**Практична робота № 25-26.** **Складання програм з використанням наслыдування.**

**Мета:** Набуття навичок в розробці найпростіших класів та роботі з об’єктами класів.

**Порядок виконання роботи**

1. Повторити лекцію №13.
2. Ознайомитися з теоретичними відомостями.
3. Продовжити працювати над програмою минулої лабораторної роботи.
   1. Створити через просте спадкування похідний до *Person к*лас *Student*, в якому створити:
4. члени-дані : курс, спеціальність, поточна кількість балів по предметах фізика, математика, програмування;
5. три конструктори (порожній, ініціалізації, копіювання) та деструктор, в яких поставити виведення повідомлення для моніторингу їх роботи;
6. селектори та модифікатори до всіх членів похідного класу,
7. метод введення даних в інтерактивному режимі (використати успадкований від базового класу, якщо такого в базовому класі не було – записати),
8. метод виведення даних на консоль (використати успадкований від базового класу, якщо такого в базовому класі не було – записати);
   1. В головній програмі (функції main()):
9. включити виклик нових методів та функцій до відповідних об‘єктів;
10. створити екземпляр класу *Student* та заповнитийого даними та вивести на консоль.
11. В першому рядку програми та заголовкового файлу повинні бути записаними в коментарі номер групи та прізвище, а також номер ЛР (через кому до попередньої).
12. Результати (програма) надсилати на електронну адресу викладача

**t.i.lumpova@gmail.com** у вигляді cpp-файлу з іменем у форматі

**<Номер групи> <Номер лабораторної><Прізвище англійською>.cpp**

Наприклад, 31-01Ivanov.cpp.

**Строк відсилки ПР для ОПІ-41 29.04.2024**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача,.В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№25-26" В темі листа вказати, номер групи, прізвище студента, номер ПР та фразу "Запитання".

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ОПІ<Номер групи>-Запитання-<Прізвище>**.

**Теоретичні відомості.**

**Конструктори і деструктори.**

Серед функцій-членів класу можуть бути такі, що визначають процеси створення, ініціювання, копіювання та знищення об’єктів свого класу. До цих функцій належать конструктори і деструктори, які, якщо явно не оголошуються, то автоматично викликаються системою за замовчуванням.

Чим і як може ініціалізуватися клас?

* Деякими замовчуваними значеннями
* Списком значень, що відповідає типам даних
* Об’єктом цього ж класу визначеним раніше
* Деякими даними сумісними за типом з типом класу

У мові С++ для ініціалізації об'єктів призначений механізм, називаний ***конструктором***. Це — функція-член, ім'я якої збігається з ім'ям класу. Вона може мати будь-які параметри, що необхідні для ініціалізації полів об'єкта. При цьому конструктор не має ніякого значення, що повертається. Його виклик являє собою визначення об'єкта. Інакше кажучи, визначаючи об'єкт, ви викликаєте конструктор.

**Види конструкторів:**

* За замовчуванням
* Ініціалізації (з параметрами)
* Копіювання
* Додаткові конструктори

Головною метою **конструкторів** є ініціювання змінних-об’єктів класу та розподілення пам’яті для їх зберігання. Конструктор викликається кожного разу при створенні об’єкта даного класу або явно, або автоматично.

**Дії, які виконують конструктори.**

* Виділяють динамічну пам’ять
* Ініціалізують дані класу
* Якщо програміст не вказав жодного конструктора або якісь поля не були проініціалізовані, то полям значимих типів присвоюється нуль, полям показникових типів — значення null;
* Не можуть нічого повернути
* Забороняють або дозволяють і виконують приведення до типу класу
* Можуть викликатися явно та неявно

**Основні правила роботи з конструкторами**:

* ім’я конструктора повинне співпадати з ім’ям свого класу;
* для конструктора не вказується тип значення, яке повертає функція;
* клас може мати декілька конструкторів або не мати жодного;
* конструктор за замовчуванням — це конструктор, який не має параметрів, або всі його параметри мають значення за замовчуванням;
* конструктор копіювання безпосередньо призначений для створення об’єкта класу шляхом копіювання даних з існуючого об’єкта.

**Конструктор за замовчуванням** – це конструктор, який дозволяє створювати екземпляри класів з неявною ініціалізацією даних**.** Такий конструктор автоматично створюється компілятором для класів, в яких не визначений власний конструктор. Проте, як тільки в класі визначається хоч один конструктор, такий автоматичний конструктор перестає діяти. В разі необхідності створення екземплярів без ініціалізації, варто визначити в класі конструктор за умовчанням. Найпростіший спосіб зробити це – перевантажити конструктор, або визначити умовчання для всіх його параметрів.

**Конструктор ініціалізації** містить окремі значення, що використовуються для ініціалізації стану полів екземпляру класу. У списку параметрів може бути зазначений нуль, один чи більше параметрів будь-якого типу. В прикладі минулої лекції це

Point (double x=0, double y=0): \_x(x),\_y(y) { };

Це скорочена форма запису, конструктор може мати вигляд

Point (double x, double y){ \_x=x; \_y=y; };

Приклад минулої лекції

#include <iostream>

using namespace std;

class Point

{// Атрибути -private закрили прямий доступ до атрибутів

double \_x, \_y;

int \*ptr; //\* Покажчик на ділянку пам‘яті

// відкрили доступ до методів

public: // Конструктор

Point (double x=0, double y=0**): \_x(x),\_y(y) { }**;

Point (const Point &obj) // Конструктор копіювання

{ cout << "\nCopy Constr\n"; }

~Point(){cout << "\nDestruct\n";}; // Деструктор

// Функції доступу до атрибутів - селектори

double& x() {return \_x;}

/\* ім'я \_х в тексті функції позначає поле \_х того екземпляру, до якого застосовано функцію \*/

double& y() {return \_y;}

void set\_x(double a){\_x=a;}

void set\_y(double a){\_y=a;}

};

void funcShow(Point object)

{ cout << "\nFunction get object as param\n"; }

Point funcReturnObject()

{ Point object;

cout << "\nFunction return object\n";

return object;

}

int main(int argc, char\* argv[])

{ system("color F0");

Point a1(1,2);

cout<< a1.x() <<";"<< a1.y() <<endl;

Point b1(1);

cout<< b1.x() <<";"<< b1.y() <<endl;

Point c1;

cout<< c1.x() <<";"<< c1.y()<<endl;

cout << "1 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

Point obj1; // створюємо об‘єкт класу

cout << "2 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

funcShow(obj1); // передаємо об‘єкт до функції

cout << "3 - 4 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

funcReturnObject(); // функція повертає об‘єкт

cout << "5 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

Point obj2 = obj1; // ініціалізація об‘єкту при створенні

Point \*ptr;

ptr = new Point (22,33);

cout<< ptr->x() <<";"<< ptr->y() <<endl;

delete ptr;

Point \*ptrM = new Point[3];

for (int j=0; j<3; j++)

{ptrM[j].set\_x(j); ptrM[j].set\_y(j);

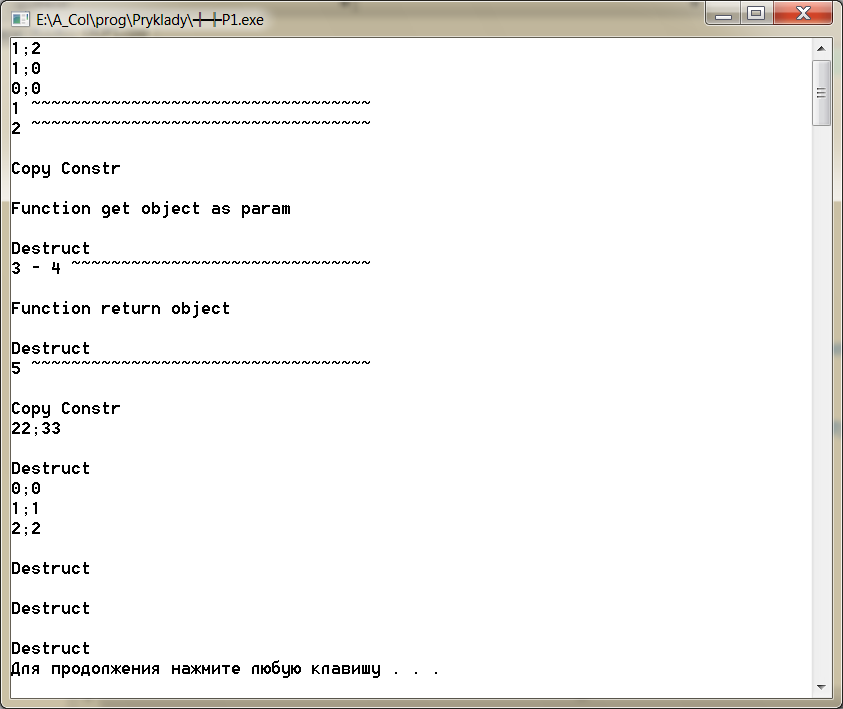
cout<< ptrM[j].x() <<";"<< ptrM[j].y() <<endl; }

delete [] ptrM ;

system("pause");

return 0;

}



Використання списку ініціалізаторів членів **переважніше**, ніж встановлення значень в тілі конструктора, оскільки він безпосередньо ініціалізує елемент. В наданому прикладі показано список ініціалізаторів членів, який складається зі всіх виразів ***ідентифікаторів (аргументів)*** після двокрапки (":"). Ідентифікатор повинен посилатися на член класу; він ініціалізується значенням аргументу. Аргумент може бути одним з параметрів конструктора, викликом функції.

*Список ініціалізації — єдиний спосіб задати значення константних змінних-членів.* Крім того, така ініціалізація буває більш ефективною. Правда, якщо клас містить велику кількість членів, список може стати занадто довгим, що знизить наочність програми.

Можна описати будь-яку кількість конструкторів класу. Вони повинні відрізнятися кількістю або типами параметрів (властивість поліморфізму).

**class** Country

{

**private**:

**char** name[40];

**double** area;

**int** population;

**public**:

Country(**double** area) { **this**->area = area; }

Country(**const** **char**\* name);

Country(**const** **char**\* name, **double** area);

Country(**const** **char**\* name, **double** area, **int** population);

. . .

};

Country::Country(**const** **char** \* name)

{

strcpy(**this**->name, name);

area = 1; // істотно, щоб територія не мала значення 0

// і не виникало помилки у функції density()

}

Country::Country(**const** **char** \* name, **double** area)

{

strcpy(**this**->name, name);

**this**->area = area;

}

Country::Country(**const** **char** \* name, **double** area, **int** population)

{

strcpy(**this**->name, name);

**this**->area = area;

**this**->population = population;

}

**Конструктор копіювання** ініціалізує стан класу значення іншого екземпляру цього класу. В списку параметрів вказується єдиний параметр, що має тип “посилання на екземпляр класу”. Для явного визначення конструктора копіювання необхідно дотримуватись особливого синтаксису:

**<ім’я\_класу> (const <ім’я\_класу> & );**

Параметром конструктора копіювання є стала змінна-посилання на екземпляр класу. Його призначення – коректне створення копії екземпляру. Особливо важливо це у випадку, коли членами класу є вказівники, пам’ять під які виділяється оператором **new**. Адже тоді поелементне копіювання копіює вказівник (поверхневе копіювання), а не об’єкт, на який він посилається, – для цього необхідне глибоке копіювання.

*Конструктор копіювання потрібен нам для того, щоб створювати «реальні» копії об'єктів класу, а не побітову копію об'єкта*. Іноді це принципово важливо. Таку «реальну» копію об'єкта треба створювати в декількох випадках:

* коли ми передаємо об'єкт в яку-небудь функцію у вигляді параметра;
* коли яка-небудь функція повинна повернути об'єкт класу в результаті своєї роботи;
* коли ми в головній функції один об'єкт класу ініціалізуємо іншим об'єктом класу.

Наприклад, ми передаємо об'єкт в функцію у вигляді параметра. Функція буде працювати не з самим переданим об'єктом, а з його побітової копією. Припустимо в конструкторі класу, при створенні об'єкта, виділяється певний об'єм пам'яті, а деструкція класу цю пам'ять звільняє. Покажчик побітової копії об'єкта буде зберігати ту ж адресу пам'яті, що й оригінальний об'єкт. І, коли при завершенні роботи функції і знищенні побітової копії об'єкта, спрацює деструктор, він обов'язково звільнить пам'ять, яка була зайнята об'єктом-оригіналом. На додачу, ще й при завершенні роботи програми, деструктор спрацює повторно і спробує ще раз звільнити цей обсяг пам'яті, що неминуче призведе до помилок програми. Та ж доля спіткає і пам'ять, виділену для покажчика об'єкта, якщо буде віддалятися побітова копія повертається функцією об'єкта, і побітова копія при ініціалізації об'єкту класу іншим об'єктом.

Щоб уникнути цих проблем і помилок існує конструктор копіювання. Його робота полягає в тому, щоб створити реальну копію об'єкта зі своєю особистою виділеної динамічною пам'яттю.

**Селектори і модифікатори**

**Селектори (Get)** — методи класу, що повертають значення приватного атрибута (даних);

**Модифікатори (Set)** — методи класу, що встановлюють коректне значення приватного атрибута(даних).

Як добратися до атрибутів, якщо вони закриті? ― За допомогою методів доступу: селекторів і модифікаторів

Селектор

double getX() { return **\_x**; };

Модифікатор

void setX (double x) { \_x = x;}

Окремий модифікатор дозволяє контролювати кожну спробу зміни значення атрибуту, а селектор ― кожне використання його значення.

***Контрольні запитання*.**

1. Визначте основні концепції наслідування.
2. В чому полягає принцип підстановки?
3. Які форми наслідування Ви можете визначити?
4. В чому полягає просте спадкування?