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

[Введение 3](#_Toc171149778)

[Глава 1 Теоретические основы 4](#_Toc171149779)

[1.1 Обучающий курс «Контроль версий с помощью Git» 4](#_Toc171149780)

[1.2 Обучающий курс «Python» 4](#_Toc171149781)

[1.2.1 Базовые конструкции Python 5](#_Toc171149782)

[1.2.2 Коллекции и работа с памятью 5](#_Toc171149783)

[1.2.3 Функции и их особенности в Python 5](#_Toc171149784)

[1.2.4 Объектно-ориентированное программирование 6](#_Toc171149785)

[1.2.5 Библиотеки для получения и обработки данных 6](#_Toc171149786)

[1.3 Выводы 7](#_Toc171149787)

[Глава 2 Индивидуальные Задания 8](#_Toc171149788)

[2.1 Класс MyList 8](#_Toc171149789)

[2.2 Класс MyStack 9](#_Toc171149790)

[2.3 Класс ListIterator 11](#_Toc171149791)

[2.4 Графический интерфейс 13](#_Toc171149792)

[2.5 Работа с JSON-файлами 15](#_Toc171149793)

[2.6 Выводы 18](#_Toc171149794)

[Заключение 19](#_Toc171149795)

[Список использованных источников 20](#_Toc171149796)

# Введение

Целью учебной вычислительной (ознакомительной) практики является закрепление полученных знаний за первый год обучения посредством выполнения специальных учебных заданий.

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 Обучающий курс «Контроль версий с помощью Git»

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

* 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 Обучающий курс «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 Базовые конструкции Python

На рисунке 1.1 представлены темы раздела «Базовые конструкции Python», а также результат выполнения упражнений каждой из тем. На изучение было затрачено 4 часа.

|  |
| --- |
|  |
| Рисунок 1.1 – Темы раздела, а также результаты упражнений |

## 1.2.2 Коллекции и работа с памятью

На рисунке 1.2 представлены темы раздела «Коллекции и работа с памятью», а также результат выполнения упражнений каждой из тем. На изучение было затрачено 5 часов.

|  |
| --- |
|  |
| Рисунок 1.2 – Темы раздела, а также результаты упражнений |

## 1.2.3 Функции и их особенности в Python

На рисунке 1.3 представлены темы раздела «Функции и их особенности в Python», а также результат выполнения упражнений каждой из тем. На изучение было затрачено 5 часов.

|  |
| --- |
|  |
| Рисунок 1.3 – Темы раздела, а также результаты упражнений |

## 1.2.4 Объектно-ориентированное программирование

На рисунке 1.4 представлены темы раздела «Объектно-ориентированное программирование», а также результат выполнения упражнений каждой из тем. На изучение было затрачено 6 часов.

|  |
| --- |
|  |
| Рисунок 1.4 – Темы раздела, а также результаты упражнений |

## 1.2.5 Библиотеки для получения и обработки данных

На рисунке 1.5 представлены темы раздела «Библиотеки для получения и обработки данных», а также результат выполнения упражнений каждой из тем. На изучение было затрачено 4 часа.

|  |
| --- |
|  |
| Рисунок 1.5 – Темы раздела, а также результаты упражнений |

## 1.3 Выводы

В процессе изучения курсов, были пройдены и усвоены необходимые базовые навыки для уверенного использования 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 представлено на рисунке 1.6.

|  |
| --- |
|  |
| Рисунок 1.6 – 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.

Объект класса MyList содержит ссылку на первый узел списка.

Класс MyList имеет следующий интерфейс:

class MyList

{

private:

SElement \*first;

public:

// Создает пустой список

MyList();

// Создает список из одного элемента

MyList(int a);

// Создает список из 3 элементов

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

// Очищает выделенную на узлы списка памяти

~MyList();

// Возвращает значение true, если список пуст

bool Empty();

// Добавляет элемент в начало списка

void AddFirst(int Value);

// Удаляет первый элемент списка

int DeleteElement();

// Выводит список в стандартный поток вывода

void Print();

// 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>).

Метод main, содержащий проверку работы MyList:

#include <iostream>

#include "List.h"

int main()

{

MyList list;

list.AddFirst(7);

list.AddFirst(4);

list.Print();

std::cout << "\n";

list.AddFirst(6);

list.AddFirst(2);

list.Print();

std::cout << "\n";

list.DeleteElement();

list.Print();

std::cout << "\n";

std::cout << (list.Empty() ? "empty\n" : "not empty\n");

std::vector<int> Vector;

list.ToVector(Vector);

return 0;

}

На рисунке 2.1 представлен результат выполнения программы.

|  |
| --- |
|  |
| Рисунок 2.1 – Консольное окно вывода 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] для реализации адаптера-класса.

Интерфейс класса MyStack:

class MyStack : private MyList

{

public:

// Создает пустой стек

MyStack() : MyList() {}

// Создает стек с 1 элементом

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

// Создает стек с 3 элементами

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

~MyStack() = default;

// Возвращает верхний элемент стека

int Pop();

// Добавляет верхний элемент

void Push(int);

// Возвращает значение true, если стек пустой

bool IsEmpty();

// Выводит стек в стандартный поток вывода

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>).

Метод main, содержащий проверку работы MyStack:

#include <iostream>

#include "MyStack.h"

int main()

{

MyStack stack;

stack.Push(7);

stack.Push(4);

stack.Push(6);

stack.Push(2);

stack.PrintStack();

std::cout << std::endl;

std::vector<int> Vector;

stack.toVector(Vector);

for(auto i : Vector)

std::cout << i << "\n";

return 0;

}

На рисунке 2.2 представлен результат выполнения программы.

|  |
| --- |
|  |
| Рисунок 2.2 – Консольное окно вывода программы |

## 2.3 Класс ListIterator

“Итератор – паттерн поведения объектов” [1].

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

Для реализации функционала класса ListIterator в класс MyList были добавлены методы int Count() и int Get(int n), которые возвращают количество элементов в списке и значение элемента на позиции n соответственно.

Интерфейс класс Iterator, от которого наследуется класс ListIterator:

class Iterator

{

public:

virtual void First() = 0;

virtual void Next() = 0;

virtual bool IsDone() = 0;

virtual int CurrentItem() = 0;

protected:

Iterator() = default;

};

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>).

Метод main, содержащий проверку работы ListIterator:

#include <iostream>

#include "List.h"

int main()

{

MyList list;

list.AddFirst(23);

list.AddFirst(1);

list.AddFirst(4);

list.AddFirst(2);

list.AddFirst(3);

for (ListIterator iterator(&list); iterator.IsDone(); iterator.Next())

{

std::cout << iterator.CurrentItem() << '\n';

}

return 0;

}

На рисунке 2.3 представлен результат выполнения программы.

|  |
| --- |
|  |
| Рисунок 2.3 – Консольное окно вывода программы |

## 2.4 Графический интерфейс

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

*Задание 1.* 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 2.4 представлено главное окно приложения. При запуске программы открывается диалог выбора текстового файла со списком натуральных чисел, и после выбора нужного txt-файла на экране программы строится гистограмма по заданным данным.

|  |
| --- |
|  |
| Рисунок 2.4 – Главное окно приложения |

*Задание 2.* 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 2.5 представлено главное окно приложения.

|  |
| --- |
|  |
| Рисунок 2.5 – Главное окно приложения |

При нажатии на кнопку «Input data» открывается диалог выбора текстового файла со списком натуральных чисел, и после выбора нужного txt-файла на экране программы строится гистограмма по заданным данным как показано на рисунке 2.6.

|  |
| --- |
|  |
| Рисунок 2.6 – Главное окно 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 2.7.

|  |
| --- |
|  |
| Рисунок 2.7 – Главное окно приложения после добавления элемента |

При на нажатии на кнопку «Pop» из нашего стека выкидывается верхний элемент. На рисунке 2.8 представлено главное окно приложения после 5 нажатий кнопки.

|  |
| --- |
|  |
| Рисунок 2.8 – Главное окно приложения после удаления 5 элементов |

## 2.5 Работа с JSON-файлами

Для работы с JSON-файлами в C++ была использована библиотека nlohmann/json, расположенная в git-репозитории (<https://github.com/nlohmann/json/tree/develop/single_include/nlohmann>).

*Задание 3.* Чтение и редактирование JSON-файлов в текстовом и графическом виде.

На вход подается input.json файл, который содержит объект со следующими полями:

{

"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 2.9.

|  |
| --- |
|  |
| Рисунок 2.9 – Консольное окно вывода программы |

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 приложений Qt Creator расположено в git‑репозитории (<https://github.com/ArpiJokle/AB_Practice/tree/master/Qt-4/Json_Qt/Json_Qt4>). На рисунке 2.10 представлено главное окно приложения.

|  |
| --- |
|  |
| Рисунок 2.10 – Главное окно приложения |

При нажатии на кнопку «Import» открывается диалог выбора JSON-файла, содержащего некоторый объект, после выбора нужного файла на экране программы строится гистограмма по заданным данным как показано на рисунке 2.11.

|  |
| --- |
|  |
| Рисунок 2.11 – Главное окно приложения с построенной гистограммой |

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 2.12.

|  |
| --- |
|  |
| Рисунок 2.12 – Главное окно приложения с добавленным полем |

При нажатии на кнопку «Save» открывается 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 изменения JSON-файлов. Были изучены способы наглядного графического отображения информации. Также были углублены навыки использования языка C++, полученные в курсе “Основы и методологии программирования” первого семестра, и изучены тонкости данного языка.

# Заключение

Прохождение практики позволило получить следующие результаты:

* Была изучена система контроля версий Git, а именно умение инициализировать репозитории, управление ветками и способы избавления от конфликтов при слиянии ветвей.
* Прохождение курса программирования на Python позволило освоить новый язык на достойном уровне.
* Освоено использование библиотеки C++ для работы с данными в формате JSON.
* Были изучены принципы проектирования тестов [2].
* Были изучены паттерны проектирования [1].

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 // Г.   Майерс. – Москва.: Финансы и статистика, 1982. – 176 с.
3. Шлее, М. 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: 05.07.2024.