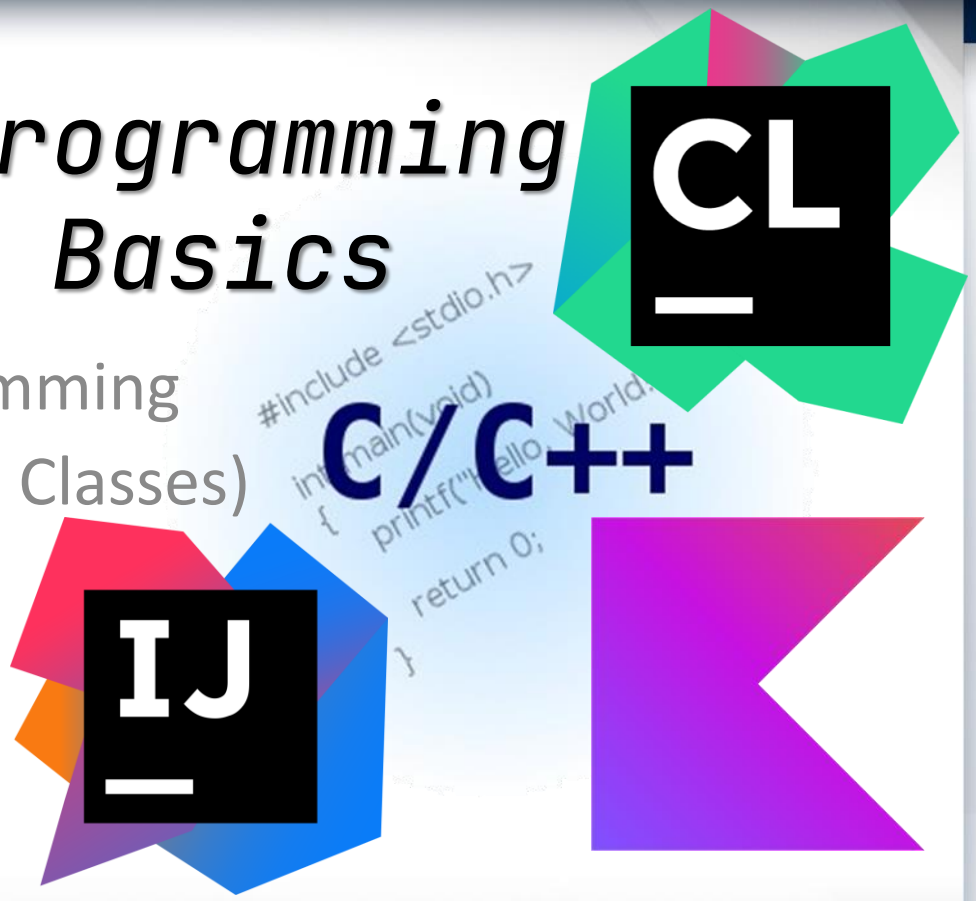
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming
Programming Basics

C/C++/Kotlin programming

(p.5 – Structures / Data Classes)



Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

Structures in C/C++

- In C/C++, a structure is a user-defined data type that groups together variables of different data types under a single name.
- A structure is a way to organize data that is related to each other, and it provides a convenient way to access and manipulate multiple variables at once.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Structures in C/C++

- A C/C++ structure is defined using the "struct" keyword, followed by the name of the structure and a set of braces that enclose the member variables of the structure.
- For example:

```
struct Person {
    char name[50];
    int age;
    double height;
};
```

Structures in C/C++

```
struct Person {
    char name[50];
    int age;
    double height;
};
```

In this example, we define a structure called "Person" that contains three member variables: a character array called "name" to store the name of the person, an integer variable called "age" to store their age, and a floating-point variable called "height" to store their height.

Structures in C/C++

- Once a structure is defined, we can create variables of that structure type and access the member variables using the dot operator.
- For example:

```
Person john;
strcpy(john.name, "John Smith");
john.age = 25;
john.height = 1.7;
```

Note: in this example "strcpy" – string function that copies value of string literal into character array

Declaring / Initialization

- In C++, structures can be declared and initialized in several ways.
- Here is an example of a structure declaration:

```
struct Person {
    string name;
    int age;
    double height;
};
```

This declares a structure named "Person" with three member variables: a string called "name", an integer called "age", and a floating-point number called "height".

Declaring / Initialization

- To initialize a structure variable, we can use either the C-style syntax or the C++11 uniform initialization syntax.
- Here's an example of initializing a structure using the C-style syntax:

```
Person johnSmith;
john.name = "John Smith";
john.age = 25;
john.height = 1.7;
```

In this example, we create a variable called "john" of type "Person" and assign values to its member variables using the dot operator.

Declaring / Initialization

- We can also initialize a structure using the C++11 uniform initialization syntax:

```
Person jane{"Jane Doe", 30, 1.6};
```

or

```
Person jane = {"Jane Doe", 30, 1.6};
```

In this example, we create a variable called "jane" of type "Person" and initialize its member variables using curly braces.

Pointer to structure

- In C++, a pointer to a structure can be used to access and manipulate the members of a structure.
- A pointer is a variable that stores the memory address of another variable, and it can be used to indirectly access the value stored in that variable.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Pointer to structure

- To declare a pointer to a structure in C++, we can use the same syntax as declaring a pointer to any other data type, with the addition of the structure name:

```
struct Person {
    char name[50];
    int age;
    double height;
};
```

```
Person *ptrPerson;
// declare a pointer to a Person structure
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Pointer to structure

```
struct Person {
    char name[50];
    int age;
    double height;
};
```

```
Person *ptrPerson;
// declare a pointer to a Person structure
```

- In this example, we declare a pointer to a structure called "Person" using the "*" symbol. The pointer variable is named "ptrPerson".

Pointer to structure

- To initialize the pointer to point to a specific instance of the structure, we can use the "address-of" operator "&" with the variable name:

```
Person johnSmith = {"John Smith", 25, 1.7};

// set the pointer to point to the "johnSmith" variable
ptrPerson = &johnSmith;
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Pointer to structure

```
Person johnSmith = {"John Smith", 25, 1.7};

// set the pointer to point to the "johnSmith" variable
ptrPerson = &johnSmith;
```

- In this example, we create an instance of the "Person" structure called "johnSmith" and initialize its member variables.
- We then set the pointer "ptrPerson" to point to the address of the "johnSmith" variable using the "&" operator.

Pointer to structure

- Once the pointer is initialized, we can access and manipulate the members of the structure using the "->" operator:

```
cout << "Name: " << ptrPerson-> name << endl;
cout << "Age: " << ptrPerson-> age << endl;
```

- In this example, we use the "->" operator to access the member variables of the structure through the pointer.
- The "->" operator is used to dereference the pointer and access the members of the structure.

Declaring / Initialization

- We can also use the "new" operator to dynamically allocate memory for a structure:

```
Person *ptrPerson2 = new Person{"Mike Smith", 35, 1.65};
```

- In this example, we create a pointer to a "Person" structure called "ptrPerson2" and dynamically allocate memory for a new "Person" structure.
- We initialize the member variables using curly braces.

Finalization

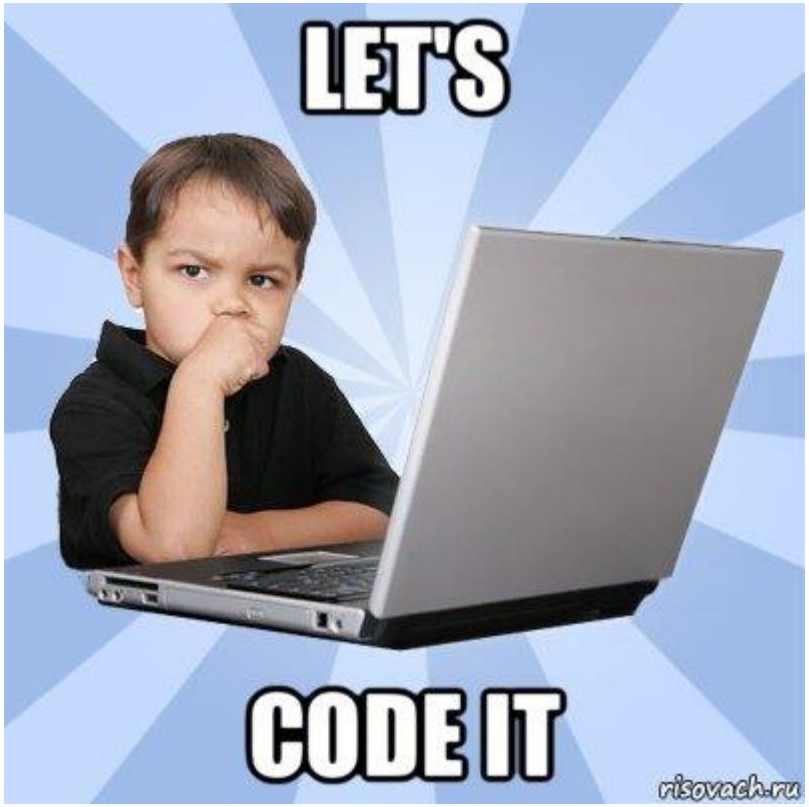
- When we're done using the dynamically allocated structure, we should free the memory using the "delete" operator:

```
delete ptrPerson2;
```

- This frees the memory that was allocated for the structure.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Demo



Arrays of structures

```
const int ARRAY_SIZE = 10;
Person people[ARRAY_SIZE];
```

- This creates an array of Person structures with a size of 10.
- You can access the elements of the array using the index notation, just like with any other array.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Arrays of structures

- To initialize the elements of the array, you can use a loop like this:

```
for (int i = 0; i < ARRAY_SIZE; i++) {
    cout << "Enter name for person #" << i+1 << ": ";
    getline(cin, people[i].name);
    cout << "Enter age for person #" << i+1 << ": ";
    cin >> people[i].age;
    cout << "Enter height for person #" << i+1 << ": ";
    cin >> people[i].height;
    cin.ignore(); // to consume the newline
                  // character left in the input stream
}
```

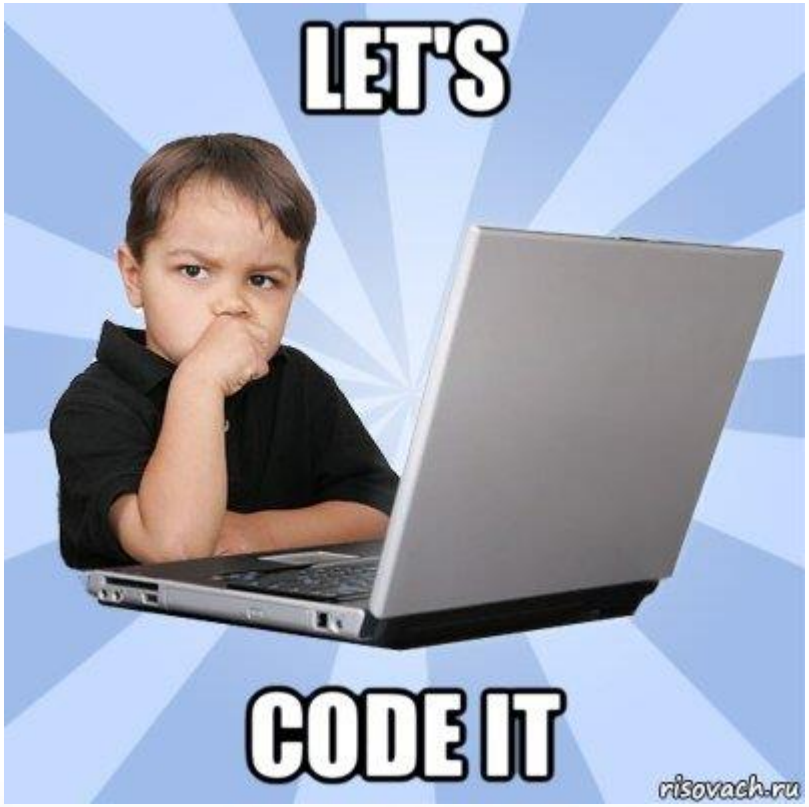
Arrays of structures

- You can then access the data in the array using the same index notation:

```
cout << "Name of person #3 is: "
     << people[2].name << endl;
cout << "Age of person #5 is: "
     << people[4].age << endl;
cout << "Height of person #9 is: "
     << people[8].height << endl;
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Demo



Kotlin: Data Class

- In Kotlin, a data class is a special type of class that is primarily used to hold data.
- It is often used to create objects that represent entities in an application such as a user, a product, or a message.

```
data class User(
    val name: String,
    val age: Int,
    val email: String
)
```

Kotlin: Data Class

```
data class User(
    val name: String,
    val age: Int,
    val email: String
)
```

- In this example, the User class has three properties: name, age, and email.
- The data keyword before the class name tells the compiler to generate some common methods such as toString(), equals(), hashCode(), and copy() for this class.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Kotlin: Data Class

- You can create an instance of this class by using the constructor:

```
val user = User("John Doe", 25, "john.doe@example.com")
```

The toString() method is automatically generated for the data class, so you can print the object like this:

```
println(user)
// prints
//"User(name=John Doe, age=25, email=john.doe@example.com)"
```

Kotlin: Data Class

- The equals() and hashCode() methods are also generated, so you can compare two User objects like this:

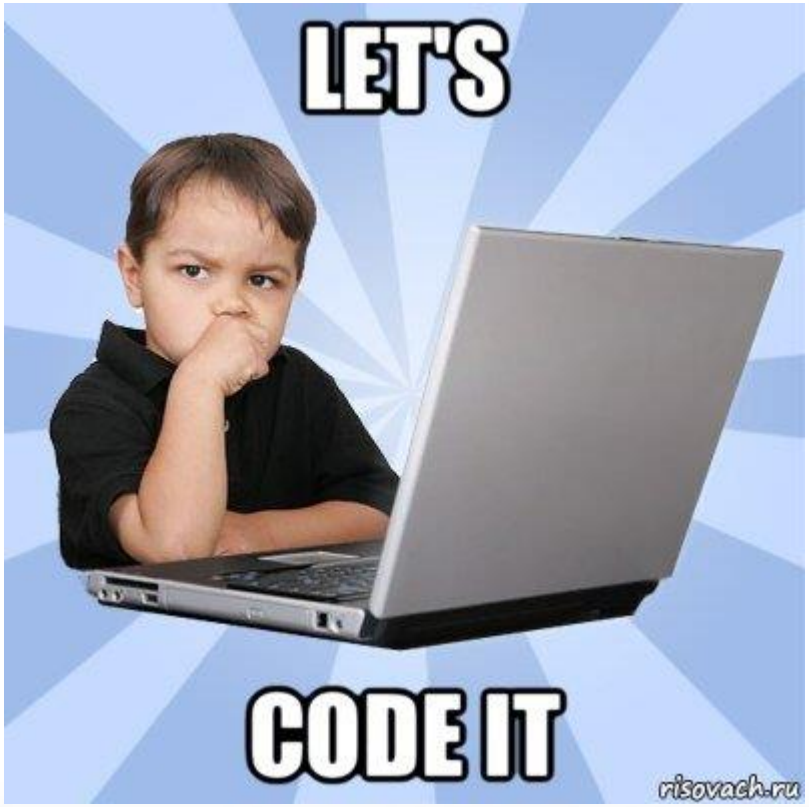
```
val user2 = User("John Doe", 25, "john.doe@example.com")
val user3 = User("Jane Doe", 30, "jane.doe@example.com")

println(user == user2) // prints true
println(user == user3) // prints false
```

The copy() method is also generated, which allows you to create a new object with some properties copied from the original:

```
val updatedUser = user.copy(name = "John Smith")
println(updatedUser) // prints "User(name=John Smith,
age=25, email=john.doe@example.com)"
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Demo

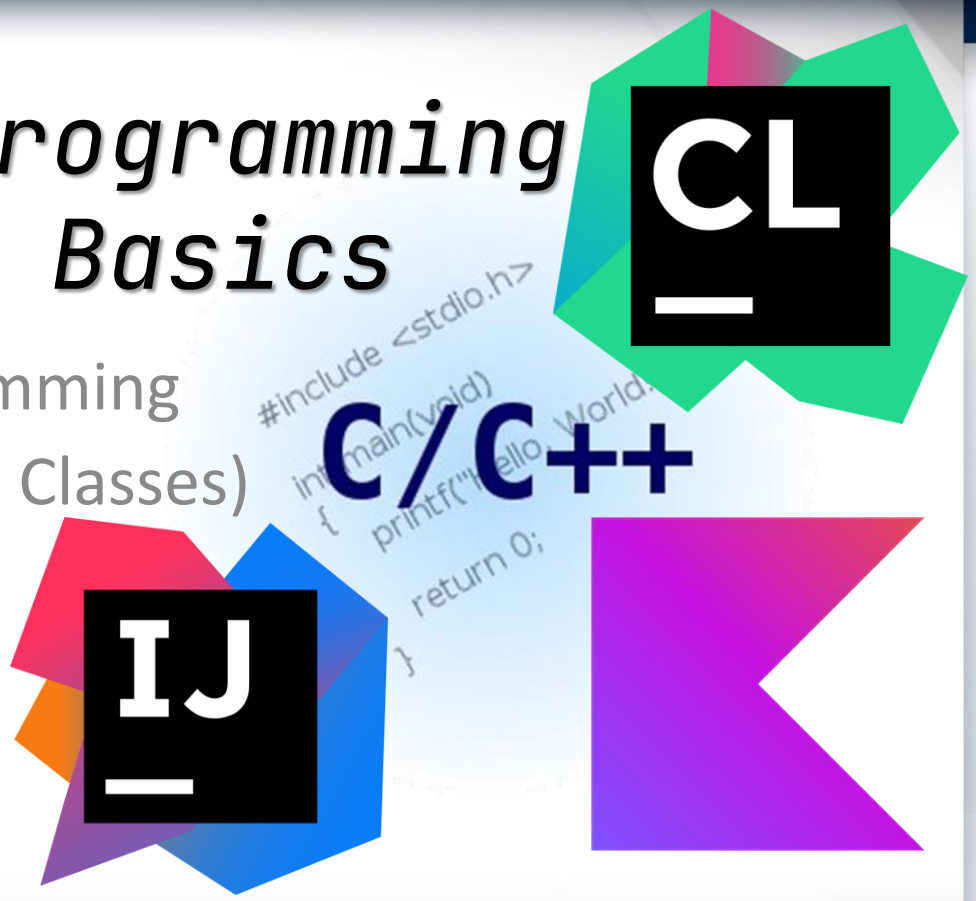


НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming
Programming Basics

C/C++/Kotlin programming

(p.4 – Structures / Data Classes)



Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua