**Лекція № 12.** **Складні типи даних – масиви, структури.**

**Використання масивів**

На практиці часто виникає необхідність в обробці даних у вигляді довільного набору значень, тобто масивів. **Масив** являє собою кінцеву іменовану послідовність величин од­ного типу, які розрізняються за порядковим номером.

Опис масиву у програмі відрізняється від опису простої змінної наявністю після імені квадратних дужок **«[ ]»**, в яких задається кількість елементів масиву (розмірність).

**У мові C++ нумерація елементів масиву починається з 0.**

**<тип> <ім’я> [n];**

**<тип> <ім’я> [n] = {значення};**

**<тип> <ім’я> [  ] = {значення}; /\*** масив відразу, можна не вказувати його розмір \*/

Приклад:

**float m [6];  
float m [6] = {3.4, 4.5, 5.6, 6.7, 8.9, 10.3};  
float m [  ] = {3.45, 4.56, 5.67, 6.78);**

Надалі кількість елементів змінити неможливо.

Для того щоб обнулити елементи оголошеного масиву, достатньо ініціювати його перший елемент: **int mas[10]={0}; тільки перший int mas[10]; mas[0]=0**

За замовчуванням, якщо в оголошеному масиві ініціюється тільки декілька перших елементів, то його інші елементи ініціюються нулями. Так, у випадку, коли **float mas[10]= {2.2,34.56};**, останні вісім елементів масиву одержать значення **0**.

Приклад.

**#include <iostream.h>**

**#include <math.h>**

**int main ( )**

**{ const int n = 7;**

**float x[n], y, a(10.5);**

**int i;**

**a= 10.5;**

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

**{**

**cin >> x[i]; //введення значення**

**y = a \* x[i] \* x[i] - sin(x[i]);**

**cout << " x[" <<i<<"] ="<<x[i]<<" y = " << y << endl;**

**}**

**system("pause");**

**return(0);**

**}**

Часто застосовуються багатовимірні масиви. У них позиція елемента визначається записом декількох індексів. Найбільш розповсюджені **двовимірні масиви** або матриці.

Масив задається або списком елементів у тому порядку, и якому вони розташовані у пам’яті, або подається як масив масивів, кожний з яких поміщається в свої фігурні дужки**«{}»**. При оголошенні і одночасному ініціюванні багатовимірних масивів можна опускати кількість індексів тільки першого виміру. Якщо ініціювання не здійснюється під час оголошення масиву, то кількість індексів треба вказувати явно.

**Для здійснення введення-виведення, а також для обробки елементів двовимірного масиву у програмі слід передбачати організацію двох циклів**: один — для завдання значень індексу рядків, другий — індексу стовпців.

Приклад.

**#include <iostream.h>**

**main()**

**{**

**const int n = 3, m = 4;**

**float M [n][m], z = 10;**

**int i, j;**

**cout << "\*\*\*\* Vvod matrix " << endl;**

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

**for (j = 0; j<m; j++)**

**{ cout << " M [" << i << "]" << "[" << j << "]=";**

**cin >> M [i][j];**

**M [i][j] += z; // M [i][j]= M [i][j] + z;**

**}**

**cout << "\n\n\*\*\*\*\* Rezult matrix: ";**

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

**{ cout << endl;**

**for (j = 0; j < m; j++)**

**cout << M [i][j] << " ";}**

**system("pause");**

**}**

У програмі при описі матриці **float M[n][m];** вказується діапазон змінення двох індексів, перший з яких призначений для індексування рядків **(і)**, другий — для індексування стовпців **(j)**. Введення, обробка і виведення елементів матриці здійснюються за допомогою двох циклів, один з яких є вкладеним в іншій. Це дозволяє при кожному значенні змінної **і**перебирати всі значення змінної **j**.

**Покажчики та масиви.**

**Покажчики** — це змінні, котрі містять адресу пам’яті, розподіленої для об’єкта відповідного типу. При оголошенні змінної-покажчика слід вказати тип даних, адресу яких буде містити змінна, та ім’я покажчика з символом «\*».

Загальний формат опису покажчика має вигляд:

**тип \*ім’я;**

де **тип** — тип значень, на який вказує покажчик;  
**ім’я** — ім’я змінної-покажчика;  
«\*» — операція над типом, що читається «покажчик на тип».

Наприклад:

**int \*рn** – покажчик на ціле значення;  
**float \*pf1, \*pf2;** — два покажчики на дійсні значення.

Покажчики не прив’язують дані до якого-небудь визначеного імені змінної і можуть містити адреси будь-якого неіменованого значення. Існує адресна константа **NULL**, що означає порожню адресу.

Мова C++ налічує лише дві операції, які стосуються адрес змінних, а саме:

**«&»** — **операція взяття адреси** («адреса значення»);

**«\*»** — **операція розіменування** («значення за адресою»).

Операція взяття адреси **«&»** застосовується разом зі змінною і повертає адресу цієї змінної. Операція розіменування «\*» використовується разом з покажчиками і бере значення, на яке вказує змінна-покажчик, розташована безпосередньо після символа «\*».

У мові C++ масиви і покажчики зв’язані між собою: ***ім’я масиву визначається як покажчик-константа на початковий (нульовий)елемент масиву.*** Так, наприклад, при оголошенні одновимірного масиву у вигляді **int mas [20];** його ім’я **mas** – покажчик на адресу початкового елемента масиву **&mas[0]**.

Існує два способи доступу до елементів масиву:

* з використанням індексу елемента масиву, наприклад, **mas[2]** або **mas[i];**
* з використанням адресного виразу, тобто виразу з покажчиком на масив, наприклад, **\*(mas + 2)** або **\*(mas + і)**.

Ім’я покажчика на масив можна записати так:

**int mas [20];**  
**int \*ptr1;**  
**ptr1 = mas;**      *//ptr1* *= &mas[0];,*

тут вирази **&mas[0] і mas** — еквівалентні.

Оскільки в комп’ютері для масивів завжди є суцільний блок комірок пам’яті, в яких розташовуються їх елементи, то адресу наступного елемента **mas[1]** можна вказати шляхом збільшення покажчика на **1**, а саме:

**р = &mas[0];**

**р++;** *//р=р  + 1;*

Перевіремо

**#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

**{ system("color F0");**

**int mas [20]={1,2,3,4,5,6,7};**

**int \*ptr1,\*p;**

**ptr1 = mas; //ptr1 = &mas[0];,**

**p = &mas[0];**

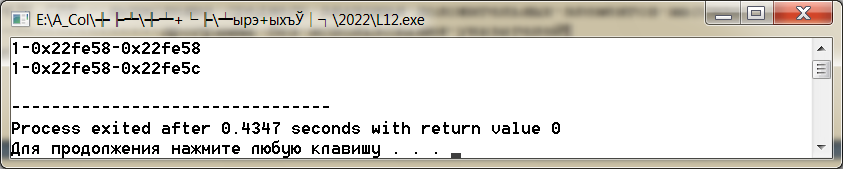
**cout<<mas[0]<<'-'<<ptr1<<'-'<<p<<endl;**

**p++; //p=p + 1;**

**cout<<mas[0]<<'-'<<ptr1<<'-'<<p<<endl;**

**return 0;**

**}**



Таким чином, адреса **і**-го елемента визначається як **р + і**. При цьому з урахуванням типу масиву і відведеної кількості байтів для кожного його елемента автоматично виконується операція збiльшення адреси, тобто:

**адреса х[і] = адреса х[0] + i\*sizeof (тип);**

***Приклад.*** Обчислити середнє значення додатних елементів одновимірного масиву. перший варіант без використання покажчиків

**#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

**{ system("color F0");**

**#include <iostream>**

**const int n = 10;**

**float mas[n], s = 0;**

**int i, kol = 0;**

**cout << "Input array " << endl;**

**for(i =0; i < n; i++) cin >> mas[i];**

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

**if (mas[i] > 0)**

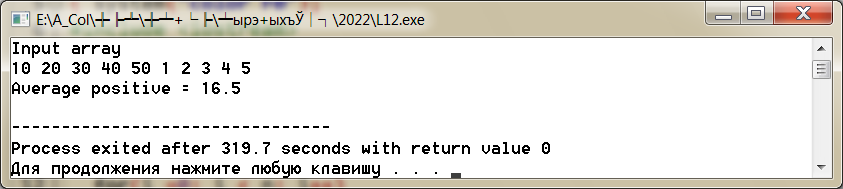
**{ s += mas[i]; //розрахунок підсумку**

**kol++; } //розрахунок кількості**

**cout.precision(3) ;**

**cout << "Average positive = " << s/kol << endl;**

**return 0;**

**}**Результати виконання програми:  
****Використовуючи ім’я масиву як покажчик на початок масиву (перший елемент), можна навести другий варіант програми:

**#indude <iostream.h>**

**main ( )**

**{ const int n = 10;**

**float mas[n], s;**

**int i, kol = 0;**

**s = 0;**

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

**{ cin >> \*(mas+i);**

**if (\*(mas+i) > 0)**

**{ s += \*(mas+i);**

**kol++; }**

**}**

**cout.precision(3);**

**cout <<"\n Average positive = " << s/kol << endl;**

**system("pause");**

**}**

Якщо описати покажчик і зв’язати його з масивом (адресувати на початок масиву), то з використанням арифметики покажчиків можна написати третій варіант цiєї програми.

*// использование арифметики указателей*

**#include <iostream.h>**

**main ( )**

**{ const int n = 10;**

**int і, kol(0);**

**float mas[n], s(0);**

**float \*pm = mas;**         *//pm= &mas[0];*

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

**{ cout << "Введите" << i << "элемент mas" << endl;**

**cin >> \*pm++;**

**cout << mas[i] << endl;**

**if (mas[i] >0)**

**{ s += mas[i];**

**kol++; }**

**}**

**cout.precision(3);**

**cout << "\n Среднее арифм. полож. элементов = " << s/kol << endl;**

**system("pause");**

**}**

У цій програмі для введення масиву застосований покажчик **\*рm**, а для роботи з масивом — ім’я масиву з індексом.

В останньому випадку використання покажчика **\*рm** призвело б до помилкового результату, оскільки цей покажчик в операціях введення збільшує свою адресу **(рm++)** після введення чергового елемента масиву і надалі вказує на ще не введений елемент.

Четвертий варіантпрограмної реалiзації прикладу:

*/\* использование указателей*

**#include <iostream.h>**

**main ( )**

**{ const int n = 10;**

**float mas[n], s = 0;**

**float \*pm = &mas[0];**  *//pm \*= &mas[0];*

**int i, kol = 0;**

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

**{ cout << "Введите" << i << "элемент mas" << endl;**

**cin >> \*pm;**

**if (\*pm >0)**

**{ s += \*pm;**

**kol++;**

**pm++; }**

**}**

**cout.precision(3);**

**cout << ”\n Среднее арифм. полож. элементов = " << s/kol << endl;**

**system("pause");**

**}**

**Визначення та використання структур**

**Структура** — це сукупність різнотипних елементів, яким присвоюється одне ім’я (воно може бути відсутнім), що займає одну ділянку пам’яті. Елементи, що складають структуру, називаються полями.

Змінна типу структура, як і будь-яка змінна, повинна бути описана. Цей опис складається з двох кроків: опису шаблону (тобто складу) або типу структури та опису змінних структурного типу.

Синтаксис опису структури має вигляд:

**struct [<ім’я структури>]**  
**{ <тип 1> ім’я поля 1;**  
**<тип 2> ім’я поля 2 . . .;**  
**} р1, р2 . . .;**

де **struct** — службове слово;

**<ім’я структури>** — ім’я типу структура (може бути відсутнім);

**<тип 1>, <тип 2>** — імена стандартних або визначених типів;

**ім’я поля 1, ім’я поля 2,…** — імена полів структури;

**р1, р2 . . .;** — імена змінних типу структура.

Наприклад, для знаходження середнього бала, отриманого студентами в період сесії з дисциплін «Математика», «Фізика» та «Програмування», визначимо таку структуру:

**struct stud**

**{ char fam [25];**            // фамилия и инициалы

**int mat, fiz, prg;**          // предметы

**float sb;**                   // средний балл

**}** **st1, st2;**

Змінні **st1 і st2** можна оголосити окремим оператором, наприклад:

**struct stud stl , st2;**

Ініціювання полів структури слід здійснювати або при її описі, або в тілі програми. При описі структури ініціювання полів виглядає, наприклад, так:

**struct stud**

**{ char fam [25];**

**int mat,fiz, prg;**

**float sb;}**

**st1 = {"Кравченко И. С.", 4, 5, 5},**

**st2 = {"Тесленко А. М.", 3, 4, 5};**

Якщо ініціювання виконується в тілі програми, то для звернення до імені поля треба спочатку записати ім’я структурної змінної, а потім ім’я поля. Ці обидва записи відокремлюються крапкою і являють собою складене ім’я.

Отже, у випадку появи змінної **st1** у програмі для її ініціювання можна записати

**stud st1** **= {“Кравченко И. С.”, 4, 5, 5};** або ініціювання виконується за допомогою складених полів. Розглянемо ілюстраційну програму:

**#include <iostream.h>**

**#include <string.h>**

**#include <stdio.h> // для puts: int puts(const char \*str)**

**#include <conio.h>**

**using namespace std;**

**main ( )**

**{ struct stud**     //----- описание структуры

**{ char fam [20];**

**int mat, fiz, prg;**

**float sb;**

**} st1, st2;**

**strcpy (st1.fam, "Кравченко I. С.");**

**st1.mat = 4;**

**st1.fiz = 5;**

**st1.prg = 5;**

**st1.sb = float (st1.fiz + st1.mat + st1.prg)/3;**

**st2 = st1;**

**puts (st2.fam);**      //---------------- вывод фамилии

**cout << st2.mat << st2.fiz << st2.prg << st2.sb << endl;**

**getch();**     //задержка экрана вывода результата

**}**

У наведеній програмі організується присвоювання всім полям структури **st1** відповідних значень.

**char strcpy (s, \*st);** — виконує операцію копіювання байтів рядка **st** у рядок **s** (включаючи  **“\0”**; повертає **s**)

Структурна змінна **st2** того ж типу, що і **st1**, тому справедлива операція **st2 = st1;.**

Якщо функція використовує тільки один структурний тип, то цей тип можна оголосити без імені. Тоді раніше розглянуту структуру можна оголосити таким чином:

**struct**

**{ char fam [25];**

**int mat, fiz, prg;**

**float sb;**

**} stl, st2;**

Коли при описі структур у деякій функції або в межах видимості змінних у різних функціях є багато (але не всі) однакових полів, то їх слід об’єднати в окрему структуру. Її можна застосовувати при описі інших структур, тобто поля структури можуть самі бути типу **struct**. Це називається **вкладеністю структур** — її можна використати, наприклад, якщо треба обробляти списки студентів та викладачів університету. Студентські списки містять дані: прізвище та ініціали, дата (день, місяць, рік) народження, група та середній бал успішності, а в списках викладачів присутні такі дані: прізвище, ініціали, дата народження, кафедра, посада. У процесі обробки списку студентів і списку викладачів можна оголосити відповідно такі структури:

**struct stud**

**{ char fio [25];**

**int den, god;**

**char mes [10];**

**char grup;**

**flout sb; }**

**struct prep  
{ char fio [25];  
int den, god;  
char mes [10];  
char kaf[10], dolg[10];  
}**

В оголошених типах однакові поля можна об’єднати в окрему структуру і застосовувати її при описі інших типів. Поетапно це виглядає так:

* загальна структура —

**struct spd**

**{ char fio [25];**

**int den, god;**

**char mes[10]; }**

* структура для опису інформації про студентів —

**struct stud**

**{ spd dr;**

**char grup;**

**float sb}**

**st1, st2;**

* структура для опису інформації про викладачів —

**struct prep**

**{ spd dr;**

**char kaf [10];**

**char dolg [15];**

**} pr1, pr2;**

У структурах **stud і prep** для оголошення поля, що містить дані про прізвище і дату народження, використовується раніше описаний тип **spd**. Тепер до поля **fio, den, god, mes**можна звернутися, використовуючи запис **st1.dr.fio**, наприклад, при зверненні до функції введення:

**gets (st1.dr.fio);** або    **gets (pr1.dr.fio);**

**Пояснення.**

Прототип функції gets: (б-ка cstdio) char \* gets( char \* string );

Отримати рядок зі стандартного потоку введення. Функція gets зчитує символи зі стандартного потоку введення до символу нового рядка n або доки, поки не буде досягнутий кінець файлу EOF, після чого зберігає лічені символи в рядок типу char.

Символ нового рядка n не копіюватиметься в рядок.

Нульовий символ автоматично додається після останнього копійованого символу в string, щоб сигналізувати про кінець рядка.

Функція gets дещо відрізняється від функції fgets, по-перше повертає рядок зі стандартного потоку введення в якості аргументу, а, по-друге, кінцевий символ нового рядка не входить до її складу . на відміну від fgets. А ще, gets не дозволяє встановити обмеження на кількість зчитувальних символів, тому потрібно бути обережними з розміром масиву, на який вказує параметр str, щоб уникнути переповнення буфера.

Параметри: **string**Покажчик на масив типу char, в якому зберігається рядок.

Значення, що повертається. У разі успіху, функція повертає той же параметр, string. Якщо відбувається помилка, повертається нульовий покажчик.

Використовуйте функції ferror або feof для перевірки внутрішніх станів показників помилки або кінця файлу – EOF.

**Об’єднання union.**

**Об’єднання** – це групування змінних, які розділяють одну й ту ж область пам’яті. В залежності від інтерпретації здійснюється звертання до тієї чи іншої змінної об’єднання. Усі змінні, що включені в об’єднання починаються з однієї границі.

*Об'єднання — це область пам'яті, у якій у різний час можуть зберігатися змінні різних типів. На відміну від структури, елементи якої записуються послідовно друг за другом, елементи об'єднання перекривають один одного в одній і тій же області пам'яті. Таким чином, у той час як розмір структури дорівнює сумі розмірів її елементів, розмір об'єднання дорівнює максимальному розміру серед усіх його елементів.*

Об’єднання дозволяє представити дані, що змінюються в компактному вигляді. Одні й ті ж дані можуть бути представлені різними способами з допомогою об’єднань.

Так як і **структури**, об’єднання вимагають оголошення типу (шаблону) та оголошення змінної цього типу.

Загальна форма оголошення типу (шаблону) об’єднання

Оголошення об’єднання (типу об’єднання або шаблону об’єднання) починається з ключового слова union.

union **ім’я\_типу\_об’єднання**

{

тип змінна1;

тип змінна2;

...

тип зміннаN;

};

де

* ім’я\_типу\_об’єднання – безпосередньо ім’я новостворюваного типу;
* змінна1, змінна2, зміннаN – змінні, що є полями об’єднання. Ці змінні можуть бути різних типів;
* тип – тип змінної, що є полем об’єднання.

Тип змінної може бути:

* **базовим типом**, прийнятим в мові C;
* тип **структура**;
* тип об’єднання;
* тип клас.

Довжина об’єднання – це розмір пам’яті в байтах, що виділяється для однієї змінної цього типу об’єднання.

Довжина об’єднання обчислюється як максимум з усіх довжин (розмірів у байтах) окремих полів шаблону. Одне поле – це оголошення однієї змінної в об’єднанні.

Приклад оголошення типу (шаблону) об’єднання та змінну цього типу

Нехай задано тип об’єднання, що містить змінні типів з плаваючою точкою

// оголошення типу "об'єднання Floats"

union Floats

{

float f; // розглядається 4 байти

double d; // розглядається 8 байт

};

Тип об’єднання Floats містить 2 змінні з іменами f та d. Змінна f є типу float, змінна d є типу double. Для змінної f типу float розглядається (береться до уваги) 4 байти. Для змінної d типу double береться до уваги 8 байт, тому що компілятор виділяє для цього типу саме 8 байт.

Щоб використати об’єднання Floats в іншому програмному коді (методі, обробнику події тощо) потрібно оголосити змінну типу Floats як показано нижче

Floats Fl;

int d;

Fl.f = 20.5; // Fl.d не визначене

Fl.d = -100.35; // тепер Fl.f не визначене

d = sizeof(Fl); // d = 8

Оскільки розміщення змінних в пам’яті умовно починається з однієї адреси, то для змінної Fl типу Floats виділяється 8 байт пам’яті. Це зв’язано з тим, що змінна типу double вимагає більше пам’яті для свого представлення ніж змінна типу float.

На рисунку 1 відображено розміщення (інтерпретація) змінних f, d з об’єднання Floats.



Рис. 1. Представлення змінних f, d в об’єднанні Floats

Доступ до полів об’єднання здійснюється так само, як і для структури:

* з допомогою символу **‘ . ‘**;
* з допомогою послідовності символів **‘->’** у випадку, коли оголошено змінну-покажчик на об’єднання.

Приклад оголошення та використання покажчика на об’єднання

Шаблон об’єднання може включати поля, що є структурами, об’єднаннями та класами.

У прикладі нижче оголошується шаблон об’єднання з іменем Types, що містить два вкладені об’єднання Floats та Ints, структуру ArrayOfChars і клас MyPoint.

Оголошення структури та об’єднань має такий вигляд

// об'єднання цілочисельних типів

union Ints

{

unsigned short int a;

unsigned int b;

unsigned long int c;

};

// структура, що містить 2 рядки

struct ArrayOfChars

{

char A[10];

char B[8];

};

// оголошення типу "об'єднання Floats"

union Floats

{

float f; // розглядається 4 байти

double d; // розглядається 8 байт

};

Оголошення типу об’єднання Types з вкладеними складними типами Ints, Floats, ArrayOfChars

// оголошення типу "об'єднання Types"

union Types

{

Floats Fl; // об'єднання

Ints I; // об'єднання

ArrayOfChars A; // структура

};

Використання об’єднання Types в деякому програмному коді:

// оголосити змінну типу "об'єднання Types"

Types T;

// змінити значення полів змінної T

T.Fl.f = (float)20.35; // об'єднання Floats

T.I.b = 230; // об'єднання Ints

T.A.A[2] = 'A'; // структура ArrayOfChars

Масив об’єднань

// Приклад оголошення та використання масиву об'єднань

Floats F[5]; // оголошується масив з 5 об'єднань типу Floats

// заповнення значень полів

for (int i=0; i<5; i++)

{

F[i].d = i\*0.2 + i\*i;

}

[**⇑**](https://www.bestprog.net/uk/2018/02/01/unions-the-keyword-union-examples-of-declaring-and-using-unions_ua/#contents)

Особливості застосування операції sizeof() для об’єднань та структур

У програмах на C++ для визначення розміру змінної типу “структура” або “об’єднання” **обов’язково потрібно використовувати операцію sizeof**. Визначення розміру “вручну” є помилковим тому що:

* розміри деяких вбудованих типів (наприклад тип int) можуть бути різними для різних комп’ютерів. Наприклад, на одних платформах для типу int буде виділено 2 байти, на інших 4 байти;
* компілятор робить так зване **“вирівнювання пам’яті”** на межі слова (2 байти) або абзацу (16 байт). Наприклад, якщо компілятор робить вирівнювання на межі абзацу, то структура (об’єднання) типу ArraysOfChars:

// структура, що містить 2 рядки

struct ArrayOfChars

{

char A[10];

char B[8];

};

може займати в пам’яті 24 байти. Тому що для масиву A виділяється 16 байт а не 10 байт. Компілятор додатково виділяє 6 байт щоб реалізувати вирівнювання на межі машинного слова .

*Під машинним словом розуміють ту кількість байт (або інших елементарних примітивів), яке процесор обробляє найбільш швидко (зазвичай це розмір основних регістрів процесора). Спочатку словом називали 8 біт (байт). Потім були 16 бітні архітектури, в яких словом називали вже 2 байта. Зараз основні процесори мають 32 або 64 біта. У них і соответствущий розмір машинного слова.*

*Якщо абстрагуватися від конкретної машини, то найчастіше словом (Word, в програмуванні) називають двубайтовую послідовність.*

Таким чином, використання операції sizeof() для визначення типу структури чи об’єднання гарантує переносність програмного коду.

**Класи та об‘єкти**

**Поняття об'єкта**

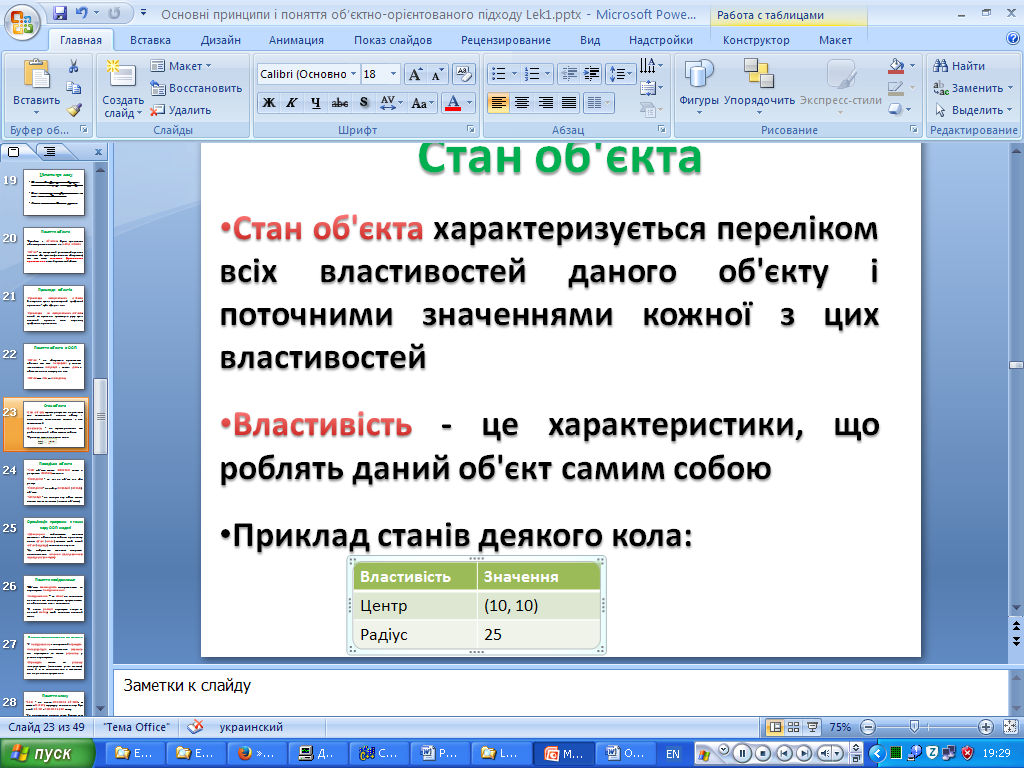
* Будь-що є **об'єктом.** Будь предметна область розглядається як набір **об'єктів**
* **Об'єкт -** це конкретний упізнаваний предмет, одиниця або сутність (реальна чи абстрактна), що має чітко визначене функціональне призначення в даній предметній області

Приклади матеріальних об'єктів: більярдна куля, тривимірний графічний примітив - куб, сфера і т.п.

Приклади не матеріальних об'єктів, подій чи процесів: траєкторія руху кулі, хімічний процес, лінія перетину графічних примітивів

**Поняття об'єкта в ООП**

* **Об'єкт -** це абстракція предметної області, що має **інтерфейс** у вигляді іменованих **операцій** і власні **дані,** з обмеженням доступу до них.
* Об'єкт має **стан** та **поведінку.**

**Стан об'єкта** характеризується переліком всіх властивостей даного об'єкту і поточними значеннями кожної з цих властивостей.

**Властивість -** це характеристики, що роблять даний об'єкт самим собою. Приклад станів деякого кола:

**Стан об'єкта** може змінитися тільки в результаті виклику методів.

* **Поведінка -** це те, як об'єкт діє або реагує
* **Поведінка -** це набір **операцій (методів)** об'єкта

**Операція -** це послуга, яку об'єкт може надати своїм клієнтам (іншим об'єктам)

**Організація програми з точки зору ООП моделі**

**Обчислення** здійснюється шляхом взаємодії об'єктів між собою, при якому один **об'єкт (клієнт)** вимагає щоб інший **об'єкт (сервер)** виконав деяку дію. Це відбувається шляхом відправки повідомлення **клієнтом (відправником) серверу (отримувачу) .**

**Поняття повідомлення**

Об'єкти **взаємодіють** відправляючи та отримуючи **повідомлення.**

**Повідомлення –** це **запит** на виконання деякої дії, що доповнюється аргументами необхідними для її виконання**.**

В якості **реакції** отримувач запускає деякий **метод,** щоб виконати наданий запит.

**Відмінність повідомлень від методів**

У **повідомлень** є конкретний **отримувач**

**Інтерпретація** повідомлення залежить від **отримувача** та може **різнитися** у різних отримувачів.

**Отримувач** може по **різному** інтерпретувати (викликати різні методи) одне й те ж повідомлення в залежності від переданих аргументів

**Поняття класу**

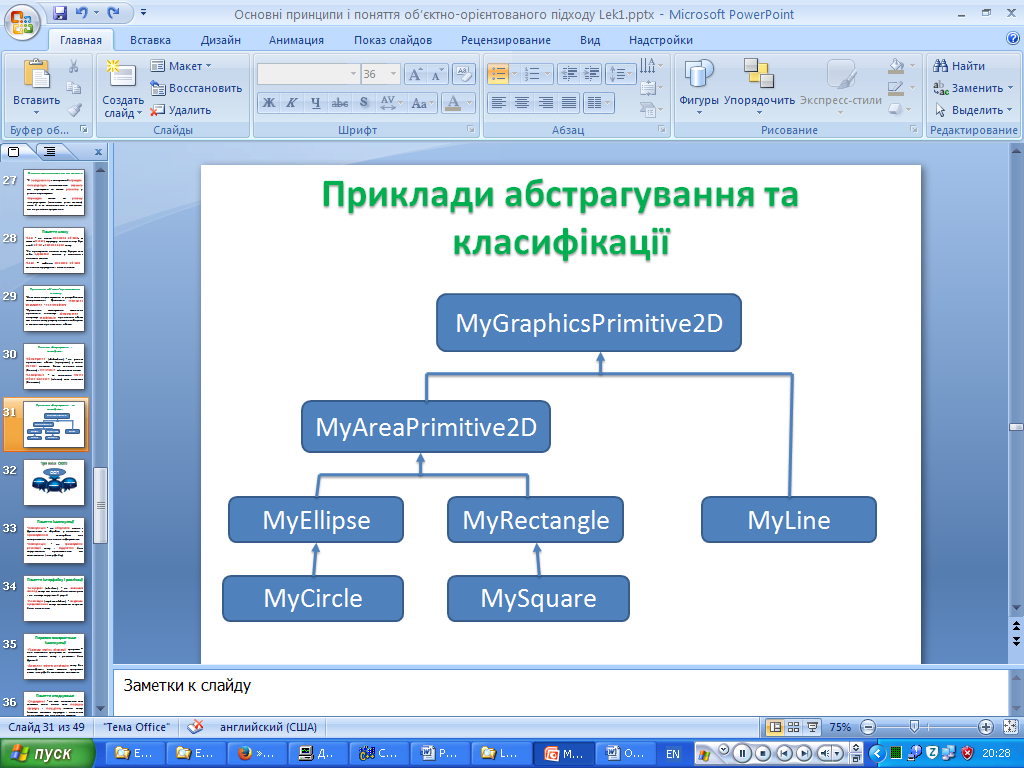
**Клас -** це деяка **множина об'єктів,** що мають **спільну структуру** та **поведінку.** Будь-який об'єкт є екземпляром класу.

Усі екземпляри одного класу будуть вести себе **однаковим** чином у відповідь на однакові запити.

**Клас** – шаблон **множини** об'єктів із спільною структурою і поведінкою.

**Принципи об'єктно-орієнтованого підходу**

Клас повинен проектуватися и розроблятися з використанням принципів **інкапсуляції, успадкування і поліморфізму.**

Правильне застосування зазначених принципів підвищує **абстрагування** та покращує **класифікацію** предметної області, що в кінцевому рахунку дозволяти боротися зі складністю предметної області.

**Абстрагування (**abstraction) - це уявлення предметної області (програми) у вигляді **меншої** кількості більш великих понять (блоків) і **мінімізація** зв'язків між ними.

**Класифікація -** це виділення **істотних, стійких** відносин (зв'язків) між поняттями (блоками).

**Поняття інкапсуляції**

**Інкапсуляція -** це **об'єднання** даних з функціями їх обробки у поєднанні з **приховуванням** непотрібної для використання цих даних інформацією**.**

**Інкапсуляція -** це **приховування реалізації** класу і **відділення** його внутрішнього представлення від зовнішнього (інтерфейсу).

**Поняття інтерфейсу і реалізації**

**Інтерфейс** (interface) - це **зовнішній вигляд** класу, що виділяє його істотні риси і не показує внутрішній устрій.

**Реалізація** (implementation) **- внутрішнє представлення** класу, включаючи секрети його поведінки.

**Переваги використання інкапсуляції**

**Підвищує ступінь абстракції** програми – для написання програми не вимагається знання даних класу і реалізації його функцій.

**Дозволяє змінити реалізацію** класу без модифікації іншої частини програми, якщо інтерфейс залишився колишнім.

**Поняття наслідування / спадкування**

**Наслідування** - це таке відношення між класами, коли один клас **повторює структуру** і **поведінку** іншого класу. Іншими словами, структура і поведінка передається від предка до нащадка.

**Наслідування** реалізує ставлення "is-a" між двома класами, тобто **дочірній** класповинен бути **частковим** або спеціалізованим випадком **батьківського** класу. Відношення типу is-a (є, являється) при якому один клас є підвидом іншого класу. При такому співвідношенні один клас розширює (деталізує) можливості іншого класу.

**Переваги використання наслідування**

При наслідуванні загальні властивості і поведінка не описуються в базовому класі (батьківському класі, класі-предку від якого ведеться наслідування) , що **скорочує** обсяг програми.

Виділення загальних рис різних класів в один клас-предок є потужним **механізмом абстракції** та **класифікації .**

**Поняття поліморфізму**

**Поліморфізм -** це використання **одного імені** для **різних сутностей.** При цьому різнорідні суті, виступаючи під одним ім'ям, сприймаються як **однотипні.**

**Поліморфізм -** це можливість обробки даних **змінного** типу.

**Поліморфізм -** це можливість оперувати об'єктами, **не** володіючи **точним** знанням їх типів.

**Переваги використання поліморфізму**

Дозволяє записувати **алгоритми** лише **одного** разу і потім **повторно** їхвикористовувати для **різних типів даних.**

**Звужує концептуальний простір**, тобто зменшує кількість інформації, яку необхідно пам'ятати програмісту.

Клас являє собою абстрактний тип даних, що визначається користувачем і являє собою модель реального об’єкта у вигляді даних та функцій для роботи з ними.

Оголошення класу має таку форму:

class <сім’я класу> : <список класів-батьків>

{ public:             // доступно для всіх **відкритий доступ**

<дані, методи, властивості, події>

protected: // доступно тільки нащадку **захищений доступ**

<дані, методи, властивості, події>

private:   // доступно тільки в класі **закритий доступ**

<дані, методи, властивості, події>

} <список змінних>

Об’єкт — це змінна типу клас. Дані класу називаються *полями,* а функції — *методами,* що призначені для обробки полів. Крім методів, тип клас може мати спеціальні функції — конструктори і деструктори.

Властивість ООП - можливість наслідування даних і функцій. У випадку, коли один клас успадковує інший, базовий клас називають **батьківським**, а той, що успадковує, — похідним або **нащадком**.

Розміщення описів змінних і функції з їх обробки, тобто полів та методів в одному класі називається **інкапсуляцією**.

Класи в С++ мають три різних рівні доступу до своїх елементів, тобто даних (полів) і функцій (методів):

* закриті елементи (private);
* захищені елементи (protected);
* відкриті елементи (public).

До даних у закритому розділі (private) мають доступ тільки функції-елементи свого класу. Класам-нащадкам забороняється доступ до закритих даних своїх базових (батьківських) класів. За замовчуванням усі дані-члени класу мають атрибут private(закритий).

До даних у захищеному розділі (protected) мають доступ функції свого класу і функції класів-нащадків.

У свою чергу до даних відкритого розділу (public) можуть звертатися будь-які функції.

**Типові помилки програмування**

1. Приклад. Різниця між сьомим елементом масиву та елементом масиву сім. Оскільки нумерація масиву починається з нуля, то сьомий елемент масиву буде мати індекс 6. Елемент масиву 7 насправді є восьмим елементом.

Часто про це забувають, що викликає помилки.

2. Елементам масиву можна присвоювати початкові значення. Наприклад,

**float m [6] = {3.4, 4.5, 5.6, 6.7, 8.9, 10.3};**

Якщо початкових значень в списку ініціалізації менше ніж елементів масиву, решта автоматично заповнюється нульовими значеннями. Автоматично нульові значення елементам масиву не присвоюються. Для заповнення масиву нульовими значеннями потрібно це значення присвоїти першому елементу масиву. Якщо потрібна ініціалізація і про це забувають, то це стає джерелом помилки.

3. Завдання в списку ініціалізації більшої кількості елементів ніж розмір масиву є синтаксичною помилкою.

4. Масив може бути оголошений як константа. Наприклад:

**const float m [6];**

Значення такому масиву присвоюють при оголошенні. Присвоювання значень в операторі під час виконання є синтаксичною помилкою.

5. Завершення директиви препроцесора **#include** крапкою з комою – директиви препроцесора не є операторами С++.

6. Посилання на елемент за межами масиву. Ефекти щодо таких посилань залежать від системи.

7. Оголошення

**char string2[20];**

створює символьний масив для 19 елементів, останній елемент - '\**0'** позначає кінець тексту. Оператор

**cin >> string2;**

зчитує рядок з клавіатури до  **string2**.

Недостатній розмір масиву, до якого вводиться символьний рядок, може привести до втрати даних в програмі та до інших помилок.

8. Помилковим є вважати, що елементи локального масиву **static,** в функції отримують нульові значення при кожному виклику функції.

9. Операція \* не поширюється на всі змінні в оголошенні. Кожний покажчик повинен оголошуватися окремо.

10. Разіменування покажчика, який не був правильно ініціалізований або якому не присвоєно значення, що вказує конкретне місце в пам’яті може викликати невиправну помилку виконання або неочікуваним чином змінити дані, що приведе до неправильних результатів.

**Техніка програмування**

1. Визначення розміру масиву через іменовану константу робить програму більш маштабованою.

**const arraySize=10;**

**float m [arraySize];**

**Хороший стиль програмування**

1. Намагайтеся програмувати зрозуміло. Інколи краще пожертвувати більш високої ефективності використання пам’яті або процесорного часу задля більшої зрозумілості програми. Але до цього процесу потрібно підходити зважено, інколи ефективність більш важлива ніж зрозумілість. В цьому випадку потрібно пам’ятати про важливість коментарів.

2. При використанні циклів з масивом індекс масиву не повинен бути меншим за нуль і повинен бути меншим за кількість елементів масиву.

3. В програмі повинна забезпечуватися правильність всіх значень, що вводяться, для унеможливлення впливу помилкової інформації на результати обчислення.

4. Хоч це і не обов’язково, доцільно в імена змінних покажчиків додавати символи Ptr.

5. Покажчики повинні інціалізуватися або при оголошенні або за допомогою оператора присвоювання. Покажчик може отримувати значення NULL, 0 або адресу. Для унеможливлення непередбачуваних результатів покажчикам потрібно присвоювати початкові значення.

6. Існує негласна домовленість, що імена змінних повинні починатися з маленької літери, а слова в середині імені з великої, наприклад, **arraySize.**

**7.** Імена змінних повинні бути короткими та наочними. Уникайте імен, які не мають сенсу.

*Для самостійного вивчення (10 годин)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Література*

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Єжова Л. Ф. Алгоритмізація і програмування процедур обробки інформації: Навч.-метод. посіб. для самост. вивч. дисц. К.: КНЕУ, 2000.
3. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

*Контрольні запитання для самоперевірки*.

1. Що таке масив та які існують різновиди масивів?
2. Як здійснюється звернення до елементів масивів?
3. Як у С++ реалізується введення-виведення елементів масиву?
4. Що таке рядки та значення елементів символьного типу?
5. Що являє собою масив символьного типу?
6. Як здійснюється введення символьних даних?
7. Як виконується порівняння даних символьного типу?
8. Як визначити кількість символів у рядку?
9. Які функції мови С++ необхідні для виділення підрядка з рядка?
10. Як описуються дані типу структура?
11. Які типи полів може містити структура?

*Контрольні запитання для надання письмових відповідей*.

1. Наведіть приклад використання операції конкатенації.
2. Охарактеризуйте функції пошуку підрядка в рядку.
3. Наведіть приклади використання структур.
4. Наведіть приклади поліморфізму