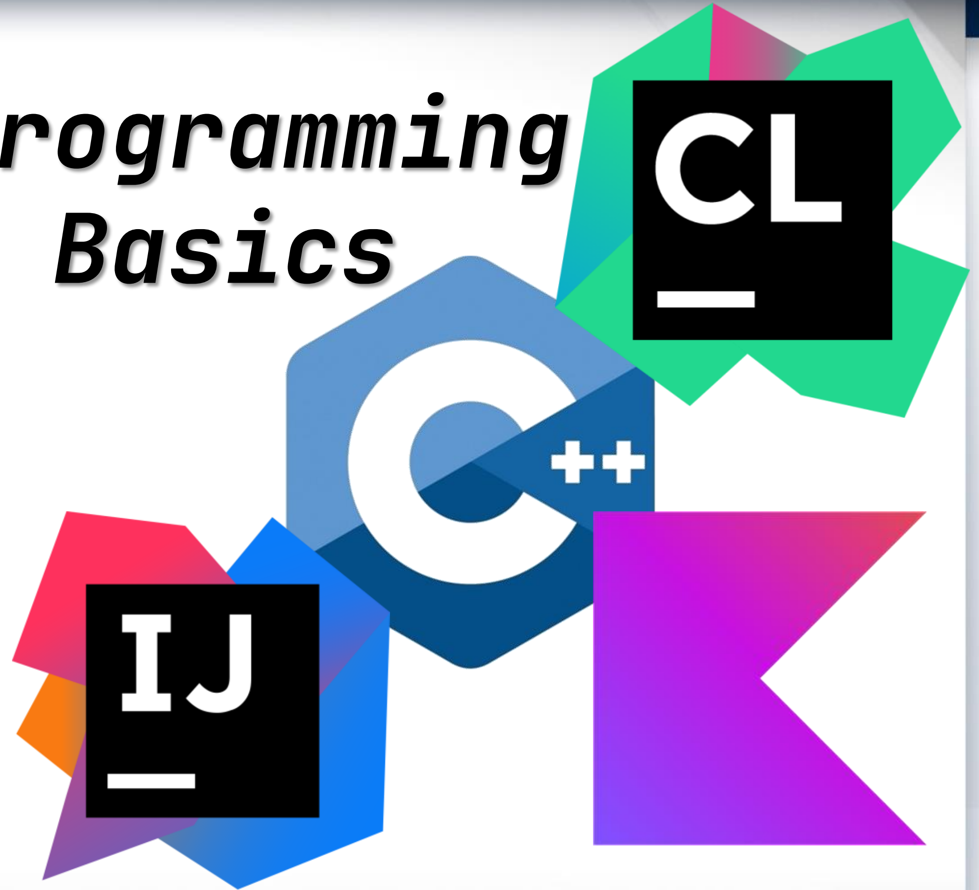
НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ
КОРАБЛЕБУДУВАННЯ
ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming
Programming Basics

C/C++/Kotlin programming

(p.7 – Pointers)



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

Pointers

- In C++, a pointer is a variable that stores the memory address of another variable. Pointers are a powerful feature of the language, as they allow programs to access and manipulate memory directly.
- To declare a pointer in C++, you use the * symbol before the variable name. For example, the following code declares a pointer variable named ptr:

```
int* ptr;
```

Pointers

```
int* ptr;
```

- This declares ptr as a pointer to an integer.
- To assign a value to a pointer, you use the & operator to get the address of a variable.
- For example, the following code assigns the address of the x variable to the ptr pointer:

```
int x = 10;
int* ptr = &x;
```

Pointers

- You can also dereference a pointer to access the value it points to using the * operator.
- For example, the following code sets the value of the x variable to 20 using the ptr pointer:

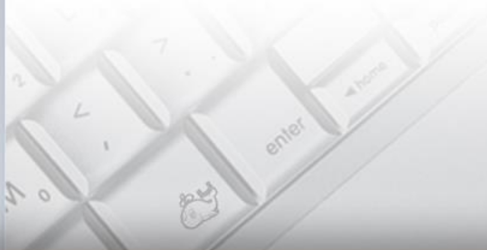
```
*ptr = 20;
```

- This sets the value of the memory location that ptr points to (which is the memory location of the x variable) to 20.

Pointers

- Pointers are commonly used in C++ for various purposes, such as dynamically allocating memory, passing variables to functions by reference, and implementing data structures like linked lists and trees.
- However, they require careful use and can lead to errors like segmentation faults if used improperly.

- The pointer refers to a block of data from the memory area, and to its start.
- A pointer can refer to a variable or a function.



Pointers

```
#include <iostream>
using namespace std;

int main() {
    int var1 = 123;
    int var2 = 99;
    int *ptrvar1 = &var1;
    int *ptrvar2 = &var2;
    cout << "var1    = " << var1 << endl;
    cout << "var2    = " << var2 << endl;
    cout << "ptrvar1 = " << ptrvar1 << endl;
    cout << "ptrvar2 = " << ptrvar2 << endl;
    if (ptrvar1 > ptrvar2)
        cout << "ptrvar1 > ptrvar2" << endl;
    if (*ptrvar1 > *ptrvar2)
        cout << "*ptrvar1 > *ptrvar2" << endl;
    return 0;
}
```

Demo

SHOW ME

YOUR CODE

risovach.ru

High-order pointers

- Pointers can refer to other pointers.
- In this case, the memory cells to which the first pointers will refer will not contain values, but the addresses of the second pointers.
- The number of characters * when declaring a pointer indicates the order of the pointer.
- To access the value pointed to by a pointer, it must be dereferenced an appropriate number of times.
- Let's develop a program that will perform some operations on indexes above the first one.

High-order pointers

- A high-order pointer is a pointer that points to another pointer. In other words, it is a pointer to a memory location that stores the address of another memory location.
- For example, consider the following code:

```
int a = 10;
int *p = &a;
int **q = &p;
```

- In this code, we have a variable a that stores the integer value 10. We also have a pointer p that points to the memory location of a.
- Finally, we have a high-order pointer q that points to the memory location of p.
- The value of q is the address of p, which is itself a pointer. So, q is a high-order pointer because it points to another pointer (p).

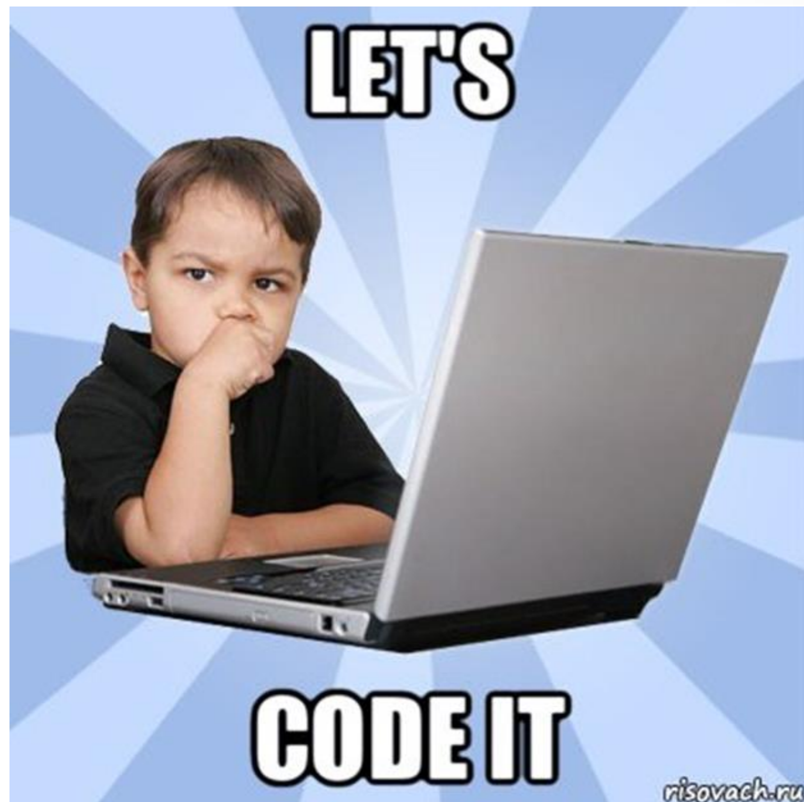
High-order pointers

- High-order pointers can be useful in situations where we need to store and access multiple levels of indirection.
- For example, in some data structures like linked lists, trees, and graphs, we may need to have pointers to other pointers to traverse and manipulate the data.

High-order pointers

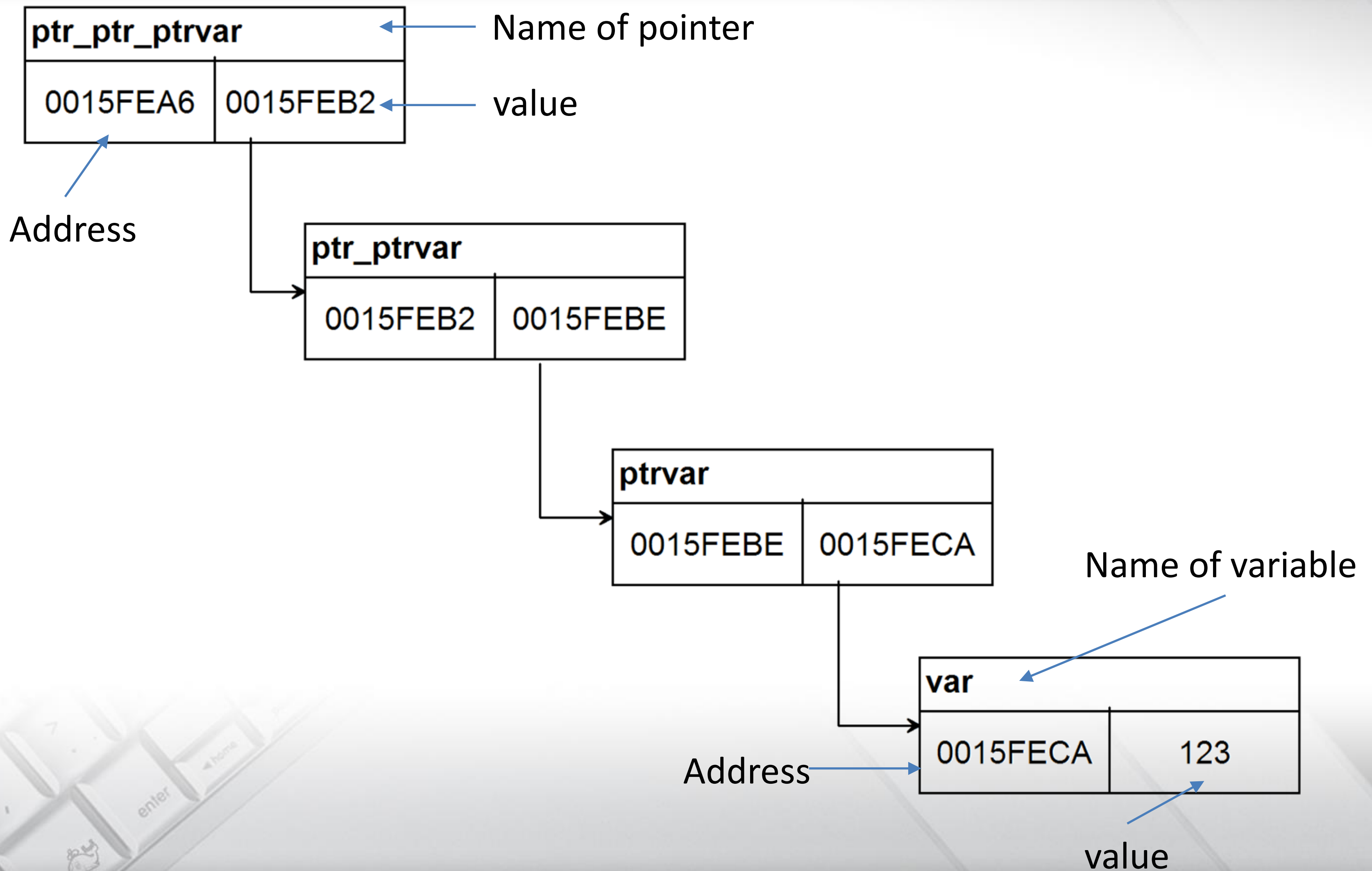
```
#include <iostream>
using namespace std;

int main()
{
    int var = 123;
    int *ptrvar = &var;
    int **ptr_ptrvar = &ptrvar;
    int ***ptr_ptr_ptrvar = &ptr_ptrvar;
    cout << " var\t\t= " << var << endl;
    cout << " *ptrvar\t= " << *ptrvar << endl;
    cout << " **ptr_ptrvar   = " << **ptr_ptrvar << endl;
    cout << " ***ptr_ptrvar  = "
         << ***ptr_ptr_ptrvar << endl;
    cout << "\n***ptr_ptr_ptrvar -> **ptr_ptrvar -> "
         << "*ptrvar -> var -> "<< var << endl;
    cout << "\t  " << &ptr_ptr_ptrvar<< " -> " << "    "
         << &ptr_ptrvar << " ->" << &ptrvar
         << " -> " << &var << " -> " << var << endl;
    return 0;
}
```

Demo

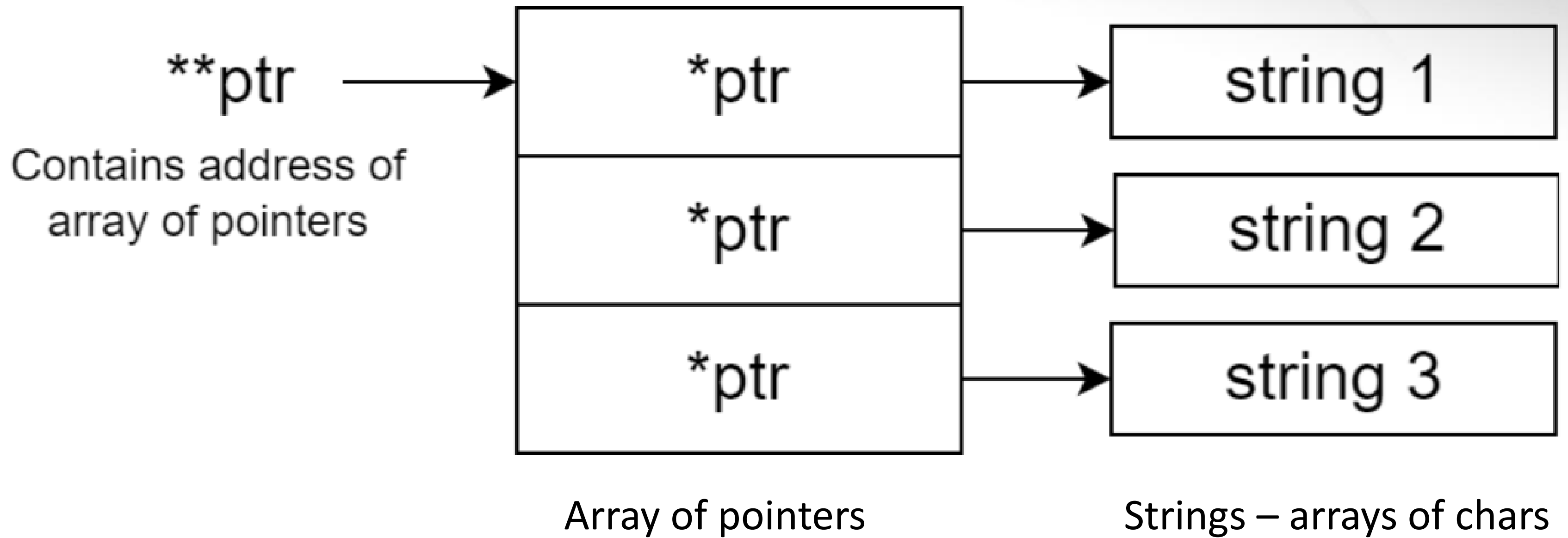


НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Pointers



High-order pointers



Dynamic memory allocation

- Dynamic memory allocation is necessary for the efficient use of computer memory.
- For example, we wrote some program that processes an array.
- When writing this program, it was necessary to declare an array with a fixed size for it (for example, from 0 to 100 elements).
- Then this program will not be universal, because it can process an array of no more than 100 elements.
- And if we need only 20 elements, but program will allocate space for 100 elements, because the array declaration was static, and such memory usage is extremely inefficient.

Dynamic memory allocation

- Dynamic memory allocation in **C** allows you to allocate memory at runtime, rather than at compile time.
- This means you can allocate memory as needed during the execution of your program.
- Dynamic memory allocation is typically done using three **C** library functions: **malloc, calloc** and **free**.

```
void * malloc( size_t size )
void * calloc( size_t number, size_t size )
```

Dynamic memory allocation

- The malloc function is used to dynamically allocate a block of memory of a specified size. It takes a single argument, which is the number of bytes to allocate, and returns a pointer to the beginning of the allocated block of memory.
- For example, to allocate a block of memory to store 10 integers, you can use the following code:

```
int *my_array = malloc(10 * sizeof(int));
```

- This code allocates a block of memory of 10 * sizeof(int) bytes, which is enough to store 10 integers.
- The sizeof operator returns the size in bytes of the type that is passed to it, so sizeof(int) gives the size of an integer in bytes.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Dynamic memory allocation

- Once you are done using the dynamically allocated memory, it is important to free it to avoid memory leaks.
- The free function is used to release the memory allocated by malloc and make it available for other uses.
- For example:

```
free(my_array);
```

This releases the block of memory that was allocated by malloc for the my_array pointer.

Dynamic memory allocation

- calloc() is used to allocate a block of memory and initialize it to zero.
- It takes two arguments: the number of elements to allocate, and the size of each element.
- The syntax of the calloc() function is:

```
ptr = (cast_type*) calloc(n, element_size);
```

- Here, n is the number of elements to allocate, and element_size is the size of each element. The calloc() function returns a pointer to the first byte of the allocated memory block.
- The memory allocated by calloc() is contiguous and initialized to zero. This is different from the malloc() function, which only allocates memory without initializing it.

Dynamic memory allocation

- Dynamic memory allocation in C++ is a way of allocating memory at runtime instead of compile-time. C++ provides two operators for dynamic memory allocation: new and delete.
- The new operator is used to allocate memory at runtime, while the delete operator is used to free that memory when it is no longer needed.
- Here's an example of using new to allocate memory for an integer:

```
int* ptr = new int;
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Dynamic memory allocation

```
int* ptr = new int;
```

- This statement creates a pointer ptr that points to a dynamically allocated integer.
- The new operator allocates memory for the integer and returns a pointer to it, which is then assigned to ptr.
- To deallocate the memory, you would use the delete operator:

```
delete ptr;
```

- This statement frees the memory allocated by new.
- It's important to note that you should always use delete to free memory that was allocated with new, otherwise you risk causing memory leaks.

Dynamic memory allocation

- You can also use new to allocate memory for arrays:

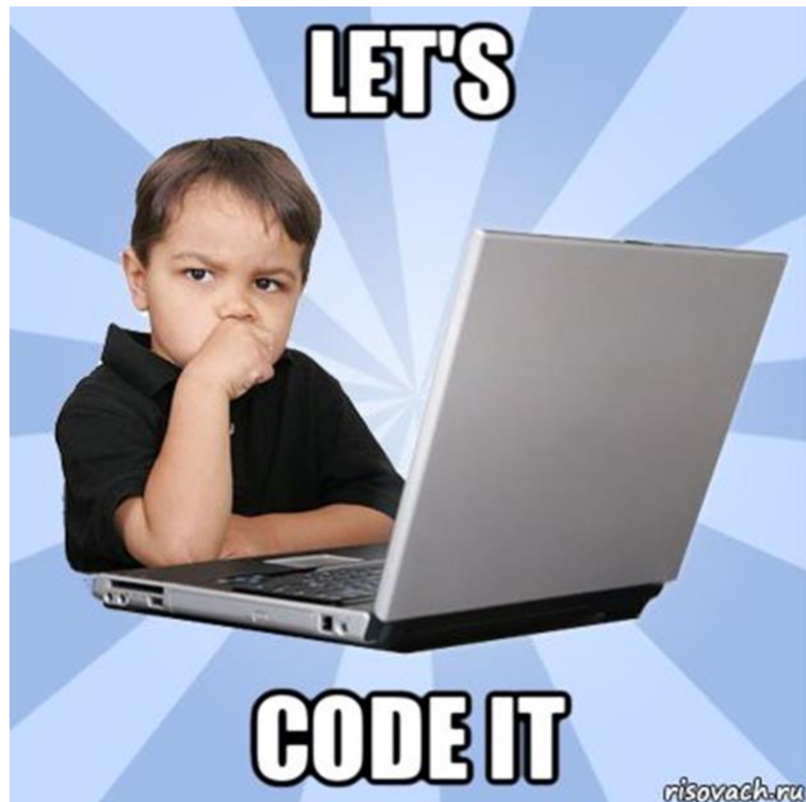
```
int* arr = new int[10];
```

- This statement allocates memory for an array of 10 integers and returns a pointer to the first element.
- To free the memory for an array, you need to use the delete[] operator:

```
delete[] arr;
```

Note: *when you use new to allocate memory, you need to ensure that you free the memory with delete when you're done with it, otherwise you'll have memory leaks in your program.*

Demo



Pointers to functions

- In C++, you can create pointers to functions, just as you can create pointers to variables.
- A pointer to a function is a variable that stores the address of a function.
- Here's an example of how to declare a pointer to a function that takes no arguments and returns an integer:

```
int (*ptr)(); // declaration of pointer to function
```

Here, int is the return type of the function, (*ptr) indicates that we are declaring a pointer to a function, and () indicates that the function takes no arguments.

Pointers to functions

- To assign a function to the pointer, you simply use the function name without the parentheses:

```
int myFunction() {
    // function code here
}

ptr = myFunction; // assign function to pointer
```

- Here, ptr is assigned the address of myFunction.
- To call the function through the pointer, you can use the function call operator:

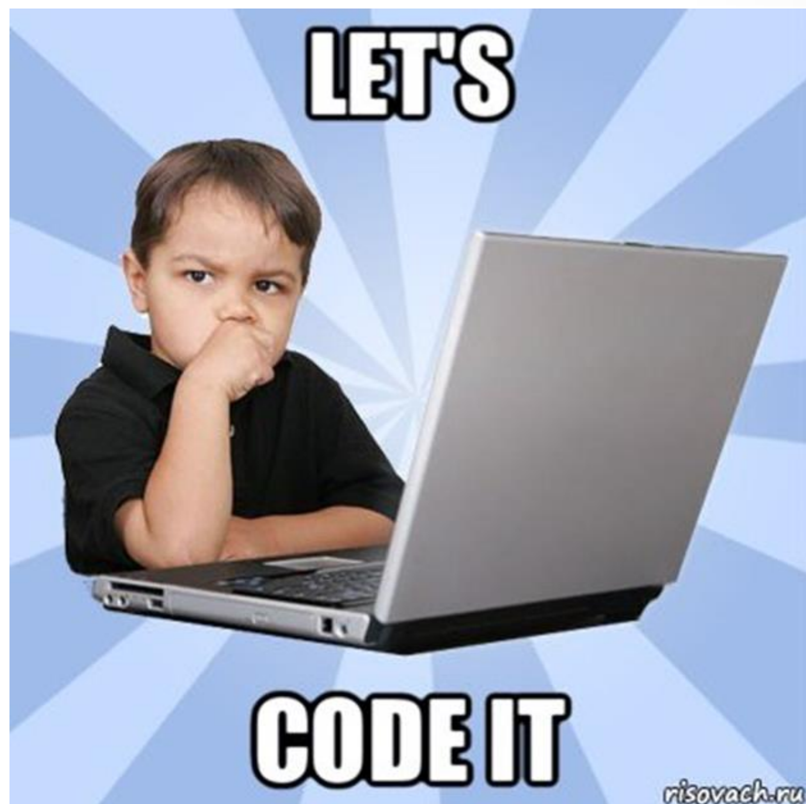
```
int result = ptr(); // call function through pointer
```

Here, ptr() calls the function that ptr points to, and the return value is assigned to result.

Pointers to functions

```
#include <iostream>
using namespace std;
int gcd(int number1, int number2) {
    if (number2 == 0)
        return number1;
    return gcd(number2, number1 % number2);
}

int main() {
    int (*ptrgcd)(int, int);
    ptrgcd = gcd;
    int a, b;
    cout << "Enter first number: ";
    cin >> a;
    cout << "Enter second number: ";
    cin >> b;
    cout << "GCD = " << ptrgcd(a, b) << endl;
    return 0;
}
```

Demo

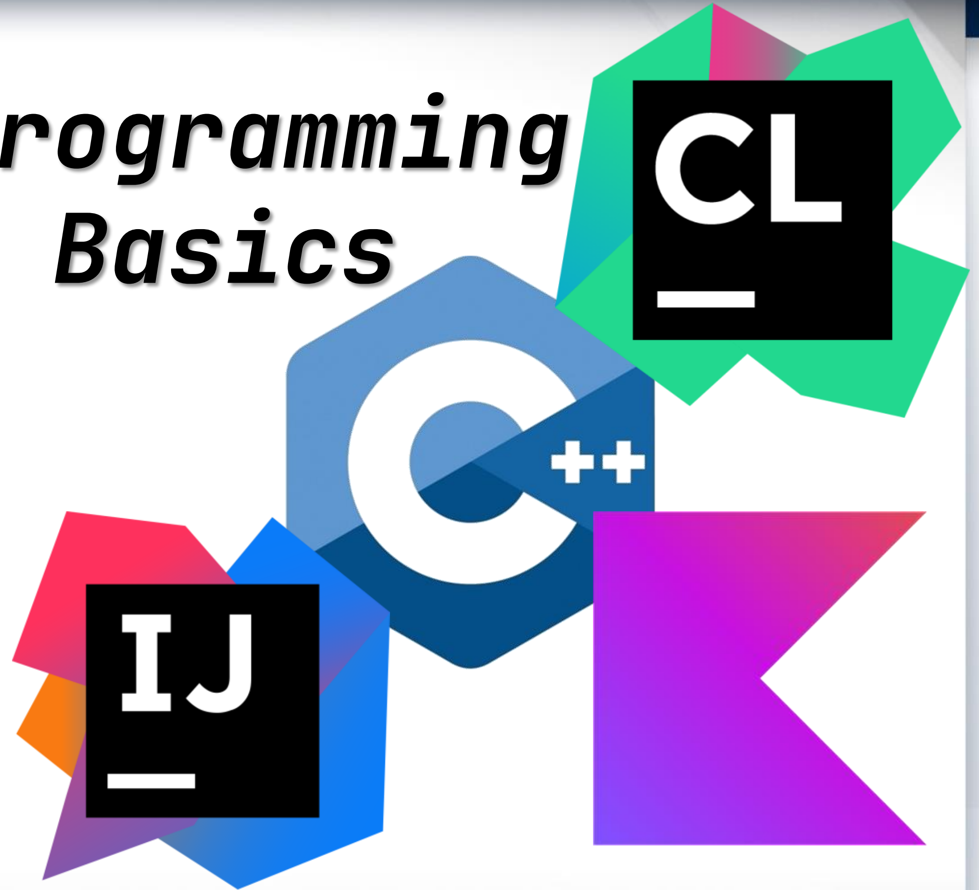


НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Algorithms & Programming
Programming Basics

C/C++/Kotlin programming

(p.7 – Pointers)



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