**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**Факультет прикладної математики**

**Кафедра системного програмування і спеціалізованих комп'ютерних систем**

**МЕТОДИЧНІ ВКАЗІВКИ ДО ЛАБОРАТОРНИХ РОБІТ**

**за курсом**

**“Об'єктно-орієнтоване програмування”**

**частина II.**

**Київ 2020**ЗМІСТ

[ЛАБОРАТОРНА РОБОТА № 1.](#Lab1)

[ЛАБОРАТОРНА РОБОТА № 2.](#Lab2)

[ЛАБОРАТОРНА РОБОТА № 3.](#Lab3)

WARNING: Start early: these labs require quite some work. Do not discuss solution approaches with others or share code. All code must be written by yourself. Do not use components from the web or from your friends. Explanations such as “Я розібрався” or “Мені допомагали” ARE NOT accepted. Разом з текстом роботи надсилати номер залікової книжки.

ЛАБОРАТОРНА РОБОТА №1.

Теоретичні відомості можна знайти в лекціях №№ 11 – 18.

**Завдання на роботу.**

Варіант 1.

*Сконструювати клас List з наступним інтерфейсом:*

List() - Конструктор за замовчуванням. Створює порожній контейнер.

List( size\_t \_Count, int \_Val) - Створює контейнер з count елементами зі значенням value.

List( const \_List& other) - Конструктор копіювання. Створює контейнер з копією вмісту other.

~List() – деструктор.

node & back() - Повертає посилання на останній елемент в контейнері.

const node & back() const - Повертає константне посилання на останній елемент в контейнері.

node & begin() - Повертає посилання на перший елемент в контейнері.

const node & begin() const - Повертає константне посилання на перший елемент в контейнері.

void clear() - Видаляє всі елементи з контейнера.

bool empty() const – Перевірка, чи є контейнер порожнім.

void erase( size\_t pos ) - Видаляє елемент в позиції pos.

void erase( size\_t first, size\_t last )- Видаляє елементи в діапазоні [first; last].

void insert(int value, unsigned pos, unsigned count)- Вставляє count копій value перед елементом, на який вказує pos.

void insert( size\_t pos, int \*v, size\_t count )- Вставляє count елементів з v перед елементом, на який вказує pos.

void merge( List& other )- Злиття двох відсортованих списків в один. Списки повинні бути відсортовані в порядку зростання. Елементи не дублюються. Контейнер other стає порожнім після операції.

void pop\_back() - Видаляє останній елемент контейнера.

void pop\_front() - Видаляє перший елемент контейнера.

void push\_back( int value )- Додає даний елемент value до кінця контейнера.

void push\_front( int value )- Додає даний елемент value до початку контейнера.

void erase( size\_t pos ) - Видаляє елемент в позиції pos.

void erase( size\_t first, size\_t last ) - Видаляє елементи в діапазоні [first; last].

void remove( int value )- видаляє всі елементи, рівні value.

void resize( size\_t count, int value);  
void resize( size\_t count ) - Змінює розмір контейнера, щоб утримувати count елементи. Якщо поточний розмір менше, ніж count, додаткові елементи додаються і ініціалізуються value. Якщо поточний розмір більше count, контейнер зводиться до його перших count елементів.

void reverse() - Змінює порядок елементів в контейнері.

size\_t size() const - Повертає кількість елементів у контейнері.

void sort() - Сортування елементів в порядку зростання.

void splice(size\_t pos, List& other) - Переміщення елементів з списку other в інший. Елементи вставляються перед елементом, на який вказує pos.

void swap( list& other )= Обмінюється вміст контейнера з іншим.

void unique() - Видаляє послідовні повторювані елементи з контейнера.

bool operator==(const List &) – перевіряє контейнери на рівність. [1,2,3] і [2,1,3] вважаються рівними.

List & operator=(const List &) – перавантажена операція присвоєння.

operator int() const – операция перетворення user defined type в built in type. Повертає суму елементів контейнеру.

void print()const;

Варіант 2.

Сконструювати клас Vector з наступним інтерфейсом:

explicit Vector( ) - Конструктор за замовчуванням. Створює порожній контейнер.

explicit Vector(size\_t sz) - Створює контейнер розміру sz.

Vector(size\_t sz, int val) - Створює контейнер розміру sz з елементами val.

Vector( const Vector& other ) - Конструктор копіювання. Створює контейнер з копією вмісту other.

Vector(const int \*v, size\_t vsz, size\_t sz) - Створює контейнер ємності sz з елементами з вектору v розміру vsz.

~Vector() – деструктор.

void assign( size\_t count, int value ) - Стирає вектор і копіює count елементів value  в порожній вектор.

void assign(const Vector &other) - Стирає вектор і копіює в нього елементи з other.

int&      at( size\_t pos )- Повертає посилання на елемент за індексом pos. Виконується перевірка виходу індексу за межі діапазону вектора. У разі виходу індексу за межі діапазону вектора генерує виняток типу std :: out\_of\_range.

int & front() - Повертає посилання на перший елемент.

const int & const\_front() - Повертає константне посилання на перший елемент.

int & back() - Повертає посилання на останній елемент.

const int & const\_back() - Повертає константне посилання на останній елемент.

size\_t capacity() - Повертає кількість елементів, які можуть розміщуватися у векторі без reallocation.

size\_t size() - Повертає кількість елементів у векторі.

void clear() – видаляє елементи контейнеру.

void eraze(size\_t pos) - видаляє елемент контейнеру в pos .

void eraze(size\_t first, size\_t last) - видаляє елементи контейнеру в діапазоні [first, last].

void sort() - сортує елементи контейнеру в порядку невбування.

size\_t unique() - видалення послідовно розташованих повторюваних елементів, , наприклад

    int vi [] = {1,2,3,1,2,3,3,4,5,4,5,6,7};

    Vector v (vi, 13);

v.sort (); // 1 1 2 2 3 3 3 4 4 5 5 6 7

auto last = v.unique ();

v зараз містить {1 2 3 4 5 6 7 x x x x x x},

last == 6;

де 'x' позначає невизначений елемент

bool empty() const – Перевірка, чи є контейнер порожнім.

void insert(size\_t \_Where, int \_Val) – вставляє \_Val в \_Where.

void insert(size\_t \_Where, size\_t \_Count, int \_Val) - вставляє \_Count елементів із значенням \_Val починаючи з \_Where.

int & at(size\_t pos) - Операція доступу до елементу вектора за індексом. Повертає посилання на елемент за індексом pos. Виконується перевірка виходу індексу за межі діапазону вектора. У разі виходу індексу за межі діапазону вектора генерує виняток типу std :: out\_of\_range.

void reverse() - Змінює порядок елементів в контейнері.

void swap( list& other ) - Обмінюється вміст контейнера з іншим.

void reserve( size\_t size ) - Задає ємність контейнера до size. Нова пам'ять виділяється при необхідності.

void resize( size\_t count, int value);  
void resize( size\_t count ) - Змінює розмір контейнера, щоб утримувати count елементи. Якщо поточний розмір менше, ніж count, додаткові елементи додаються і ініціалізуються value. Якщо поточний розмір більше count, контейнер зводиться до його перших count елементів.

operator int() const – операция перетворення user defined type в built in type. Повертає суму елементів контейнеру.

bool operator==(const Vector &) – перевіряє контейнери на рівність. [1,2,3] і [2,1,3] вважаються рівними.

Vector & operator=(const Vector &) – перавантажена операція присвоєння.

void pop\_back() - видалення елемента з кінця контейнера.

void push\_back(int) - вставка елемента в кінець контейнера.

void print()const;

Варіант 3.

*Сконструювати клас Deque з наступним інтерфейсом:*

Deque() - Конструктор за замовчуванням. Створює порожній контейнер.

Deque(size\_t \_Count, int \_Val ) - Створює контейнер розміру \_Count з елементами \_Val .

Deque(const Deque&) – Конструктор копіювання.

Deque(const int \*v, size\_t vsz, size\_t sz) - Створює контейнер розміру sz з елементами з вектору v розміру vsz.

~Deque() – деструктор

Deque & operator=(const Deque &) - перавантажена операція присвоєння.

int & at(size\_t pos) - Операція доступу до елементу вектора за індексом. Повертає посилання на елемент за індексом pos. Виконується перевірка виходу індексу за межі діапазону вектора. У разі виходу індексу за межі діапазону вектора генерує виняток типу std :: out\_of\_range.

void pop\_front() – видалення елемента з початку контейнера.

void push\_front(int) - вставка елемента в початок контейнера.

void pop\_back() - видалення елемента з кінця контейнера.

void push\_back(int) - вставка елемента в кінець контейнера.

int & front() - Повертає посилання на перший елемент.

const int & const\_front() - Повертає константне посилання на перший елемент.

int & back() - Повертає посилання на останній елемент.

const int & const\_back() - Повертає константне посилання на останній елемент.

size\_t capacity() - Повертає кількість елементів, які можуть розміщуватия у векторі без reallocation.

size\_t size() - Повертає кількість елементів у векторі.

void clear() – видаляє елементи контейнеру.

void eraze(size\_t pos) - видаляє елемент контейнеру в pos .

void eraze(size\_t first, size\_t last) - видаляє елементи контейнеру в діапазоні [first, last].

void sort() - сортує елементи контейнеру в порядку невбування.

size\_t unique() - видалення послідовно розташованих повторюваних елементів, наприклад

    int vi [] = {1,2,3,1,2,3,3,4,5,4,5,6,7};

    Deque v (vi, 13);

v.sort (); // 1 1 2 2 3 3 3 4 4 5 5 6 7

auto last = v.unique ();

v зараз містить {1 2 3 4 5 6 7 x x x x x x},

last == 6;

де 'x' позначає невизначений елемент

void reverse() - Змінює порядок елементів в контейнері.

void swap( Deque& other )= Обмінюється вміст контейнера з іншим.

void reserve( size\_t size ) - Задає ємність контейнера size. Нова пам'ять виділяється при необхідності.

void assign( size\_t count, int value ) - Стирає чергу і копіює count елементів value  в порожню чергу.

void assign(const  Deque &other) - Стирає чергу і копіює в неї елементи з other.

operator int() const – операция перетворення user defined type в built in type. Повертає суму елементів контейнеру.

bool operator==(const Deque &) – перевіряє контейнери на рівність. [1,2,3] і [2,1,3] вважаються рівними.

void print()const;

Варіант 4.

*Сконструювати клас String з наступним інтерфейсом:*

String() - Конструктор за замовчуванням. Створює порожній контейнер.

String( size\_t count, char ch ) - Створює контейнер розміру count з елементами ch.

String( const String& other, size\_t pos, size\_t count = npos) - Створює рядок з підрядка діапазону [pos, pos + count) параметра other. Якщо запитуваний підрядок виходить за межі кінця рядка або якщо count == npos, діапазон повертається підрядка буде [pos, size ()). Якщо pos > = other.size (), буде згенеровано виняток std :: out\_of\_range.

npos - значення статичного члена. Це значення, коли використовується як значення параметра len (або sublen) у функціях-членах рядка, означає "до кінця рядка". Як повернене значення зазвичай використовується для позначення не збігів. Будемо вважати npos =1024.

String( const char\* s) - Створює рядок з символів рядка, на який вказує s

String( const String &) – Коструктор копіювання.

size\_t size() const - Повертає кількість символів в рядку.

String& operator=( const String& str ) - Замінює вміст копією str.

String& operator=( const char\* s ) - Замінює вміст цього рядка вмістом рядка, на яку вказує s.

String& operator=( char ch ) - Замінює вміст символом ch.

String& operator+=( const String& str ) - Додає рядок str.

String& operator+=( char\* s ) - Додає рядок із завершальним нулем, на яку вказує s.

void resize( size\_t count ) - Змінює розмір рядка, щоб він міг містити count символів.

void resize( size\_t count, char ch ) - Змінює розмір рядка, , щоб він міг містити count символів.

Якщо поточний розмір менше, ніж count, додаткові символи будуть додані за допомогою Char () – функція генерує довільний символ Якщо поточний розмір більше, ніж count, рядок скорочується до перших count символів.

void swap( String& other ) - Обмінює вміст рядка з вмістом other.

friend bool operator==( String & lhs, String & rhs ) - Перевіряє, чи є вміст lhs і rhs однаковим, тобто

lhs.size () == rhs.size () і кожен символ в lhs має ідентичний символ в rhs в тій же позиції.

size\_t find( const String& str, size\_type pos = 0 ) const - Знаходить перший підрядок, рівний переданій послідовності символів. Пошук починається з позиції pos.

size\_t find\_first\_of( const сhar\* s, size\_t pos, size\_t count ) const - Знаходить перший символ рядка, рівний одному з count перших символів рядка в переданій послідовності символів s. Пошук починається з позиції pos, тобто знайдений символ не може перебувати в позиції, що передує pos.

size\_t find\_first\_not\_of( const char\* s, size\_t pos, size\_t count ) const - Знаходить перший символ, що не рівний жодному з перших count символів рядка, на яку вказує s. Пошук починається з позиції pos, тобто знайдений символ не може перебувати в позиції, що передує pos.

size\_t find\_last\_not\_of( const char\* s, size\_t pos, size\_t count ) const - Знаходить останній символ, що не рівний жодному з перших count символів рядка, на яку вказує s. Пошук закінчується на pos, тобто в пошуку бере участь лише підрядок в діапазоні [0, pos).

size\_t rfind( const String & str, size\_t pos = npos ) const - Знаходить останній підрядок, рівний переданій символьної послідовності. Пошук починається з позиції pos, тобто в пошуку бере участь лише підрядок в діапазоні [pos, size). Якщо npos передано в якості позиції pos npos, пошук буде проведений по всьому рядку.

String & append( const String & str, size\_t pos, size\_t count ) - Додає підрядок [pos, pos + count) з str. Якщо запитаний підрядок виходить за межі кінця рядка, або якщо count == npos, діапазоном підрядка, що додається, буде [pos, size ()). Якщо pos> = str.size (), буде згенеровано std :: out\_of\_range

void print()const;

Результати роботи оформити у вигляді проекту С++-програми, що включає наступні файли:

1. Файли оголошення класів.

2. Файли визначення класів.

3. Файл тест-драйву.

Імена файлів і проектів повинні бути змістовними.

Номер варіанта визначає номер ЗАЛІКОВОЇ КНИЖКИ за модулем 4:

00 – варіант 1

01 – варіант 2

10 – варіант 3

11 – варіант 4

ЛАБОРАТОРНА РОБОТА №2.

Просте успадкування.

Мета роботи – оволодіти способами проектування й використання поліморфних класів, способами приведення типів і використання параметризованих контейнерів стандартної бібліотеки С++.

Теоретичні відомості (Див. Лекції №№ 15-17).

Загальна схема для всіх варіантів. Є 3 класи пов’язані простим успадкуванням.

Базовий абстрактний клас

//An abstract class

class CAbstract {

public:

CAbstract() {};

CAbstract(string \_prop1) : prop1(\_prop1) { }

virtual void Process() = 0;

protected:

string prop1;// name

};

і 2 похідних

class CDerived1 : public CAbstract {

public:

CDerived1() {};

CDerived1(string \_prop1, int \_prop2) :

CAbstract(\_prop1), prop2(\_prop2) {};

CDerived1(const CDerived1 &e);

CDerived1& operator=(const CDerived1 &e);

void SetProp2(int n) { prop2 = n; };

int GetProp2() { return prop2; };

virtual void Process() { printf("name = %s val = %d\n", prop1.c\_str(), prop2); };

//Add here whatever you need

protected:

int prop2;//value

};

class CDerived2 : public CAbstract {

public:

CDerived2() {};

CDerived2(string \_prop1, int \_prop3) :

CAbstract(\_prop1), prop3(\_prop3) {};

CDerived2(const CDerived1 &e) {};

CDerived2& operator=(const CDerived2 &e);

void SetProp3(int n) { prop3 = n; };

int GetProp3() { return prop3; };

virtual void Process() { printf("name = %s number = %d\n", prop1.c\_str(), prop3); };

//Add here whatever you need

protected:

int prop3;//number of

};

Екземпляри цих класів зберігаються в контейнері

class Container {

public:

Container() {};

//display all container elementa

void Display();

//add element to container

void AddElement(CAbstract \*p) ;

private:

vector<CAbstract \*> elements;//elements list

};

Таким чином, об’єкти представлені вектором vector< CAbstract \*> elements, тобто покажчиками на екземпляр абстрактного класу. При обробці списку потрібно мати можливість розрізняти тип об'єкта (CDerived1 або CDerived2), адресу якого містить покажчик типу CAbstract \*p. Зробити це можна, наприклад, з використанням операції typeid.

if (typeid(\*p) == typeid(CDerived1))// CDerived1

if (typeid(\*p) == typeid(CDerived2))// CDerived2

Використання (потребує #include <typeinfo>)

typeid( type-id )

typeid( expression )

The **typeid** operator allows the type of an object to be determined at run time.

The result of **typeid** is a **const** **type\_info&**. The value is a reference to a **type\_info** object that represents either the *type-id* or the type of the *expression*, depending on which form of **typeid** is used. See [type\_info Class](https://msdn.microsoft.com/ru-ru/library/70ky2y6k.aspx) for more information.

The **typeid** operator does not work with managed types (abstract declarators or instances), see [typeid (C++ Component Extensions)](https://msdn.microsoft.com/ru-ru/library/kwd9abya.aspx) for information on getting the Type of a specified type.

The **typeid** operator does a run-time check when applied to an l-value of a polymorphic class type, where the true type of the object cannot be determined by the static information provided. Such cases are:

* A reference to a class
* A pointer, dereferenced with \*
* A subscripted pointer (i.e. [ ]). (Note that it is generally not safe to use a subscript with a pointer to a polymorphic type.)

If the *expression* points to a base class type, yet the object is actually of a type derived from that base class, a **type\_info** reference for the derived class is the result. The *expression* must point to a polymorphic type (a class with virtual functions). Otherwise, the result is the**type\_info** for the static class referred to in the *expression*. Further, the pointer must be dereferenced so that the object it points to is used. Without dereferencing the pointer, the result will be the **type\_info** for the pointer, not what it points to.

Тут операція dynamic\_cast використовується для приведення типу Person\* до типу Manager\*. Це так званий *down cast*, тобто приведення покажчика на базовий клас до покажчика на похідний клас.

Використання

dynamic\_cast < type-id > ( expression )

The *type-id* must be a pointer or a reference to a previously defined class type or a "pointer to void". The type of *expression* must be a pointer if *type-id* is a pointer, or an l-value if *type-id* is a reference.

See [static\_cast](https://msdn.microsoft.com/ru-ru/library/c36yw7x9.aspx) for an explanation of the difference between static and dynamic casting conversions, and when it is appropriate to use each.

There are two breaking changes in the behavior of **dynamic\_cast** in managed code:

* **dynamic\_cast** to a pointer to the underlying type of a boxed enum will fail at runtime, returning 0 instead of the converted pointer.
* **dynamic\_cast** will no longer throw an exception when *type-id* is an interior pointer to a value type, with the cast failing at runtime. The cast will now return the 0 pointer value instead of throwing.

If *type-id* is a pointer to an unambiguous accessible direct or indirect base class of *expression*, a pointer to the unique subobject of type *type-id* is the result.

Таким чином, в функції Display() повинно бути

if (typeid(\*\*it) == typeid(CDerived1) ) {

dynamic\_cast<CDerived1\*>(\*it)->Process();

},

Де

vector<CAbstract\*>::iterator it = elements.begin();

Варіант 1.

Написати додаток Database, яке зберігає відомості про співробітників компанії. У додатку використовується наступна ієрархія поліморфних класів.

Базовим є абстрактний клас

//An abstract class

class Person{

public:

Person(){};

Person(string \_f\_name, string \_l\_name, int \_age) : age(\_age),

f\_name(\_f\_name), l\_name(\_l\_name){ }

virtual void Display(bool) = 0;

protected:

string f\_name;//first name

string l\_name;//last name

int age;

};

Відомості про рядових співробітників представлені класом

class Employee : public Person{

public:

Employee(){};

Employee(string \_f\_name, string \_l\_name, int \_age, int \_id) :

Person(\_f\_name, \_l\_name, \_age), id(\_id){};

Employee(const Employee &e);

Employee& operator=(const Employee &e);

void SetSalary(int s);

void SetDepartment(string dept);

void SetId(int n);

int GetId();

string GetDepartment();

virtual void Display();

//Add here whatever you need

protected:

string department;

int salary;

int id;

};

Відомості про менеджерів представлені класом

class Manager : public Employee {

public:

Manager(){};

Manager(string \_f\_name, string \_l\_name, int \_age, int \_id) :

Employee(\_f\_name, \_l\_name, \_age, \_id){};

Manager(const Manager &m);

Manager& operator=(const Manager &m);

virtual void Display(bool);

//add an employee to the subordinates list

Person\* AddSubordinate(Person \*p);

void DisplaySubordinates();

//Add here whatever you need

private:

list<Person \*> subordinates;//список підлеглих

};

Тобто, крім відомостей, спільних із класом Employee, клас Manager містить список його підлеглих (усі вони працюють у тому ж відділі, що й менеджер).

База даних представлена класом

class Database{

public:

Database(){};

~Database(){};//no need in destructor

//creates “flat” database

bool LoadFromFile(const char \*file);

//arranges "flat" database after loading from the file

void ArrangeSubordinates();

//hire a new employee

Person\* HireEmployee(Person \*p);

void DisplayDepartmentEmployees(string \_department);

//fire the employee

bool FireEmployee(int id);

void DisplayAll();

vector<Person\*> SQL(const char \*field, const char \* cond,

const char\* value);

void ShowRecordSet(vector<Person\*> rs)

//Add here whatever you need

private:

vector<Person\*> employees;

};

Методи класу Database.

bool Loadfromfile(const char \*file) – зчитує дані з csv–файлу й створює “плоский” список employees, тобто з порожніми списками subordinates. Наприклад.

//input.csv

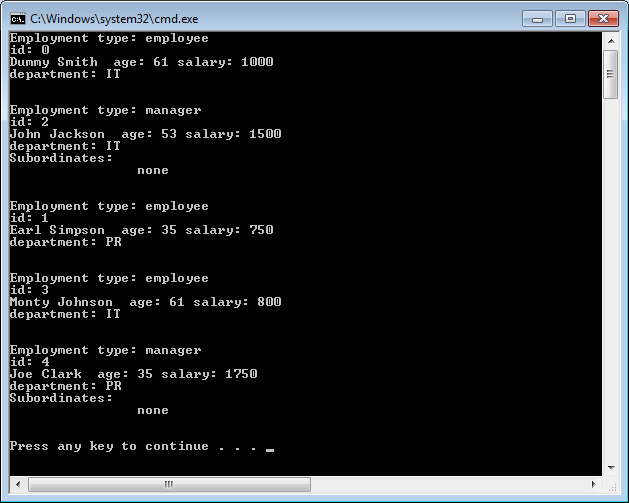
0;0;Dummy;Smith;61;IT;1000

1;2;John;Jackson;53;IT;1500

0;1;Earl;Simpson;35;PR;750

0;3;Monty;Johnson;61;IT;800

1;4;Joe;Clark;35;PR;1750



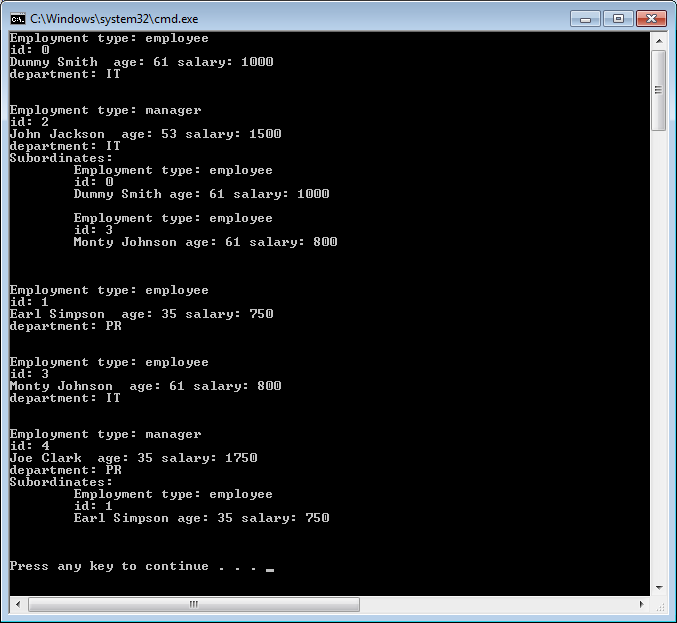
void ArrangeSubordinates() – заповнює списки subordinates кожного з менеджерів.

Після виклику

db.ArrangeSubordinates();

db.Displayall();

Одержимо



Структура рядка csv-файлу

Тип зайнятості – 0 –employee, 1 – manager;

Id;

First name;

Last name;

Age;

Department;

Salary

0;0;Dummy;Smith;61;IT;1000

Person\* HireEmployee(Person \*p) – додавання нового співробітника, повертає покажчик на запис у випадку успіху й NULL в разі неуспіху.

bool FireEmployee(int id) – видаляє запис про співробітника із заданим id, у тому числі й зі списку subordinates, повертає *true* у випадку успіху й *false* - а якщо ні..

vector<Person\*> SQL(const char \*field, const char \* cond, const char\* value) – SQL-запит до БД.

field – може приймати значення age або salary;

cond - може приймати значення le (less or equal) або ge (greater or equal);

value – значення відповідного поля.

void ShowRecordSet(vector<Person\*> rs) – друкує зміст набору засобів, яки повернено SQL. Наприклад,

vector<Person\*> rs = db.SQL("age", "le", "44");

db.ShowRecordSet(rs);

Employment type: manager

id: 2

John Jackson age: 43 salary: 1500

department: IT

Subordinates:

Employment type: employee

id: 0

Dummy Smith age: 61 salary: 1000

Employment type: employee

id: 3

Monty Johnson age: 67 salary: 800

Employment type: employee

id: 1

Earl Simpson age: 35 salary: 750

department: PR

Employment type: manager

id: 4

Joe Clark age: 44 salary: 1750

department: PR

Subordinates:

Employment type: employee

id: 1

Earl Simpson age: 35 salary: 750

В разі Employment type: manager друкуються його Subordinates.

Варіант 2.

Написати додаток Depot, яке зберігає відомості про транспортні засоби. У додатку використовується наступна ієрархія поліморфних класів.

Базовим є абстрактний клас

//An abstract class

class CVehicle {

public:

CVehicle() {};

CVehicle(int \_id, int \_v\_type, double \_average\_speed, double \_load\_capacity,

double \_cost\_per\_mile) :

average\_speed(\_average\_speed), load\_capacity(\_load\_capacity),

cost\_per\_mile(\_cost\_per\_mile),

id(\_id), v\_type(\_v\_type){};

virtual void Display() = 0;

//вартість перевезення weight кг на dist км

//якщо перевезення здійснюється вантажівкою, або дізелем і dist > //max\_distance повертає -1

virtual double CalculateCost( int weight, int dist) = 0;

// час перевезення вантажів

//якщо перевезення здійснюється вантажівкою або дизелем і dist > //max\_distance повертає -1

virtual double CalculateTime( int dist) = 0;

virtual int GetId() { return id; }

protected:

int v\_type;//0- car, 1 - train

int id;

double average\_speed;

double load\_capacity;

double cost\_per\_mile;//вартість перевезення 1 кг на 1 милю

};

class CCar : public CVehicle{

//Add here whatever you need

. . .

protected:

int max\_distance;// max distance without refueling

string make;//manufactorer

};

class CTrain: public CVehicle{

//Add here whatever you need

. . .

protected:

string type;//diesel / electrical

int max\_distance;// max distance without refueling if diesel

};

Ці транспортні засоби розміщені в

class CDepot {//add whatever you need

public:

bool loadDataFromCSV(string path) ;

void AddCar(…)

void AddTrain(…);

void RemoveVehicle(int id);

void ShowAll()

List<CVehicle\*> FindCheapest(double weight, double dist);

list< CVehicle\*> SQL(const char \*field, const char \* cond, const char\* value);

void ShowRecordSet(list< CVehicle\*> rs);

CVehicle\* VehiclesAvailable(double weight, double dist, double cost);

Void ChangeCostPerMile(int id, double new\_cost);

private:

list< CVehicle \*> vehicles;

};

Методи класу Depot.

bool Loadfromfile(const char \*file) – зчитує дані з csv–файлу input.csv. Наприклад

0;0;Marcedes;10000;25.5;70;500

1;1; Diezel;150000;11;80;1800

Структура рядка csv-файлу

id

v\_type - Тип транспортного засобу – 0 – truck, 1 – train;

make - if truck, type if train

load capacity

cost\_per\_mile;//вартість перевезення 1 кг на милю

average speed

max\_dist – if truck or diesel

Vehicle\* AddCar (…) – додавання нового транспортного засобу, повертає покажчик на запис у випадку успіху й NULL в разі неуспіху.

Vehicle\* AddTrain (…) – додавання нового транспортного засобу, повертає покажчик на запис у випадку успіху й NULL в разі неуспіху.

void ShowAll() - Друкує інфо про всі транспортні засоби

bool Remove Vehicle (int id) – видаляє запис про транспортний засіб із заданим id, повертає *true* у випадку успіху й *false* - а якщо ні..

list<CVehicle\*> SQL(const char \*field, const char \* cond, const char\* value) – повертає набір записів, які задовольняють заданим умовам.

field – може приймати значення average\_speed або max\_distance;

cond - може приймати значення le (less or equal) або ge (greater or equal);

value – значення відповідного поля.

void ShowRecordSet(list< CVehicle \*> rs) – друкує зміст набору засобів, яки повернено SQL. Наприклад,

list<CVehicle\*> rs = depot.SQL(“max\_distance”, “ge”, “500”);

depot. ShowRecordSet(rs);

0 Marcedes 10000 150.5 70 500

2 Man 7000 120.6 50 600

CVehicle\* VehiclesAvailable(double weight, double dist, double cost) – повертає набір записів про транспортні засоби, які здатні перевезти weight кг на dist км за ціну, що менше або дорівнює cost.

Примітка. Не повінні бути враховані вантажівки й дизелі для яких dist > max\_distance.

Результати роботи оформити у вигляді проекту C++-програми, що включає наступні файли:

1. Файли *оголошення* класів (\*.h) залежно від варіанта.

2. Файли *визначення* класів(\*cpp) залежно від варіанта.

3. Файл тест-драйву.

Номер варіанта визначає номер залікової книжки узятий за модулем 2.

ЛАБОРАТОРНА РОБОТА №3.

Теоретичні відомості можна знайти в лекціях №№ 21 – 23.

Мета роботи – оволодіти способами проектування й використання параметризованих класів і використання ітераторів.

**Завдання на роботу.**

Спроектувати параметризований клас однозв’язний список List з наступним інтерфейсом

template <class T>

class List{

struct ListNode{

T data;

ListNode \* next;

ListNode(){

next = NULL;

}

ListNode( T dat ){

data = dat;

}

void Print(){

cout<<data;

}

};

public:

typedef ListNode node\_type;

typedef \_iterator<node\_type> iterator;

//constructors / destructors

List();

~list();

//methods

iterator begin();//Returns an iterator addressing the first element

iterator end();//Returns an iterator that addresses the location

//succeeding the last element

void clear();//Erases all the elements of a list.

bool empty();//Tests if a list is empty.

iterator find(const node\_type & val);//Returns an iterator to the

// first element in a list that

//match a specified value.

void pop\_front();//Deletes the element at the beginning of a list.

void push\_front( const node\_type val);//Adds an element to the beginning

//of a list.

void remove(const node\_type val);//Erases first element in a list that

//match a specified value.

int size();//Returns the number of elements in a list.

void splice( iterator \_Where, List<T>& \_Right);//Removes element from //the target list and inserts it in first

// position of the argument list.

void Print();//Dumps list into the screen

private:

node\_type \*head;

iterator \*first, \*last;

};

**Mетоди.**

List() – створює порожній список.

List<int> l;

void push\_front( const node\_type val) – вставляє елемент у начло списку

l.push\_front( 10 );

iterator begin() – повертає ітератор перший елемент, що адресує, списку

iterator end()- повертає ітератор, що адресує позицію за останнім елементом

списку

List<int>::iterator it;

for(it = l.begin(); it = l.end; it++)

//do something

bool load(const char \*filename) – ініціалізує список з csv-file

void clear()- видаляє всі елементи списку

bool empty() – повертає true у випадку порожнього списку

iterator find(const node\_type & val) – повертає ітератор на перший елемент із

заданим значенням val

void pop\_front() – видаляє перший елемент списку

void remove(const node\_type val); – видаляє перший елемент списку з

заданим значенням val

int size() – повертає довжину списку

void splice( iterator \_Where, List<T>& \_Right) – видаляє елемент зі списку, з яким викликається метод, у позиції \_Where і вставляє його в початок списку \_Right

void Print() – виводить зміст списку на екран

Для переміщення за списком використовувати наступний параметризований ітератор

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*file: Iterator.h

\*purpose: template class used to traverse through the List container

\*author: Yu.Zorin

\*written: 13/03/2008

\*last modified:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

template <typename T>

class \_iterator

{

private:

T\* ptr;

public:

typedef T value\_type;

typedef \_iterator<T> it\_type;

\_iterator() {ptr = 0;}

\_iterator(T\* \_ptr) {ptr = \_ptr;}

T& operator \*();

void operator++();

//dummy operator to keep compiler quiet

void operator++(int);

bool operator==(const it\_type& \_iter) const;

bool operator!=(const it\_type& \_iter) const;

bool empty(){ return ptr == NULL;}

};

template <typename T>

T& \_iterator<T>::operator \*()

{

return \*ptr;

}

template <typename T>

void \_iterator<T>::operator ++()

{

ptr = ptr->next;

}

template <typename T>

void \_iterator<T>::operator ++(int)

{

ptr = ptr->next;

}

template <typename T>

bool \_iterator<T>::operator ==(const it\_type &\_iter) const

{

return (ptr == \_iter.ptr);

}

template <typename T>

bool \_iterator<T>::operator !=(const it\_type &\_iter) const

{

return !(\*this == \_iter);

}

Залежно від варіанта спроектувати один з наступних класів.

Варіант 1.

class Bus

{

public:

/\*

methods

\*/

friend ostream& operator<<(ostream& stream, Bus &obj);

private:

char \*driver;

char \*bus\_number;

char \*route\_number;

};

Відомості про автобус містять:

Номер автобуса (унікальний) - bus\_number ;

Ім'я водія - driver;

Номер маршруту - route\_number.

Методи класу Bus повинні забезпечити можливість створення й обробки списку List<Bus>.

Відомості про автобуси зберігаються у двох списках типу List<Bus>. Перший список - дані про всі автобуси в парку (ініціалізується з csv-файлу). Другий список – автобуси, що перебувають на маршруті.

При виїзді автобуса з парку програма видаляє дані про цей автобус зі списку автобусів, що перебувають у парку, і записує ці дані в список автобусів, що перебувають на маршруті за допомогою функції splice.

При в'їзді автобуса в парк програма видаляє дані про цей автобус зі списку автобусів, які перебувають на маршруті, і записує ці дані в список автобусів, що перебувають у парку за допомогою функції splice. Передбачити функцію друк списку автобусів, що перебувають у цей момент на заданому маршруті й функцію, що дозволяє змінити ім'я водія автобуса, що перебуває в парку.

Варіант 2.

class CFile{

public:

/\*

methods

\*/

friend ostream& operator<<(ostream& stream, CFile &obj);

private:

char \*\_filename;

char \*\_creation\_date;

int \_size;

};

Відомості про файл містять:

ім'я файлу - \_filename;

дата створення - \_creation\_date;

розмір файлу - \_size .

Методи класу CFile повинні забезпечити можливість створення й обробки списку

List<CFile>.

Відомості про файли зберігаються у двох списках типу List<CFile>. Перший список - дані про всі файли даного каталогу (ініціалізується з csv-файлу). Другий список – файли, вилучені з каталогу.

При видаленні файлу з каталогу, він переноситься з першого списку в другий за допомогою функції splice.

При відновленні файлу, він переноситься із другого списку в перший за допомогою функції splice.

Передбачити функцію видалення файлів, дата створення яких менше заданої, і функцію, що дозволяє змінити ім'я файлу в каталозі.

Варіант 3.

class References{

public:

public:

/\*

methods

\*/

friend ostream& operator<<(ostream& stream, References &obj);

private:

char \*word;

int \*pages;//pages numbers

int \_size;//number of pages

};

Відомості про слово містять:

слово - word;

номера сторінок, на яких воно зустрічається - pages;

число сторінок, на яких воно зустрічається - \_size.

Методи класу References повинні забезпечити можливість створення й обробки списку

посилань List<References>.

Відомості про посилання зберігаються у двох списках типу List<References>. Перший список - дані про всі посилання книги (ініціалізується з файлу). Другий список – вилучені посилання. Кількість сторінок книги не перевищує 300.

При видаленні посилання, вона переноситься з першого списку в другий за допомогою функції splice.

При відновленні посилання, вона переноситься із другого списку в перший за допомогою функції splice.

Передбачити функцію видалення посилань, число сторінок для яких менше заданого й функцію, що дозволяє відредагувати слово в списку посилань.

Результати роботи оформити у вигляді проекту C++-програми, що включає наступні файли:

1. Файли *оголошення* класів - <List.h>, <Iterator.h> і <Bus.h>, <CFile.h> <References.h> залежно від варіанта.

2. Файли *визначення* класів - <Bus.cpp>, <CFile.cpp>, <References.cpp> залежно від варіанта.

3. Файл тест-драйву, наприклад, <Test.cpp>.

Номер варіанта визначає номер залікової книжки, узятий за модулем 3.