

студента 1 курса, 2 группы

специальность «Информатика»

Руководитель практики:

старший преподаватель

О. М. Кондратьева

Минск, 2024

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc171022093)

[Введение 3](#_Toc171022094)

[Глава 1 Теоретические основы 4](#_Toc171022095)

[1.1 Общая характеристика курса «Контроль версий с помощью Git» 4](#_Toc171022096)

[1.2 Общая характеристика курса «Python» 4](#_Toc171022097)

[1.2.1 Введение 4](#_Toc171022098)

[1.2.2 Базовые конструкции Python 5](#_Toc171022099)

[1.2.3 Коллекции и работа с памятью 5](#_Toc171022100)

[1.2.4 Функции и их особенности в Python 5](#_Toc171022101)

[1.2.5 Объектно-ориентированное программирование 6](#_Toc171022102)

[1.2.6 Библиотеки для получения и обработки данных 6](#_Toc171022103)

[1.3 Выводы 6](#_Toc171022104)

[Глава 2 Индивидуальные Задания 8](#_Toc171022105)

[2.1 Класс MyList 8](#_Toc171022106)

[2.2 Класс MyStack 9](#_Toc171022107)

[2.3 Класс ListIterator 11](#_Toc171022108)

[2.4 Графический интерфейс 12](#_Toc171022109)

[2.5 Работа с Json файлами 15](#_Toc171022110)

[2.6 Выводы 17](#_Toc171022111)

[Заключение 18](#_Toc171022112)

[Список использованных источников 19](#_Toc171022113)

# Введение

Целью учебной вычислительной (ознакомительной) практики является закрепление полученных знаний за первый год обучения посредством выполнения специальных учебных заданий, участия в работе над командным проектом.

Задачи практики:

* Знакомство с IT компанией, возможностями обучения и повышения квалификации, знакомство с разными направлениями профессиональной подготовки IT специалистов.
* Изучение инструментов для совместной разработки проектов и контроля версий.
* Подготовка и настройка программных сред и средств тестирования для прохождения тренинга по выбранному направлению.
* Выполнение стандартных заданий.
* Выполнение заданий повышенного уровня.
* Оформление отчета.

# Глава 1 Теоретические основы

## 1.1 Общая характеристика курса «Контроль версий с помощью Git»

В рамках курса по системе контроля версий Git мы пользовались курсом от EPAM Systems (<https://learn.epam.com/detailsPage?id=601f195a-d408-4439-a16d-0630ed2a412e>), в котором содержится вся достаточная информация для самостоятельного изучения. Данный курс содержит следующие темы:

* VCS concept
* Version control types
* Why Git
* Download, install, configure
* Create a github repo and clone it
* Pull from remote
* Git Gui & gitk
* Inside .git folder
* Undoing changes
* Git reset
* Git revert
* . gitignore
* Branching and merge
* Conflict solving
* Rebase
* Cherry-pick
* Tags
* Stashing
* Remotes
* Branching strategies
* Extras

## 1.2 Общая характеристика курса «Python»

В рамках курса по Python мы проходили общедоступный курс от Яндекс (<https://education.yandex.ru/handbook/python>), рассчитанный на изучение с нуля. Данный курс содержит как и самые базовые темы, например ввод и вывод информации в консоль, так и изучение внешних широко используемых библиотек.

## 1.2.1 Введение

На рисунке 1.1 перечислены темы данной главы.

|  |
| --- |
|  |
| Рисунок 1.1 – темы первой главы |

## 1.2.2 Базовые конструкции Python

На рисунке 1.2 перечислены темы данной главы, а также результат выполнения упражнений каждой из тем.

|  |
| --- |
|  |
| Рисунок 1.2 – темы первой главы, а также результаты упражнений |

## 1.2.3 Коллекции и работа с памятью

На рисунке 1.3 перечислены темы данной главы, а также результат выполнения упражнений каждой из тем.

|  |
| --- |
|  |
| Рисунок 1.3 – темы первой главы, а также результаты упражнений |

## 1.2.4 Функции и их особенности в Python

На рисунке 1.4 перечислены темы данной главы, а также результат выполнения упражнений каждой из тем.

|  |
| --- |
|  |
| Рисунок 1.4 – темы первой главы, а также результаты упражнений |

## 1.2.5 Объектно-ориентированное программирование

На рисунке 1.5 перечислены темы данной главы, а также результат выполнения упражнений каждой из тем.

|  |
| --- |
|  |
| Рисунок 1.5 – темы первой главы, а также результаты упражнений |

## 1.2.6 Библиотеки для получения и обработки данных

На рисунке 1.6 перечислены темы данной главы, а также результат выполнения упражнений каждой из тем.

|  |
| --- |
|  |
| Рисунок 1.6 – темы первой главы, а также результаты упражнений |

## 1.3 Выводы

В процессе изучения выше представленных курсов, были пройдены и усвоены все необходимые базовые навыки для уверенного пользования системы контроля версий Git и языка Python. Выделенного времени на курсы было вполне достаточно для освоения приобретенных навыков. В частности курс по Python был усвоен на 76%, что показано на рисунке 1.7.

|  |
| --- |
|  |
| Рисунок 1.7 – общее количество решенных задач по Python |

# Глава 2 Индивидуальные Задания

## 2.1 Класс MyList

Класс MyList представляет собой реализацию односвязного списка. Односвязный список — это структура данных, состоящая из элементов-узлов. Каждый узел состоит из данных и ссылки на следующий узел списка. Последний узел, не имеющий следующего узла, указывает на nullptr.

Класс MyList имеет следующий интерфейс:

class MyList

{

private:

SElement \*first;

public:

// Конструктор без параметров создает пустой список

MyList();

// Конструктор с 1 параметром создает список из одного элемента

MyList(int a);

// Конструктор с 3 параметрами создает список из 3 элементов

MyList(int a, int b, int c);

// Деструктор MyList очищает выделенную на узлы списка памяти

~MyList();

// Возвращает состояние списка

bool Empty();

// Добавляет элемент в начало списка

void AddFirst(int Value);

// Удаляет первый элемент списка

int DeleteElement();

// Выводит на экран весь список

void Print();

// Возвращает количество элементов в списке

int Count();

// Возвращает значение первого элемента списка

int Get(int);

// Преобразует список в вектор

void ToVector(std::vector<int>&);

};

Фрагменты кода с реализацией:

void MyList::AddFirst(int Value)

{

SElement \*NEW = new SElement(Value, first);

first = NEW;

}

void MyList::Print()

{

for (SElement \*temp = first; temp != nullptr; temp = temp->next)

std::cout << temp->data << " ";

}

int MyList::DeleteElement()

{

if (first == nullptr)

return -1;

SElement \*Temp = first;

first = first->next;

int Value = Temp->data;

delete Temp;

return Value;

}

Полная реализация класса MyList расположена в git репозитории (<https://github.com/ArpiJokle/AB_Practice/tree/master/MyList>).

На рисунке 2.1 представлен пример использования класса MyList, а на рисунке 2.2 представлен результат выполнения программы.

|  |
| --- |
|  |
| Рисунок 2.1 – содержание файла main.cpp |

|  |
| --- |
|  |
| Рисунок 2.2 – результат выполнения программы |

## 2.2 Класс MyStack

“Адаптер – паттерн, структурирующий классы и объекты” [1].

Адаптеры используются для преобразования интерфейса одного класса в другой интерфейс. Адаптеры обеспечивают совместную работу классов, у которых несовместимые интерфейсы.

Класс MyStack это класс-адаптер, который приватно наследуется от класса MyList. Другими словами, MyStack наследует функционал MyList но скрывает его от внешнего доступа. MyStack был реализован пользуясь инструкцией из [1].

Интерфейс класса MyStack:

class MyStack : private MyList

{

public:

// Конструктор без параметров создает пустой стек

MyStack() : MyList() {}

// Конструктор с 1 параметром содает стек с 1 элементом

MyStack(int a) : MyList(a) {}

// Конструктор с 3 параметрами содает стек с 3 элементами

MyStack(int a, int b, int c) : MyList(a, b, c) {}

// Деструктор по умолчанию

~MyStack() = default;

// Возвращает верхний элемент стека

int Pop();

// Добавляет верхний элемент

void Push(int);

// Возвращает состояние стека

bool IsEmpty();

// Выводит стек

void PrintStack();

// Преобразует стек в вектор

void toVector(std::vector<int> &);

};

Код реализации:

int MyStack::Pop()

{

return DeleteElement();

}

void MyStack::Push(int a)

{

AddFirst(a);

}

bool MyStack::IsEmpty()

{

return Empty();

}

void MyStack::PrintStack()

{

Print();

}

void MyStack::toVector(std::vector<int> &Vec)

{

ToVector(Vec);

}

Вся реализация класса MyStack расположена в git репозитории (<https://github.com/ArpiJokle/AB_Practice/tree/master/MyStack>).

На рисунке 2.3 представлен пример использования класса MyStack, а на рисунке 2.4 представлен результат выполнения программы.

|  |
| --- |
|  |
| Рисунок 2.3 – содержание файла main.cpp |

|  |
| --- |
|  |
| Рисунок 2.4 – результат выполнения программы |

## 2.3 Класс ListIterator

“Итератор – паттерн поведения объектов” [1].

Итераторы позволяют реализовать последовательное обращение ко всем элементам составного объекта, не раскрывая внутреннее представление.

Интерфейс класса ListIterator:

class ListIterator : public Iterator

{

public:

// Конструктор ListIterator

ListIterator(MyList \*aList);

// Установнение начальной позиции итератора

virtual void First();

// Переход к следующему элементу списка

virtual void Next();

// Проверка выхода за границы списка

virtual bool IsDone();

// Возвращение элемента списка

virtual int CurrentItem();

private:

MyList \*\_list;

int \_current;

};

Код реализации:

ListIterator::ListIterator(

MyList \*aList) : \_list(aList), \_current(0) {}

void ListIterator::First()

{

\_current = 0;

}

void ListIterator::Next()

{

\_current++;

}

bool ListIterator::IsDone() {

return \_current <= \_list->Count();

}

int ListIterator::CurrentItem() {

return \_list->Get(\_current);

}

Вся реализация класса ListIterator расположена в git репозитории (<https://github.com/ArpiJokle/AB_Practice/tree/master/MyListIterator>).

На рисунке 2.5 представлен пример использования класса ListIterator, а на рисунке 2.6 представлен результат выполнения программы.

|  |
| --- |
|  |
| Рисунок 2.5 – содержание файла main.cpp |

|  |
| --- |
|  |
| Рисунок 2.6 – результат выполнения программы |

## 2.4 Графический интерфейс

У нас было 2 задания, в которых было необходимо реализовать графический интерфейс с использованием фреймворка для разработки кроссплатформенных приложение Qt.

Первым заданием было: для заданного списка натуральных чисел нарисовать горизонтальную гистограмму. Для выполнения данной задачи был использован класс QPainter. Решение этой задачи расположено в git репозитории (<https://github.com/ArpiJokle/AB_Practice/tree/master/Task_1_25.06>).

На рисунке 2.7 представлен снимок окна приложения. При его открытии открывается диалог выбора файла со списком натуральных чисел, и после выбора нужного .txt файла на экране программы строится гистограмма по заданным данным.

|  |
| --- |
|  |
| Рисунок 2.7 – снимок окна приложения |

Вторым заданием было построение горизонтальной гистограммы используя реализованный ранее класс MyStack. Решение этой задачи расположено в git репозитории (<https://github.com/ArpiJokle/AB_Practice/tree/master/Tasks_26.06/QT2>).

На рисунке 2.8 представлен снимок окна данного приложения.

|  |
| --- |
|  |
| Рисунок 2.8 – снимок окна приложения |

При нажатии на кнопку Input data появляется окно диалога выбора файла со списком натуральных чисел, и после выбора нужного .txt файла на экране программы строится гистограмма по заданным данным как показано на рисунке 2.9.

|  |
| --- |
|  |
| Рисунок 2.9 – снимок окна приложения с построенной гистограммой |

Также при выборе числа и нажатии на кнопку Push в наш стек натуральных чисел добавляется введенное, и соответственно гистограмма дополняется как показано на рисунке 2.10.

|  |
| --- |
|  |
| Рисунок 2.10 – снимок окна приложения после добавления элемента |

При на нажатии на кнопку Pop из нашего стека выкидывается верхний элемент. На рисунке 2.11 представлен снимок окна приложения после 5 нажатий кнопки.

|  |
| --- |
|  |
| Рисунок 2.11 – снимок окна приложения после удаления 5 элементов |

## 2.5 Работа с Json файлами

Для работы с .json файлами в c++ была использована библиотека nlohmann/json расположенной в git репозитории (<https://github.com/nlohmann/json/tree/develop/single_include/nlohmann>).

У нас было задание реализации программы по чтению и редактированию .json файлов в текстовом и графическом виде.

Для чтения я использовал input.json файл, который содержит следующую информацию:

{

"id": 1,

"length": 8,

"mass": 16,

"width": 9

}

Решение задачи в текстовом виде расположено в git репозитории (<https://github.com/ArpiJokle/AB_Practice/tree/master/Qt-4/Json_console>) а ее интерфейс представлен на рисунке 2.12

|  |
| --- |
|  |
| Рисунок 2.12 – снимок окна приложения |

В данной программе реализован выбор файла для чтения, возможность добавления новых полей, возможность сохранения изменений, а также выход из программы.

Решение задачи в графическом виде с помощью фреймворка для разработки кроссплатформенных приложение Qt расположено в git репозитории (<https://github.com/ArpiJokle/AB_Practice/tree/master/Qt-4/Json_Qt/Json_Qt4>) а ее снимок окна приложения представлен на рисунке 2.13

|  |
| --- |
|  |
| Рисунок 2.13 – снимок окна приложения |

При нажатии на кнопку Import появляется окно диалога выбора файла со списком натуральных чисел, и после выбора нужного .json файла на экране программы строится гистограмма по заданным данным как показано на рисунке 2.14.

|  |
| --- |
|  |
| Рисунок 2.14 – снимок окна приложения |

При введении названия нового поля и заданию ему некоторого значения, а далее нажатия на кнопку Add field в нашем json объекте добавляется новое поле, а гистограмма показывающая хранящиеся значения обновляется как показано на рисунке 2.15.

|  |
| --- |
|  |
| Рисунок 2.15 – снимок окна приложения |

При нажатии на кнопку Save появляется окно диалога выбора файла для сохранения измененных данных.

## 2.6 Выводы

В результате выполнения практических задач, были усвоены паттерны проектирования, изучен формат хранения данных Json а также манипуляция файлов .json. Были изучены способы наглядного графического отображения информации. Также были освежены навыки использования языка c++, полученные на курсе “Основы и методологии программирования” первого семестра, и изучены тонкости данного языка.

# Заключение

За период выполнения практических занятий мы на деле удостоверились в удобстве и необходимости использования системы контроля версий Git. Был почти полностью пройден курс по языке Python, а также на практике познакомились с форматом хранения данных Json, и изучили паттерны проектирования. А главное в итоге углубилось понимание того что нужно подробнее изучить и как двигаться дальше.

# Список использованных источников

1. Гамма Э. Паттерны объектно-ориентированного проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. . – СПб.: Питер, 2020. — 448 с.: ил. — (Серия «Библиотека программиста»).
2. Майерс, Г. Искусство тестирования программ / Пер. с англ. под ред. Б. А. Позина. – Москва.: Финансы и статистика, 1982. – 176 с. (или Майерс, Г. Искусство тестирования программ // Г. Майерс. – Москва.: Финансы и статистика, 1982. – 176 с.)
3. Шлее, Г. Qt 5.10. Профессиональное программирование на С++ // М. Шлее. – СПб.: БХВ-Петербург, 2018. – 1072 с.
4. C++ - Сериализация (json) [Электронный ресурс] // Youtube. – Режим доступа: https://www.youtube.com/watch?v=x8wsONn1IcQ&list=LL&index=5. – Дата доступа: 19.07.2020.