Lesson №6.1

Subject: Pointers in C++

The purpose of a lesson is to learn the basic principles of using pointers in C++

pointers

Every variable is a memory location and every memory location has its address defined.

A **pointer** is a variable whose value is the **address of another variable**.

Like any variable or constant, a pointer must be declared before using it in a program. The general form of a pointer variable declaration:

type *var-name;

Where,

type is the pointer's base type (the type of a value that is stored in a cell);

var-name is the name of the pointer variable.

pointers

Some valid pointer declarations:

```
int    *ip;       // pointer to an integer
double *dp;       // pointer to a double
float  *fp;       // pointer to a float
char   *ch;       // pointer to character
```

The actual data type of the value of all pointers (whether integer, float, character, or otherwise) is the **same, a long hexadecimal number** that represents a **memory address**. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

pointers: reference operator (&)

To know where the data is stored, C++ has an & operator. The & (reference) operator gives you the address occupied by a variable.

If **a1** is a variable then, **&a1** gives the address of that variable in a memory.

For example,

```
int main()
{       int a=15;
        int b=-6;
        float c=2.76;
        cout<<&a<<&b<<&c;}
```

On the screen you will see something like this:

0x7fff5fbff8ac 0x7fff5fbff8a8 0x7fff5fbff8a4

pointers: dereference operator (*)

To get the value stored in the memory address, we use the dereference operator (* - asterisk).

Example,

```
int *pc, c;
c = 5;
cout << "Address of c: " << &c << endl;
cout << "Value of c: " << c << endl ;
```

Output:

```
Address of c: 0x7fff5fbff80c
Value of c: 5
```

pointers: dereference operator (*)

To get the value stored in the memory address, we use the dereference operator (* - asterisk).

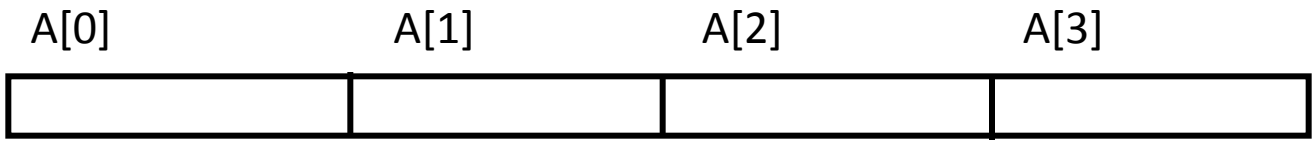
Example,

```
int *pc, c;
c = 5;
pc = &c;
cout << "Address that pointer pc holds: "<< pc << endl;
cout << "Content of the address pointer pc holds: " << *pc << endl;
```

Output:

```
Address that pointer pc holds: 0x7fff5fbff80c
Content of the address pointer pc holds: 5
```

pointers and arrays



Let's see the example:

int A[4];

int *ptr;

Suppose, we need to hold the address of third element of an array, i.e. we need the pointer **ptr** to point to third array's element.

pointers and arrays

So, if we write

ptr=&a[2];

then **ptr** will store the address of third array's element.

But we have more powerful tool to deal with pointers to an array.

ptr=&a; //pointer **ptr** points to whole array, i.e. stores the address of the first array's element

ptr+=1; //**ptr** holds the address of the second element of an array

And so on:

ptr+2 – address of a third element of an array

ptr+3 – address of a fours element

task

Solve the problems:

1. Define the array of 10 integers. Create 2 pointers and set the address of first arrays element into first pointer, the last arrays element into second pointer.
2. Define the array of 100 real numbers. Output the addresses of all arrays elements.