

ТЕМА: Класи і об'єкти. Будова класу.

Зміст класів і об'єктів

Модель процедурного програмування: програма визначає окремо всі необхідні елементи даних (пам'ять) і окремо – функції для опрацювання таких даних:

```
// деякі дані
int testdata[6] = {5,9,0,-4,-2,12};
// функції опрацювання даних
void ShowList(int * dt, int size)
{ // друкувати масив окремо додатні, окремо від'ємні
  int i;
  for (i=0; i<size; i++)  if (dt[i]>=0) cout << dt[i] << '\t';
  cout << endl;
  for (i=0; i<size; i++)  if (dt[i]<0) cout << dt[i] << '\t';
  cout << endl;
}

int main() // головна функція
{
  ShowList(testdata,6); // зв'язування функцій і даних
  system("pause"); // потрібна затримка вікна консолі
  return 0 ; // вихід з функції
}
```



Для великих і складних програм потрібно будувати багато елементів даних і багато різних функцій для опрацювання даних. Тому комбінування даних і функцій буде значно ускладненим.

Виникає ідея: об'єднати дані і функції для опрацювання в єдине ціле, що називають об'єкт на основі класу:



Об'єкт пам'ятає інформацію про самого себе і вміє сам таку інформацію опрацьовувати, тобто має самостійну поведінку.

Клас – спеціально організована компонента для реалізації нових програмних типів. Головна ідея класу – створити об'єкт, який вміє самостійно розв'язувати задачі.




Клас – це проект об'єкта, потрібного для розв'язування групи споріднених задач.

Об'єкт – це "власна суть", тобто результат реалізації класу.

Схема визначення класів в С++

Перший спосіб визначення класу:

```
class ім'я
{
    protected:    // або private
        // поля і прототипи захищених методів
    public:        
        // прототипи відкритих методів
} ; // крапка з комою в кінці оголошення
. . . . . реалізація методів (повне визначення)
```

Спосіб доступу (специфікатор доступу, дозвіл на використання) до окремих частин класу визначають ключовими словами:

private: все, що записано нижче за текстом, доступне лише всередині цього класу і відповідних об'єктах;

protected: все, що записано нижче за текстом, доступне всередині класу і в дочірніх класах; зовні не доступне;

public: все записане нижче доступне як всередині класу, так і ззовні в інших частинах програми, класах, функціях. Це є відкрита для всіх частина класу.

Приклад базової задачі. Проектування класу

Побудувати клас для опрацювання масиву числових даних цілого типу. Масив може змінювати розмір в процесі виконання. Клас має забезпечити збереження даних на час роботи програми, редагування даних і прості статистичні функції щодо них.

Перше, що потрібно зробити – виконати проектування класу. Проектування потрібно починати з визначення переліку полів, тобто всіх необхідних даних для розв’язування задачі. Не забувати, що клас стосується цілої групи споріднених задач, а не лише одного варіанту постановки.

Отже, буде потрібно:

- місце в пам’яті для масиву даних; якщо масив має змінний розмір, тоді потрібно не статичне виділення пам’яті, а вказівник на масив:

```
int * mas;
```

- вказівник сам по собі розмір пам’яті не визначає, тому потрібно фіксувати фактичний розмір в кожний момент виконання програми:

```
int size;
```

- треба вирішити, принаймі для початку, яка конкретна статистична інформація нам потрібна; нехай це буде середнє і максимальне значення масиву:

```
int ave, max;
```

Друге, що потрібно зробити – визначити способи взаємодії з даними, які ми визначили першим кроком. Типові способи взаємодії складаються з трьох елементів: ініціалізація, оновлення і відображення. Ці способи визначають інтерфейс користувача, реалізований за допомогою методів.

Ініціалізація. Початкове створення масиву. Прийmemo, що масив на початку має заданий розмір, а всі елементи – однакове значення:

```
void Create(int S, int val); // S – розмір, val – значення
```

Забігаючи вперед, зазначимо, що пізніше можна додати інші способи початкового створення масиву, тому зупинимось зараз лише на одному варіанті.

Оновлення. Прочитати нові значення для всіх елементів масиву:

```
void Read();
```

Оновлення. Значення одного елемента масиву замінити іншим значенням:

```
void Repl(int N, int x); // елементу N надати значення x
```

Відображення. Отримати (показати) середнє і максимальнє значення масиву (можна окремо):

```
int GetAve(); // отримати середнє значення  
int GetMax(); // отримати значення найбільшого
```

Відображення. Зберегти поточні значення масиву в файлі:

```
void Print(char * filename);
```

Відображення. Надрукувати поточні значення на екрані:

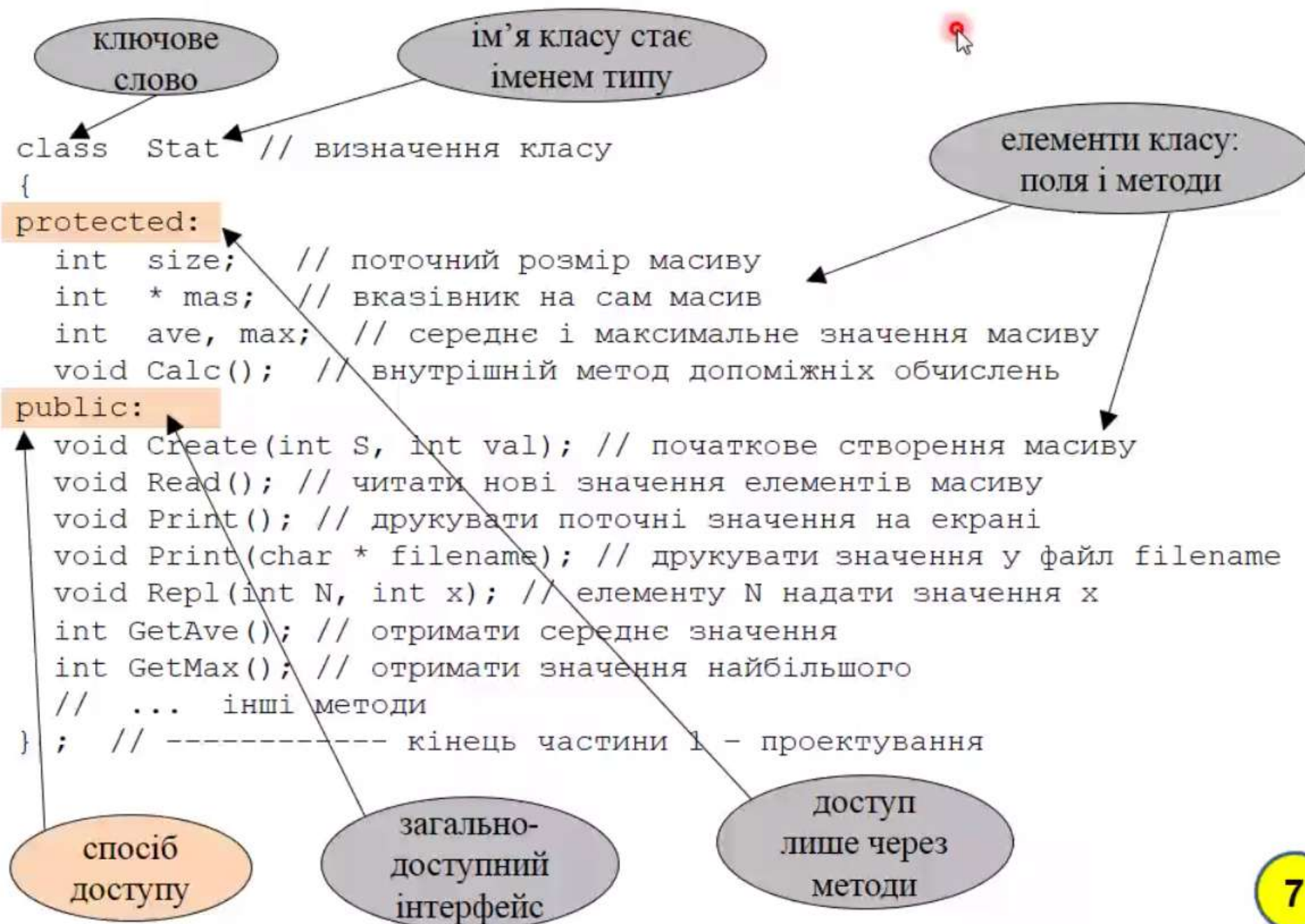
```
void Print();
```

Третє, що потрібно зробити – визначити внутрішні обчислювальні процедури (методи) опрацювання масиву, які до інтерфейсу не належать, але потрібні для автоматичного виконання операцій.

Внутрішній метод. Обчислити найбільше і середнє значення:

```
void Calc();
```


Отже, проект класу матиме такий вигляд:



Програмна реалізація класу

```
#include <iostream> // читання і запис стандартних потоків
#include <fstream> // файловий ввід-вивід
using namespace std;
class Stat // визначення класу
{
protected:
    int size; // поточний розмір масиву
    int * mas; // вказівник на сам масив
    int ave, max; // середнє і максимальне значення масиву
    void Calc(); // внутрішній метод допоміжних обчислень
public:
    void Create(int S, int val); // початкове створення масиву
    void Read(); // читати нові значення елементів масиву
    void Print(); // друкувати поточні значення на екрані
    void Print(char * filename); // друкувати значення у файл filename
    void Repl(int N, int x); // елементу N надати значення x
    int GetAve(); // отримати середнє значення
    int GetMax(); // отримати значення найбільшого
    // ... інші методи
} ; // ----- кінець частини 1 - проектування
```



```
// далі записуємо частину 2 - реалізацію методів в класу
void Stat::Calc() // внутрішній метод допоміжних обчислень
{
    int i;
    max=mas[0]; // обчислення найбільшого
    for(i=1; i<size; i++)
    {
        if(mas[i]>max) max=mas[i];
    }
    int S=0; // обчислення середнього
    for(i=0; i<size; i++) S+=mas[i];
    ave = S / size;
} // void Stat::Calc()

void Stat::Create(int S, int val) // початкове створення масиву
розміру S, значення val
{ // вважаємо, що S>0 і не дуже велике
    size = S;
    mas = new int[size];
    for(int i=0; i<size; i++) mas[i]=val;
    max=ave=val; // потрібно визначати всі поля
} // void Stat::Create(int S, int val)
```

```
void Stat::Repl(int N, int x) // елементу N надати значення x
{
    if(N>=0 && N<size) { mas[N]=x; Calc(); } // присвоїти і
    переобчислити !
} // void Stat::Repl(int N, int x)
```

```
void Stat::Read() // читати нові значення елементів масиву з
клавіатури
{
    cout << "Print " << size << " integer numbers:\n";
    for(int i=0; i<size; i++) cin >> mas[i];
    Calc(); // заново переобчислити
} // void Stat::Read()
```

```
int Stat::GetAve() // отримати середнє значення
{ return ave; } // int Stat::GetAve()
```

```
int Stat::GetMax() // отримати значення найбільшого
{ return max; } // int Stat::GetMax()
```

```
void Stat::Print() // друкувати поточні значення на екрані
{
    cout << "Current values:\n";
    for(int i=0; i<size; i++) cout << mas[i] << "  ";
    cout << endl;
    cout << "ave=" << ave << "  max=" << max << endl;
} // void Stat::Print()
```

```
void Stat::Print(char * filename) // друкувати поточні значення у
файл filename
{
    ofstream outres(filename); // потік виведення
    outres << "Current values:\n";
    for(int i=0; i<size; i++) outres << mas[i] << "  ";
    outres << endl;
    outres << "ave=" << ave << "  max=" << max << endl;
    outres.close();
} // void Stat::Print(char * filename)
```



Зауваження: 1) порядок перелічення полів і методів і порядок реалізації методів не мають значення; 2) реалізація класу має бути повною, не можна оголошувати не реалізовані методи; 3) доступ до полів за прийнятою методологією ООП має бути лише через методи; 4) звернути увагу на форму заголовків в частині реалізації методів.

Другий спосіб визначення класу

Всі методи або частину з них можна реалізувати безпосередньо в класі:

```
class Stat
{
protected:
. . . . .
public:
    void Create(int S, int val) // початкове створення масиву
    { // вважаємо, що S>0 і не дуже велике
        size = S;
        mas = new int[size];
        for(int i=0; i<size; i++) mas[i]=val;
        max=ave=val; // потрібно визначати всі поля
    }
. . . . .
} ;
```

Зауваження:

```
void Create(int S, int val) // заголовок метода
void Create(int S, int val) ; // прототип метода
```



Перевантаження методів класу

Методи – це функції, тому їх можна перевантажувати за правилами, розглянутими при вивченні функцій. Перевантажені методи мають однакові імена, але різні сигнатури. Таких методів маємо два:

```
void Stat::Print()  
void Stat::Print(char * filename)
```



визначення

Застосування класу. Об'єкти



В тексті програми потрібно спершу подати визначення класу. Далі в головній програмі або в довільній функції визначаємо потрібні об'єкти і виклики методів, наприклад:

```
void main()
{
    Stat A, B; // пусті об'єкти, визначені як статичні
    A.Create(4, 100); // виклик методів - через крапку
    A.Print();
    A.Read();  A.Print();
    A.Repl(3, -1);  A.Print();  A.Print("test1.txt");
    cout << A.GetAve() << "  " << A.GetMax() << endl;
    // . . . . .
    system("pause"); // затримати вікно консолі
}
```




Модель організації файлів

В технології ООП прийнято визначати класи окремими файлами. Об'єкти визначають в інших файлах і директивою `#include` приєднують файл класу:

визначення класу –
файл **stat.h**

```
// підключати однократно
#pragma once
#include <iostream>
#include <fstream>
using namespace std;
class Stat // визначення класу
{ . . . . . } ;
// реалізація методів класу
. . . . .
int Stat::GetMax()
{ return max; }
```

робота з об'єктами –
файл **L1_3.cpp**

```
#include <iostream>
#include <fstream>
// приєднати файл класу
#include "stat.h"
using namespace std;
void main()
{ Stat A, B;
  . . . . .
  system("pause");
}
```

