**Лабораторна робота №12. Розробка програм із використанням перевантаження операторів**

**Тема:** Розробка ієрархії класів з використанням перевантаження операторів та функцій

**Мета:** Засвоєння навичок розробки програм з використанням перевантаження функцій

**Порядок виконання роботи**

1. Ознайомитися з теоретичними основами розробки програм, що використовують перевантаження операторів та функцій.
2. Розробити ієрархією класів відповідно завдання з використанням перевантаження операторів.
3. Розробити програму, яка використовує ієрархію класів п.2 для роботи з не менш ніж двома об’єктами.
4. Розробити 2-3 тести для перевірки правильності роботи розробленої програми.
5. В першому рядку програми та заголовкового файлу повинні бути записаними в коментарі номер групи та прізвище, а також номер ЛР (через кому до попередньої).
6. Результати надсилати на електронну адресу викладача

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

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

Наприклад, 31-01Ivanov.cpp.

Тему в заголовку листа записати

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

**Строк відсилки ЛР ІПЗ-31 22.11.2020**

**ІПЗ-32 20.11.2020**

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

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

**Завдання**

Примітка: у всіх класах повинне бути хоча б одне поле, пам'ять під яке виділяється динамічно.

##### Описати базовий клас РЯДОК.

Обов'язкові поля класу:

* покажчик на char — зберігає адресу динамічно виділеної пам'яті для розміщення символів рядка;
* значення типу int — зберігає довжину рядка в байтах.

Обов'язкові методи повинні виконувати наступні дії:

* конструктор без параметрів;
* конструктор, що приймає як параметр Сі-рядок (закінчується нульовим байтом);
* конструктор, що приймає як параметр символ;
* конструктор копіювання;
* одержання довжини рядка;
* очищення рядка (зробити рядок порожнім);
* деструктор.

Описати похідний від РЯДОК клас ДЕСЯТКОВИЙ\_РЯДОК.

Рядки даного класу можуть містити тільки символи десяткових цифр і символи – і +, що задають знак числа. Символи – чи + можуть знаходитися тільки в першій позиції числа, причому символ + може бути відсутнім, у цьому випадку число вважається додатнім. Якщо в складі ініціалізуючого рядка будуть зустрінуті будь-які символи, відмінні від припустимих, ДЕСЯТКОВИЙ\_РЯДОК приймає нульове значення. Вміст даних рядків розглядається як десяткове число.

Обов'язкові методи:

* конструктор без параметрів;
* конструктор, що приймає як параметр Сі-рядок (закінчується нульовим байтом);
* конструктор копіювання;
* деструктор;
* метод, що визначає, чи можна представити дане число у форматі int;

Перевизначити наступні операції:

* присвоювання (=);
* віднімання (–) — арифметична різниця рядків;
* операція > — перевірка на більше (за значенням);
* операція < — перевірка на менше (за значенням);

Розроблювач вправі вводити кожне (з обґрунтуванням необхідності) число додаткових полів і методів.

Написати тестову програму, яка:

* динамічно виділяє масив покажчиків на базовий клас (4–6);
* в режимі діалогу заповнює цей масив покажчиками на похідні класи, при цьому екземпляри похідних класів створюються динамічно з завданням початкових значень;
* для створених екземплярів похідних класів виконує перевірку всіх розроблених методів з виводом вихідних даних і результатів на дисплей.

Для конструкторів копіювання кожного класу передбачити діагностичну печатку кількості його викликів у визначене місце дисплею (рекомендується використовувати статичні члени класу).

Режим діалогу забезпечується за допомогою ієрархічного меню.

**Теоретичні відомості**

**Перевантаження операторів**

Як і більшість інших мов, C++ підтримує набір операторів для вбудованих типів. Однак більшість концепцій, для яких звичайно використаються оператори, не є вбудованими типами C++, тому вони повинні бути представлені у вигляді типів, обумовлених користувачем. Наприклад, якщо вам потрібна комплексна арифметика, матрична алгебра, сигнальна логіка або символьні рядки в C++, ви користуєтесь механізмом класів для подання цих понять. Визначення операторів для таких класів іноді дозволяє програмістові реалізувати більш звичну і зручну форму запису для маніпулювання об'єктами, ніж та, котра доступна з використанням тільки базової функціональної форми запису. Наприклад,

class complex { *//* дуже спрощений клас complex

double re, im; public:

complex {double r, double i) re [r], in (i) {}

complex operator –(complex);

complex operator +(complex};

};

визначає просту реалізацію концепції комплексного числа. Клас complexпредставлений парою чисел із плаваючою крапкою подвійної точності й двома операторами + і \*. Програміст вводить оператори comptex::operator+() і complexv.operator\*(), які визначають зміст операцій додавання й множення. Наприклад, якщомають тип *complex,* b+cозначає b.operator+(c)*.* Тепер ми можемо записати комплексні вирази у формі, близької до загальноприйнятої:

voidf()

{

complex a=complex {1,3.1); complex b=complex {1.2,2);

a = b+c;

b = b+c\* a;

с = a\*b+complex (1,2)*;*

Виконуються звичайні правила пріоритету операцій, тому друга інструкція означає b=b+(c\*a)*,* а не b=(b+c)\*a*.*

Найбільш очевидне використання перевантажених операторів - застосування їх до конкретних типів. Однак користь від операторів, обумовлених користувачем, не обмежується тільки конкретними типами. Наприклад, проектування узагальнених і абстрактних інтерфейсів часто приводить до використання операторів, таких як ->, [] і ().

Операторні функції

Можна оголосити функції, що визначають зміст наступних операторів:

**+ - \* / % ^ &**

| ~ ! = < > +=

- = . \*= /= %= Л= &= |=

<< >> >>= «= == != <=

>= && || ++ — ->\* ,

-> [] () newnew[]deletedelete[]

Ім’я операторної функції починається із ключового слова **operator***.* Операторна функція оголошується й може бути викликана, як будь-яка інша функція. Використання операторної функції як оператора є просто скороченою формою її явного виклику. Наприклад:

void j(complex a, complex b)

{

complex c = a + b; *//* скорочена форма

complex d = a.operator+ (b); *//* явний виклик

}

При наявності попереднього визначення complexобидві інструкції є синонімами.

В С++ є можливість поширення дії стандартних операцій на операнди абстрактних типів даних. Для того, щоб перевизначити одну зі стандартних операцій для роботи з операндами абстрактних типів, програміст повинен написати функцію з ім'ям *operator знак*, де знак - позначення цієї операції (наприклад, *+ - | + =* і т.д.).

Проте в мові С++ існує декілька обмежень, накладених на перевизначення операторів:

1. Не можна створювати нові символи операцій.

2. Не можна перевизначити операції*:* :: ; \* (розіменування, а не бінарне множення); ?: ; sizeof; &&; ||.

3. Символи унарної операції не може використовуватися для зміни бінарної операції і навпаки. Наприклад, символ <<можна використовувати тільки для бінарної операції,!- тільки для унарної, а & - і для унарної, і для бінарної.   
 4. Перевизначення операцій не міняє ні їх пріоритетів, ні порядку їх виконання (зліва направо або справа наліво).

5. При перевантаженні операції комп'ютер не робить ніяких припущень про її властивості. Це означає, що якщо стандартна операція + = може бути виражена через операції + і =, тобто а + = b еквівалентно а = а + b, то для перевизначення операцій у загальному таких співвідношень не існує, хоча, звичайно, програміст може їх забезпечити.

6. Ніяка операція не може бути перевизначена для операндів стандартних типів.

7. Як для унарної, так і для бінарної операції число аргументів функції operator () має точно відповідати числу операндів цієї операції. Причому в перевантаження бінарного оператора прийнято передавати один аргумент, тому що другий - неявний. Його має будь-яка функція - член класу, той самий покажчик this - покажчик на об'єкт, для якого викликаний метод. Таким чином, в перевизначенні унарного оператора не слід передавати нічого. Зручно передавати значення параметрів у функцію operator () не за значенням, а по посиланню.

Розглянемо приклад використання перевантаження чотирьох операторів + - \* % .

**Лістинг програми**

#include <iostream>

#include <Windows.h>

using namespace std;

class Digit{

private:

int dig; // число

public:

Digit(){dig=0;}

Digit(int iDig){ dig=iDig;}

void Show(){cout<<dig<<"\n";}

//перевантажуємо чотири оператора

//зверніть увагу, усі оператори бінарні, тому ми передаємо в них один параметр - це операнд, який буде знаходитися праворуч від оператора у виразі

//лівий операнд передається за допомогою this

Digit operator+(const Digit &N)

{

Digit temp;

temp.dig=dig+N.dig;

return temp;

}

Digit operator-(const Digit &N)

{

Digit temp;

temp.dig=dig-N.dig;

return temp;

}

Digit operator\*(const Digit &N)

{

Digit temp;

temp.dig=dig\*N.dig;

return temp;

}

Digit operator%(const Digit &N)

{

Digit temp;

temp.dig=dig%N.dig;

return temp;

}

};

int main()

{ system("color F0");

// перевіряємо роботу операторів

Digit A(8),B(3);

Digit C;

cout<<"\Digit A:\n"; A.Show();

cout<<"\Digit B:\n"; B.Show();

cout<<"\noperator+:\n"; C=A+B; C.Show();

cout<<"\noperator-:\n"; C=A-B; C.Show();

cout<<"\noperator\*:\n"; C=A\*B; C.Show();

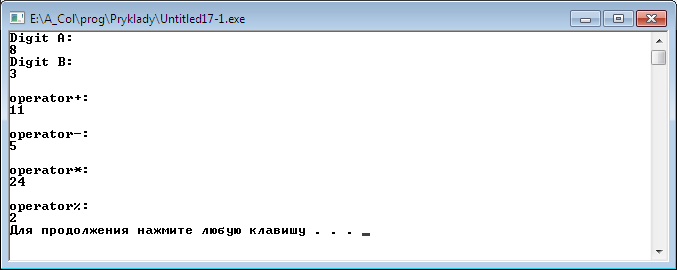
cout<<"\noperator%:\n"; C=A%B; C.Show();

system ("pause");

return 0;

}

**Результат роботи**



На відміну від бінарних і унарних операторів, оператор виклику функції може приймати будь-яку кількість аргументів (від 0 і більше), іншими словами кількість параметрів функції обмежується тільки здоровим глуздом. Коли оператор виклику функції перевантажений, він модифікує інтерпретацію застосування круглих дужок до об’єктів того класу, в якому він об’явлений, а не те, як функції викликаються.

Розглянемо приклад перевантаження оператора виклику функції.

**Лістинг програми**

#include <iostream>

#include <Windows.h>

using namespace std;

class MyPoint{

int X;

int Y;

public:

MyPoint(){ X=0; Y=0; }

// перевантажений оператор ()

operator()(int x,int y){ X=x; Y=y;}

Show(){ cout<<X<<" "<<Y<<"\n\n"; }

};

int main()

{ system("color F0");

MyPoint P;

P.Show();

//зміна значень полів об’єкта

P(2,3);

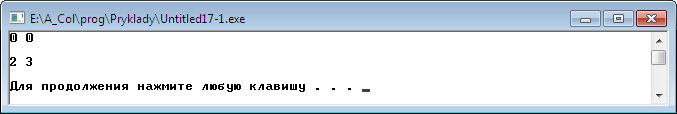
P.Show();

system("pause");

return 0;

}

**Результат роботи**



Приклад використання перевантаження оператора ***+*** на прикладі класу “Рядок”

**Лістинг програми**

#include<iostream>

#include<string.h>

#include <Windows.h>

Using namespace std;

class string\_

{

private:

char\* S; // рядок

int len; // довжина рядка

public:

// Конструктор за замовчуванням без параметрів

string\_();

// Перевантажений конструктор з параметром

string\_(char\* s);

// Конструктор копіювання

string\_(const string\_& s);

// Деструктор

~string\_(){delete [] S;}

// Метод сортування

Sort(string\_ s[], int n);

// Константний метод що повертає вміст строки

сonst char\* GetStr()const { return S;}

// метод, що дозволяє змінити вміст за допомогою користувача

SetStr()

{

// якщо рядок не порожній - очистити

if(S!=NULL)

delete[]S;

// створюємо массив і запитуємо у користувача дані

char a[256];

cin.getline(a,256);

// прораховуємо розмір

len=strlen(a)+1;

// виділяємо пам’ять

S = new char[len];

// переписуємо в об’єкт введену стрічку

strcpy(S,a);

}

// Перевантаження бінарного оператора

// Перший параметр передається неявно за допомогою покажчика this

// функція реалізує зчеплення стрічок

string\_ operator+(const string\_&);

// Перевантаження бінарного оператора

// Перший параметр передається неявно за допомогою покажчика this

//Функція реалізує коректне присвоєння обєктів один одному в ситуації обєкт1=обєкт2.

string\_&operator=(const string\_&);

// Перевантаження типу

// Функція реалізує перетворення обєкта класу до типу char\*

operator char\*() { return S; }

};

string\_::string\_()

{

// Початкова ініціалізація

S = NULL;

len = 0;

}

string\_::string\_(char\* sVar)

{

len = strlen(sVar);

S = new char[len + 1];

// Ініціалізація рядком, який переданий користувачем

strcpy(S, sVar);

}

string\_::string\_(const string\_& sVar)

{

len = sVar.len;

// Безпечне копіювання

S = new char[len + 1];

strcpy(S, sVar.S);

}

void string\_::Sort(string\_ s[], int n)

{

// Сортування рядків методом бульбашки

string\_ temp;

for(int i=0;i<n-1;i++)

{

for(int j=n-1;j>i;j--)

{

// порівняння двох рядків

if(strcmp(s[j].S,s[j-1].S)<0)

{

//запис рядка s[j] в temp

temp=s[j];

// запис рядка s[j-1] в s[j]

s[j]=s[j-1];

// запис рядка temp в s[j-1]

s[j-1]=temp;

}

}

}

}

// Функція зчеплення рядків (перевантажений бінарний плюс)

string\_ string\_::operator+(const string\_ &str)

{

// створення тимчасового об’єкта

string\_ sVar;

// обчислюємо нову довжину рядка

sVar.len = len + str.len;

// Виділяємо пам’ять під новий рядок

sVar.S = new char[s.len + 1];

// Ініціалізація першої частини рядка

strcpy(sVar.S, S);

//Ініціалізація другої частини рядка

strcat(sVar.S, str.S);

// Повернення нового обєкта

return sVar;

}

// Функція, що реалізує безпечне витягування

string\_& string\_::operator=(const string\_ &str)

{

// Передбачення варіанта STRING = STRING; //присвоєння самому собі,де STRING змінна класу string

if(this == &str) return \*this;

// якщо розміри рядків не співпадають, або рядок, в якому відбувається //запис, несформований

if(len != str.len || len == 0)

{

// видалення старого рядка

delete [] S;

// вирахування нової довжини рядка

len = str.len;

// виділення пам’яті під новий рядок

S = new char[len + 1];

}

// ініціалізація рядка

strcpy(S, str.S);

// повернення посилання на "самого себе"

// Завдяки цьому можливе багатократне присвоєння об’єктів один одному, наприклад, string\_ a, b, c; a = b = c;

return \*this;

}

int main()

{ system("color F0");

int n,i;

// Вводимo кількість рядків

cout<<"Input the number of string s:\t";

cin>> n;

if(n < 0)

{

cout<<"Error number:\t"<< n << endl;

return 0;

}

// Забираємо з потока символ Enter ("\n")

char c[2];

cin.getline(c, 2);

// Створюємо массив з n рядків

string\_ \*s = new string\_[n];

// Введення рядків з клавіатури

for(i = 0; i < n; i++)

s[i].SetStr();

// Сортування рядків

// Виклик через покажчик,так як функція працює для групи обєктів, а не для одного конкретного

s->Sort(s, n);

// Вивід відсортованих рядків

for(i = 0; i < n; i++)

cout<<"\n"<<s[i].GetStr()<<"\n";

// Видалення масиву рядків

delete [] s;

cout<<"\n\n\n\n";

//Перевіряємо на практиці оператор + і перетворення

string\_ A,B,C,RES;

A="Ivanov ";

B="Ivan ";

C="Ivanovich";

RES=A+B+C;

cout<<RES.GetStr()<<"\n\n";

system("pause");

return 0;

}

**Приклад виконання завдання 1**

Написати програму, яка демонструє роботу з перевантаженими операторами “**+**”, ”-“, ”**= =**“ на прикладі роботи з рядками.

Лістинг програми

class string{

public:

string(char \*); // конструктор

char \* operator +(char \*);

char \* operator -(char);

int operator ==(string);

void show\_string(void);

private:

char data[256];

};

int main(){

string title( "study C++");

string lesson("operators");

string str("operators");

if(title==lesson)

cout<<"title i lesson rivni"<<endl;

if(str == lesson)

cout <<"str i lesson rivni"<<endl;

if (title == str)

cout <<"title i str rivni"<< endl;

title.show\_string();

title = title + " I'm studing";

title.show\_string() ;

lesson.show\_string();

lesson = lesson - 'm';

lesson.show\_string();

system("pause");

}

string::string(char \*str)

{

strcpy(data, str);

}

char \* string::operator +(char \*str)

{

return(strcat(data, str));

}

char \* string::operator -(char letter)

{

char temp[256];

int i, j;

for (i = 0, j = 0; data[i]; i++)

if (data[i] = letter)

temp[j++] = data[i];

temp[j] = NULL;

return(strcpy(data, temp));

}

int string::operator ==(string str)

{

int i;

for (i = 0; data[i] == str.data[i]; i++)

if ((data[i] == NULL)&&(str.data[i] == NULL))

return(1); // рівно

return (0); //He рівно

}

void string::show\_string(void)

{

cout << data << endl;

}

### Приклад виконання завдання

Створити динамічний клас для роботи з рядками. Максимальна довжина послідовності – 254. Перший байт повинен містити інформацію про фактичну кількість елементів масиву. Здійснити перевантаження операцій:

" = "– присвоєння,

" + " – конкатенація (з’єднання) рядків,

" <= " – відношення "менше чи дорівнює",

" >= " – відношення "більше чи дорівнює",

" == " – відношення "дорівнює",

" != " – відношення "не дорівнює".

" <= " – відношення включення (перевіряється, чи є множина, надана лівим операндом, підмножиною множини, наданої правим),

" == " – еквівалентність множин,

" != " – нееквівалентність множин,

" << " – належність (перевіряється, чи належить значення, що надане лівим  операндом, до множини, що надана правим операндом).

**Лістинг програми**

#include <iostream>

#include <conio.h>

#include <stdio.h>

#include <windows.h>

using namespace std;

char bufRus[256];

char\* Rus(const char\* text)

{

CharToOem(text,bufRus);

return bufRus;

}

class String{

char \*str;

public:

String();

String &operator = (String &obj);

bool operator>= (String &obj);

bool operator<= (String &obj);

bool operator == (String &obj);

bool operator != (String &obj);

bool operator<< (char ch);

bool operator = (char \*s);

friend String &operator +(String &A, String &B);

friend ostream &operator<<(ostream &out,String obj);

friend istream &operator>>(istream &in,String obj);

};

String ::String()

{

str=new char [100];

str[0]=strlen(str);

}

String &String::operator =(String &obj)

{

if(this==&obj) return \*this;

else

{

delete [] str;

str=new char[(obj.str[0]+1)];

int i;

for(i=0; i<=obj.str[0]; i++)

str[i]=obj.str[i];

return \*this;

}

}

String &operator + (String &A, String &B)

{

String S;

delete [] S.str;

S.str=new char[(A.str[0]+B.str[0]+1)];

int i;

for(i=0; i<=A.str[0]; i++)

{

S.str[i]=A.str[i];

}

for(i=0; i<=B.str[0]; i++)

{

S.str[i+strlen(A.str)]=B.str[i];

}

return S;

}

bool String::operator<=(String &obj)

{

if(obj.str[0]<=str[0]) return true;

else return false;

}

bool String::operator>=(String &obj)

{

if(obj.str[0]>=str[0]) return true;

else return false;

}

bool String::operator==(String &obj)

{

if(str[0]==obj.str[0]) return true;

else return false;

}

bool String::operator !=(String &obj)

{

if(str[0]!=obj.str[0]) return true;

else return false;

}

bool String::operator<<(char ch)

{

int i, n=0;

for(i=1; i<=str[0]; i++)

{

if(str[i]==ch) return true;

else n++;

}

if(n==str[0]) return false;

}

bool String::operator = (char \*s)

{

if(strstr(str,s)!=0) return true;

else return false;

}

ostream &operator<< (ostream &out, String obj)

{

out<<obj.str<<endl;

return out;

}

istream &operator>> (istream &in, String obj)

{

cout<<Rus("Введіть стрічку:");

in>>obj.str;

return in;

}

int main(){

String A,B;

cin>>A;

cin>>B;

cout<<Rus("\n\tПеревантаження operator>=")<<endl;

if(A>=B)

cout<<Rus("1-ша стрічка більше 2-ї стрічки”)<<endl;

else

cout<<Rus("1-ша стрічка НЕ більше 2-ї стрічки")<<endl;

cout<<Rus("\n\tПеревантаження operator<=")<<endl;

if(A<=B)

cout<<Rus("1-ша стрічка менше 2-ї стрічки")<<endl;

else

cout<<Rus("1-ша стрічка НЕ менше <= 2-ї стрічки")<<endl;

cout<<Rus("\n\tПеревантаження operator==")<<endl;

if(A==B)

cout<<Rus("1-ша стрічка рівна 2-й стрічці")<<endl;

else

cout<<Rus("1-ша стрічка НЕ рівна 2-й стрічці")<<endl;

cout<<Rus("\n\tПеревантаження operator!=")<<endl;

if(A!=B)

cout<<Rus("1-ша стрічка НЕ рівна 2-й стрічці")<<endl;

else

cout<<Rus("1-ша стрічка рівна 2-й стрічці")<<endl;

char ch;

cout<<Rus("\n\tПеревантаження operator<<")<<endl;

cout<<Rus("Введіть символ: "); cin>>ch;

if(A<<ch)

cout<<Rus("Символ входить в 1-шу стрічку")<<endl;

else

cout<<Rus("Символ НЕ входить в 1-шу стрічку")<<endl;

char s[20];

cout<<Rus("\n\tПеревантаження operator=")<<endl;

cout<<Rus("Введіть підстрічку: "); cin>>s;

if(A=s)

cout<<Rus("1-ша стрічка входить в підстрічку")<<endl;

else

cout<<Rus("Підстрічка входить в 1-шу стрічку")<<endl;

cout<<endl;

getch();

return 0;

}

**Обмеження щодо перевантаження операторів.**

Оператори присвоювання (=), індексу ([]), виклику функції (()) та вибору члену (->) перевантажуються через методи класу — це вимога мови C++.

Перевантаження операторів через методи класу не використовується, якщо лівий операнд не є класом (наприклад, int), або, якщо це клас, який ми не можемо змінити (наприклад, std::ostream).

**Приклади невизначеності при перевантаженні функцій**

Основна причина, появи невизначеності, пов’язана з автоматичним перетворенням типів в С++.

Приклад 1.

**#include <iostream.h>**

**float myfunc(float i);**

**double myfunc(double i);**

**int main()**

**{**

**cout << myfunc (10.1) << " "; // невизначеності нема,**

**// виклик myfunc(double)**

**cout << myfunc(10); // невизначеність до якого типу**

**// приводити число float чи double**

**return 0;**

**}**

**float myfunc (float i)**

**{**

**return i;**

**}**

**double myfunc(double i)**

**{**

**return -i;**

**}**

|  |  |  |
| --- | --- | --- |
| float |  | 4 байта |
| double |  | 8 байт |

Приклад 2.

**#include <iostream.h>**

**char myfunc (unsigned char ch);**

**char myfunc(char ch);**

**int main()**

**{  
cout << myfunc('с'); // виклик myfunc(char)**

**cout << myfunc(88) << " "; // до якого типу приводити**

**// число char або unsigned char**

**return 0;  
}  
char myfunc (unsigned char ch)  
{  
return ch-1;  
}  
char myfunc(char ch)  
{  
return ch+1;  
}**

|  |  |
| --- | --- |
| **char** | **-128...127** |
| **unsigned char** | **0...255** |

Приклад 3.

**#include <iostream.h>  
int myfunc (int i);  
int myfunc (int i, int j = 1);  
int main()  
{  
cout << myfunc (4, 5) << " "; // невизначеності нема  
cout << myfunc(10); // невизначеність  
return 0;  
}  
int myfunc (int i)  
{  
return i;  
}  
int myfunc (int i, int j)  
{  
return i\*j;  
}**

**Контрольні запитання**

* 1. Сутність перевантаження функцій.
  2. Область використання перевантаження функцій.
  3. Чи можна перевантажити конструктор? Якщо так, то для чого?
  4. Наведіть приклад перевантаження конструктора?
  5. Чи можна перевантажити деструктор?
  6. Що таке аргумент по замовчуванню?
  7. Для чого можуть вводитись аргументи по замовчуванню?
  8. Приклад використання аргументу по замовчуванню.
  9. При використані перевантаження може виникати неоднозначність?
  10. Як задати адресу перевантаженої функції?

## Що таке перевантаження операцій?

## Які операції можуть бути перевантажені?

## Які операції не можуть бути перевантажені?

## Для чого використовується ключове слово operator?

## [Що таке **унарні** та **бінарні оператори**?](https://www.bestprog.net/uk/2019/02/11/operator-overloading-in-c-operator-function-keyword-operator-overload-of-basic-arithmetic-operators-ua/#q01)

## [Які **обмеження** накладаються на перевантажені оператори?](https://www.bestprog.net/uk/2019/02/11/operator-overloading-in-c-operator-function-keyword-operator-overload-of-basic-arithmetic-operators-ua/#q08)

1. Чи можна реалізувати дві і більше операторних функцій в класі, які перевантажують однаковий оператор і отримують різні (відмінні між собою) параметри?