**Лабораторна робота №7-2.** **Конструктори. Ініціалізація даних.**

**Мета:** Набуття навичок в розробці програм, де використовуються класи та об’єкти класів.

**Порядок виконання роботи**

1. Повторити синтаксис оголошення конструкторів.
2. Ознайомитися з теоретичними відомостями.
3. Продовжити працювати над програмою минулої лабораторної роботи. Створити для класу *Person* три конструктори (порожній, ініціалізації, копіювання) та деструктор та продемонструвати їх роботу на окремих об’єктах. Для тих, хто працює над темою власної курсової роботи виконати ці дії до вже розроблених класів курсової, а саме: хоч би до одного класу надати всі види конструкторів. Бажано, щоб це було логічно і доцільно.
4. Створити масив об’єктів класу *Person* фіксованого розміру та проініціалізувати його конкретними значеннями. Для тих, хто працює над темою власної курсової роботи виконати ці дії хоч би до одного класу. Бажано, щоб це було логічно і доцільно.
5. Надати коментарі до всіх членів класу та методів щодо їх призначення.
6. В першому рядку програми та заголовкового файлу повинні бути записаними в коментарі номер групи та прізвище, а також номер ЛР (через кому до попередньої) , в другому рядку в коментарі записати тему курсової.
7. Результати надсилати на електронну адресу викладача

[**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді заголовкового файлу та cpp-файлу з іменем у форматі

Для заголовкового файлу

**<Ім‘я класу>.h**

Наприклад, *Person*.h

При розділенні інтерфейсу і реалізації, реалізацію надавати, відповідно

**<Ім‘я класу>.срр**

Наприклад, *Person*.срр

Для cpp-файлу:

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

Наприклад, 31-01Ivanov.cpp.

Тему в заголовку листа записати

**ООП<Номер групи>-ЛР<Номер лабораторної>-<Прізвище>**

**Строк відсилки ЛР ІПЗ-31 24.02.2024**

**ІПЗ-32 23.02.2024**

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

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

**Теоретичні відомості.**

Розглянемо приклад.

#include <iostream>

using namespace std;

class Student {

private:

char \*name;

public:

Student (char \*); //конструктор з 1 параметром

void show (void);

void setname (char \*);

};

Student::Student (char \*name=NULL) {

//реализація конструктора

if (name) {

this->name = new char [strlen(name)+1];

if (this->name) setname (name);

}

else this->name=NULL;

}

void Student::setname (char \*name) {

strcpy (this->name, name);

// Відсутня перевірка чи достатньо місця!

}

void Student::show (void) {

cout << endl;

if (this->name) cout << this->name;

else cout << "NULL";

}

int main() {

Student \*Ivanov = new Student ("Ivanov");

Ivanov->show();

Student Petrov("Petrov");

Petrov.show();

return 0;

}

Щоб уникнути краху програми у випадку завдання прізвища, що перевищує виділену ділянку (довжина більша за ініціалізоване значення) функцію setname можна задати у вигляді, коли "зайве" не копіюється:

void Student::setname (char \*name) {

if (this->name) {

int len = strlen(this->name);

strncpy (this->name, name, len);

this->name[len]='\0';

}

}

Але в цьому випадку можуть виникнути проблеми при копіюванні в не ініціалізований об‘єкт, як наприклад, у випадку

Student Copeikin = \*Ivanov;

У такого об‘єкта name буде заповнене "сміттям", що може викликати проблеми при вивільненні пам‘яті.

Можна використати таку конструкцію

void Student::setname (char \*name) {

if (this->name) delete[] this->name;

int len = strlen(name);

if (len) {

this->name = new char [len+1];

if (this->name!=NULL) strcpy(this->name,name);

}

}

Ця конструкція буде працювати, якщо властивість this->name попередньо ініціалізувалася до виклику setname.

Ми уникнемо цієї проблеми, якщо будемо використовувати конструктор копіювання, який компілятор розпізнає за тим, що його параметром є покажчик (посилання) на об‘єкт класу. Це об‘єкт, що знаходиться праворуч від знаку "присвоїти", а об‘єкт ліворуч від "=" доступний через this. Ось такою має бути реалізація конструктора копіювання:

public:

//...

Student (Student &); //конструктор копіювання

//...

Student::Student (Student &from) {

//реалізація конструктора копіювання

this->name = new char [strlen(from.name)+1];

if (this->name) setname (from.name);

}

*Конструктор копіювання* для класу **X** має один аргумент типа **&X** і виконується при присвоюванні екземплярів класу:

Student \*Ivanov = new Student ("Ivanov");

Student Dudka = \*Ivanov;

Для нашого класу деструктор повинен мати вигляд:

Student::~Student () { //реалізація деструктора

if (name) delete[];

}

Оскільки в середині деструктора нема інших змінних на ім‘я name, то вираз this->name можна не писати.

Якщо ми при явному деструкторі не задали конструктора копіювання, то замовчуваний конструктор виконає дії Dudka.name = Ivanov->name, тобто, не скопіює рядок, а лише встановить покажчик Dudka.name на ту ж адресу пам‘яті, куди показував Ivanov->name. Після явного або неявного вилучення об‘єкту Ivanov наслідком спроби показати значення Dudka.name буде крах програми.

**Для всіх класів існує правило "великої трійки": якщо в класі є явний деструктор, то потрібен явний конструктор копіювання та оператор "=".** Можливо, доцільно визначати в цьому випадку і явний конструктор без аргументів.

Якщо для збереження рядків використовувати не покажчики char\*, а готовий клас string, створення класу значно спроститься.

#include <iostream>

#include <string>

using namespace std;

class Class {

string prop1, prop2;

int n1, n2;

public:

Class() : prop1(""), prop2(""), n1(0), n2(0) {} //Конструктор 1

Class(char \*, char \*, int = 0, int = 0); //Конструктор 2

};

Class::Class (char \* p1, char \*p2, int n1, int n2) { //Конструктор 2

prop1 = p1; prop2 = p2; this->n1 = n1; this->n2 = n2;

}

int main () {

Class c1; //Викликаємо конструктор 1

Class \*c2 = new Class(); // Викликаємо конструктор 1

// Викликаємо конструктор 2:

Class \*c3 = new Class("str1", "str2", 100, 200);

// конструктор 2 з 4 аргументами

Class \*c4 = new Class("ctr1", "ctr2");

// конструктор 2с 2 аргументами

cin.get(); return 0;

}

Об‘єкти c2, c3 та c4 створені в "кучі", під них динамічно надається пам‘ять оператором new і вона може бути вивільнена застосуванням до об‘єктів оператора delete.

Об‘єкт c1 займають місце в стеці програми, зайнята ними пам‘ять буде вивільнена тільки коли закінчиться область видимості змінної (найближчі зовнішні по відношенню до її опису фігурні дужки) і до них не можна явно застосовувати delete.

Стек і куча - це **одна й та ж сама оперативна пам‘ять (ОП)**, якщо всю ОП вважати "кучею", то стек - її частина, що виділяється при запуску потоку програми під ті дані, для яких відомий об‘єм пам‘яті, що вони займають (локальні змінні, масиви фіксованої розмірності тощо).

Стек - це кількість пам‘яті, про яку відомо, що вона потрібна **на момент запуску потоку застосунку** (в багатопоточних програмах у кожного потоку - свій стек), а куча - потенційно доступна для виділення пам‘ять, її розмір може статися і більше, ніж є вільної ОП (за рахунок дискових кешей).

Пам‘ять в стеці "виділяється" простим переміщення вказівника стека (регістр процесора SP/ESP/RSP), це виконується постійно, при завантаженні аргументів функції, при виклику будь-якої функції, в самій функції виділяється місце під локальні змінні тощо. Місце під сам стек виділяється один раз при запуску потоку.

Кучею керує менеджер пам‘яті, пам‘ять виділяється і звільняється "вручну" операторами new/delete або функціями malloc/free. Куча працює значно повільніше, ніж стек.

По замовчуванню створені в куче об‘єкти живуть "вічно", тобто до завершення програми.

Але програма за необережністю або навмисно може переписати стек "по живому", дані, або навпаки, переповнити стек та "залізти" в кучу, зіпсувавши дані там.

**Про "статичні класи"**

В C++ відсутній "static class". Найближчою концепцією буде клас, що містить тільки статичні методи. Наприклад:

// header

class MyClass

{

public :

static void myMethod() ;

} ;

// source

void MyClass::myMethod()

{

// etc.

}

**Контрольні запитання**

1. Якщо в програмі повинні виконуватися якісь дії по обробленню введених даних перед створенням/модифікацією об‘єкта певного класу або по перетворенню даних перед записом до файлу, чи доцільно створювати в цьому класі методи читання / запису даних до файлу?
2. Як здійснюється ініціалізація даних-членів класу?
3. Що таке об’єкт класу?
4. Наведіть приклад об’єкту класу.
5. Як задати початкові значення даним-членам класу (всі способи)?
6. У чому полягає роль конструкторів та деструкторів у класі?
7. Які способи можна застосувати для ініціалізації об’єкта класу? Як для цього застосовуються конструктори?
8. Уява об’єкта у пам’яті.
9. В чому різниця між *inline*-функціями та іншими функціями-членами класу?
10. Чи можна присвоювати значення одного об’єкта іншому.
11. Які специфікатори видимості для членів класу ви знаєте?
12. Чи можливо створювати масиви об’єктів?
13. Для чого використовується покажчик this?
14. Чи можливо передавати об’єкти як аргументи функціям?
15. Чи можливо щоб функція повертала значення типу клас?