МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА

МЕТОДИЧНІ ВКАЗІВКИ

до практичних занять та самостійної роботи

з навчальної дисципліни

«ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

(для студентів освітньо-кваліфікаційного рівня бакалавр у галузі знань 12 – Інформаційні технології спеціальності 122 – Комп'ютерні науки та інформаційні технології)

Харків

ХНУМГ ім. О.М. Бекетова

2017

Методичні вказівки до практичних занять та самостійної роботи з навчальної дисципліни «Об’єктно-орієнтоване програмування» (для студентів освітньо-кваліфікаційного рівня бакалавр у галузі знань 12 – Інформаційні технології спеціальності 122 – Комп’ютерні науки та інформаційні технології) / І.Л. Яковицький ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2017. – 34 с.

Автор – канд. техн. наук, доц. І. Л. Яковицький

Рецензент д-р фіз.-мат. наук, проф. кафедри мікроелектроніки, електронних приладів та пристроїв Харківського національного університету радіоелектроніки О. В. Грицунов.

Рекомендовано кафедрою прикладної математики і інформаційних технологій, протокол № 2 від 29.08.2015 р.

Зміст

Типи даних створювані програмістом

Структури. Класи

Тип є уявлення деякої концепції (поняття). Наприклад, тип float з операціями: +, -, *, / і т.і. забезпечує обмежену, але конкретну версію математичного поняття дійсного числа.

Мова С ++ надає можливості для визначення нового типу даних, який описує деяке поняття.

Наприклад, одним з таких понять є «Календарна дата». У повсякденному житті ми часто виконуємо різні операції з цим поняттям.

Спробуйте відповісти на наступні питання:

1. Який день тижня сьогодні?
2. В який день тижня Ви народилися?
3. Яка подія світової важливості відбулася 29.02.2001 року?
4. Скільки днів минуло з дати Вашого народження до сьогоднішнього дня?
5. Скільки Вам буде років 13 жовтня 2020 року?
6. Як часто Новий рік починався у неділю в минулому столітті?
7. Скільки недільних днів було в грудні 2000 року?
8. Скільки днів тривала Вітчизняна війна?

    - Початок війни 22 червня 1941 року.
    - Закінчення війни 9 травня 1945 року.

9. Яка календарна дата буде через 147 днів від поточної дати?

Новий тип створюється для того, щоб дати визначення поняттю, якому прямо і очевидно серед вбудованих типів мови програмування ніщо не відповідає.

Для опису в програмі нового типу даних використовується поняття КЛАС. Призначення поняття класу, полягає в тому, щоб надати програмісту інструмент для створення нових типів, настільки ж зручних в обігу як і вбудовані типи. В ідеалі тип, визначений користувачем, способом використання не повинен відрізнятися від вбудованих типів, тільки способом створення.

Визначення 1. Клас - це визначений користувачем тип даних.

Для чого потрібні класи?

- для визначення поняття, для якого немає відповідного вбудованого типу даних мови програмування;
- для скорочення розміри програм;
- для формування більш чітких програм;
- дозволяє компілятору виявляти неприпустимі використання об’єктів, які в іншому випадку залишаться виявленими до тестування програми;
- щоб обмежити доступ до даних безліччю функцій доступу;
- щоб захистити структури даних,

Основна ідея визначення нового типу в тому, щоб відокремити несуттєві подробиці реалізації від значущих якостей, які важливі для правильного використання типу.

Такий поділ можна описати так, що робота з даними і внутрішніми адміністративними функціями здійснюється за допомогою спеціального інтерфейсу.

Клас - це визначений користувачем тип даних. Ми знайомимося з основними засобами визначення класу, створення об’єкту класу, роботи з такими об’єктами і, нарешті, знищення таких об’єктів після використання.

Зміст практикуму

У практикумі пропонується послідовно виконати 6 вправ і 11 завдань для самостійної роботи. Матеріали для роботи (тексти програмних модулів) розміщені на сайті ЦДН ХНУМГ.

Робота 1.

У роботі розглядається:

1. Форма опису нового типу даних.
2. Ключове слово - struct. Структура даних.

Опис нового типу даних.

Для опису нових типів даних в мові С ++ визначені ключові слова

struct - структура

class - клас

Опис нового типу даних виконується в наступній формі:

```
<ключове слово> <ІМ'Я ТИПУ ДАНИХ>
{
<тіло опису>
};
```

Структури

Розглянемо реалізацію поняття «Календарна дата» (далі тип «Дата») з використанням **struct** і безлічі функцій для роботи зі змінними цього типу:

Вправа 1. Текст програми 1. Файл *Date.h* (робота в складі проекту)

```
struct Date // структура дата: ПОЧАТОК опису
{
int day; // структура дата - поле "день"
int month; // структура дата - поле "місяць"
int year; // структура дата - поле "рік"
}; // структура дата: КІНЕЦЬ опису
// Список функцій, які працюють з новим типом
void set (Date *, int, int, int); // функція "ініціалізація дати"
-
// задає початкове значення змінної типу дата
void next (Date *); // функція "визначити наступну дату"
void print (Date *); // функція "вивести дату на екран"
// Кінець файлу
```

Вправа 1. Текст програми 1. Файл *Date.cpp* (робота в складі проекту)

```
#include <iostream>
#include "date.h"
using namespace std;
// функція "ініціалізація дати"
void set (Date * dt, int d, int m, int y)
{
dt-> day = d; // поле day отримує значення формального параметра d
dt-> month = m; // поле month отримує значення формального
параметра m
dt-> year = y; // поле year отримує значення формального параметра
y
}
// функція "визначити наступну дату"
void next (Date * dt)
{
dt-> day ++; // значення поля day збільшуємо на одиницю
}
// функція "вивести дату на екран"
void print (Date * dt) // друкує дату в формі "день.місяць.рік"
// 12 квітня 2016 року - 12.4.2016
{
cout << "Вивід дати:" << dt-> day << "." << dt-> month << "." <<
dt-> year << "\ n";
}
// Кінець файлу
```

Вправа 1. Текст програми 1. Файл *main.cpp* (робота в складі проекту)

```
#include <iostream> // Підключення заголовка для роботи з потоками
введення-виведення
#include "date.h" // Підключення заголовка - опис типу Date
void main() {
int d, m, y; // Оголошення змінних цілого типу
Date TheDate; // Оголошення змінної TheDate типу Date
cout << "Введіть через пробіл число місяць рік:";
cin >> d >> m >> y;
set (& TheDate, d, m, y); // Ініціалізація змінної TheDate
print (& TheDate); // Висновок змінної TheDate
next (& TheDate); // Зміна змінної TheDate
print (& TheDate); // Висновок змінної TheDate
}
// Кінець файлу
```

Завдання 1

У функції main реалізувати висновок зазначеної кількості n послідовних дат. Доповнити функцію main викликом функції My_Present (з минулого семестру).

Завдання 2

Доповнити список функцій роботи зі структурою Date функцією prev

Призначення функції: «Визначити попередню дату»

Прототип функції: void prev (Date *);

Реалізація: Аналогічно функції «Визначити наступну дату»

Доповнити функцію main викликом функції prev.

Зауваження. Для виконання завдання редагувати всі файли вправи 1.

Робота 2

У роботі розглядається:

1. Форма опису нового типу даних.
3. Структура даних і функції-члени.
4. Реалізація функції-члена.
5. Виклик функції-члена для об’єкта певного типу даних

Явного зв'язку між функціями і типом даних у вправі 1 немає. Такий зв'язок можна встановити, описавши функції як члени:

Функція-член - це функція, яка оголошена в <тілі опису>, і має безпосередній доступ до полів об’єкта розробленого типу.

Функції-члени викликаються тільки для змінної відповідного типу з використанням стандартного синтаксису для доступу до членів структури.

Різні структури можуть мати функції-члени з однаковими іменами !!!

При визначенні функції-члена необхідно вказувати ім’я структури, якій вона належить.

У функції-члені імена полів можуть використовуватися без явної посилання на об’єкт. У цьому випадку ім’я відноситься до члену того об’єкта, для якого функція була викликана.

Вправа 2. Текст програми 2. Файл *Date.h* (робота в складі проекту)

```
struct Date // структура дата: ПОЧАТОК опису
{
int day; // структура дата - поле "день"
int month; // структура дата - поле "місяць"
int year; // структура дата - поле "рік"
// Список функцій-членів
void set (int, int, int); // структура дата - функція-член
"ініціалізація дати" -
// задає початкове значення змінної типу дата
void next (); // структура дата - функція-член "визначити наступну
дату"
void print (); // структура дата - функція-член "вивести дату на
екран"
}; // структура дата: КІНЕЦЬ опису
// Кінець файлу
```

Вправа 2. Текст програми 2. Файл *Date.cpp* (робота в складі проекту)

```
#include <iostream> // Підключення заголовка для роботи з потоками
введення-виведення
#include "date.h" // Підключення заголовка містить опис типу Date
using namespace std;
// Реалізація функції-члена містить ім’я типу даних, якому вона
належить
//
// <тип повернення> <Ім’я типу>::<Ім’я функції-члена> (<список
аргументів>)
// {
// <тіло функції>
//}
// функція "ініціалізація дати"
void Date::set (int d, int m, int y)
{
day = d; // значенням поля day присвоюється значення формального
параметра d
month = m; // значенням поля month присвоюється значення
формального параметра m
year = y; // значенням поля year присвоюється значення формального
параметра e
}
// функція "визначити наступну дату"
void Date::next ()
{
day ++; // значення поля day збільшується на одиницю
}
// функція "вивести дату на екран"
void Date::print () // друкує дату в формі "день.місяць.рік"
// 14 лютого 2002 року - 14.2.2002
{
cout<<"Вивід дати:"<<day<<"."<<month << "." << year << "\ n";
}
// Кінець файлу
```

Вправа 2. Текст програми 2. Файл *main.cpp* (робота в складі проекту)

```
#include <iostream> // Підключення заголовка для роботи з
//чпотоками введення-виведення
#include "date.h" // Підключення заголовка опис типу Date
void main() {
int d, m, y; // Оголошення змінних цілого типу
Date TheDate; // Оголошення змінної TheDate типу Date
cout << "Введіть через пробіл число місяць рік:";
```

```
cin >> d >> m >> y;
// Виклик функції-члена виконують для конкретного об’єкта
(екземпляра класу)
//
// <об’єкт>. <Функція-член>
//
TheDate.set(d, m, y); // Ініціалізація змінної TheDate
TheDate.print(); // Висновок змінної TheDate
TheDate.next(); // Зміна змінної TheDate
TheDate.print(); // Висновок змінної TheDate
}
// Кінець файлу
```

Вправа 2. Текст програми 2.1. Файл main1.cpp (робота в складі проекту)

```
// Оголошення змінних, використання функції-члена
#include <iostream> // Підключення заголовка для роботи з потоками
введення-виведення
#include "date.h" // Підключення заголовка містить опис типу Date
Date today; // оголошення змінної today - сьогодні
Date my_birthday; // оголошення змінної my_birthday - мій день
народження
void main()
{
my_birthday.set(30,12,1950); // ініціалізація змінної my_birthday
today.set(18,1,1985); // ініціалізація змінної today
my_birthday.print(); // вивід на екран змінної my_birthday
today.next(); // змінної today встановити наступну дату
today.print(); // вивід на екран змінної today
}
// Кінець файлу
```

Завдання 3

Доповнити тип Date наступними функціями-членами:

Тип повернення	ім’я функції	формальні параметри	виконувана операція	зауваження
void	Blank	відсутні	Очищає всі поля об’єкта (присвоює їм значення 0)	
int	IsDate	відсутні	Перевіряє є, чи об’єкт календарною датою	повертає: 1, якщо об’єкт є датою 0, якщо об’єкт не є датою

int	WeekDay	відсутні	Обчислює день тижня для календарної дати	повертає: 1, якщо понеділок 2, якщо вівторок 3, якщо середовище 4, якщо четвер 5, якщо п'ятницю 6, якщо суботу 7, якщо неділя
int	IsHiYear	відсутні	Перевіряє чи є дата датою високосного року.	повертає: 1, якщо об’єкт є датою 0, якщо об’єкт не є датою
void	PrintText	відсутні	Виводить на екран дату в формі «13 жовтень 2002 г.»	

Застосувати нові функції-члени до об’єкта TheDate (вправа 2, файл main.cpp)

Завдання 4

Змінити функції-члени типу Date:

- void next ();
- void prev ();

Робота 3

У роботі розглядається:

1. Опис класового типу.
6. Ключове слово - **Class.**
7. Розділи опису класу.
8. Ініціалізація. Конструктори.
9. Очищення. Деструктори.

Опис Date у вигляді struct дає безліч функцій для роботи з Date, але не вказує, що ці функції повинні бути єдиними для доступу до об’єктів типу Date.

При використанні структури все елементи структури як поля даних, так і функції завжди доступні для використання. Таким чином, при некоректному поводженні з полями даних, інформація, що зберігається в них, може бути пошкоджений.

Обмеження використання можна накласти, використовуючи замість struct опис class.

```
class <ІМ'Я ТИПУ ДАНИХ>
{
// <тіло класу>
private:
<розділ тіла класу в закритій області>
protected:
<розділ тіла класу в захищеній області>
public:
<розділ тіла класу у відкритій області>
};
```

Захистити інформацію від «руйнування» можна засобами класів. При описі класового типу використовуються специфікатор доступу:

- private;
- protected;
- public.

Специфікатор ***PRIVATE*** вказує, що елемент знаходиться в області його дії є доступним лише для функцій-членів класу і дружнім функціям класу.

Специфікатор ***PROTECTED*** вказує, що елемент знаходиться в області його дії є доступним для функцій-членів класу і дружнім функціям похідних класів.

Специфікатор ***PUBLIC*** вказує, що елемент знаходиться в області його дії є загальнодоступним для будь-яких операцій і функцій.

Приклад. Опис класу Date. (файл <date.h>)

```
class Date // клас дата: ПОЧАТОК опису
{
private:
int day; // клас дата - поле "день"
int month; // клас дата - поле "місяць"
int year; // клас дата - поле "рік"
public:
// Список функцій-членів
Date (); // клас дата - КОНСТРУКТОР
Date (int, int, int); // клас дата - КОНСТРУКТОР
~ Date (); // клас дата - деструкторів
void system (); // функція "ініціалізація дати по календарю
комп'ютера"
void set (int, int, int); // клас дата - функція-член
"ініціалізація дати"
// задає початкове значення змінної типу дата
void next (); // клас дата - функція-член "визначити наступну
дату"
void print (); // клас дата - функція-член "вивести дату на екран"
void printText (); // клас дата - функція-член "вивести дату на
екран"
}; // клас дата: КІНЕЦЬ опису
// Кінець файлу
```

Мітка public ділить тіло класу на дві частини. Імена в першій, закритої частини, можуть використовуватися тільки функціями членами. Друга, відкрита частина, становить інтерфейс до об’єкту класу.

Struct - це просто class, у якого всі члени загальні, тому функції-члени класу визначаються і використовуються точно так же, як в попередньому випадку структури.

У тому, що доступ до структури даних обмежений явно описаним списком функцій, є кілька переваг. Будь-яка помилка, яка призводить до того,

що Дата приймає неприпустиме значення (наприклад, 36 Грудня 1985), повинна бути викликана кодом функції члена. Тому перша стадія налагодження, *локалізація,* виконується ще до того, як програма буде запущена.

Це окремий випадок загального твердження, що будь-яка зміна в поведінці типу Date може і повинно викликатися змінами в його членах.

Інша перевага в тому, що потенційному користувачеві класового типу потрібно знати визначення функцій-членів, щоб навчитися їх використовувати.

Програмування без приховування даних (із застосуванням структур) вимагає меншої продуманості, ніж програмування з приховуванням даних (з використанням класів).

Структуру можна визначити, чи не надто замислюючись про те, як її використовувати.

При визначенні класу, вся увага зосереджується на забезпеченні нового типу повним безліччю операцій. Це важливе зміщення акценту!

Час, витрачений на розробку нового типу, зазвичай багаторазово окупається при розробці та тестуванні програми.

Ініціалізація і конструктори

Використання для ініціалізації об’єкта класу функцій на зразок *set (int, int, int)* (встановити дату) неелегантно і значною кількістю помилок. Оскільки ніде не стверджується, що об’єкт слід буде почати, то програміст може забути це зробити, або зробити це двічі. Обидва випадки мають майже рівноцінні руйнівних наслідків.

Є більш хороший підхід: дати можливість описати функцію, явно призначену для ініціалізації об’єктів. Оскільки така функція конструює значення даного типу, вона називається конструктором.

Конструктор розпізнається по тому, що має те ж ім’я, що і сам клас.

Коли клас має конструктор, всі об’єкти цього класу будуть ініціалізовані. Якщо для конструктора потрібні параметри, їх слід давати.

Часто буває добре забезпечити кілька способів ініціалізації об’єкта класу.

Це можна зробити, поставивши кілька конструкторів.

Спеціальні функції-члени при оголошенні КЛАСУ мають власну назву і призначення. До таких функцій відносяться КОНСТРУКТОРИ.

Конструктор ініціалізує об’єкт класу.

Конструктор автоматично викликається компілятором в наступних випадках:

- при оголошенні змінної даного класу;
- при необхідності перетворень:
- у виразах,
- в аргументах функцій,
- в повертаються значеннях
- при створенні динамічної змінної за допомогою оператора *new*

Конструктор може розташовуватися в будь-якій частині класу private (закритою), protected (захищеної), public (відкритої).

Очищення і деструктори

Визначається користувачем тип частіше має, чому не має, конструктор, який забезпечує належну ініціалізацію.

Для багатьох типів також потрібно зворотну дію, деструктор, щоб забезпечити відповідне очищення об’єктів цього типу.

Ім’я деструктора для класу Date є ~ Date () ("додаток конструктора").

Багато класові типи використовують вільну пам’ять, яка виділяється конструктором, і звільняється деструктором.

Примітка. У тексті вправ 3, 4 в тіло функцій конструктора і деструктора додано рядок для виведення інформаційного повідомлення про виконуваної функції. Це дозволяє зрозуміти механізм виклику і роботи конструктори і деструкції.

Вправа 3. Текст програми 3. Файл *Date.cpp* (робота в складі проекту d3.prj)

```
#include <iostream> // Підключення заголовка для роботи з потоками
введення-виведення
#include <system>
#include "date" // Підключення заголовка містить опис типу Date
using namespace std;
// КОНСТРУКТОРИ
// Призначення КОНСТРУКТОРА - створити в програмі об’єкт КЛАСУ
// функція "Конструктор" - Створює об’єкт типу Date. (всі поля
рівні 0)
Date::Date ()
{
cout << "Конструктор 1. Створює об’єкт типу Date. (всі поля рівні
0) \ n";
day = 0; // значенням поля day присвоюється 0
month = 0; // значенням поля month присвоюється 0
year = 0; // значенням поля year присвоюється 0
}
// деструктор
// Призначення деструкторами - знищити в програмі об’єкт КЛАСУ
// функція "Деструктор"
Date::~ Date ()
{
cout << "Деструктор \ n";
}
// функція "ініціалізація дати"
void Date::set (int d, int m, int y)
{
day = d; // значенням поля day присвоюється значення формального
параметра d
month = m; // значенням поля month присвоюється значення
формального параметра m
year = y; // значенням поля year присвоюється значення формального
параметра e
}
// функція "визначити наступну дату"
void Date::next ()
{
day ++; // значення поля day збільшується на одиницю
}
// функція "вивести дату на екран"
void Date::print () // друкує дату в формі "день.місяць.рік"
{
cout << "Вивід дати:" << day << "." << month << "." << year << "\
n";
```

```
}
void Date::printText () // друкує дату в текстовій формі
{
cout << day;
switch (month) {
case 1: cout << "січня"; break;
case 2: cout << "лютого"; break;
case 3: cout << "березня"; break;
case 4: cout << "квітня"; break;
case 5: cout << "травня"; break;
case 6: cout << "червня"; break;
case 7: cout << "липня"; break;
case 8: cout << "серпня"; break;
case 9: cout << "вересня"; break;
case 10: cout << "жовтня"; break;
case 11: cout << "листопада"; break;
case 12: cout << "грудня"; break;
}
cout << year << "року. \ n";
}
// Кінець файлу
```

Вправа 3. Текст програми 3. Файл *Util.cpp* (робота в складі проекту d3.prj)

```
#include <iostream> // Підключення заголовка для роботи з потоками
введення-виведення
// Набір функцій для оформлення прикладів
void Start () {cout << "*** Початок програми
************************** \ n"; }
void End () {cout << "************************ Закінчення
програми *** \ n"; }
// Кінець файлу
```

Вправа 3. Текст програми 3. Файл *main.cpp* (робота в складі проекту d3.prj)

```
#include <iostream> // Підключення заголовка для роботи з потоками
введення-виведення
#include "date.h" // Підключення заголовка містить опис типу Date
#include "util.h" // Тема з описом допоміжних функцій
using namespace std,
void main()
{
Start ();
char Name [10]; // Оголошення змінної символьного типу
int d, m, y; // Оголошення змінних цілого типу
// При оголошенні змінної класового типу автоматично
// виконується виклик спеціальної функції КОНСТРУКТОРА
```

```
Date my_birthday; // Оголошення змінної my_birthday
cout << "Як Вас звати?";
cin >> Name;
     cout << "Введіть через пробіл число місяць рік Вашого
народження:";
cin >> d >> m >> y;
my_birthday.set (d, m, y); // Ініціалізація змінної my_birthday
cout << "Здрастуйте," << Name << "! \ n"; cout << "Ваша дата
народження";
my_birthday.printText ();
End ();
}
// Кінець файлу
```

Примітка. Текст вправи 4 доповнює текст вправи 3, тому наведено з деякими скороченнями.

Зверніть увагу:

1. Реалізовано конструктор 2. Створення та ініціалізація змінної Date з використання 3 цілих чисел
2. Реалізована функція-член <system>.

Вправа 4. Текст програми 4. Файл *Date.cpp* (робота в складі проекту d4.prj)

```
#include <iostream.h> // Підключення заголовка для роботи з
потоками введення-виведення
#include <system>
#include "date.h" // Підключення заголовка містить опис типу Date
using namespace std,
// КОНСТРУКТОРИ
// функція "Конструктор" - Створює об’єкт типу Date. (Всі поля
рівні 0)
Date::Date ()
{
cout << "Конструктор 1. Створює об’єкт типу Date. (всі поля рівні
0) \ n";
day = 0; // поле day отримує 0
month = 0; // поле month отримує 0
year = 0; // поле year отримує 0
}
// функція "Конструктор" - Створює об’єкт типу Date по трьом
числам.
Date::Date (int d, int m, int y)
{
```

```
cout << "Конструктор 2. Створює об’єкт типу Date по трьом числам.
\ n";
day = d; // поле day отримує значення параметра d
month = m; // поле month отримує значення параметра m
year = y; // поле year отримує значення параметра e
}
// функція "ініціалізація дати за поточним календарем комп'ютера"
void Date::system ()
{
struct date d;
getdate (& d);
day = d.da_day;
month = d.da_mon;
year = d.da_year;
}
// ... ..
// Кінець файлу
```

Вправа 4. Текст програми 4. Файл *main.cpp* (робота в складі проекту d4.prj)

```
#include <iostream> // Тема для роботи з потоками введення-
виведення
#include "date.h" // Тема з описом типу Date
#include "util.h" // Тема з описом допоміжних функцій
Date today; // Оголошення змінної today - сьогодні
Date my_birthday (13,10,1961); // Оголошення змінної my_birthday
void main() {
Start ();
Date TheDate; // Оголошення змінної TheDate типу Date
today.system (); // ініціалізація змінної today
my_birthday.print (); // вивід на екран змінної my_birthday
cout << "Сьогодні:";
today.printText (); // вивід на екран змінної today
End ();
}
// Кінець файлу
```

Завдання 5

Виконати аналіз виконання вправ 3,4. Пояснити результати роботи програм.

Завдання 6

Функції-члени, створені в завданнях 3 і 4 включити в опис класу Date.

Завдання 7

Написати тестову програму, яка демонструє роботу всіх функцій-членів класу Date (аналог завдання 10 першого семестру).

Робота 4

У роботі розглядається:

1. Функції-члени. Перевантаження операцій.
2. Операційні функції.
3. Конструктори. Конструктор копіювання. Ініціалізація об’єктів за замовчуванням.
4. Ключове слово this.

Зумовлені в мові операції можуть бути перевантажені для роботи з операндами класового типу.

Перевантаження операцій дозволяє розширювати на класові типи будь вбудовані операції мови.

Перевантаження операцій дозволяє максимально інтегрувати класовий тип в стандартний синтаксис вбудованих операцій.

Перевантаження операції виконується за допомогою введення спеціальної функції-члена - операторної функції.

Операційна функція (функція-член) в якості формального параметра бере всього один аргумент.

Синтаксис оголошення:

<тип значення> **operator** *<символ операції>* (*<тип аргументу>*);

Операційні функції як правило є елементами класу. Якщо операційна функція є елементом класу, неявний аргумент **this** є її першим операндом.

Зауваження.

У відповідних каталогах наведені файли «result.txt». Ці файли містять результати виконання вправ.

Вправи 5, 6 демонструють реалізацію операційних функцій:

- *«Інкремент»;*
- *«Сума Date і ціле число».*

Слід звернути увагу (вправа 5) на реалізацію «Конструктора копіювання». Конструктор активно використовується при роботі операційних функцій.

Демонструється використання неявного аргументу ***this*** .

Аргумент функції-члена ***this*** *застосовується в разі, якщо в тілі цієї функції слід звернутися до елементів об'єкта класу, для якого вона викликана.*

this *є покажчиком на об'єкт класу, тому підкоряється всім правилам роботи з покажчиками в мові С ++.*

Вправа 5. Текст програми 5. Файл *Date.h* (робота в складі проекту d5.prj)

```
class Date // клас дата: ПОЧАТОК опису
{
private:
int day; // клас дата - поле "день"
int month; // клас дата - поле "місяць"
int year; // клас дата - поле "рік"
public:
// Список функцій-членів
Date (); // клас дата - КОНСТРУКТОР
Date (int d, int m, int y); // клас дата - КОНСТРУКТОР
Date (Date &); // клас дата - КОНСТРУКТОР копіювання
~ Date (); // клас дата - деструкторів
void system (); // функція "ініціалізація дати по календарю
комп'ютера"
void set (int, int, int); // клас дата - функція-член
"ініціалізація дати"
void next (); // клас дата - функція-член "визначити наступну
дату"
void prev (); // клас дата - функція-член "визначити наступну
дату"
int IsHiYear (); // клас дата - функція-член "високосний чи рік"
// *** Арифметичні операції
void operator ++ (); // операція інкремента
void operator - (); // операція декремента
int operator - (Date & d); // віднімання Date-Date
Date operator - (int); // віднімання Date-int
Date operator + (int); // додавання Date + int
int Full_Year ();
// *** Логічні операції
int operator == (Date & d);
int operator! = (Date & d);
int operator> (Date & d);
int operator <(Date & d);
int operator> = (Date & d);
int operator <= (Date & d);
void print (); // клас дата - функція-член "вивести дату на екран"
void printText (); // клас дата - функція-член "вивести дату на
екран"
}; // клас дата: КІНЕЦЬ опису
// Кінець файлу
```

Вправа 5. Текст програми 5. Файл *Date.cpp* (робота в складі проекту d5.prj)

```
#include <iostream.h> // Тема для роботи з потоками введення-
виведення
```

```
#include <system>
#include "date.h" // Тема опису типу Date
static int Month_days [13] =
{365,31,28,31,30,31,30,31,31,30,31,30,31};
int vol_days [13];
using namespace std,
// КОНСТРУКТОРИ
// функція "Конструктор" - Створює об’єкт типу Date. (Всі поля
рівні 0)
Date::Date ()
{
cout << "Конструктор 1. Створює об’єкт типу Date. (всі поля рівні
0) \ n";
day = 0; // полю day присвоюється 0
month = 0; // полю month присвоюється 0
year = 0; // полю year присвоюється 0
}
// функція "Конструктор" - Створює об’єкт типу Date по трьом
числам.
Date::Date (int d, int m, int y)
{
cout << "Конструктор 2. Створює об’єкт типу Date по трьом числам.
\ n";
day = d; // значенням поля day присвоюється значення параметра d
month = m; // значенням поля month присвоюється значення параметра
m
year = y; // значенням поля year присвоюється значення параметра e
}
// функція "Конструктор" - Конструктор копіювання.
Date::Date (Date & d)
{
cout << "Конструктор 3. Конструктор копіювання. \ n";
day = d.day;
month = d.month;
year = d.year;
}
// деструктор
Date::~ Date ()
{
cout << "Деструктор \ n";
}
// функція "ініціалізація дати за поточним календарем комп'ютера"
void Date::system ()
{
struct date d; // структура бібліотеки
getdate (& d); // функція бібліотеки
```

```
day = d.da_day;
month = d.da_mon;
year = d.da_year;
}
// функція "ініціалізація дати"
void Date::set (int d, int m, int y)
{
day = d; // полю day присвоюється d
month = m; // полю month присвоюється m
year = y; // полю year присвоюється e
}
// функція "визначити наступну дату"
void Date::next ()
{
day ++; // значення поля day збільшуємо на одиницю
}
// функція "вивести дату на екран у формі DD.MM.YY"
void Date::print ()
{
cout << day << "." << month << "." << year << "\ n";
}
// функція "вивести дату на екран в текстовій формі DD.MM.YY"
void Date::printText ()
{
cout << day;
switch (month)
{
case 1: cout << "січня"; break;
case 2: cout << "лютого"; break;
case 3: cout << "березня"; break;
case 4: cout << "квітня"; break;
case 5: cout << "травня"; break;
case 6: cout << "червня"; break;
case 7: cout << "липня"; break;
case 8: cout << "серпня"; break;
case 9: cout << "вересня"; break;
case 10: cout << "жовтня"; break;
case 11: cout << "листопада"; break;
case 12: cout << "грудня"; break;
}
cout << year << "року. \ n";
}
// функція-член "Чи є рік високосним"
int Date::IsHiYear ()
{
return (year == ((year / 4) * 4)? 1: 0);
```

```
}
// Розподіл днів по місяцях поточного року
int Date::Full_Year () {
for (int i = 0; i <= 12; i ++) vol_days [i] = Month_days [i];
i = (year == ((year / 4) * 4)? 1: 0);
vol_days [2] = vol_days [2] + i;
return (* vol_days + i);
}
// Операторная функція "Інкремент"
void Date::operator ++ () {
if (++ day> vol_days [month])
{
day = 1;
if (++ month> 12) {month = 1; year ++; }
}
}
// Операторная функція "Сумма Date + int (повертає Date за
календарем)"
Date Date::operator + (int d)
{
Date a = * this;
while (d--)
a ++;
return a;
}
// Кінець файлу
```

Вправа 5. Текст програми 5. Файл *main.cpp* (робота в складі проекту d5.prj)

```
#include <iostream> // Тема для роботи з потоками
#include "date.h" // Тема з описом типу Date
#include "util.h" // Тема з описом допоміжних функцій
using namespace std,
Date today; // Оголошення змінної today
Date my_birthday (13,10,1961); // Оголошення змінної my_birthday
void main() {
Start ();
today.system (); // Ініціалізація змінної today
today.Full_Year (); // Ініціалізація календаря поточного року
Date TheDate = today; // Оголошення TheDate
Date TestDate; // Оголошення TestDate
cout << "My_birthday:";
my_birthday.print (); // вивід
cout << "Сьогодні:";
TheDate.printText (); // вивід
TheDate ++;
```

```
cout << "Інкремент:";
TheDate.printText (); // вивід
int test = 5;
TestDate = TheDate + test;
cout << "Склали:";
TestDate.printText (); // вивід
End ();
}
// Кінець файлу
```

Вправа 5. Результат роботи програми 5. Файл *Result.txt*

```
Конструктор 1. Створює об’єкт типу Date. (Всі поля рівні 0)
Конструктор 2. Створює об’єкт типу Date по трьом числам.
*** Початок програми **************************
Конструктор 3. Конструктор копіювання.
Конструктор 1. Створює об’єкт типу Date. (Всі поля рівні 0)
My_birthday: 13.10.1961
Сьогодні: 12, березня 2016 року.
Інкремент: 13 березня 2016 року.
Конструктор 3. Конструктор копіювання.
Конструктор 3. Конструктор копіювання.
деструктор
Склали: 18 березня 2016 року.
********************** Закінчення програми ***
деструктор
деструктор
деструктор
деструктор
деструктор
```

Закінчення файлу *Result.txt*

Зауваження.

Розмноження конструкторів в прикладі з класом Date є типовою ситуацією. При розробці класу є спокуса забезпечити "все", оскільки здається простіше забезпечити якийсь спосіб на будь-який випадок, можливо комусь знадобиться або тому, що має витончений вигляд, ніж вирішити, що потрібно насправді.

Останнє вимагає ретельних міркувань, але зазвичай призводить до програм, які менше за розміром і більш зрозумілі.

Один із способів скоротити число родинних функцій - використовувати параметри за замовчуванням. У разі Date для кожного параметра можна задати значення за замовчуванням, що інтерпретується як "за замовчуванням приймати: нуль" (порожня дата).

У вправі 6 наведено приклад реалізації та використання Конструктора, за допомогою якого можна виконати ініціалізацію об'єкта класу за замовчуванням.

Наведено нову версія "Конструктора копіювання".

Вправа 6. Текст програми 6. Файл *Date.h* (робота в складі проекту d6.prj)

```
class Date // клас дата: ПОЧАТОК опису
{
private:
int day; // клас дата - поле "день"
int month; // клас дата - поле "місяць"
int year; // клас дата - поле "рік"
public:
// Список функцій-членів
Date (int d = 0, int m = 0, int y = 0); // КОНСТРУКТОР
ініціалізація за замовчуванням
Date (Date &); // клас дата - КОНСТРУКТОР копіювання
.....................................................................
..................
// Відповідає вправі 5
.....................................................................
..................
// Кінець файлу
```

Вправа 6. Текст програми 6. Файл *Date.cpp* (робота в складі проекту d6.prj)

```
#include <iostream> // Тема для роботи з потоками введення-
виведення
#include <system>
#include "date.h" // Тема опису типу Date
static int Month_days [13] =
{365,31,28,31,30,31,30,31,31,30,31,30,31};
int vol_days [13];
// КОНСТРУКТОРИ
// функція "Конструктор" - Створює об'єкт типу Date по трьом
числам.
DateDate (int d, int m, int y)
{
```

```
cout << "Конструктор 1. Створює об’єкт Date (порожній або по 3
числах). \ n";
day = d; // полю day присвоюється d або 0
month = m; // полю month присвоюється m або 0
year = y; // полю year присвоюється e або 0
}
// функція "Конструктор" - Конструктор копіювання.
Date::Date (Date & d)
{
cout << "Конструктор 2. Конструктор копіювання. \ n";
* This = d;
}
.....................
// Відповідає вправі 5
.....................
// Кінець файлу
```

Вправа 6. Результат роботи програми 6. Файл *Result.txt*

```
Конструктор 1. Створює об’єкт Date (порожній або по 3 числах).
Конструктор 1. Створює об’єкт Date (порожній або по 3 числах).
*** Початок програми ***************************
today: 00 року.
Конструктор 2. Конструктор копіювання.
Конструктор 1. Створює об’єкт Date (порожній або по 3 числах).
My_birthday: 13.10.1961
Сьогодні: 12, березня 2016 року.
Інкремент: 13 березня 2016 року.
Конструктор 3. Конструктор копіювання.
Конструктор 3. Конструктор копіювання.
деструктор
Склали: 18 березня 2016 року.
************************ Закінчення програми ***
деструктор
деструктор
деструктор
деструктор
деструктор
```

Закінчення файлу *Result.txt*

Завдання 8

Виконати аналіз виконання вправ 5,6. Пояснити результати роботи програм.

Завдання 9

Розробити і реалізувати операторні функції, оголошені в описі класу (файл ***date.h*** вправу 6).

Завдання 10

Доповнити програму завдання 7 відповідними демонстраційними тестами.

Завдання 11

Використовуючи розроблений клас **Date,** скласти програму для вирішення наступного завдання.

РИТМИ ЖИТТЯ.

«Життя будь-якої людини проходить на тлі трьох періодичних циклів:

- фізичного - період 23 доби;
- емоційного - період 28 діб;
- інтелектуального - період 33 доби.

Всі цикли починають свій відлік від дати народження людини, і описують форму синусоїди. Перший напівперіод характеризує підйом відповідної діяльності, другий - спад. Критичним станів для людини є дні, в які відбувається накладення різних циклів.

Для конкретної людини видати календар циклів для зазначеного року.

Список використаних джерел

1. Авдеев В. А. Периферийные устройства. Интерфейсы, схемотехника, программирование / В. А. Авдеев. – Москва : ДМК Пресс, 2009 – ISBN: 978-5-94074-505-1.
2. Адельсон-Вельский Г. М Программирование игр /Г. М. Адельсон-Вельский, В. Л. Арлазаров, М. В. Донской. – Москва : Наука, 1978. – 256 с. : ил.
3. Бондарев В. М. Основы программирования / В. М. Бондарев, В. И. Рублинецкий, Е. Г. Качко. – Харьков : Фолио ; Ростов-на-Дону : Феникс, 1998. – 368 с. : ил. – ISBN 966-03-0313-0.
4. Буч Г. Язык UML : руководство пользователя / Г. Буч, Рамбо, А. Джекобсон ; пер. с англ. – Москва : ДМК, 2000. – 432 с. : ил. – (Серия для программистов). – ISBN 5-93700-009-9.
5. Бьерн Страуструп Язык программирования C++ / Бьерн Страуструп. – [Б. м.] : Бином, 2011.
6. Вайнер Р. C++ изнутри / Р. Вайнер, Л. Пинсон ;пер. с англ. – Киев : ДиаСофт, 1993. – 304 с. : ил. – ISBN 5-87554-079.
7. Гарбер М. Введение в SQL / Мартин Гарбер ; пер. с англ. – Москва : Лори, 1996.
8. Гарднер М. Крестики-нолики / М. Гарднер ; пер.с англ. – Москва : Мир, 1988. – 352 с. ил. ISBN 5-03-001234-6.
9. Дунаев В. В. Самоучитель JavaScript / В. В. Дунаев. – Санкт-Петербург : Питер, 2003. – 395 с. : ил. – ISBN 5-94723-533-1.
10. Дьюхарст С. Программирование на C++ /С. Дьюхарст, К. Старк ; пер. с англ. – Киев : ДиаСофт, 1993. – 272 с. : ил. – ISBN 5-87458-441-2.
11. Кнут Д. Искусство программирования для ЭВМ В 2-х т. Т. 2 Получисленные алгоритмы/ Д. Кнут ; пер. с англ. – Москва : Мир, 1977.
12. Кнут Д. Искусство программирования для ЭВМ. В 2-х т. Т. 1 Основные алгоритмы / Д. Кнут ; пер. с англ. – Москва : Мир, 1976.

13. Крупник А. Ассемблер : самоучитель / Александр Крупник. – Санкт-Петербург : Питер, 2005. – ISBN: 5-469-00825-8.
14. Кубенский А. А. Структуры и алгоритмы обработки данных: объектно-ориентированный подход и реализация на C++ / А. А. Кубенский. – Санкт-Петербург : БХВ-Петербург, 2004. – 463 с. : ил. – ISBN 5-94157-506-8.
15. Кулаков В. Программирование на аппаратном уровне : специальный справочник / В. Кулаков.–2-е изд. – Санкт-Петербург : Питер, 2003. – ISBN:
5-94723-487-4.
16. Ларман Крэг Применение UML и шаблонов проектирования / Ларман Крэг ; пер. с англ. – 2-е изд. – Москва : Вильямс, 2004. – 624 с. : ил. – ISBN
5-8459-0250-9 (рус.).
17. Лукас П. C++ под рукой / П. Лукас ;пер. с англ. – Киев : ДиаСофт, 1993. – 176 с. : ил. – ISBN 5-87554-079.
18. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы / В. Г. Олифер, Н. А. Олифер. Санкт-Петербург : Питер, 2007. – 960 с.
19. Пауэре Л. Microsoft Visual Studio 2008 / Л. Пауэре, М. Снелл ; пер. с англ. – Санкт-Петербург : БХВ-Петербург, 2009. – 1200 с. : ил. – (В подлиннике) – ISBN 978-5-9775-0378-5.
20. Прата С. Язык программирования C++ : лекции и упражнения / Стивен Прата ; пер. с англ. – 5-е изд. – Москва : ООО И. Д. Вильямс, 2007. – 1184 с. – ISBN: 5-8459-1127-3, 0-672-32697-3.
21. Рочкинд М. Программирование для UNIX. / М. Рочкинд ; пер. с англ. – 2-е изд. перераб. и доп. –Москва : Русская Редакция; Санкт-Петербург : БХВ- Петербург, 2005. – 704 с. : ил. – ISBN 5-94157-749-4.
22. Стерлинг Л. Искусство программирования на языке Пролог / Л. Стерлинг, Э. Шапиро.–Москва : Мир, 1990.

23. Уолл Л. Программирование на Perl / Ларри Уолл, Том Кристиансен, Джон Орвант ; пер. с англ. – 3-е изд. – Санкт-Петербург : Символ-Плюс, 2002.
24. Уэзерелл Ч. Этюды для программистов / Ч. Уэзерелл ;пер. с англ. – Москва : Мир, 1982. – 288 с. : ил.
25. Фаронов В. В. Программирование на языке C#. / В. В. Фаронов. – Санкт-Петербург : БХВ-Петербург, 2007. – 240 с. : ил. – ISBN 978-5-91180-369-8.
26. Флэнаган Д. JavaScript : подробное руководство / Д. Флэнаган ; пер. с англ. – Санкт-Петербург : Символ-Плюс, 2008. – 992 с. : ил. – ISBN-10: 5-93286-103-7, ISBN-13: 978-5-93286-103-5.
27. Фридл Д. Регулярные выражения / Джеффри Фридл ; пер. с англ. – 3-е изд. – Санкт-Петербург : Символ-Плюс, 2008.
28. Хендерсон П. Функциональное программирование. Применение и реализация / П. Хендерсон ;пер.с англ. – Москва : Мир, 1983. – 349 с. : ил.
29. Шлеер С. Объектно-ориентированный анализ: моделирование мира в состояниях / С. Шлеер, С. Меллор ; пер. с англ. – Киев : Диалектика, 1993. – 240 с. : ил. – ISBN 0-13-629940-7 (англ.), ISBN 5-7707-5541-5.
30. Visual C# 2008 : базовый курс / Карл Уотсон, Кристиан Нейгел, Якоб Педерсен, Хаммер Рид Джон Д., Скиннер, Морган, Эрик Уайт ; пер. с англ. – М. : ООО "И. Д. Вильяме", 2009. – 1216 с. : ил. –(Парал. тит. англ.). – ISBN 978-5-8459-1532-0 (рус.).

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ

до практичних занять та самостійної роботи

з навчальної дисципліни

«ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

(для студентів освітньо-кваліфікаційного рівня бакалавр у галузі знань 12 – Інформаційні технології спеціальності 122 – Комп'ютерні науки та інформаційні технології)

Відповідальний за випуск *І. Л. Яковицький*

За авторською редакцією

Комп'ютерне верстання І. *Л. Яковицького*

План 2015, поз. 392м

Підп. до друку 07.04.2017 р. Формат 60×84/16
Друк на ризографі Ум. друк. арк. 1,02
Зам. № Тираж 50 пр.

Видавець і виготовлювач:
Харківський національний університет міського господарства імені О. М. Бекетова,
вул. Маршала Бажанова,17, Харків, 61002
Електронна адреса: rectorat@kname.edu.ua
Свідоцтво суб'єкта видавничої справи:
ДК № 5328 від 11.04.2017 р.