**3 варіант**

1. **Об'є́ктно-орієнто́ване програмува́ння** (ООП) — одна з [парадигм](https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [програмування](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають три основні концепції: [інкапсуляція](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F), [успадкування](https://uk.wikipedia.org/wiki/%D0%A3%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) та [поліморфізм](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%96%D0%BC%D0%BE%D1%80%D1%84%D1%96%D0%B7%D0%BC_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Одною з переваг ООП є краща [модульність](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в [1960-тих](https://uk.wikipedia.org/wiki/1960-%D1%82%D1%96) роках, вона не мала широкого застосування до [1990-тих](https://uk.wikipedia.org/wiki/1990-%D1%82%D1%96), коли розвиток комп'ютерів та комп'ютерних мереж дозволив писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато [мов програмування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) або підтримують ООП ([PHP](https://uk.wikipedia.org/wiki/PHP), [Lua](https://uk.wikipedia.org/wiki/Lua)) або ж є цілком об'єктно-орієнтованими (зокрема, [Java](https://uk.wikipedia.org/wiki/Java_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [C#](https://uk.wikipedia.org/wiki/C_Sharp_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [C++](https://uk.wikipedia.org/wiki/C%2B%2B), [Python](https://uk.wikipedia.org/wiki/Python), [Ruby](https://uk.wikipedia.org/wiki/Ruby) та [Objective-C](https://uk.wikipedia.org/wiki/Objective-C), [ActionScript 3](https://uk.wikipedia.org/wiki/ActionScript), [Swift](https://uk.wikipedia.org/wiki/Swift_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [Vala](https://uk.wikipedia.org/wiki/Vala)).
2. abstract class A {  
       int p1;  
       A() {  
           p1 = 1;  
       }  
       void print() {  
           System.out.println(p1);  
       }  
   }  
   class B extends A {  
   }  
   public class Main {  
       public static void main(String[] args) {  
           A ob1;  
           // ошибка: ob1 = new A();  
           B ob2 = new B(); // будет вызван конструктов по умолчанию из A  
           ob2.print();  
       }  
   }

Пасивна поведінка – дія, яку виконує об’єкт під впливом іншого об’єкту.

Активна поведінка – дія, яку об’єкт виконує над іншим об’єктом.

**3)Написати перевантажувані функції, що обчислюють середнє масиву чисел типу float або double. Продемонструвати виконання.**

#include <stdio.h>

double func(float\* arr, int length);

double func(double\* arr, int length);

int main(void)

{

static double arr1[] = { -2,4,6,8,9,5,3 };

float arr2[] = { 2.3,4.1,4.6 };

printf("Function with float is:%0.1f\n", func(arr2, 3));

printf("Function with double is:%0.1f\n", func(arr1, 7));

return 0;

}

double func(float\* arr, int length)

{

float sum = 0;

double result = 0;

if (length > 0)

{

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

sum += arr[i];

result = sum / length;

}

return result;

if (length <= 0)

return 0;

}

double func(double\* arr, int length)

{

double sum = 0;

double result = 0;

if (length > 0)

{

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

sum += arr[i];

result = sum / length;

}

return result;

if (length <= 0)

return 0;

}

4) Написати програму, що виводить у стандартний потік масив чисел типу int по 4 значення в рядку з вирівненням за стовпцями.

#include <iostream>

#include <iomanip>

using namespace std;

int main(int argc, char\* argv[]) {

int n;

cout << "Enter size:";

cin >> n;

int \*a = new int[n];

for (int i = 1; i <= n; i++) {

cin >> a[i];

}

for (int i = 1; i <= n; i++) {

cout.width(4);

cout <<" " << a[i] <<" " << setw(12);

if (i % 4 == 0)

cout << "\n";

}

cout << endl;

delete[] a;

}

**5)** // ConsoleApplication5.cpp: определяет точку входа для консольного приложения.

#include <iostream>

#include <iomanip>

#include <math.h>

using namespace std;

class Vector

{

private:

int x,x1;

int y, y1;

int z, z1;

public:

Vector(int s\_x = 0, int s\_y = 0, int s\_z = 0);

virtual ~Vector();

float moving(int, int, int);

float zoom(int, int, int);

float get\_length(int,int,int);

//float SetCoord(int, int, int);

//float SetCoord1(int, int, int);

float get\_directing\_vector(int, int, int);

};

Vector::Vector(int s\_x, int s\_y, int s\_z)

{

x = s\_x;

y = s\_y;

z = s\_z;

}

Vector::~Vector()

{

}

float Vector::moving(int n, int k, int j)

{

x += n;

y += k;

z += j;

}

float Vector::zoom(int n, int k, int j)

{

x \*= n;

y \*= k;

z \*= j;

}

float Vecotr::get\_length()

{

float L = sqrt((n - x) \* (n - x)

+ (k - y) \* (k - y) + (j - z) \* (j - z) );

return L;

}

float Vecotr::get\_directing\_vector()

{

float M = ( y \* j) - ( k \* z ) + ( n \* z)

- (x \* j ) + (x \* k) - (k \* y);

return M;

}

float Vector::SetCoord(int n, int k, int j)

{

x = n;

y = k;

z = j;

}

float Vector::SetCoord1(int n, int k, int j)

{

x1 = n;

y1 = k;

z1 = j;

}

/\*int Vector::GetSecond()

{

if (second <= 0)

return 0;

return (hour \* 60 \* 60) + (minute \* 60) + second ;

}

int Vector::GetMinute()

{

if (minute <= 0)

return 0;

return (hour \* 60) + minute;

}

\*/

int main()

{

Vector a;

a.moving(4, 4, 4);

a.zoom(4, 4, 4);

a.length(30, 12,25);

a.directing\_vector(15,24,24);

system("pause");

}

**4 варіант**

1. **Об'є́ктно-орієнто́ване програмува́ння** (ООП) — одна з [парадигм](https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [програмування](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають три основні концепції: [інкапсуляція](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F), [успадкування](https://uk.wikipedia.org/wiki/%D0%A3%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) та [поліморфізм](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%96%D0%BC%D0%BE%D1%80%D1%84%D1%96%D0%B7%D0%BC_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Одною з переваг ООП є краща [модульність](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в [1960-тих](https://uk.wikipedia.org/wiki/1960-%D1%82%D1%96) роках, вона не мала широкого застосування до [1990-тих](https://uk.wikipedia.org/wiki/1990-%D1%82%D1%96), коли розвиток комп'ютерів та комп'ютерних мереж дозволив писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато [мов програмування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) або підтримують ООП ([PHP](https://uk.wikipedia.org/wiki/PHP), [Lua](https://uk.wikipedia.org/wiki/Lua)) або ж є цілком об'єктно-орієнтованими (зокрема, [Java](https://uk.wikipedia.org/wiki/Java_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [C#](https://uk.wikipedia.org/wiki/C_Sharp_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [C++](https://uk.wikipedia.org/wiki/C%2B%2B), [Python](https://uk.wikipedia.org/wiki/Python), [Ruby](https://uk.wikipedia.org/wiki/Ruby) та [Objective-C](https://uk.wikipedia.org/wiki/Objective-C), [ActionScript 3](https://uk.wikipedia.org/wiki/ActionScript), [Swift](https://uk.wikipedia.org/wiki/Swift_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [Vala](https://uk.wikipedia.org/wiki/Vala)).
2. **Інкапсуляція** — один з трьох основних механізмів [об'єктно-орієнтованого програмування](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). Йдеться про те, що об’єкт вміщує не тільки дані, але і правила їх обробки, оформлені в вигляді виконуваних фрагментів ([методів](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))). А також про те, що доступ до стану об'єкта напряму заборонено, і ззовні з ним можна взаємодіяти виключно через заданий інтерфейс (відкриті поля та методи), що дозволяє знизити [зв'язність](https://uk.wikipedia.org/wiki/%D0%97%D0%B2%27%D1%8F%D0%B7%D0%BD%D1%96%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Таким чином контролюються звернення до полів класів та їхня правильна ініціалізація, усуваються можливі помилки пов'язані з неправильним викликом методу. Оскільки користувачі працюють лише через відкриті елементи класів, то розробники класу можуть як-завгодно змінювати всі закриті елементи і навіть перейменовувати та видаляти їх, не турбуючись, що десь хтось їх використовує у своїх програмах.

**3) Написати перевантаження функції, що обчислюють середнє масиву чисел типу int або double. Продемонструвати їх використання.**

#include <iostream>

using namespace std;

double func(int\* arr, int length);

double func(double\* arr, int length);

int main(void)

{

static double arr1[] = { -2,4,6,8,9,5,3 };

int arr2[] = { 2.3,4.1,4.6 };

cout << "Function with int is:" << func(arr1, 7) << endl;

cout << "Function with double is:" << func(arr2, 3) << endl;

return 0;

}

double func(int\* arr, int length)

{

double sum = 0;

double result = 0;

if (length > 0)

{

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

sum += arr[i];

result = sum / length;

}

return result;

if (length <= 0)

return 0;

}

double func(double\* arr, int length)

{

double sum = 0;

double result = 0;

if (length > 0)

{

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

sum += arr[i];

result = sum / length;

}

return result;

if (length <= 0)

return 0;

}

**4)** Написати програму, що виводить у стандартний потік масив чисел типу float по 4 значення в рядку з вирівнянням за стовпцями. Точність 3 знаки після коми.

#include <iostream>

#include <iomanip>

using namespace std;

float main() {

int n;

cout << "Enter size:";

cin >> n;

float \*a = new float [ n ];

for (int i = 1; i <= n; i++) {

cin >> a[i] ;

}

for (int i = 1; i <= n; i++) {

cout.width(4);

printf(" %0.3f ",a[i]);

if (i % 4 == 0)

cout << "\n";

}

cout << endl;

delete[] a;

}

**5)**

// ConsoleApplication5.cpp: определяет точку входа для консольного приложения.

#include <iostream>

#include <iomanip>

#include <math.h>

using namespace std;

class Vector

{

private:

int x,x1;

int y, y1;

int z, z1;

public:

Vector(int s\_x = 0, int s\_y = 0, int s\_z = 0);

virtual ~Vector();

float moving(int, int, int);

float zoom(int, int, int);

float get\_length(int,int,int);

//float SetCoord(int, int, int);

//float SetCoord1(int, int, int);

float get\_directing\_vector(int, int, int);

};

Vector::Vector(int s\_x, int s\_y, int s\_z)

{

x = s\_x;

y = s\_y;

z = s\_z;

}

Vector::~Vector()

{

}

float Vector::moving(int n, int k, int j)

{

x += n;

y += k;

z += j;

}

float Vector::zoom(int n, int k, int j)

{

x \*= n;

y \*= k;

z \*= j;

}

float Vecotr::get\_length()

{

float L = sqrt((n - x) \* (n - x)

+ (k - y) \* (k - y) + (j - z) \* (j - z) );

return L;

}

float Vecotr::get\_directing\_vector()

{

float M = ( y \* j) - ( k \* z ) + ( n \* z)

- (x \* j ) + (x \* k) - (k \* y);

return M;

}

float Vector::SetCoord(int n, int k, int j)

{

x = n;

y = k;

z = j;

}

float Vector::SetCoord1(int n, int k, int j)

{

x1 = n;

y1 = k;

z1 = j;

}

/\*int Vector::GetSecond()

{

if (second <= 0)

return 0;

return (hour \* 60 \* 60) + (minute \* 60) + second ;

}

int Vector::GetMinute()

{

if (minute <= 0)

return 0;

return (hour \* 60) + minute;

}

\*/

int main()

{

Vector a;

a.moving(4, 4, 4);

a.zoom(4, 4, 4);

a.length(30, 12,25);

a.directing\_vector(15,24,24);

system("pause");

}

**5 варіант**

1. **Об’єктно-орієнтований аналіз.** Об’єктно-орієнтований аналіз спрямований на створення моделей реальної дійсності на основі об’єктно-орієнтованого світогляду.

*Об’єктно-орієнтований аналіз – це методологія, при якій вимоги до системи сприймаються з погляду класів і об'єктів, виявлених у предметній області.*

**2)Приклад інкапсуляції**

1. **class** **Point** {
2. *//можливий доступ лише з методів даного класу*
3. int x, y;
4. bool visibility;
5. **public**:
6. void createPoint(int a, int b) {
7. x = a; y = b;
8. visibility = true;
9. }
10. void setVisibility(bool visibility) {
11. **this**->visibility = visibility;
12. }
13. int getX() {
14. **return** x;
15. }
16. int getY() {
17. **return** y;
18. }
19. };

У цьому прикладі клас Point інкапсулює (приховує) координати точки. Доступ до них можливий лише за певними правилами, які реалізовуються через відповідні методи. Такими правилами можуть бути, наприклад, операція створення точки (установка значень координат), а також операції «увімкнення» і «вимкнення» видимості точки, отримання координат.

Як видно з прикладу, для створення точки необхідно вказати координати точки і «увімкнути» дану точку (зробити її видимою). Якщо б клас був повністю відкритий, то можна б було вручну встановити відповідні поля класу х, y та встановити visible в true. Проте програміст може легко забути встановити якусь з координат або забути встановити видимість. Метод createPoint забезпечує виконання усіх необхідних дій, а закриття доступу до координат змушує діяти лише через використання даного методу. Інкапсульованими також можуть бути і методи класу.

В ООП рекомендується з самого початку створювати закриті поля і лише в разі крайньої необхідності надавати ширший доступ до них. Для роботи із закритими полями класів краще використовувати відповідний метод доступу (getter) та метод мутатор (setter).

**3) Написати перевантаження функції, що обчислюють середнє масиву чисел типу int або short. Продемонструвати їх використання.**

#include <iostream>

using namespace std;

double func(int\* arr, int length);

double func(short\* arr, int length);

int main(void)

{

short arr1[] = { 2,4,6,8,9,5,3 };

int arr2[] = { 2,3,1,4,6 };

cout << "Function with short is:" << func(arr1, 7) << endl;

cout << "Function with int is:" << func(arr2, 5) << endl;

return 0;

}

double func(int\* arr, int length)

{

double sum = 0;

double result = 0;

if (length > 0)

{

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

sum += arr[i];

result = sum / length;

}

return result;

if (length <= 0)

return 0;

}

double func(short\* arr, int length)

{

double sum = 0;

double result = 0;

if (length > 0)

{

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

sum += arr[i];

result = sum / length;

}

return result;

if (length <= 0)

return 0;

}

**4)** Написати програму, що виводить у стандартний потік масив чисел типуdouble по 3 значення в рядку з вирівнянням за стовпцями. Точність 5 знаки після коми.

#include <iostream>

#include <iomanip>

using namespace std;

int main() {

int n;

cout << "Enter size:";

cin >> n;

double \*a = new double [ n ];

for (int i = 1; i <= n; i++) {

cin >> a[i] ;

}

for (int i = 1; i <= n; i++) {

cout.width(4);

printf(" %0.5f ",a[i]);

if (i % 3 == 0)

cout << "\n";

}

cout << endl;

delete[] a;

return 0;

}

**5) Створити клас, що асоціюється з часом(год,хв,сек) та підтримує операції збільшення, - , ініціалізація сек та хв. Видаючи результат у хв або сек. Продемонструвати використання.**

#include <iostream>

#include <conio.h>

#include <fstream>

#include "Time.h"

using namespace std;

class Time

{

private:

unsigned int hours;

unsigned int seconds;

unsigned int minutes;

public:

Time::Time() {};

Time::Time(unsigned int sec)

{

hours = sec / (60 \* 60);

minutes = sec % (60 \* 60) / 60;

seconds = sec % (60 \* 60) % 60;

};

void MinusTime(unsigned sec)

{

if ((sec / (60 \* 60)) > hours)

hours = 0;

else hours -=(sec / (60 \* 60));

if ((sec % (60 \* 60) / 60) > minutes)

{

minutes = 0;

}

else minutes-=(sec % (60 \* 60) / 60);

if ((sec % (60 \* 60) % 60) > seconds)

{

seconds = 0;

}

else seconds-= (sec % (60 \* 60) % 60);

}

void PlusTime(unsigned sec)

{

cout << "Plus time" << endl;

hours = hours + (sec / (60 \* 60));

minutes = minutes + (sec % (60 \* 60) / 60);

seconds = seconds + (sec % (60 \* 60) % 60);

}

Time::~Time() {};

friend ostream &operator«(ostream &stream, Time ex)

{

stream << "Hours:" << ex.hours << endl;

stream << "Minutes:" << ex.minutes << endl;

stream << "Seconds:" << ex.seconds << endl;

return stream;

}

};

int main()

{

Time obj(4501);

obj.MinusTime(1000);

obj.PlusTime(2000);

cout << obj;

\_getch();

return 0;

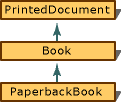
}

**Варіант 6**

1. **абстрагува́ння** — спосіб та метод відокремлення деталей з метою отримання можливості зосередитись на найважливіших особливостях об'єкта. .Стан об’єкта – описує дані, що зберігаються в об’єкті

Грубо кажучи, в програмуванні, абстрагування може бути або керування, або даних. **Абстрагування керування** є абстрагуванням від деталей дій, а **абстрагування даних** є абстрагуванням структур даних. Наприклад, абстрагування керування в [структурному програмуванні](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) полягає у використанні [підпрограм](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0) та визначених керівних конструкцій. Абстрагування даних дозволяє обробляти одиниці даних у змістовний спосіб. Наприклад, абстрагування є основною мотивацією створення [типів даних](https://uk.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85). [Об'єктно-орієнтоване програмування](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) може розглядатись як спроба абстрагувати як керування так і дані.

2.



Модульність - властивість системи, яка зв’язана з можливістю декомпозиції на ряд тісно зв’язаних модулів (частин).

3) Написати вбудовану функцію, що підносить значення аргументу типу int до квадрату, та продемонструвати її використання.

#include <iostream>

using namespace std;

inline void sqr(int a)

{

int x = a \* a;

cout << "the number of square is:" << x;

}

int main()

{

int x;

sqr(5); //Call it like a normal function...

cin.get();

}

4) Написати програму, що вводить зі стандартного потоку масив довжину N чисел типу float, обчислює середнє значення елементів цього масиву.

#include "stdafx.h"

#include <iostream>

#include <iomanip>

using namespace std;

void input(float\* arr, int n);

double average(float\* arr, int n);

int main()

{

int n = 0;

float\* arr = new float[n];

n = 5;

input(arr,n);

double aver = average(arr, n);

cout << aver;

delete[] arr;

system("pause");

return 0;

}

void input(float\* arr,int n)

{

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

{

cin >> arr[i];

}

}

double average(float\* arr, int n)

{

float sum = 0;

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

{

sum += arr[i];

}

double ser = sum / n;

return ser;

}

5) Створити клас, що асоціюється з датою(рік, місяць, день) та підтримує операції збільшення , зменшення , ініціалізацію місцями та днями і днями, видачу результату у місяцях або днях. Продемонструйте його виконання.

// ConsoleApplication5.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <iomanip>

using namespace std;

class Date

{

private:

int year;

int mounth;

int day;

public:

Date(int s\_year = 1, int s\_mounth = 1, int s\_day = 1);

virtual ~Date();

void add(int,int,int);

void minus(int,int,int);

void SetMDay(int, int);

void SetDay(int);

int GetDay();

int GetMounth();

};

Date::Date(int s\_day, int s\_mounth, int s\_year)

{

day = s\_day;

mounth = s\_mounth;

year = s\_year;

}

Date::~Date()

{

}

void Date::add(int n, int k, int j)

{

day += n;

mounth += k;

year += j;

}

void Date::minus(int n, int k, int j)

{

day -= n;

mounth -= k;

year -= j;

}

void Date::SetMDay(int s\_day, int s\_mounth)

{

day = s\_day;

mounth = s\_mounth;

}

void Date::SetDay(int s\_day)

{

day = s\_day;

}

int Date::GetDay()

{

if (day > 31 || day < 1)

return 0;

return day + 30 \* mounth + 365 \* year

}

int Date::GetMounth()

{

if (mounth > 12 || mounth < 1)

return 0;

return mounth + year \* 12;

}

int main()

{

Date a;

a.add(4, 4, 4);

a.minus(4, 4, 4);

a.SetMDay(30, 12);

a.SetDay(15);

cout << a.GetDay() << endl;

cout << a.GetMounth() << endl;

system("pause");

}

**Варіант 7**

1. Основні елементи об’єктно – орієнтованого стилю програмування.

Для об’єктно-орієнтованого стилю концептуальна основа полягає в об’єктному підході. Цьому підходу відповідають чотири головних елементи: абстрагування, обмеження доступу, модульність та ієрархія. Ці елементи є головними у тому розумінні, що за одним з класиків об'єктно-орієнтованого проектування програм Граді Бучем [6] без будь-якого з них підхід не буде повністю об’єктно-орієнтованим.

**2)** Приклад модульності

1. Первый пример:   
   в коде приложения расставляем инструкции сдедующего содержания: соединиться, передать пароль, перейти в папку, забрать файл. Все предельно просто и понятно.   
     
   Пример второй, несколько сложнее:  
   Создаем отдельный класс, которому передадим, хост, порт, пароль и путь до файла. Использовать класс будем примерно так:   
     
   записать\_в\_файл(локальный\_путь, новое\_соединение(хост, порт, пароль, путь).закачать())  
     
   Здесь несколько усложняется код внутри класса, но несколько упрощается его использование, не так ли?  
     
   Пример третий, еще более сложный:   
   создадим класс соединения, который будем использовать так:  
     
   Соединение({конфигурация})   
     
   Конфигурация для разных соединений может быть разная. Примеры:   
   {protocol: ftp, port: 224, user: vasia, password: secret}   
   или   
   [proto->http; domain->example.com; file->foo/bar/1.zip]

**3)** Написати вбудовану функцію, що підносить значення аргументу типу float до квадрату, та продемонструвати її використання.

#include <iostream>

using namespace std;

inline void sqr(float a)

{

float x = a \* a;

cout << "the number of square is:" << x;

}

float main()

{

float x;

sqr(5.5); //Call it like a normal function...

cin.get();

}

**4)Написати програму, що вводить зі стандартного потоку масив довжини N чисел типу double, обчислює середнє значення елементів цього масиву.**

#include "stdafx.h"

#include <iostream>

#include <iomanip>

using namespace std;

void input(double\* arr, int n);

double average(double\* arr, int n);

int main()

{

int n = 0;

double\* arr = new double[n];

cin >> n;

input(arr,n);

double aver = average(arr, n);

cout << aver;

delete[] arr;

system("pause");

return 0;

}

void input(double\* arr,int n)

{

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

{

cin >> arr[i];

}

}

double average(double\* arr, int n)

{

float sum = 0;

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

{

sum += arr[i];

}

double ser = sum / n;

return ser;

}

**5) Створити клас, що асоціюється з цілими дробами та підтримує операції + - ініціалізацію видачу знаменника та чисельника.Продем. використання**

#include <iostream>

#include <iomanip>

using namespace std;

class Fraction

{

private:

int numerator;

int denominator;

public:

Fraction(int s\_numerator = 1, int s\_denumerator = 1);

virtual ~Fraction();

void addNum(int);

void addDeno(int);

void minusNum(int);

void minusDeno(int);

void SetNum(int);

void SetDeno(int);

int GetNum();

int GetDeno();

};

Fraction::Fraction(int s\_numerator, int s\_denominator)

{

numerator = s\_numerator;

denominator = s\_denominator;

}

Fraction::~Fraction()

{

}

void Fraction::addNum(int n)

{

numerator += n;

}

void Fraction::addDeno(int n)

{

denominator += n;

}

void Fraction::minusNum(int n)

{

numerator -= n;

}

void Fraction::minusDeno(int n)

{

denominator -= n;

}

void Fraction::SetNum(int s\_numerator)

{

numerator = s\_numerator;

}

void Fraction::SetDeno(int s\_denominator)

{

denominator = s\_denominator;

}

int Fraction::GetNum()

{

return numerator;

}

int Fraction::GetDeno()

{

if (denominator == 0)

return NULL;

return denominator;

}

int main()

{

Fraction a;

a.SetNum(8);

a.SetDeno(9);

a.addNum(4);

a.addDeno(4);

a.minusNum(2);

a.minusDeno(2);

cout << a.GetNum() << endl;

cout << a.GetDeno() << endl;

system("pause");

}

**Варіант 9**

**1) абстрагува́ння** — спосіб та метод відокремлення деталей з метою отримання можливості зосередитись на найважливіших особливостях об'єкта.

Грубо кажучи, в програмуванні, абстрагування може бути або керування, або даних. **Абстрагування керування** є абстрагуванням від деталей дій, а **абстрагування даних** є абстрагуванням структур даних. Наприклад, абстрагування керування в [структурному програмуванні](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) полягає у використанні [підпрограм](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0) та визначених керівних конструкцій. Абстрагування даних дозволяє обробляти одиниці даних у змістовний спосіб. Наприклад, абстрагування є основною мотивацією створення [типів даних](https://uk.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85). [Об'єктно-орієнтоване програмування](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) може розглядатись як спроба абстрагувати як керування так і дані.

**2)Приклад ієрархії одиничного наслідування**

lass ArrayWithAdd : public Array {  
  
ArrayWithAdd(int n) : Array(n) {  
  
} ArrayWithAdd() : Array() {}  
  
ArrayWithAdd(const Array& a) : Array(a) {}  
  
void Add(const Array& a);

**3)** Написати вбудовану функцію, що обчислює двійковий логарифм зі значенням аргументу short, та продемонструвати її використання.

#include <iostream>

#include <cstdlib>

#include <cmath>

inline short log(short b)

{

return log(b) / log(2);

}

int main()

{

std::cout << log(4)<< std::endl;

return EXIT\_SUCCESS;

}

**4)Написати програму, що вводить зі стандартного потоку масив довжини N**

**Чисел типу short, обчислює середнє значення елементів цього масиву.**

*#include "stdafx.h"*

*#include <iostream>*

*#include <iomanip>*

*using namespace std;*

*void input(short\* arr, int n);*

*double average(short\* arr, int n);*

*int main()*

*{*

*int n = 0;*

*short\* arr = new short[n];*

*n = 5;*

*input(arr, n);*

*double aver = average(arr, n);*

*cout << aver;*

*delete[] arr;*

*system("pause");*

*return 0;*

*}*

*void input(short\* arr, int n)*

*{*

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

*{*

*cin >> arr[i];*

*}*

*}*

*double average(short\* arr, int n)*

*{*

*float sum = 0;*

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

*{*

*sum += arr[i];*

*}*

*double ser = sum / n;*

*return*

***5) Створити клас, що асоціюється з відстаннями та підтримує операції + - ініціалізацію , видачу значень об’єкту в межах або …..***

*#include "stdafx.h"*

*#include <iostream>*

*#include <iomanip>*

*using namespace std;*

*class Distance*

*{*

*private:*

*int distance;*

*public:*

*Distance(int s\_distance = 1);*

*virtual ~Distance();*

*void operator +(int);*

*void operator -(int);*

*void SetDistance(int);*

*int GetDistance();*

*static float Duym();*

*};*

*Distance::Distance(int s\_distance)*

*{*

*distance = s\_distance;*

*}*

*Distance::~Distance()*

*{*

*}*

*void Distance:: operator+(int n)*

*{*

*distance += n;*

*}*

*void Distance::operator -(int n)*

*{*

*distance -= n;*

*}*

*void Distance::SetDistance(int s\_distance)*

*{*

*distance = s\_distance;*

*}*

*int Distance::GetDistance()*

*{*

*return distance;*

*}*

*float Distance::Duym()*

*{*

*static float duym = 0.0254;*

*return duym;*

*}*

*int main()*

*{*

*Distance a;*

*a.SetDistance(198);*

*a + 5;*

*a - 1;*

*cout << a.GetDistance() \* a.Duym() << endl;*

*system("pause");*

*}*