**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 2018

ЗМІСТ

[ЛАБОРАТОРНА РОБОТА № 1.](#Lab1)

[ЛАБОРАТОРНА РОБОТА №. 2.](#Lab2)

ЛАБОРАТОРНА РОБОТА №1.

Мета роботи – оволодіти способами проектування й використання параметризованих класів, способами приведення типів.

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

**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

bool load(const char \*filename);//Initializes object from csv-file

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.

iterator remove(iterator \_where);// Erases element from specified //position and returns iterator to position next to removed element.

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 на початок списку

l.push\_front( 10 );

iterator begin() – повертає посилання на перший елемент

iterator end()- повертає посилання на 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 - 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 печатки 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 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:

/\*

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 С++-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 по модулю 3.

Варіант 4.

Спроектувати параметризований клас deque з наступним інтерфейсом

///////////////// deque class ///////////////////////

template <typename Type>

class deque{

public:

typedef deque<Type> dq\_type;

typedef Type val\_type;

typedef \_iterator<Type> iterator;

typedef unsigned int size\_type;

public:

////// constructors / destructors /////////////

deque();

deque(size\_type \_Count, const Type& \_Val);

deque(iterator \_First, iterator \_Last);

////////// member functions //////////////

void push\_back( const Type &);

void push\_front( const Type &);

void pop\_front() ;

void pop\_back() ;

void insert( iterator \_Where, const Type& \_Val){};

void insert(iterator \_Where, iterator \_First, iterator \_Last);

Type& operator[]( size\_type );

iterator begin() const;

iterator end() const;

size\_type size() const;

void resize( size\_type \_Newsize);

void resize( size\_type \_Newsize, const Type& \_Val);

void swap(deque<Type> &dq);

bool empty()const;

void erase(size\_type indx) ;

iterator erase(iterator \_First, iterator \_Last){};

void clear();

Type& back() ;

Type& front() ;

~deque();

void Print();

bool load(const char \*filename);

private:

Type\* buf;

size\_type \_size;

};

**Mетоди.**

deque()– створює порожню чергу.

deque(size\_type \_Count, const Type& \_Val) - створює чергу розміру \_Count, елементи якої мають значення \_Val.

deque(iterator \_First, iterator \_Last) - створює чергу з елементами, що належать дыапазону \_First, \_Last черги-аргумента.

void push\_front( const Type &); – вставляє елемент на початок черги

void push\_back( const Type &); – вставляє елемент в кінець черги

void pop\_front() - видаляє елемент з початку черги

void pop\_back() - видаляє елемент з кінця черги

iterator begin() – повертає ітератор на перший елемент

iterator end()- повертає ітератор на позицію за останнім елементом

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

Type& operator[]( size\_type n) - повертає посилання на елемент з індексом n

void insert( iterator \_Where, const Type& \_Val) - вставляє елемент \_Val в позицію \_Where

void insert(iterator \_Where, iterator \_First, iterator \_Last) – вставляє набір елементів з черги-аргументу, що належать діапазону \_First, \_Last, в цільову чергу, починаючи с позиції \_Where

d1.insert( ++d1.begin( ), d2.begin( ),--d2.end( ) );

void clear()- видаляє всі елементи черги

bool empty() – повертає true у випадку порожньої черги

size() – повертає кількість елементів черги

resize( size\_type \_Newsize) – якщо \_Newsize менше за поточний розмір, видаляються елементи з кінця черги, поки розмір черги не дорівнютиме \_Newsize. якщо \_Newsize більше за поточний розмір, додаються елементи в кінець черги, поки розмір черги не дорівнютиме \_Newsize. Значення елементів, що додаються - за замоченням.

resize( size\_type \_Newsize, const Type& \_Val) – все як у попередньому методі, значення елементів, що додаються - \_Val.

void swap(deque<Type> &dq) – виконує обмін елементами черги-аргумента й цільової черги.

erase(size\_type indx) – видаляє елемент з індексом indx.

iterator erase(iterator \_First, iterator \_Last) - видаляє елементи з діапазону \_First, \_Last і повертає ітератор на елемент за останнім видаленим.

Type& back() – повертає посилання на останній елемент черги.

Type& front()– повертає посилання на перший елемент черги.

Print() – виводить зміст черги на екран

class Student

{

public:

Student();

Student(int \_id, char\* first\_name,char\* last\_name,float grade);

Student(const Student &);

~Student();

void operator=(const Student &);

int get\_id()const;

friend ostream& operator<<(ostream& stud\_out,Student& stud);

friend istream& operator>>(istream& stud\_in,Student& stud);

private:

int id;

char \*f\_name;

char \*l\_name;

float average\_grade;

};

Написати код тест-драйву, який забезпечує:

* ініціалізацію черги з csv-файлу;
* друк черги з використанням перевантаженої операції вилучення <<;
* видалення з черги запису з заданим id;
* редагування запису з заданим id;
* додавання в чергу запису з контролем унікальності id;
* тест функций swap, erase.

Структура сsv-файла

id f\_name l\_name average\_grade

1;"bill";"Gates";3.14

2;"george";"bush";2.98

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

1. Файли *оголошення* класів - - <Student.h> , <Iterator.h> і <deque.h>.

2. Файли *визначення* класів - < Student.cpp >, <Test.cpp>.

ЛАБОРАТОРНА РОБОТА №2.

Просте успадкування. Додаток Database.

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).

Завдання на роботу.

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();

//Add here whatever you need

private:

vector<Person\*> employees;

};

Таким чином, відомості про співробітників представлені вектором vector<Person\*> employee, тобто покажчиками на екземпляр абстрактного класу. При обробці списку потрібно мати можливість розрізняти тип об'єкта (Employee або Manager), адресу якого містить покажчик типу Person \*p. Зробити це можна, наприклад, з використанням операції typeid.

if (typeid(\*p) == typeid(Employee))//employee

if (typeid(\*p) == typeid(Manager))//manager

Використання

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.

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

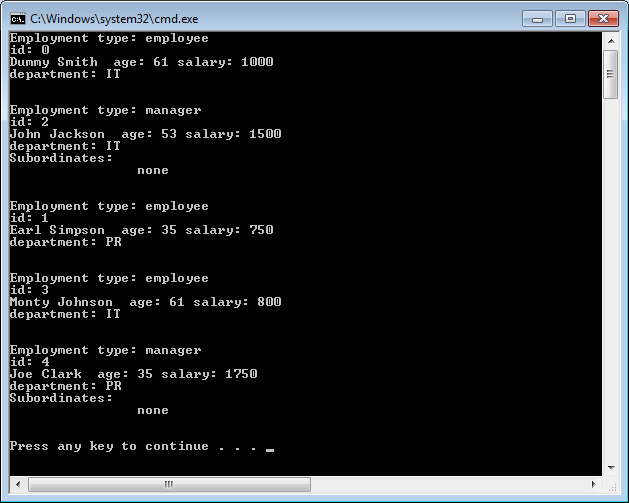
0;0;Dummy;Smith;61;IT;1000

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



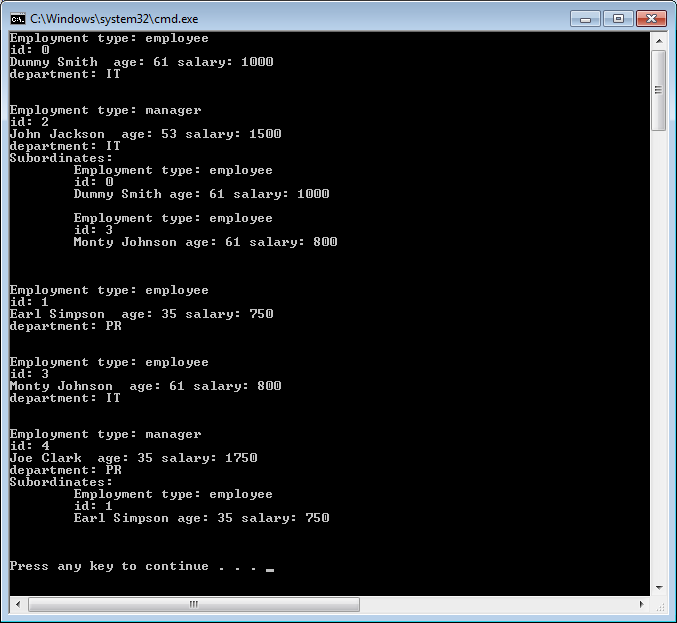
void ArrangeSubordinates() – заповнює списки subordinates кожного з менеджерів.

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

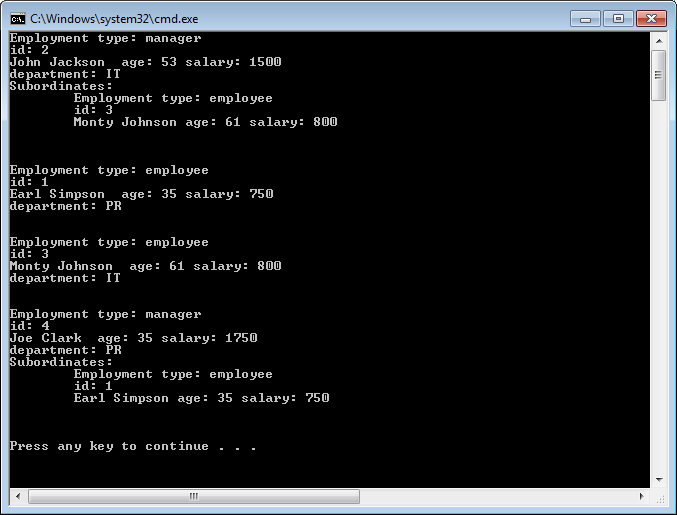
Person\* HireEmployee(Person \*p) – додавання нового співробітника, повертає покажчик на запис у випадку успіху й NULL в разі неуспіху.

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, то. Наприклад, після

db.FireEmployee(0);

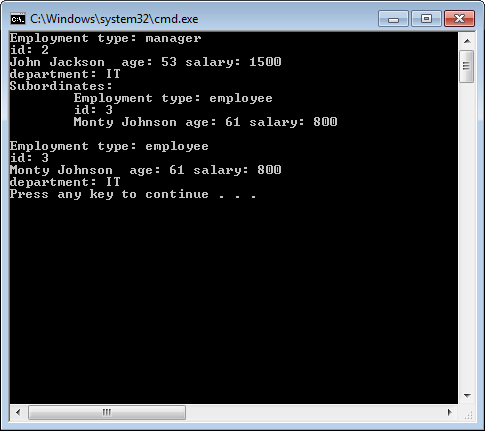
db.DisplayAll();

Одержимо



void DisplayDepartmentEmployees(string \_department) – виводить на консоль відомості про всіх співробітників department.

db.DisplayDepartmentEmployees("IT");



Для запитів до бази даних застосовуються дещо спрощені Structured Query Language (SQL). В нашому випадку вони мають вигляд

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

де

field – поле, за яким виконується запит,

value – значення поля,

cond – умова, якому відповідає value.

Наприклад

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

Отримати відомості про співробітників, вік яких менше за 44 роки.

Після виклику

db.LoadFromFile("input.txt");

db.ArrangeSubordinates();

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

db.ShowRecordSet(rs);

Одержимо

//Display all

Employment type: employee

id: 0

Dummy Smith age: 61 salary: 1000

department: IT

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: employee

id: 3

Monty Johnson age: 67 salary: 800

department: IT

Employment type: manager

id: 4

Joe Clark age: 31 salary: 1750

department: PR

Subordinates:

Employment type: employee

id: 1

Earl Simpson age: 35 salary: 750

Selected by age less than 44

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: 31 salary: 1750

department: PR

Subordinates:

Employment type: employee

id: 1

Earl Simpson age: 35 salary: 750

Отже, базу даних треба доповнити методами.

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

void ShowRecordSet(vector<Person\*> rs);

Поля, за якими треба забезпечити виклики

department;

salary;

id;

age.

Поле cond може приймати значення

“gt” – greater than;

“lt” – less than;

“eq” – equal.