DOI: 10.36910/6775-2524-0560-2020-39-32
УДК: 004.415.3
**Пех Петро Антонович**., к.т.н., доцент
https://orcid.org/0000-0002-6327-3319
**Костюк Юрій Юрійович**, студент
**Кравченко Максим Богданович**, студент
Луцький національний технічний університет

# ДО ПИТАННЯ КОНСТРУЮВАННЯ КЛАСІВ З КОНСТРУКТОРАМИ РІЗНОГО ТИПУ

**Пех П.А., Костюк Ю.Ю., Кравченко М.Б. До питання конструювання класів з конструкторами різного типу.** В статті запропоновано кілька програм мовою С++ на базі класів з різними типами конструкторів. Досліджуються також питання передачі об'єктів з головної функції у підпрограми-функції і навпаки. Наведені коди запропонованих програм.
**Ключові слова:** клас, функція, формальний параметр, конструктор з параметрами, конструктор копіювання

**Пех П.А., Костюк Ю.Ю., Кравченко М.Б. К вопросу конструироавния классов с конструкторами различных типов.** В статье предложены несколько программ на языке С++ на базе классов с различными типами конструкторов. Исследуются также вопросы передачи объектов с главной функции подпрограммам-функциям и наоборот. Приведены коды предложенных программ.
**Ключевые слова**: класс, функция, формальный параметр, конструктор с параметрами, конструктор копирования

**Pekh Petro, Kostyuk Yuri, Kravchenko Maxim. On the question of designing classes with constructors of different types.** The article proposes several C ++ language programs based on classes with different types of constructors. The transfer of objects from the main function to the subroutines-functions and vice versa is also investigated. The codes of the offered programs are resulted.
**Keywords:** class, function, formal parameter, constructor with parameters, copy constructor

**Постановка задачі**. Загальновідомо, що програми мовою С++ найчастіше розробляються на базі класів [1,2,4,7,8]. Клас є фундаментальним поняттям мови С++. Класи опрацьовують дані за допомогою методів [3,5]. Однією з функцій класу є конструктор. Саме конструктор є функцією, за якою програми створюють об'єкти даного класу і виконують їх попередні налаштування [4,6]. Конструктори бувають різних типів. Задача полягає у дослідженні дії різного типу конструкторів на дані класу.

**Метою нашого дослідження** було розроблення засобами мови `C++` комплексу програм на базі класів з різними типами конструкторів.

**Новизна** полягає у підході до вирішення проблеми з позицій об'єктно-орієнтованого програмування [1,4,6].

**Основна частина.** Оскільки під час розробки класів використовуються конструктори різних типів – конструктор за замовчуванням, конструктори з одним, двома або більшою кількістю параметрів, конструктор копіювання, ми розробили комплекс програм, частина яких пропонується в даній статті для всебічного дослідження даногопитання. Крім того, ми досліджували процес передачі об'єктів з головної функції у функції-підпрограми і навпаки.

Програма, код якої наведений нижче, розроблена на базі класу `PlanePoint`, вирішує досить просту задачу. Вона дозволяє ввести координати `x,y` довільної точки `M(x,y)` площини і обчислити віддаль від цієї точки до точки початку координат `O(0,0)`. Ми передбачили у програмі три конструктори: конструктор без параметрів `PlanePoint(void)`, конструктор з двома параметрами `PlanePoint(double nx, double ny)` та конструктор копіювання `PlanePoint(PlanePoint & ref_Point)`. Клас `PlanePoint` має два власні методи: `int GetX(void){return x;}` та `int GetY(void){return y;}`,з допомогою яких можна отримати значення координат `x,y` довільної точки `M` площини. Функція `double GetLength(PlanePoint tmp)` не належить до методів класу `PlanePoint`, тому вона є зовнішньою по відношенню до класу функцією. Родзинкою є те, що ця функція має у якості формального параметра об'єкт `tmp` класу `PlanePoint` і з його допомогою передаються координати точки `M`.

Після запуску програми на виконання спочатку спрацьовує конструктор з двома параметрами під час виконання команди `PlanePoint Ob1(5, 8)`, який створює об'єкт `Ob1` з парметрами `x=5,y=8`, а потім під час виконання команди `PlanePoint Ob2` спрацьовує конструктор без

параметрів, який створює об'єкт `Ob1` з парметрами `x=7,y=21`. Процес контролюємо шляхом виведення на екран значень абсцис обох об'єктів за допомогою методу `GetX()`. Далі виконується команда побітового копіювання об'єктів `Ob2 = Ob1`. Тут конструктор копіювання не використовується, оскільки обидва об'єкти вже існували. Процес побітового копіювання об'єктів контролюємо виведенням на екран абсциси точки `x=5` за допомогою методу `GetX()`. Конструктор копіювання спрацьовує під час виконання команди `PlanePoint Ob3 = Ob1`, яка забезпечує створення нового об'єкта `Ob3` і копіювання у нього значень `x=5,y=8` об'єкта `Ob1`, у чому переконуємося шляхом виведення на екран значення `x=5`, отриманого методом `Ob3.GetX()`. Далі команда `PlanePoint Ob4(5,5)` створює об'єкт `Ob4`, а команда `len = GetLength(Ob4)` передасть цей об'єкт з головної функції у зовнішню функцію `double GetLength(PlanePoint tmp)`, яка обчислить і передасть у змінну `len` головної функції значення віддалі від точки `M(5,5)` до точки `O(0,0)`. Головна функція виведе це значення на екран.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
#include <windows.h>
// клас PlanePoint
class PlanePoint {
  double x, y;
  public:
   // конструктор класу без параметрів:
   PlanePoint(void);
   // конструктор класу з двома параметрами:
   PlanePoint(double nx, double ny);
   // конструктор копіювання класу PlanePoint -
   // передається посилання на об'єкт класу PlanePoint:
   PlanePoint(PlanePoint & ref_Point);
   // методи, реалізовані в класі
   int GetX(void) { return x; };
   int GetY(void) { return y; };
};
PlanePoint:: PlanePoint(void) {
 x =7;
 y = 21;}
PlanePoint:: PlanePoint(double nx, double ny) {
  x = nx;
  y = ny;}
PlanePoint:: PlanePoint(PlanePoint & ref_Point) {
  // створення нового об'єкта та копіювання даних в нього
  x = ref_Point.x;
  y = ref_Point.y;}
// зовнішня функція, що обчислює відстань від точки
// з координатами tx, ty до початку координат
// об'єкт PlanePoint mp є вхідним параметром функції
double GetLength(PlanePoint tmp);
int main() {
  clrscr();
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);
  int d;
  double len;
  // створення об'єкта Ob1 за допомогою конструктора
  PlanePoint Ob1(5, 8);
  // створення об'єкта Ob2 – викликається
  // конструктор за замовчуванням
  PlanePoint Ob2;
  // Перевірка роботи конструкторів
   d = Ob1.GetX();
  cout<<"\n Працює функція Ob1.GetX():";
```

```
  cout<<"\n Абсциса точки об\'єкта Ob1 дорівнює "<<d<<endl;
  cout<<"\n Натисніть будь-яку клавішу, щоб продовжити...\n";
  getch();
  d = Ob2.GetX();
  cout<<"\n Працює функція Ob2.GetX():";
  cout<<"\n Абсциса точки об\'єкта Ob2 дорівнює "<<d<<endl;
  cout<<"\n Натисніть будь-яку клавішу, щоб продовжити...\n";
  getch();
  // Побітове копіювання: конструктор копіювання не викликається
  Ob2 = Ob1;
  // дані скопіювались, але не за допомогою
  // конструктора копіювання
  d = Ob2.GetX();
  cout<<"\n Працює функція Ob2.GetX():";
  cout<<"\n після побітового копіювання Ob2 = Ob1:";
  cout<<"\n Абсциса точки об\'єкта Ob2 дорівнює "<<d<<endl;
  cout<<"\n Натисніть будь-яку клавішу, щоб продовжити...\n";
  getch();
  // Код, що викликає до роботи конструктор копіювання
  PlanePoint Ob3 = Ob1;
  // Створюється новий об'єкт Ob3
  // і в нього копіюється об'єкт Ob1,
  // для чого викликається конструктор копіювання
  d = Ob3.GetX();
  cout<<"\n Працює функція Ob3.GetX() після ";
  cout<<"\n виклику конструктора копіювання PlanePoint Ob3 = Ob1";
  cout<<"\n Абсциса точки об\'єкта Ob3 дорівнює "<<d<<endl;
  cout<<"\n Натисніть будь-яку клавішу, щоб продовжити...\n";
  getch();
  PlanePoint Ob4(5,5);
  // оголошено об'єкт Ob4 - екземпляр класу PlanePoint
  // передача об'єкта Ob4 у функцію GetLength,
  // викликається конструктор копіювання
  len = GetLength(Ob4);
  cout<<"\n Працює конструктор копіювання PlanePoint tmp, який";
  cout<<"\n створює об'єкт tmp і копіює його параметри в об'єкт Ob4";
  cout<<"\n функції GetLength(Ob4), що обчислює віддаль між точками.";
  cout<<"\n Віддаль від точки Ob4(5,5) до початку координат: ";
  cout<<"\n len = "<<len<<endl;
  cout<<"\n Розв'язок задачі завершено!"<<endl;
  getch();
  return 0;
}
// зовнішня функція GetLength(PlanePoint tmp)
double GetLength(PlanePoint tmp) {
  double length;
  double tx, ty;
  tx = tmp.GetX();
  ty = tmp.GetY();
  length = pow((tx*tx + ty*ty),1./2.);
  return length;}
```

Результати тестування програми:

```
Працює функція Ob1.GetX():
Абсциса точки об'єкта Ob1 дорівнює 5
Натисніть будь-яку клавішу, щоб продовжити...
Працює функція Ob2.GetX():
Абсциса точки об'єкта Ob2 дорівнює 7
Натисніть будь-яку клавішу, щоб продовжити...
Працює функція Ob2.GetX():
після побітового копіювання Ob2 = Ob1:
Абсциса точки об'єкта Ob2 дорівнює 5
```

```
Натисніть будь-яку клавішу, щоб продовжити...
Працює функція Ob3.GetX() після
виклику конструктора копіювання PlanePoint Ob3 = Ob1
Абсциса точки об'єкта Ob3 дорівнює 5
Натисніть будь-яку клавішу, щоб продовжити...
Працює конструктор копіювання PlanePoint, який
створює об'єкт tmp і копіює в нього дані об'єкта Ob4.
Функція GetLength(Ob4)обчислює віддаль між точками.
Віддаль від точки Ob4(5,5) до початку координат:
len = 7.07107
Розв'язок задачі завершено!
```

Програма, код якої подано нижче, розроблена на базі класу `Person` з конструктором за замовчуванням – він у програмі не згадується - та двох зовнішніх функцій `void rename(Person &x, string s)` та `void print(const Person &x)`. Як бачимо з прототипів обох функцій, вони в якості формального параметра мають посилання на об'єкт класу `Person`. У цій програмі потрібно змінити одне ім'я персони на інше.

Після запуску програми на виконання команда `Person ob` створює об'єкт `ob`, а команда `ob.name="Петро"` задає ім'я персони. Команда виклику зовнішньої функції `rename(ob, s)` передає об'єкт `ob` у зовнішню функцію `void rename(Person &x, string s)`, яка змінить ім'я персони на інше і поверне його у головну функцію. Команда виклику зовнішньої функції `print(ob)` передає об'єкт `ob` у зовнішню функцію `void print(const Person &x)`, яка виведе змінене ім'я персони на екран.

```
#include <string>
#include <iostream.h>
#include <conio.h>
#include <windows.h>
using namespace std;
class Person {
  public:
    string name;
};
// Функції, що не входять в клас Person
void rename(Person &x, string s);
void print(const Person &x);
int main() {
  clrscr();
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);
  string s;
  // Задання імені персони за допомогою
  // об'єкта ob класу Person
  cout<<"\n Задання імені персони за допомогою";
  cout<<"\n об'єкта ob класу Person: \n";
  Person ob;
  ob.name="Петро";
  print(ob);
  cout<<"\n Натисніть будь-яку клавішу, щоб продовжити...\n";
  getch();
  // Зміна імені персони за допомогою
  // зовнішньої функції rename(ob, s):
  cout<<"\n Зміна імені персони за допомогою";
  cout<<"\n зовнішньої функції rename(ob, s): \n";
  s= "Микола";
  rename(ob, s);
  print(ob);
  cout<<"\n Розв'язок задачі закінчено!\n";
  getch();
  return 0;
}
void rename(Person &x, string s) {
```

```
 x.name = s;
};
void print(const Person &x) {
 cout<<"\n "<<x.name<<endl;
};
Результати тестування програми:
 Задання імені персони за допомогою
 об'єкта ob класу Person:
 Петро
 Натисніть будь-яку клавішу, щоб продовжити...\
 Зміна імені персони за допомогою
 зовнішньої функції rename(ob, s):
 Микола
 Розв'язок задачі закінчено!\
```

Програма, код якої подано нижче, також розроблена на базі класу `Person` з конструктором за замовчуванням та двох зовнішніх функцій `Person create1()` та `Person create2()`. кожна з яких має такий самий тип, як і клас. Це означає, що ці дві функції повертають об'єкти. Як бачимо з прототипів обох функцій, вони не мають формальних параметрів. Призначення першої функції полягає у тому, що вона повертає об'єкт з порожніми (обнуленими) полями. Призначення другої функції полягає у тому, що вона повертає об'єкт з наперед заданими полями. У цій програмі потрібно обнулити або ж задати ім'я персони.

Після запуску

```
#include <string>
#include <iostream.h>
#include <conio.h>
#include <windows.h>
using namespace std;
class Person {
  public:
    string name;
};
// Функції, що не входять в клас Person
Person create1();
Person create2();
int main() {
  clrscr();
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);
  // Задання імені персони за допомогою
  // об'єкта ob класу Person
  Person ob;
  ob.name="Peter";
  cout<<"\n Задання імені персони за допомогою";
  cout<<"\n об'єкта ob класу Person: \n";
  cout<<" ob.name="<<ob.name<<endl;
  cout<<"\n Натисніть будь-яку клавішу, щоб продовжити...\n";
  getch();
  // Передано об'єкт класу Person з обнуленими полями
  cout<<"\n Передано об\'єкт класу Person ";
  cout<<"\n з обнуленими полями:"<<endl;
  ob=create1();
  cout<<" ob.name="<<ob.name<<endl;
  cout<<"\n Натисніть будь-яку клавішу, щоб продовжити...\n";
  getch();
  // Передано об'єкт tmp класу Person з визначеними
  // у функції create2() полями
  cout<<"\n Передано об\'єкт tmp класу Person з визначеними ";
  cout<<"\n у функції create2() полями:"<<endl;
```

```
  ob=create2();
  cout<<" ob.name="<<ob.name<<endl;
  cout<<"\n Розв'язок задачі закінчено!\n";
  getch();
  return 0;
}
// Функція create1 поверне через своє ім'я
// об'єкт класу Person з обнуленими полями
Person create1() {
  return Person();
};
// Функція create2 поверне через своє ім'я
// об'єкт tmp класу Person з визначеними
// у тілі функції полями та їх значеннями
Person create2() {
  Person tmp;
  tmp.name = "Paul";
  return tmp;
};
Результати тестування програми:
 Задання імені персони за допомогою
 об'єкта ob класу Person:
 ob.name=Peter
 Натисніть будь-яку клавішу, щоб продовжити...
 Передано об'єкт класу Person
 з обнуленими полями:
 ob.name=
 Натисніть будь-яку клавішу, щоб продовжити...
 Передано об'єкт tmp класу Person з визначеними
 у функції create2() полями:
 ob.name=Paul
 Розв'язок задачі закінчено!
```

**Висновок.** В статті запропоновано комплекс програм мовою С++, за допомогою яких досліджується робота конструкторів різних типів на дані класу. Проблема вирішується з позицій об'єктно-орієнтованого програмування Наведені коди запропонованих програм та результати їх тестування

**Список бібліографічного опису.**

1. Архангельский А.Я. Программирование в С++ Builder – М.: ООО "Бином-Пресс", 2010. – 896 с.
2. Архангельский А.Я. Язык С++ в С++Builder. Справочное и методическое пособие –М: Бином, 2007. – 1012 с.
3. Бобровский С. Самоучитель программирования на языке С++ в системе Borland Builder 5.0. –М.: ДЕСС КОМ, I-PRESS, 2000. –272 с.
4. Браян В. Керніган, Денс М. Річі Мова програмування мовою С++. –Київ: КНЕУ, 2014. – 232 с.
5. Глинський Я.М., Анохін В.Є., Ряжська В.А. С++ і С++Builder – Львів:СПД Глинський, 2011. – 192 с.
6. Грицюк Ю.І., Рак Т.Є. Програмування мовою С++. - Львів: ЛБУ БЖД, 2011. - 292 с.
7. Дейтел Х.М., Дейтел П.Дж. Как программировать на С++. –М.: М.: ООО "Бином-Пресс", 201. – 1456с.
8. В. Ермолаев, Т. Сорока С++ Builder: Книга рецептов. –КУДИЦ–Образ, 2006.-208с.

**References.**

1. Arkhangelsk A.Y. Programming in C ++ Builder - M .: Binom-Press LLC, 2010. - 896 p.
2. Arkhangelsk A.Y. C ++ language in C ++ Builder. Reference and methodological manual –M: Binom, 2007. - 1012 p.
3. Bobrovsky S. Tutorial programming in C ++ in the Borland Builder 5.0 system. –M .: DESS COM, I-PRESS, 2000. –272 p.
4. Brian V. Kernigan, Dance M. Richi Mova program my C ++. –Kiev: KNEU, 2014. - 232 p.
5. Glinsky Ya.M., Anokhin V.Є., Ryazhska V.A. C ++ i C ++ Builder - Lviv: SPD Glinsky, 2011 .-- 192 p.
6. Gritsyuk Yu.I., Cancer T.Є. Programming my C ++. - Lviv: LBU BZD, 2011 .-- 292 p.
7. Daytel H.M., Daytel P.J. How to program in C ++. –M .: M .: Binom-Press LLC, 201. - 1456s.
8. V. Ermolaev, T. Soroka C ++ Builder: Recipe Book. ¬ – KUDIC – Obraz, 2006.-208s.