ЗМІСТ

[АНОТАЦІЯ](#_Toc390260760)

[ВСТУП 5](#_Toc390260761)

[1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ 7](#_Toc390260762)

[2 ОБГРУНТУВАННЯ АЛГОРИТМУ І СТРУКТУРИ ПРОГРАМИ 8](#_Toc390260763)

[3 РОЗРОБКА ПРОГРАМИ 10](#_Toc390260764)

[4 ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ 18](#_Toc390260765)

[ВИСНОВОК 23](#_Toc390260766)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 24](#_Toc390260767)

ДОДАТОК А………………………………………………………………..25

ДОДАТОК Б………………………………………………………………..28

ДОДАТОК В………………………………………………………………..30

# АНОТАЦІЯ

У цій курсовій роботі описано об’єктно-орієнтований підхід розробки програмного продукту мовою програмування С++. Використано уніфіковану мову моделювання (UML) для відображення акторами та прецедентами потреб до системи. В кінцевому результаті отримали програму для обробки результатів сесії студентів ІІ-го курсу.

# ВСТУП

Курсовий проект є комплексним завданням по застосуванню та закріпленню вивченого матеріалу за курс з ведучої дисципліни (в даному випадку з об'єктно-орієнтованого програмування). Він включає в себе розробку повноцінної системи (згідно з варіантом) і дає можливість проявити креативність і свої здібності в програмуванні та на кодовій частині проекту.

Розробляючи проект дуже важливим аспектом є створення раціональної системи та її структури, які б підходили під усі вимоги, та найголовніше – виконувала свою задачу якомога найкраще. В даному курсовому проекті представлена програма обробки результатів сесії студентів ІІ-го курсу.

Програмування – це процес написання, тестування і підтримки комп'ютерних програм. З розвитком програмування виникла ідея поєднати в межах однієї сутності дані і код, що безпосередньо опрацьовує ці дані. Така сутність отримала назву об’єкт, а відповідний підхід до створення програм називають об’єктно-орієнтованим програмуванням.

Об’єктно-орієнтоване програмування (ООП) – це парадигма програмування, яка розглядає програму як сукупність гнучко пов’язаних між собою об’єктів.

Основні переваги концепції ООП:

* можливість створювати користувацькі типи даних (класи);
* приховування деталей реалізації (інкапсуляція);
* можливість повторного використання коду (наслідування);

Метою даної роботи є створення програми для обробки результатів сесії студентів ІІ-го курсу. Для цього використовувалось середовище розробки – Microsoft Visual Studio 2010.

Git — розподілена система керування версіями файлів та спільної роботи. Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і

комітів. (Додаток А)

Обробка винятків (також опрацьовування виняткових ситуацій, англ. exception handling) — механізм мов програмування, призначений для обробки помилок часу виконання і інших можливих проблем (винятків), які можуть виникнути при виконанні програми. (Додаток Б)

Модульне тестування (англ. Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні — інтерфейс, клас. Модульні тести, або unit-тести, розробляються в процесі розробки програмістами та, іноді, тестувальниками білої скриньки (white-box

testers).(Додаток В)

# АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

Дана програма створена для обробки результатів сесії студентів ІІ-го курсу. Середовище розробки програми: Microsoft Visual Studio 2010. Програма написана на мові програмування С++, вона високого рівня з підтримкою декількох парадигм програмування: об’єктно-орієнованої, узагальненої та процедурної. С++ додає до С об'єктно-орієнтовані можливості. Вона вводить класи, які забезпечують три найважливіші властивості ООП: інкапсуляцію, успадкування і поліморфізм.

Інкапсулювання – це механізм в програмуванні, який пов’язує в одне ціле функції і дані, якими вони маніпулють, а також захищає їх від зовнішнього доступу і неправильного застосування.

Успадкування – це властивість, з допомогою якої один об’єкт може набувати властивостей іншого. При цьому підтримується концепція ієрархічної класифікації.

Поліморфізм дозволяє писати більш абстрактні програми і підвищити коефіцієнт повторного використання коду.

Програма виконує:

* Зчитування даних про студентів з файлу;
* Виведення середнього балу студента та його оцінок;
* Виведення кращого і гіршого студента;
* Сортування студентів по середньому балу від кращого до гіршого і від гіршого до кращого;
* Сортування студентів по балах з вказаного предмета від кращого до гіршого та від гіршого до кращого;
* Виведення студентів які здали сесію і котрі не здали сесії;
* Виведення студентів в яких бал вищий вказаного по вказаному предмету, і в яких нижчий.

# ОБґРУНТУВАННЯ АЛГОРИТМУ І СТРУКТУРИ ПРОГРАМИ

Клас — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій.

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

8

ТНТУ КРКН 15.006.084.006 ПЗ

Розроб.

*Добривода Н.І.*

Перевір.

*Бревус В.М.*

Реценз.

Н. Контр.

Затверд.

Розділ 2

Літ.

Аркушів

2

ТНТУ ФІС гр. СН-21

У програмі створено кілька класів. Клас Parser виконує парсинг стрічки. Синтаксичний аналіз (парсинг) (англ. parsing) — це процес аналізу вхідної послідовності символів, з метою розбору граматичної структури згідно із заданою формальною граматикою. Синтаксичний аналізатор (англ. parser) — це програма або частина програми, яка виконує синтаксичний аналіз.Під час синтаксичного аналізу текст оформлюється у структуру даних, зазвичай — в дерево, яке відповідає синтаксичній структурі вхідної послідовності, і добре підходить для подальшої обробки. Зазвичай синтаксичні аналізатори працюють в два етапи: на першому ідентифікуються осмислені токени (виконується лексичний аналіз), на другому створюється дерево розбору. Тобто парсинг займається, безпосередньо, збором, аналізом і перетворенням необхідної інформації. Клас Student повертає ім’я студента, його рейтинг по предметах, середній бал, виводить значення на екран.

Клас Loder зберігає у собі ім’я файлу даних студентів і використовує клас Parser для розбору стрічок.

Клас Util містить всі методи якими можна впливати на масив студентів. Виконує основне завдання програми, обробляє дані.

Класи та функції які знаходяться в програмі розбиті на модулі. Модуль — функціонально завершений фрагмент програми, оформлений у вигляді окремого файлу, призначений для використання в інших програмах. Модулі дозволяють розбивати складні задачі на менші відповідно до принципу модульності. Тобто кожен наслідуваний клас має відповідний

набір методів, які виконують роль реалізації програми.

Програма реалізує вікно, в якому можна побачити результати сесії студентів 2 курсу. В наслідуваних файлах заголовках включено основні файли заголовки зі стандартної бібліотеки С++, такі як: global.h, loader.h, parser.h, student.h та utils.h.

Також до програми підключається текстовий файл даних результатів сесії студентів students.txt для відповідної подальшої обробки цих результатів програмою.

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

9

ТНТУ КРКН 15.006.084.0006 ПЗ

# РОЗРОБКА ПРОГРАМИ

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

10

ТНТУ КРКН 15.006.084.006 ПЗ

Розроб.

*Добривода Н.І.*

Перевір.

*Бревус В.М.*

Реценз.

Н. Контр.

Затверд.

Розділ 3

Літ.

Аркушів

8

ТНТУ ФІС гр. СН-21

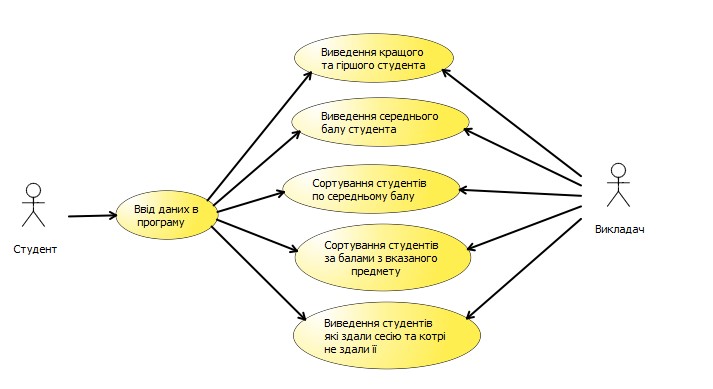


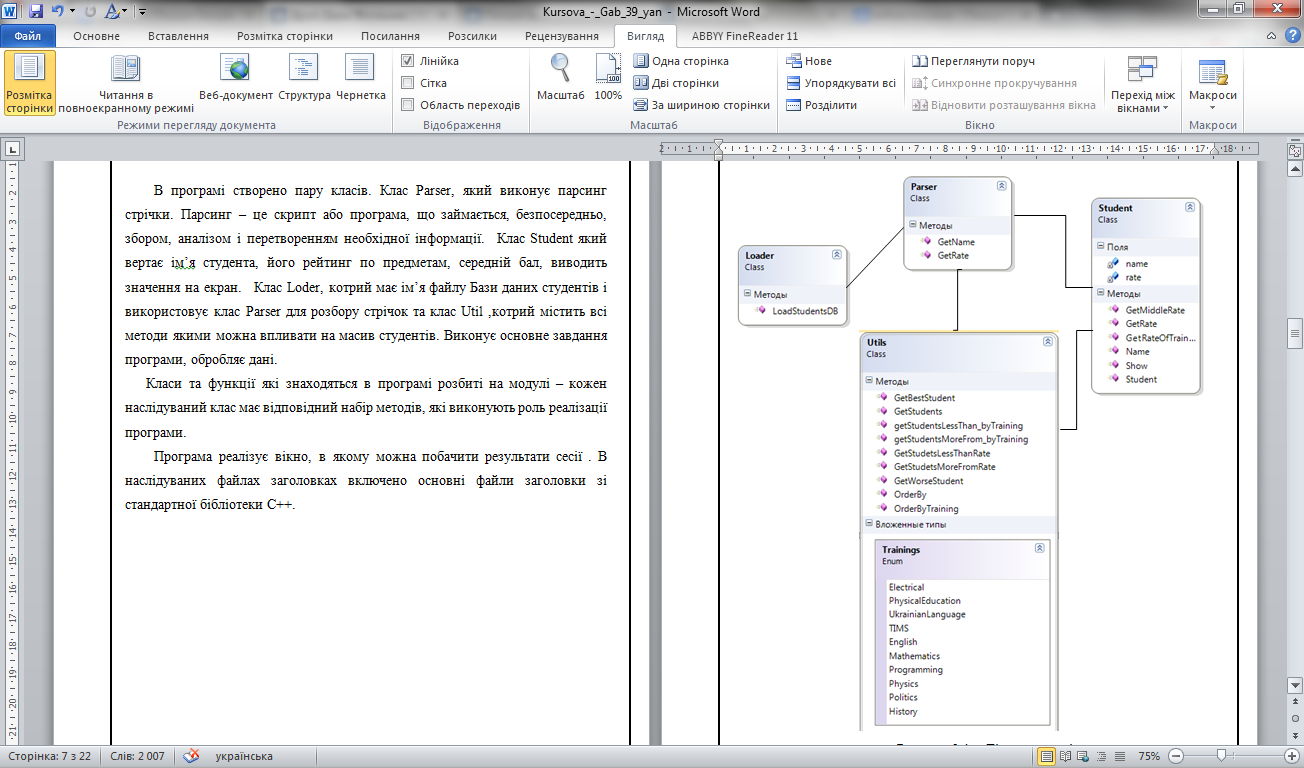
Рисунок 3.1 – Діаграма прецедентів

Діаграма прецедентів — в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання. Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (англ. use case) використовують для описання послуг, які система

надає актору. Іншими словами, кожен варіант використання визначає деякий

набір дій, який виконує система при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.



Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

11

ТНТУ КРКН 15.006.084.0006 ПЗ

Рисунок 3.2 – Діаграма класів

Діаграма класів — статичне представлення структури моделі. Відображає статичні елементи, такі як: класи, типи даних, їх зміст та відношення. Класом називається іменований опис сукупності об'єктів із загальними атрибутами, операціями, зв'язками і семантикою. Графічно клас зображується у вигляді прямокутника. Атрибутом класу називається   
  
іменована властивість класу, що описує безліч значень, котрі можуть приймати екземпляри цієї властивості. Клас може мати будь-яке число атрибутів (зокрема, не мати жодного атрибута).

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

12

ТНТУ КРКН 15.006.084.0006 ПЗ

Клас Parser (Рис. 3.3) виконує парсинг стрічки. У цьому класі представлені два методи. Метод GetName – парсит( дістає) з стрічки ім’я студента та метод GetRate, який витягає вектор оцінок

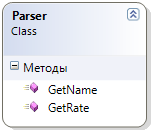


Рис. 3.3 Клас Parser

Клас Loder (Рис 3.4) використовує клас Parser для розбору стрічок. У цьому класі представлений метод LoadStudentsDB , який загружає дані у масив студента.

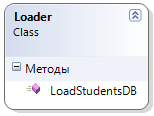
vector<Student> studentDB = vector<Student>();

for(int i=0;i<lines.size();i++)

studentDB.push\_back(Student(Parser::GetName(lines[i]), Parser::GetRate(lines[i])));

return studentDB;

}

Рис. 3.4 Клас Loder

Клас Student (Рис. 3.5.) вертає ім’я студента і масив оцінок по предметам, переліченим в enum Trainings. Він має такі поля як name (ім’я) та rate (рейтинг), тобто оцінки по предмету доступні по індексу предмета.

Клас Student представлений такими методати , як:

GetRateofTraining – повертає рейтинг по вказаному предмету.

int Student::GetRateOfTraining(int trainingIndex)

{

return (trainingIndex < rate.size()) ? this->rate[trainingIndex] : -1;

}

GetMiddleRate – повертає середній бал по всім предметам

double Student::GetMiddleRate()

{

double tmpMiddleRate = 0;

for (int i = 0; i < this->rate.size(); i++)

tmpMiddleRate += ((double)this->rate[i]);

return tmpMiddleRate /= ((double)(this->rate.size()));

}

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

13

ТНТУ КРКН 15.006.084.0006 ПЗ

Show – виводить на екран інформацію про студента

void Student::Show()

{

cout << this->name << endl;

for (int i = 0; i < this->rate.size(); i++)

cout << rate[i] << ' ';

cout << endl;

}

Name – вивиодить ім’я студента

string Student::Name()

{

return this->name;

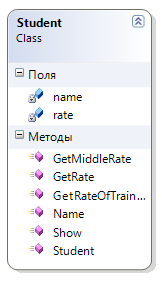
}

GetRate – виводить список оцінок студента

vector<int> Student::GetRate()

{

return this->rate;



Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

14

ТНТУ КРКН 15.006.084.0006 ПЗ

Рис. 3.5 Клас Student

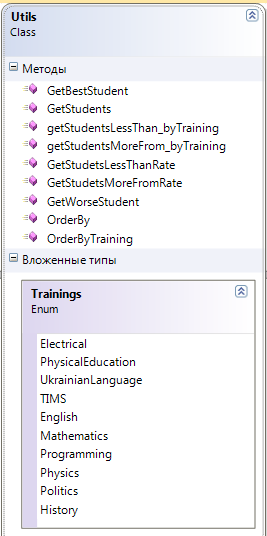
Клас Utils (Рис. 3.6) містить всі методи якими можна впливати на масив студентів. А також виконує основне завдання програми, обробляє дані.

Рис. 3.6 клас Utils

Клас Utils представлений такими методати як: GetBestStudent – виводить кращого студента (по середньому балу)

Student Utils::GetBestStudent(vector<Student> \_dbStudent)

{

double minMiddleRate = 0;

int index = -1;

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

15

ТНТУ КРКН 15.006.084.0006 ПЗ

for (int i = 0; i < \_dbStudent.size(); i++)

if (\_dbStudent[i].GetMiddleRate() > minMiddleRate)

{

minMiddleRate = \_dbStudent[i].GetMiddleRate();

index = i;

}

return \_dbStudent[index];

}

GetWorsesStudent - виводить гіршого студента (по середньому балу)

Student Utils::GetWorseStudent(vector<Student> \_dbStudent)

{

double maxMiddleRate = 100;

int index = -1;

for (int i = 0; i < \_dbStudent.size(); i++)

if (\_dbStudent[i].GetMiddleRate() < maxMiddleRate)

{

maxMiddleRate = \_dbStudent[i].GetMiddleRate();

index = i;

}

return \_dbStudent[index];

}

OrderBy – сортує студентів від меншого до гіршого і навпаки

vector<Student> Utils::OrderBy(vector<Student> \_dbStudent, bool reverse)

{

for (int i = 0; i < \_dbStudent.size() - 1; i++)

for (int j = i + 1; j < \_dbStudent.size(); j++)

if (reverse)

{

OrderByTraining – сортує студентів від гіршого до кращого по вказаному предмету

vector<Student> Utils::OrderByTraining(vector<Student> \_dbStudent, Trainings trainingIndex, bool reverse)

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

16

ТНТУ КРКН 15.006.084.0006 ПЗ

{

for (int i = 0; i < \_dbStudent.size() - 1; i++)

for (int j = i + 1; j < \_dbStudent.size(); j++)

if (reverse)

{

GetStudents – повертає студентів які здали або нездали сесію

vector<Student> Utils::GetStudents(vector<Student> \_dbStudent, double minMiddleRate, bool sessionComplete)

{

return (sessionComplete) ?

GetStudetsMoreFromRate(\_dbStudent, minMiddleRate) :

GetStudetsLessThanRate(\_dbStudent, minMiddleRate);

}

GetStudentsMoreFromRate – вертає студентів, чий середній рейтинг більший-рівний вказаному

vector<Student> Utils::GetStudetsMoreFromRate(vector<Student> \_dbStudent, double middleRate)

{

vector<Student>resDB\_students = vector<Student>();

for (int i = 0; i < \_dbStudent.size(); i++)

if (\_dbStudent[i].GetMiddleRate() >= middleRate)

resDB\_students.push\_back(\_dbStudent[i]);

return resDB\_students;

}

GetStudentsLessThan – вертає студентів, чий середній рейтинг менший ніж вказаний

vector<Student> Utils::GetStudetsLessThanRate(vector<Student> \_dbStudents, double middleRate)

{

vector<Student>resDB\_students = vector<Student>();

for (int i = 0; i < \_dbStudents.size(); i++)

if (\_dbStudents[i].GetMiddleRate() < middleRate)

resDB\_students.push\_back(\_dbStudents[i]);

return resDB\_students;

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

17

ТНТУ КРКН 15.006.084.0006 ПЗ

}

GetStudentsMoreFromRatebyTraining – повертає студентів, чий середній рейтинг більший-рівний вказаному рейтингу по заданому предмету

vector<Student> Utils::getStudentsMoreFrom\_byTraining(vector<Student> \_dbStudent, int rate, Trainings indexOfTraining)

{

vector<Student>resDB\_students = vector<Student>();

for (int i = 0; i < \_dbStudent.size(); i++)

if (\_dbStudent[i].GetRate()[(int)indexOfTraining] >= rate)

resDB\_students.push\_back(\_dbStudent[i]);

return resDB\_students;

GetStudentsMoreLessThanbyTraining – повертає студентів, чий середній рейтинг менший за вказаний рейтинг по заданому предету

vector<Student> Utils::getStudentsLessThan\_byTraining(vector<Student> \_dbStudents, int rate, Trainings indexOfTraining)

{

vector<Student>resDB\_students = vector<Student>();

for (int i = 0; i < \_dbStudents.size(); i++)

if (\_dbStudents[i].GetRate()[(int)indexOfTraining] < rate)

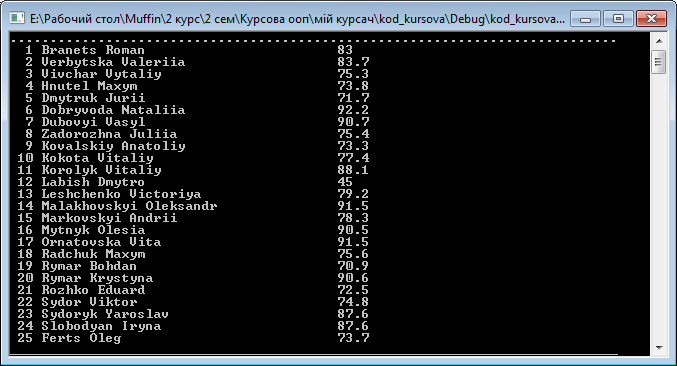
resDB\_students.push\_back(\_dbStudents[i]);

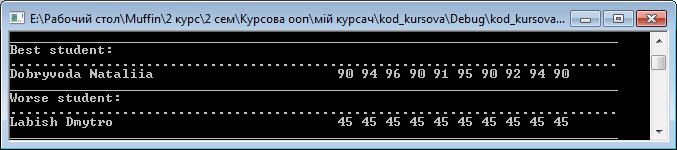
return resDB\_students;

}

# ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ

При запуску програми відображається вікно на Рис. 4.1.

Рис. 4.1 Виведення середнього балу студента

Рис. 4.2 Виведення кращого та гіршого студента по балах

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

18

ТНТУ КРКН 15.006.084.006 ПЗ

Розроб.

*Добривода Н.І.*

Перевір.

*Бревус В.М.*

Реценз.

Н. Контр.

Затверд.

Розділ 4

Літ.

Аркушів

5

ТНТУ ФІС гр. СН-21

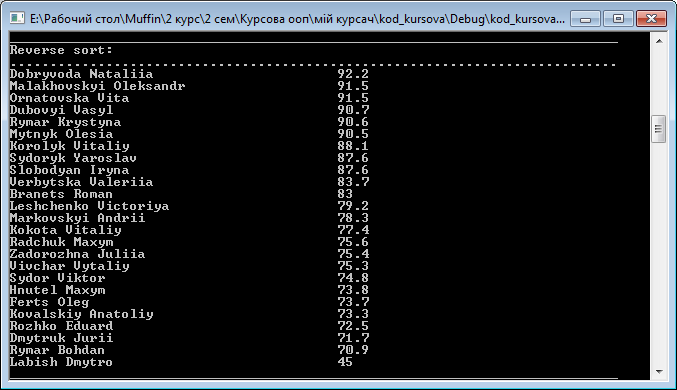


Рис. 4.3 Сортування студентів по середньому

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

19

ТНТУ КРКН 15.006.084.0006 ПЗ

балу від кращого до гіршого

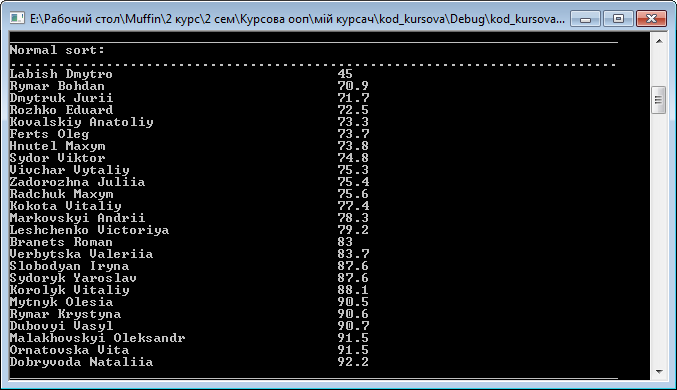


Рис. 4.4 Сортування студентів по середньому

балу від гіршого до кращого

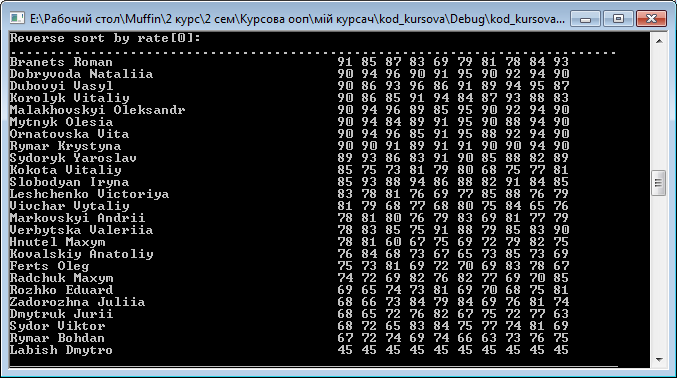


Рис. 4.5 Сортування студентів по балах з вказаного предмета

від кращого до гіршого

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

20

ТНТУ КРКН 15.006.084.0006 ПЗ

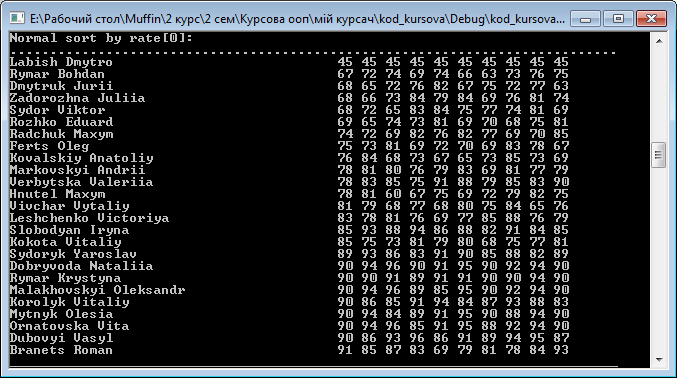


Рис. 4.6 Сортування студентів по балах з вказаного предмета

від гіршого до кращого

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

21

ТНТУ КРКН 15.006.084.0006 ПЗ

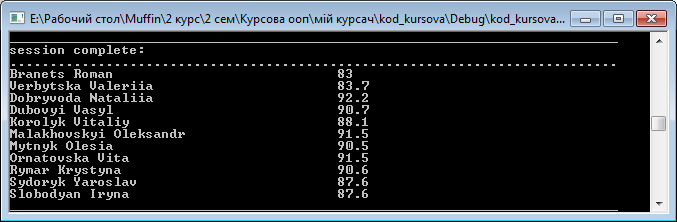


Рис. 4.7 Студенти які здали сесію

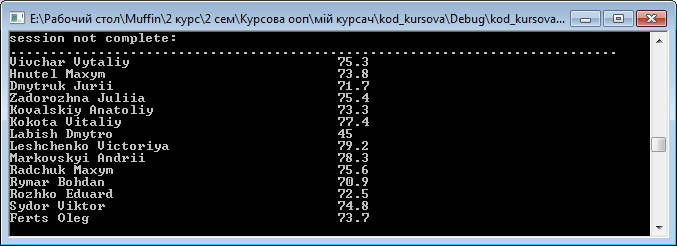


Рис. 4.8 Студенти котрі не здали сесії

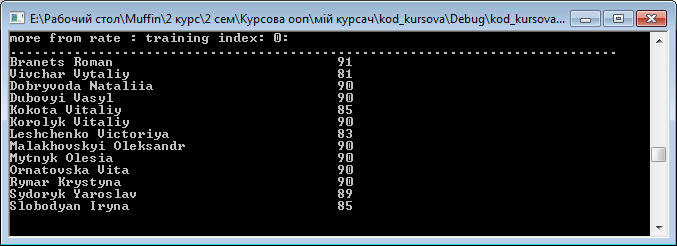


Рис. 4.9 Студенти в яких бал вищий вказаного по вказаному предмету

Змн.

Арк.

№ докум.

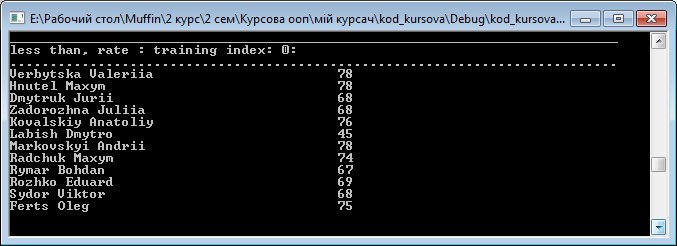
Підпис

Дата

Арк.

22

ТНТУ КРКН 15.006.084.0006 ПЗ



*Рис. 4.10 Студенти в яких бал нижчий вказаного по вказаному предмету*

# ВИСНОВОК

В результаті виконання роботи було розроблено програму для обробки результату сесії студентів ІІ-го курсу, вона створена з використанням компілятора Microsoft Visual Studio 2010, на мові програмування С++.

Дана програма надає користувачеві можливість легко і зручно обробляти і зберігати дані про успішність студентів. Завдяки інтуїтивно зрозумілому інтерфейсу будь-який користувач не зіткнеться з жодними проблемами при її використанні. у якій користувач створити власний розклад або відкрити вже створений раніше. Було використано мову програмування С++.

Організовано роботу з файлами: створення файлу, зчитування файлу, зміни файлу та вивід даних на екран.

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

23

ТНТУ КРКН 15.006.084.0006 ПЗ

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

24

ТНТУ КРКН 15.006.084.0006 ПЗ

1. Орлов С. А. Технологии разработки программного обеспечения: Учебник. - СПб.: Питер, 2002. – 464 с. ISBN: 5-94723-145-X(рус.).
2. Прата С. Язык программирования С++. Лекции и упражнения,  
   6-е изд. / Стивен Прата : Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2012. – 1248 с. ISBN 978-5-8459-1778-2 (рус.).
3. Страуструп Б. Язык программирования С++: Специальное издание. / Бьерн Страуструп. Пер. с англ. – М.: Издательство Бином, 2011. – 1136 с. ISBN 978-5-7989-0425-9 (рус.).
4. Дейтел Х.М. Как программировать на С++: 5-е издание. /  
    Х.М. Дейтел, П. Дж. Дейтел : Пер. с англ. – М.: ООО «Бином-Пресс», 2008. – 1456 с. ISBN 978-5-9518-0224-8 (рус.)

# ДОДАТОК А

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

25

ТНТУ КРКН 15.006.084.0006 ПЗ

Git — розподілена система керування версіями файлів та спільної роботи. Проект створив Лінус Торвальдс для управління розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано (англ. Junio C. Hamano). Система спроектована як набір програм, спеціально розроблених з врахуванням їхнього використання у скриптах. Це дозволяє зручно створювати спеціалізовані системи управління версіями на базі Git або користувацькі інтерфейси. Тобто ґіт – це репозиторій коду (програмний інструмент), основне завдання якого – зберігання коду та історії змін до даного коду. Також його ще називають Системою Контролю Версій (VCS). Маються на увазі версії коду.

Репозиторій коду дозволяє:

* зберігати ваш код;
* запам’ятовувати історію змін до вашого коду (та дозволяти у будь-який момент побачити хто саме зробив зміни, коли зробив зміни);
* відкотити до будь-якої версії коду у будь-який момент;
* з легкістю об’єднувати зміни різних версій, станів та розробників;
* розробляти проект командою розробників, одночасно працюючи над одними і тими ж модулями і навіть стрічками коду;
* вести статистику змін до коду.

Ось як виглядають усі етапи вашого коду від вашої папочки локальної і аж до кінцевого місця зберігання коду, яким користуються усі розробники у вашій команді:



Рис. А.1 Чотири етапи вашого коду: Workspace-, Staging (Index)-,

Local-, and Remote- репозиторії

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

26

ТНТУ КРКН 15.006.084.0006 ПЗ

Тобто ви працюєте у своїй Робочій Папці (workspace), потім закидаєте ваші зміни у так званий Перевалочний Пункт (index або staging), звідки кладете код у ваш Локальний Репозиторій (local repository), і вже при потребі, у Віддалені Репозиторії (remote repository), щоб інші також мали доступ до ваших змін.

Тому локальний репозиторій складається із трьох так званих дерев (tree):

* робоча папка (working space) – там ви працюєте,
* індекс (staging, index) – перевалочний пункт між робочою папкою та локальним репозиторієм,
* head – вказівник на ваш останній коміт в локальному репозиторії.

Робоча Папка (Working Directory) – це ваша локальна папка у файловій системі, яка відображає певний стан коду в репозиторії + ваші локальні зміни (якщо такі є). Ви весь час працюєте над кодом будучи всередині вашої робочої папки.

Коміт (commit) – запис змін коду у репозиторій.

Мердж (merge) – об’єднання змін з однієї гілки коду в іншу.

Гілка (branch) – версія коду, яку створють всередині того ж репозиторію, і ведуть розробку зазвичай до моменту поки не зроблять мердж (об’єднання) змін даної версії коду із основною гілкою (зазвичай master). Гілки використовують для роботи над окремими завданнями одночасно, не заважаючи один одному. Головна гілка у будь-якому git репозиторії – це master.

Пул (pull) – отримання останніх змін з віддаленого репозиторію у ваш власний локальний репозиторій.

Push (push) – закинути зміни локального репозиторію у віддалений репозиторій.

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

27

ТНТУ КРКН 15.006.084.0006 ПЗ

Ремоут (remote) – один із віддалених репозиторіїв. Один код може мати багато клонів та копій на вашому та зовнішніх серверах. Кожна така копія, окрім вашої локальної, називається ремоутом (віддаленим репозиторієм).

Індекс (Stage або Index) – підготовче місце ваших файлів з робочої папки перед тим як закинути зміни робочої папки у локальний репозиторій. Зазвичай коміт включає у себе лише ті зміни, що є в індексі. Давайте для простоти називати дане місце Перевалочним Пунктом, кращого терміну поки не придумав.

Локальний репозиторій – ваша власна копія репозиторію, у яку безпосередньо йдуть коміти з вашої робочої папочки (насправді все йде через Перевалочний Пункт – Stage (Index)). Для роботи з даним репозиторієм вам не потрібно інтернету.

Git HEAD – вказівник на коміт в репозиторії, на якому зараз перебуває ваш локальний репозиторій і гілка у цьому репозиторії. Вказівник по дефолтну вказує на останній коміт, але його можна вручну “переміщати” між комітами, гілками і навіть тегами (у цьому випадку його називають Detached Head – від’єднаний вказівник).

Тег (tag) – фіксація стану коду для подальшого використання його при релізах. Фіксуємо зазвичай код під тегом з певною назвою.

# ДОДАТОК Б

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

28

ТНТУ КРКН 15.006.084.0006 ПЗ

Обробка виняткових ситуацій (exception handling) дозволяє правильно реагувати на помилки, що виникають в ході виконання програми. Використовуючи механізм виняткових ситуацій, програма може автоматично викликати процедуру обробки помилок. Це позбавляє програміста від утомливого кодування.

Механізм обробки виняткових ситуацій в мові C++ заснований на трьох ключових словах: try, catch і throw. Фрагменти програми, що підлягають контролю, містять блок try. Якщо в ході виконання програми в блоці try виникає виняткова ситуація (тобто помилка), вона генерується за допомогою ключового слова throw. Потім виняткова ситуація перехоплюється блоком catch і обробляється.

Тобто код, що підлягає контролю, повинен виконуватися усередині блоку try. (Функції, що викликаються усередині блоку try, також можуть генерувати виняткові ситуації.) Виняткові ситуації перехоплюються оператором catch, який слідує безпосередньо за блоком try, в якому вони виникли. Розмір блоку try може варіюватися. Він може містити як декілька операторів, так і цілу програму (в цьому випадку функція main() цілком поміщається в блок try).

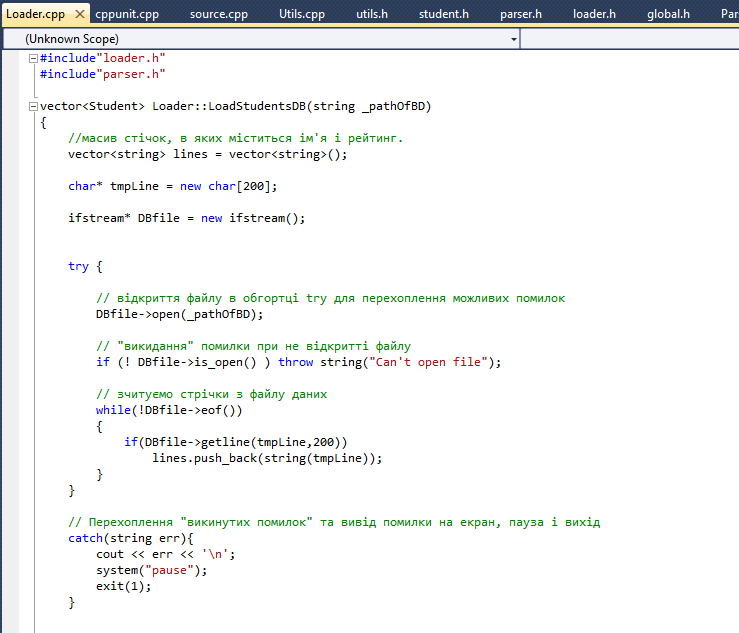
З одним блоком try може бути зв'язане декілька операторів catch. Вибір правильного оператора catch визначається типом виняткової ситуації. З декількох варіантів вибирається оператор catch, тип аргументу якого співпадає з виниклою винятковою ситуацією (решта варіантів ігнорується). В процесі перехоплення виняткової ситуації аргументу привласнюється якесь значення. Аргумент може бути об'єктом вбудованого типу або класу. Якщо фрагмент не генерує ніяких виняткових ситуацій (тобто в блоці try помилки не виникають),

не виконується жоден оператор catch.

Оператор throw генерує вказану виняткову ситуацію. Якщо в програмі передбачено її перехоплення, оператор throw повинен виконуватися або усередині блоку try, або усередині функції, явно або що неявно викликається усередині блоку try. Якщо генерується виняткова ситуація, для якої не передбачена обробка, програма може припинити своє виконання. В цьому випадку викликається стандартна функція terminate (), яка за умовчанням викликає функцію abort (). Проте, як буде показано нижче, програміст може

передбачити власну обробку помилки.

Важливо чітко розуміти, що код, пов'язаний з оператором catch, виконується тільки при перехопленні виняткової ситуації. Інакше оператор catch просто ігнорується. (Інакше кажучи, потік управління ніколи не проходить через тіло оператора catch.) Наприклад, в наступній програмі виняткові ситуації взагалі не генеруються, і оператор catch не виконується.



Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

29

ТНТУ КРКН 15.006.084.0006 ПЗ

Рис. Б.1 Приклад оформлення обробки виняткових ситуацій на програмі для обробки результатів сесії студентів

# ДОДАТОК В

Модульне тестування (Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. В об'єктно-орієнтованому програмуванні модулем вважають інтерфейс, клас.

Зазвичай unit-тести застосовують для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

Метою модульного тестування є ізоляція кожної частини програми та впевненість у тому, що кожна окрема частина є коректною. Модульний тест забезпечує жорсткий «контракт», за яким має працювати тестований код. Як результат, це надає деякі переваги. Модульне тестування допомагає знайти помилки раніше в циклі розробки ПЗ, що робить розробку дешевшою та швидшою.

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

30

ТНТУ КРКН 15.006.084.0006 ПЗ

Unit-тест перевіряє критичні характеристики модулю. Відповідність чи невідповідність цим характеристикам демонструє коректність модуля. Модульний тест «документує» ці критичні характеристики, але не треба покладатися лише на код в документуванні ПЗ під час розробки.

Слід відзначити, що звичайна письмова документація дуже повільно реагує на зміни в коді, тоді як модульні тести завжди відображають поточний стан модуля.

Фреймворки для модульного тестування зазвичай не розповсюджуються в комплекті з компіляторами — це програмне забезпечення пишуть окремо. Вони розроблені для багатьох мов програмування й допомагають спростити процес тестування. Серед відомих фреймворків є проекти open source, наприклад ті, що зазвичай називають xUnit, та комерційні рішення, такі як TBrun, Testwell CTA++ and VectorCAST/C++.

Також можливо виконувати модульне тестування без додаткових бібліотек, створюючи код, що застосовує механізми assertion, виняткових ситуацій англ. exception, та інші механізми контролю виконання програми, які можуть сигналізувати про неуспіх.

Змн.

Арк.

№ докум.

Підпис

Дата

Арк.

31

ТНТУ КРКН 15.006.084.0006 ПЗ

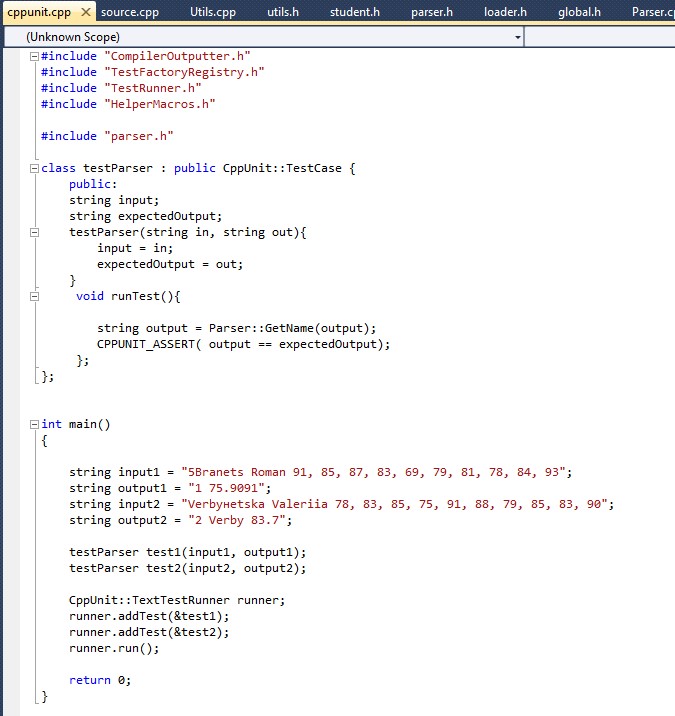


Рис. В.1 Приклад оформлення unit-тесту на програмі для обробки результатів сесії студентів