**Тема 12.** **Перевантаження операторів.** (4 години)

**Лекція 1. Перевантаження функцій та операторів в С++**

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.

1. Всі еnum-типи 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)

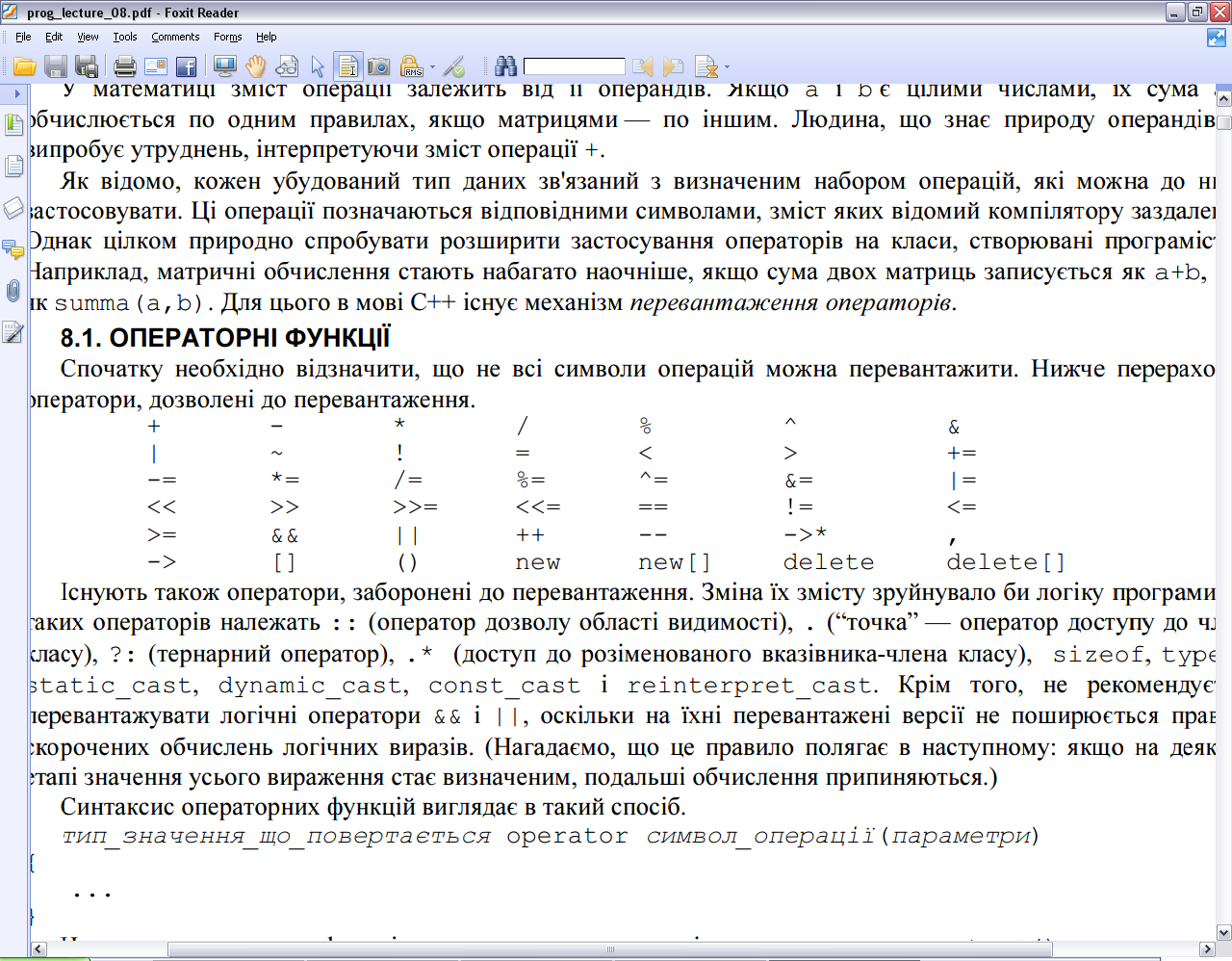
{

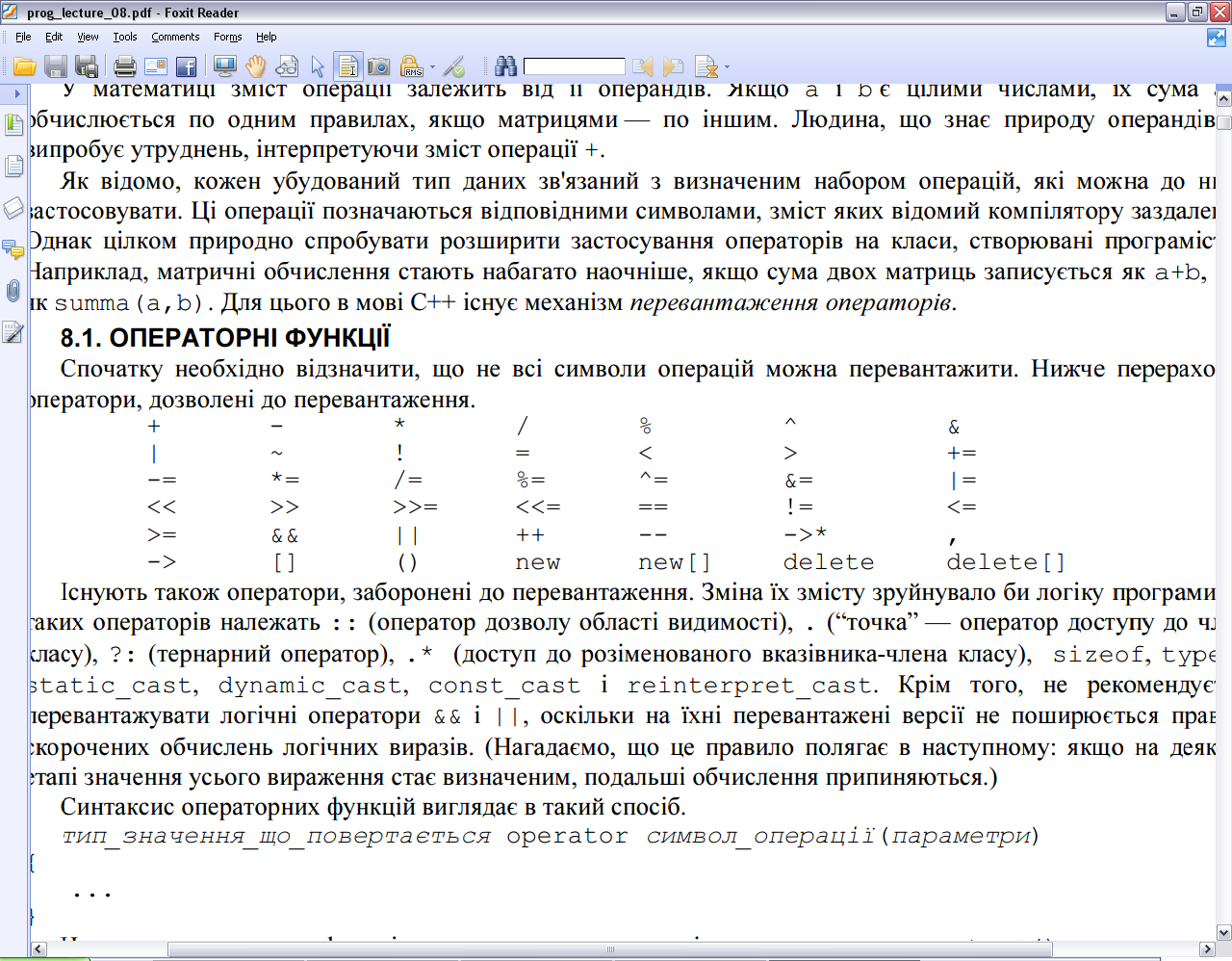
...

}

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, лише власних: перерахування, об’єднання, структури чи класу
2. Кількість операндів фіксована: жодного, один чи два.
3. Не можна змінювати пріоритет перевантажуваних операцій. 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.
4. 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.
5. 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.
6. Значення операндів не можна задавати за замовчуванням.

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 +, -, ++, --, &, ~ і !.

**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 приклад перевантаження операторів.

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 \*);

void operator -(char);

void show\_string(void);

private:

char data[256] ;

};

strings::strings(char \*str){

strcpy(data, str);

}

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

strcat(data, str);

}

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\toverloading \*\*\*");

title.show\_string();

title + " I try overload too!";

title.show\_string() ;

lesson.show\_string();

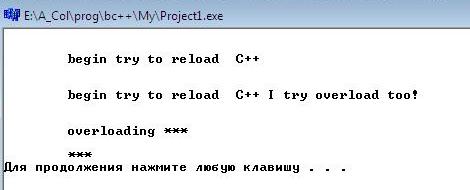
lesson -'\*';

lesson.show\_string();

system("pause");

}

Результати виконання програми.



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

*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.

*Література* [1, 4, 11, 12, 18]

*Контрольні запитання для самоперевірки*.

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?

*Контрольні запитання для надання письмових відповідей*.

1. Чи буде помилковим виклик f(1.5, 1.5); для функцій

void f(int, float);

void f(float, int);

Відповідь поясніть.

1. Які дії виконують перевантажені оператори в наведеному наприкінці лекції прикладі.
2. Наведіть власні приклади перевантаження функцій.
3. Наведіть власні приклади перевантаження операторі.