**Лекція 17. Перевантаження функцій та операторів. Базові поняття та основні принципи**

В ході вивчення С++ розглядаються:

* перевантаження функцій,
* перевантаження операторів /операцій,

Обидві ці можливості є проявом поліморфізму С++.

**Перевантаження функцій**.

Перевантаження (overloading) - це можливість одночасно використовувати декілька різних функцій з одним і тим же іменем. Перевантажені функції – це функції з одним й тим же ім'ям, що мають різні списки параметрів. Набори параметрів можуть вирізнятися порядком слідування, типами, кількістю. Таким чином перевантаження дозволяє уникнути дублювання імен функцій, які виконують схожі дії, але за різними алгоритмами, з різною логікою програми. Тип повертаємого функцією значення до уваги не береться. Наприклад:

void f(int);

void f(char);

void f(long);

void f(float, int);

void f(int, int, int);

Якщо визначені декілька функцій з однаковим ім'ям та різними списками параметрів (перевантажені функції) і в програмі зустрічається виклик функції, компілятор повинен вибрати одну з перевантажених функцій. Існує певний алгоритм вибору функцій, у відповідності з яким вибирається функція, яка найкращим чином відповідає виклику. Якщо не буде встановлена відповідність жодній з перевантажений функцій чи буде встановлена неоднозначна відповідність, на етапі компіляції генерується повідомлення про помилку.

**Обмеження на перевантажені функції**:

1. Будь-які перевантажені функції повинні мати різні списки параметрів (при цьому аргумент даного типу і посилання на цей тип розглядаються як одне те саме).
2. Не допускається перевантаження функцій із співпадаючими списками параметрів лише на основі типу повертаємих значень.
3. Функції-члени не можуть бути перевантажені лише на основі того, що одна з них – статична, а друга – ні.
4. Typedef- визначення не впливають на механізм перевантаження, оскільки вони не вводять нові типи, а є лише синонімами існуючих. Наприклад, якщо є визначення

typedef  char \*ptr;

то дві функції

void setval (char \*s)  та  void setval ( ptr s)

не є різними. Це буде помилкою.

Більш детально про Typedef наприкінці лекції.

1. Всі еnum-типи[[1]](#endnote-1) розглядаються як різні і тому можуть використовуватись для перевантаження функцій.
2. Типи “масив (чогось)” і “покажчик (на щось)” розглядаються як ідентичні з точки зору перевантаження.  Наприклад,

void setval(char pz);

void setval (char \*ptr);

є помилкою при перевантаженні.

Але це стосується тільки одномірних масивів. Для багатомірних масивів друга, третя,. розмірності розглядаються як частина типу даних.

Наприклад, не буде помилки:

void setval (char sz [ ]);

void setval (char sz [ ][4]);

**Перевантаження операторів**.

**Перевантаження оператора** полягає в зміні сенсу оператора (наприклад, оператора плюс (+), який звичайно в C++ використовується для додавання) при використанні його з певним класом.

**Синтаксис операторних функцій**

<тип\_значення\_що\_повертається> **operator** <символ\_операції(параметри)>

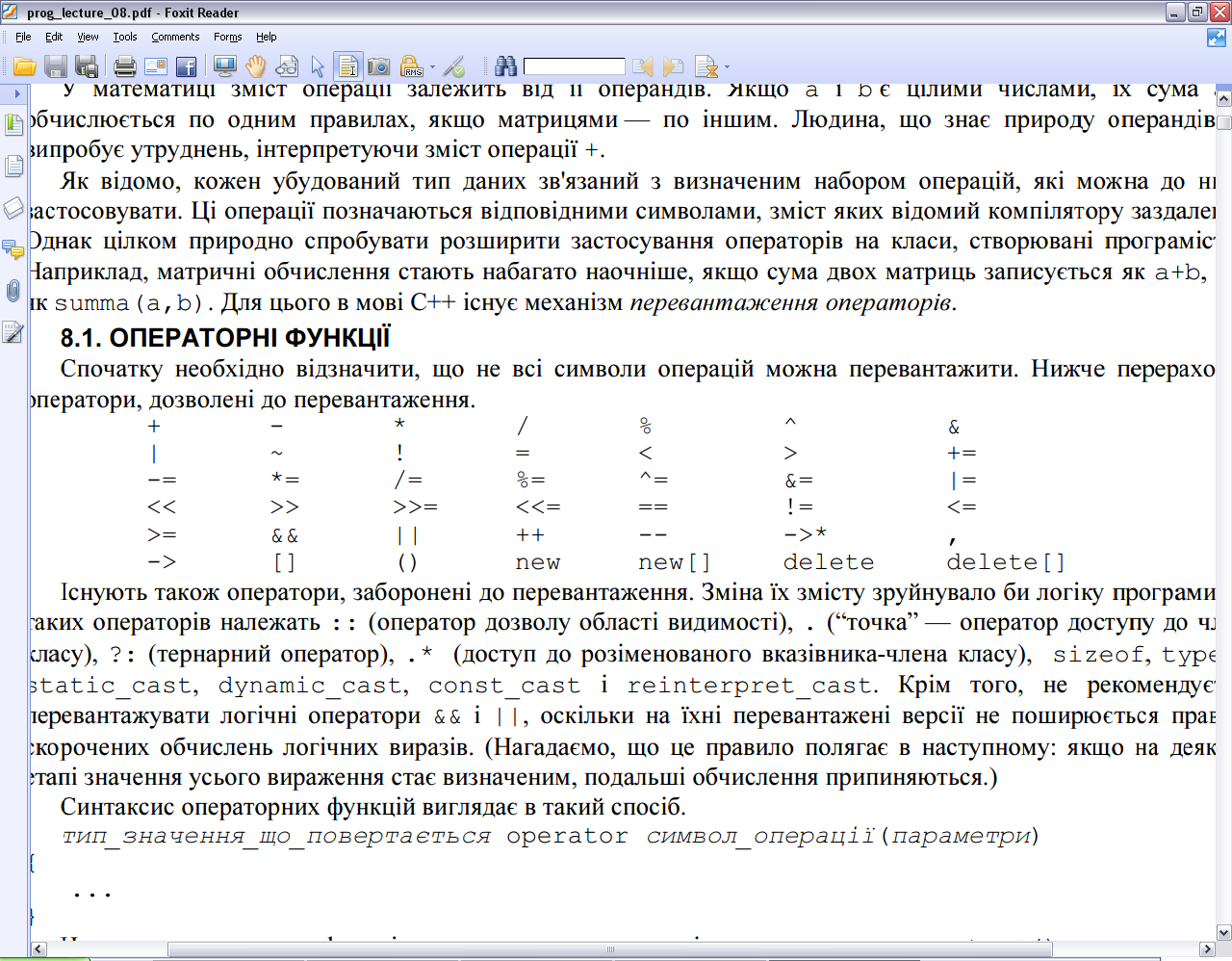
{

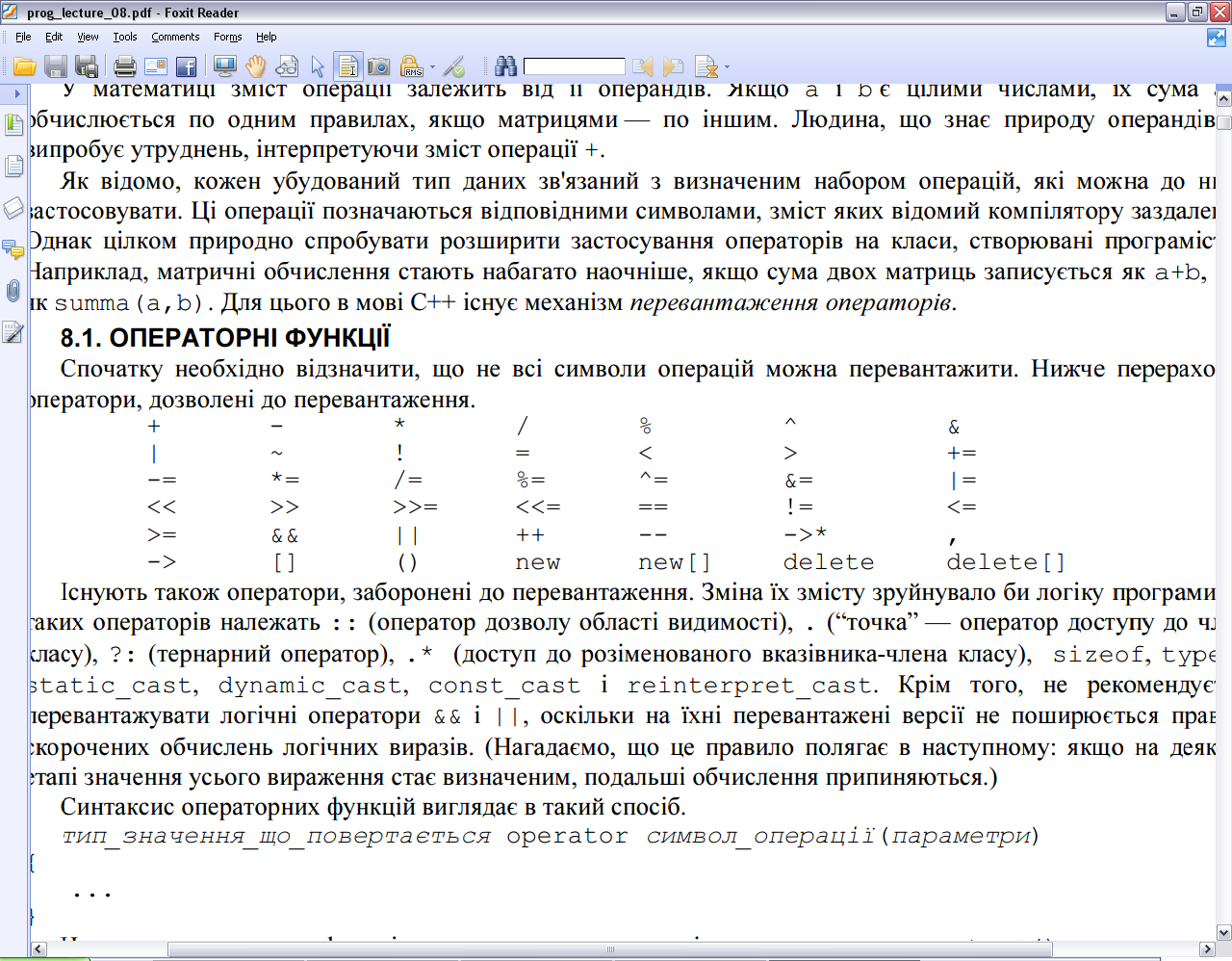
...

}

Наприклад, операторна функція, що перевантажує операцію +, називається operator+().

*Оператори, дозволені до перевантаження*





*Оператори, заборонені до перевантаження*

Існують також оператори, заборонені до перевантаження. Зміна їх змісту зруйнувало би логіку програми. До таких операторів належать

**::** (оператор дозволу області видимості),

**.**  (“точка” — оператор доступу до члена класу),

**?:**  (тернарний оператор),

**.\***  (доступ до розіменованого вказівника-члена класу),

**sizeof, typeid, static\_cast, dynamic\_cast, const\_cast і reinterpret\_cast.**

Крім того, не рекомендується перевантажувати логічні оператори **&&** і **||**, оскільки на їхні перевантажені версії не поширюється правило скорочених обчислень логічних виразів.

**Обмеження щодо перевантаження операторів**.

1. Не можна перевантажувати операції стандартних типів даних[[2]](#footnote-1), лише власних: перерахування, об’єднання, структури чи класу
2. Кількість операндів фіксована: жодного, один чи два.
3. Не можна змінювати пріоритет перевантажуваних операцій. Змінити пріоритет перевантажуваних операцій можна за допомогою дужок ().
4. Унарна операція має мати один операнд, а бінарна два. Єдина операція, яка не має фіксованої кількості операндів - це операція виклику функції (). Операції “+”, ”-”, “&”, “\*” можна перевантажувати і як бінарні, і як унарні.
5. Операції можна перевантажувати трьома способами: як методи класу, як незалежні зовнішні функції і як дружні функції. Лише як методи класу можна перевантажувати операції: “=”, “[]”, “()”, “->”, “\*”(розіменування), new, delete.
6. Якщо операція перевантажується як метод класу то її лівим аргументом є об’єкт класу (this). Зовнішні функції мають на один аргумент більше ніж аналогічні внутрішні. У зовнішніх функція покажчик this недоступний.
7. Значення операндів не можна задавати за замовчуванням.
8. Перевантажувати можна лише тільки існуючі операції, створювати нові операції не можна.

Операторні функції повинні мати прямий доступ до членів класу. Отже, необхідно, щоб вони були або членами класу, або дружніми функціями.

**Перевантаження унарних операторів за допомогою функцій-членів.**

Оператори можуть бути унарними і бінарними.

**Унарний оператор** має один операнд, а бінарний — два.

Нагадаємо, що до унарних операторів, що перевантажуються, належать такі оператори, як +, -, ++, --, &, ~ , !.

**До бінарних операторів**, що перевантажуються, належать всі інші оператори, перераховані в приведеній вище таблиці.

Операторні функції-члени, що перевантажують унарний оператор, мають одну особливість: їх операнди передаються неявно за допомогою вказівника this. Отже, така функція-член класу не має явних параметрів.

Оператори заперечення (!), взяття адреси (&) і побітового заперечення (~) допускають перевантаження, але не мають універсальних альтернатив, що варто було б реалізувати. Їх можна перевантажувати, наприклад, для підвищення наочності програми. Скажемо, за допомогою оператора ! можна позначати операцію обернення матриці, а за допомогою символу ~ — її транспонування. Щоправда, застосування тильди закріплене за деструкторами, тому варто виявляти обережність, щоб не створити плутанину. У будь-якому випадку зміст перевантаження операторів залежить від конкретної задачі.

**Що приймають та повертають оператори.**

1. Можуть приймати і повертати об’єкти власних типів, стандартних типів та користувацьких типів.
2. Об’єкт власного типу найчастіше приймається за константним посиланням.
3. Якщо операція повертає об’єкт власного типу, то може повертати за значенням чи за посиланням:

Перший тип 1. Створює новий об’єкт результат повертає його копію.

Другий тип 2. Змінює об’єкт операнд та повертає посилання на нього.

**Таблиця 1. Форма перевантаження операцій**

|  |  |  |
| --- | --- | --- |
| Всі унарні операції | -, +, ! ,++, -- | Як метод класу |
| Присвоєння і ін. | =, [], (), \*, ->, new, delete | Лише як метод класу |
| Комбіновані | -=, +=, \*=, /=, %=, &=, |=, >>=, <<= | Як метод класу |
| Всі бінарні операції | -, +, \*, /, %, >>, <<, &&, &, |, || | Як дружні зовнішні функції |
| Порівняння | ==, !=, <=, >=, <, > |

**Таблиця 2. Способи перевантаження операцій**

Позначки в графі 3.

1. Створює новий об’єкт результат повертає його копію
2. Змінює об’єкт операнд та повертає посилання на нього
3. Перевантажуються лише методами класу

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| Унарні | -, + | 1 |
| Бінарні | -, +, \*, /, %, >>, <<, &&, &, |, || | 1 |
| Порівняння | ==, !=, <=, >=, <, > | повертають 0 або 1 |
| Присвоєння | = | 2, 3 |
| Комбіновані | -=, +=, \*=, /=, %=, &=, |=, ^= | 2 |
| Префіксний інкремент та декремент | ++, -- (++x, --x) | 2 |
| Постфіксний інкремент та декремент | ++, -- (x++, x--)  \*приймає фіктивний параметр int | 1 |
|  | [],(),\*->,new, delete | 3 |

**Операція індексування**

При наявності поля-масиву в класі часто буває необхідно перевантажити операцію індексування []. Щодо неї існують такі обмеження:

* операція перевантажується тільки як метод класу;
* вона є бінарною;
* лівий аргумент - це поточний об'єкт, а правий - аргумент функції-операції.

Вираз об'єкт [індекс] трактується як:

**об'єкт.ореrаtоr [] (індекс)**

Операція [] (як і операція присвоювання) зобов'язана повертати посилання, оскільки вираз “ім’я [індекс]” може знаходитися як справа, так і зліва від операції присвоєння. Повинні виконуватися такі правила:

* тип індексу може не збігатися з типом, що повертається посиланням;
* тип посилання, що повертається не обов'язково повинно бути типом поточного класу;
* тип індексу не зобов'язаний бути цілочисловим.

Зазвичай реалізують два методи: константних і неконстантний.

**тип\_1& ореrаtоr[] (const тип\_2 &index)**

**const тип\_2& ореrаtоr[] (const тип\_2 &index) const**

Неконстантний метод працює, коли вираз “ім'я [індекс]” стоїть ліворуч від знаку операції присвоєння.

Константний метод викликається для параметрів, переданих по константному посиланню.

Розглянемо приклади перевантаження операторів.

**Приклад 1.** Нижче наведено визначення класу strings. Цей клас містить один елемент даних, який являє собою власне символьний рядок. Крім того, цей клас містить декілька перевизначених операторів:

#include <iostream>

#include <string.h>

#include <locale.h>

#include <Windows.h>

using namespace std;

class strings

{

public:

strings(char \*); // Конструктор

void operator +(char \*);//конкатенація

void operator -(char);//вилучення символу з тексту

void show\_string(void);

private:

char data[256] ;

};

strings::strings(char \*str){

strcpy(data, str); // копіювання рядка в інший рядок

}

void strings::operator +(char \*str){

strcat(data, str); // Об'єднання рядків

}

void strings::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;

strcpy(data, temp);

}

void strings::show\_string(void){

cout << "\n\t"<<data << endl;

}

void main(void){

system("color F0");

strings title( "\n\tbegin try to reload C++");

strings lesson("\n\tover\*\*\*loading \*\*\*");

title.show\_string();

title + " I try overload too!";

title.show\_string() ;

lesson.show\_string();

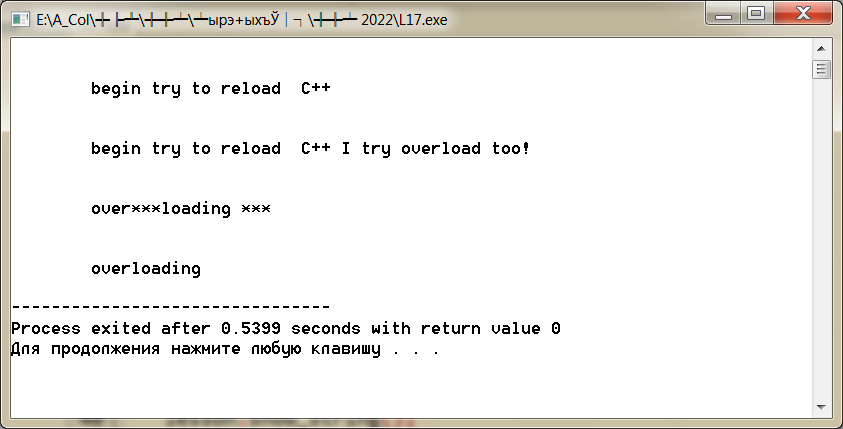
lesson -'\*';

lesson.show\_string();

system("pause");

}

Результати виконання програми.



##### Приклад 2. Приклад перевантаження унарних та бінарних операторів для класу, який містить одиночні дані. Операторна функція реалізована всередині класу

Оголошується клас Point, що реалізує точку на координатній площині. У класі реалізовано:

* дві внутрішні змінні x, y, що є координатами точки;
* два конструктори класу;
* методи доступу до внутрішніх змінних класу GetX(), GetY(), SetX(), SetY();
* дві операторні функції operator+() та operator-(). Операторна функція operator+() перевантажує бінарний оператор **‘+’**. Операторна функція operator-() перевантажує унарний оператор **‘-‘**.

#include <iostream>

#include <Windows.h>

using namespace std;

class Point

// Клас, що реалізує точку на координатній площині

// клас містить дві операторні функції

class Point

{

private:

int x, y; // координати точки

public:

// конструктори класу

Point()

{

x = y = 0;

}

Point(int nx, int ny)

{

x = nx;

y = ny;

}

// методи доступу до членів класу

int GetX(void) { return x; }

int GetY(void) { return y; }

void SetX(int nx) { x = nx; }

void SetY(int ny) { y = ny; }

// перевантажений бінарний оператор '+'

Point operator+(Point pt)

{

// p - тимчасовий об'єкт, який створюється з допомогою

// конструктора без параметрів

Point p;

p.x = x + pt.x;

p.y = y + pt.y;

return p;

}

// перевантажений унарний оператор '-'

Point operator-(void)

{

Point p;

p.x = -x;

p.y = -y;

return p;

}

};

Як видно з вищенаведеного коду, операторна функція operator+() отримує один параметр. Це означає, що ця функція реалізує бінарний оператор **‘+’**. Цей параметр відповідає операнду, що розміщується у правій частині бінарного оператора **‘+’**. Операнд, що розміщується в лівій частині оператора **‘+’** передається операторній функції неявно з допомогою покажчика this цього класу.

Виклик операторної функції здійснює об’єкт, який розміщується в лівій частині оператора присвоювання.

Демонстрація використання перевантажених операторів класу Point в іншому методі:

int main()

{system("color F0");

// оголошення змінних - об'єктів класу CPoint

Point P1(3,4);

Point P2(5,7);

Point P3;

int x, y; // додаткові змінні

// 1. Використання перевантаженого бінарного оператора '+'

P3 = P1 + P2; // об'єкт P1 викликає операторну функцію

// перевірка

x = P3.GetX(); // x = 8

y = P3.GetY(); // y = 11

cout << "x="<< x << " y="<< y << endl;

// 2. Використання перевантаженого унарного оператора '-'

P3 = -P2;

x = P3.GetX(); // x = -5

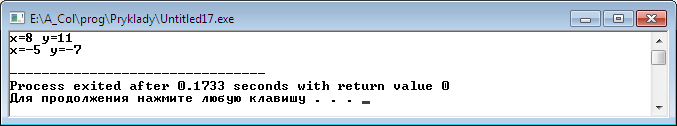
y = P3.GetY(); // y = -7

cout << "x="<< x << " y="<< y << endl;

return 0;

}

Результат роботи



У вищенаведеному коді в операції додавання **‘+’** об’єкт P1 викликає операторну функцію. Тобто фрагмент рядка

P1 + P2

замінюється викликом

P1.operator+(P2)

Реалізувати операторну функцію operator+() в класі можна й по іншому

Point operator+(Point pt)

{

// виклик конструктора з двома параметрами

return Point(x+pt.x, y+pt.y); // створюється тимчасовий об'єкт, який потім копіюється

}

У вищенаведеній функції в операторі return створюється тимчасовий об’єкт шляхом виклику конструктора з двома параметрами, який реалізований в класі. Якщо (у даному випадку) з тіла класу забрати конструктор з двома параметрами

// конструктор з двома параметрами

Point(int nx, int ny)

{

// ...

}

то вищенаведений варіант функції operator+() працювати не буде, тому що для створення об’єкту типу Point ця функція використовує конструктор з двома параметрами. У цьому випадку компілятор видасть повідомлення

Point::Point : no overloaded function takes 2 arguments

що означає: немає методу (конструктора) Point::Point() який приймає 2 аргументи.

##### Приклад 3. Приклад перевантаження оператора ‘\*’, що обробляє клас, який містить масив дійсних чисел. Операторна функція реалізована всередині класу

У прикладі реалізується операторна функція operator\*(), яка множить поелементно значення внутрішніх масивів об’єктів класу ArrayFloat. Якщо розмір масивів неоднаковий, то перемножується тільки та кількість елементів, яка є мінімальною між двома розмірами масивів.

#include <iostream>

#include <Windows.h>

using namespace std;

// масив дійсних чисел

class ArrayFloat

{

private:

float A[10]; // масив дійсних чисел, фіксований розмір масиву

int size;

public:

// конструктори

ArrayFloat()

{

size = 0;

}

ArrayFloat(int nsize, float nA[])

{

size = nsize;

for (int i=0; i<nsize; i++)

A[i] = nA[i];

}

// методи доступу

float GetAi(int i)

{

if ((i>=0) && (i<=size-1))

return A[i];

else

return 0;

}

void SetAi(int i, float value)

{

if ((i>=0) && (i<=size-1))

A[i] = value;

}

// перевантажений оператор '\*'

ArrayFloat operator\*(ArrayFloat AF)

{

ArrayFloat tmp;

int n;

if (size>AF.size)

n = AF.size;

else

n = size;

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

tmp.A[i] = A[i] \* AF.A[i];

tmp.size = n;

return tmp;

}

};

int main()

{system("color F0");

// додаткові змінні та масиви

float x, y;

float AF1[] = { 2, 5, 7, 9, 12 };

float AF2[] = { 3, 4, 9, 8, 10, 13 };

// створити об'єкти класу ArrayFloat

ArrayFloat A1(5, AF1);

ArrayFloat A2(6, AF2);

ArrayFloat A3;

// виклик операторної функції operator\*

A3 = A1 \* A2; // здійснюється поелементне множення

// перевірка

x = A3.GetAi(0); // x = 6

y = A3.GetAi(1); // y = 20

cout << "x="<< x << " y="<< y << endl;

x = A3.GetAi(2); // x = 63

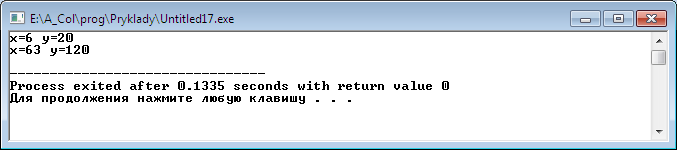
y = A3.GetAi(4); // y = 120

cout << "x="<< x << " y="<< y << endl;

return 0;

}

Результат роботи



На наступній лекції більш детально розглянемо перевантаження операторів на конкретних прикладах.

***Додаткова інформація***

Оператор **typedef** дозволяє перейменувати один з існуючих типів даних. На перший погляд, це зовсім зайве. Як правило, це роблять, щоб полегшити подальшу модифікацію програми. Наприклад, деяка програма виконує обчислення з числами типу float. У якійсь ситуації можуть знадобитися більш точні обчислення. Якщо не використовувати перейменування типу, прийдеться пройтися по всій програмі і замінити float на double. А от якщо в програмі тип float був перейменований, то його

псевдонім, скажемо real, стає деяким параметром, якому можна знову перевизначити. Для того щоб перейти до обчислень з підвищеною точністю, досить змінити лише один оператор: замість

typedef float real;

варто написати

typedef double real;

Тоді програма прийме наступний вид.

Перейменування типу

**#include <iostream.h>**

**int main()**

**{**

**typedef double real;**

**real a=1.0e300, b=1.0e200, c=a\*b;**

**cout << c << endl;**

**return 0;**

**}**

Ця програма дуже підступна. Значення змінних a і b лежать у припустимому діапазоні типу double, але їхній добуток виходить за його межі. Однак у мові С++ такі ситуації за замовчуванням не відслідковуються. Програма без коливань виводить на екран невірну відповідь: 1.127201e-231

Зовсім очевидно, що нам необхідно розширити діапазон, задавши тип long double. У приведеному вище прикладі для цього досить замінити оператор

typedef double real;

оператором

typedef long double real;

Тепер відповідь буде вірною; 1.0e+500

Звичайно, у такій короткій програмі застосування оператора typedef не занадто помітно, однак у великій програмі, що складається з багатьох файлів, він дозволяє уникнути стомлюючої перевірки і заміни оголошень змінних типу double на змінні типу long double.

Варто мати на увазі, що оператор typedef не створює новий тип, він просто створює псевдонім існуючого типу. До речі, це перейменування діє в межах усієї програми. Інакше кажучи, один тип можна перейменувати тільки один раз.

Розглянемо приклад. Повторне перейменування

**#include <iostream.h>**

**int main()**

**{**

**typedef double real;**

**real a=1.0e300, b=1.0e200, c=a\*b;**

**cout << c << endl;**

**typedef long double real; // Помилка! Повторне перейменування!**

**real d=1.0e300, e=1.0e200, f=d\*e;**

**cout << f << endl;**

**return 0;**

**}**

У той же час використання псевдоніму не забороняє застосування вихідного імені типу. Застосування псевдоніма і старого імені

#**include <iostream.h>**

**int main()**

**{**

**typedef long double real;**

**real a=1.0e400, b=1.0e500, c=a\*b;**

**cout << c << endl;**

**long double d;**

**d=c\*c;**

**cout << d << endl;**

**return 0;**

**}**

У результаті обчислень на екран будуть виведені числа.

1.0e+900

1.0e+1800

Оператор typedef дуже часто зустрічається в заголовних файлах. Він забезпечує машинну незалежність програм. Зокрема, саме за допомогою оператора typedef визначені типи wchar\_t і size\_t.

***Контрольні запитання для письмових відповідей (0,5 балів за запитання)***

1. Для чого використовують перевантаження функцій?
2. Які обмеження на перевантажені функції?
3. Як компілятор діє з перевантаженими функціями?
4. В чому сенс перевантаження операторів?
5. Наведіть приклади перевантаження унарних та бінарних операторів.
6. Які оператори не можна перевантажувати?
7. Для чого перевантажують конструктори класів?
8. Чи буде помилковим виклик f(1.5, 1.5); для функцій

void f(int, float);

void f(float, int);

***Для самостійного вивчення***: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

**Рекомендована література**

1. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL: <http://www.ph4s.ru/bookprogramir_1.html>
2. Страуструп Б. Язык программирования С++. — СПб.; М.: Невский диалект — ЗАО “Изд-во “Бином”, 1999.
3. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

4. Липпман С. Б., Лажойе Ж. Язык программирования С++: Вводный курс. — М.: ДМК, 2001. URL: <http://www.insycom.ru/html/metodmat/inf/Lipman.pdf>

5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL: <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>

6. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++ . [2-е изд.] / Буч Г. - СПб.: Невский Диалект, 1998. - 560 с.

7. Бадд Т. Объектно-ориентированное программирование в действии. [2-е изд.] – СПб.: Изд-во "Питер". 1997. URL: <http://khizha.dp.ua/library/Timothy_Budd_-_Introduction_to_OOP_(ru).pdf>

8. Скотт Мейерс*.* Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов. *"ДМК", 2000; 240 с.*

1. enum Colors

   {

   COLOR\_YELLOW, // присвоюється 0

   COLOR\_WHITE,  // присвоюється 1

   COLOR\_ORANGE, // присвоюється 2

   COLOR\_GREEN,  // присвоюється 3

   COLOR\_RED,    // присвоюється 4

   COLOR\_GRAY,   // присвоюється 5

   COLOR\_PURPLE, // присвоюється 6

   COLOR\_BROWN   // присвоюється 7

   };

   int main()

   {

   Colors paint(COLOR\_RED);

   std::cout << paint;

   return 0;

   } [↑](#endnote-ref-1)
2. До основних типів мови C++ відносяться:

   * **int** (цілий);
   * **char** (символьний);
   * **bool** (логічний);
   * **float** (дійсний);
   * **double** (дійсний з подвійною точністю);
   * **void** (порожній, не має значення).

   [↑](#footnote-ref-1)