Training - pointers

Years ago, when Qoli was still a nobody among programmers and had not yet traveled to the future (he didn't even know Rick and Morty), something happened that was actually the beginning of his interest in programming and finally giving up. do it

Of course, that story has nothing to do with pointers, we just felt it necessary to introduce you to Qoli's professional background.

The first semester of the university was one of these autumn days when the professor entered the class and presented a slide show called "C pointers" to the class. Qoli was sitting in class with a negative mindset about pointers and thought that pointers were very ugly things. Of course, he thought completely right, but that is a story for another day. In the following, we bring you all the material that Professor Qoli taught them that day. Maybe you can use them usefully one day, maybe not. But what is important is that it forms a very large part of your final exam and therefore bears an uncanny resemblance to Aunt John's famous "Curd Ash".

Pointers actually store the address of a variable or the location of a variable in memory. The general way of writing the definition of a pointer is as follows:

```
// General syntax
datatype *var_name;

// An example pointer "ptr" that holds
// address of an integer variable or holds
// address of a memory whose value(s) can
// be accessed as integer values through "ptr"
int *ptr;
```

In order to be able to use pointers in C, we need to know the following two operators:

- To access the address of a variable, the unary ampersand operator (ampersand - &) is used. For example, x& gives us the address of variable x. Consider the following example:

```
// The output of this program can be different
// in different runs. Note that the program
// prints address of a variable and a variable
// can be assigned different address in different
// runs.

#include <stdio.h>

int  main()

{
    int x;
    // Prints address of x
    printf("%p", &x);
    return 0;`
}
```

- Another operator is the unary star operator (Asterisk - *) which is used for two things:

1. To define a pointer: When a pointer is defined in C/C++, we must put a * after its name.

Questions

Training - pointers

Sereja and Dima

Sort of ridiculous

Library

Welfare of workers

All submissions

Final submissions

Scoreboard

```
// C program to demonstrate declaration of
// pointer variables.
#include <stdio.h>
int main()
{
    int x = 10;

    // 1) Since there is * in declaration, ptr
    // becomes a pointer varaible (a variable
    // that stores address of another variable)
    // 2) Since there is int before *, ptr is
    // pointer to an integer type variable
    int *ptr;

    // & operator before x is used to get address
    // of x. The address of x is assigned to ptr.
    ptr = &x;

    return 0;
}
```

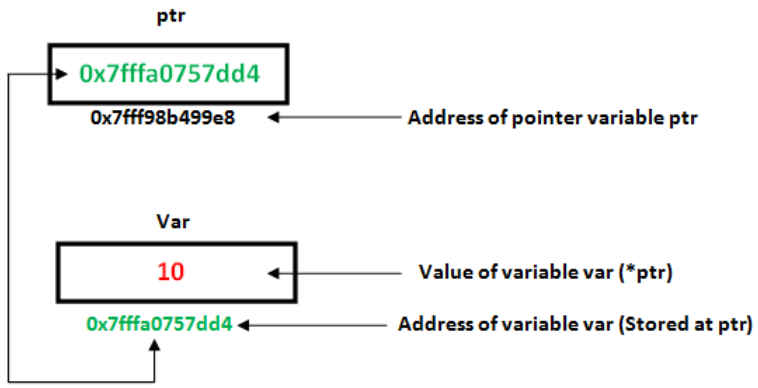
2. We also use the same operator to access the value pointed to by a pointer, which will return the value stored in the address of its operation.

```
// C program to demonstrate use of * for pointers in C
#include <stdio.h>

int main()
{
    // A normal integer variable
    int Var = 10;

    // A pointer variable that holds address of var.
    int *ptr = &Var;

    // This line prints value at address stored in ptr.
    // Value stored is value of variable "var"
    printf("Value of Var = %d\n", *ptr);

    // The output of this line may be different in different
    // runs even on same machine.
    printf("Address of Var = %p\n", ptr);

    // We can also use ptr as lvalue (Left hand
    // side of assignment)
    *ptr = 20; // Value at address is now 20

    // This prints 20
    printf("After doing *ptr = 20, *ptr is %d\n", *ptr);

    return 0;
}
```

Output: (note that addresses may differ from system to system)

```
Value of Var = 10
Address of Var = 0x7fffa057dd4
After doing *ptr = 20, *ptr is 20
```

In the image below, you can see a demonstration of the above program:



Arithmetic operations on pointers

A limited number of operations can be performed on pointers, a pointer can:

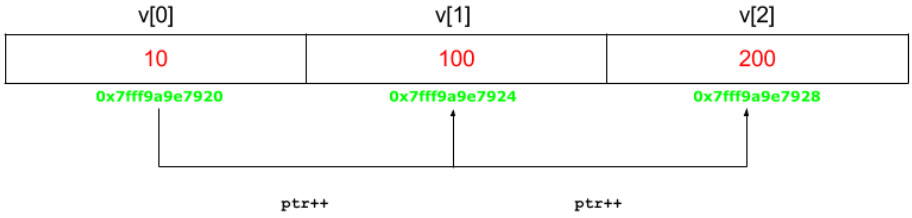
- Be incremented! (++)
- Be decremented! (--)
- A number can be added to a pointer (+ or =+).
- A number can be subtracted from a pointer (- or =-).

(Note that arithmetic operations on pointers are meaningless except when using arrays. We will explain more later.)

```
// C++ program to illustrate Pointer Arithmetic
// in C/C++
#include <bits/stdc++.h>

// Driver program
int main()
{
    // Declare an array
    int v[3] = {10, 100, 200};

    // Declare pointer variable
    int *ptr;

    // Assign the address of v[0] to ptr
    ptr = v;

    for (int i = 0; i < 3; i++)
    {
        printf("Value of *ptr = %d\n", *ptr);
        printf("Value of ptr = %p\n\n", ptr);

        // Increment pointer ptr by 1
        ptr++;
    }
}
```

Output:

```
Value of *ptr = 10
Value of ptr = 0x7ffcae30c710

Value of *ptr = 100
Value of ptr = 0x7ffcae30c714

Value of *ptr = 200
Value of ptr = 0x7ffcae30c718
```



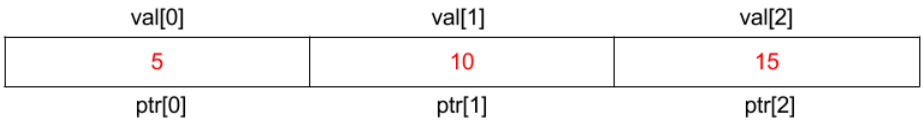
Array names as pointers

One of the interesting points about pointers is their relationship with arrays. The name of an array actually acts like a pointer to the address of the first element of that array. For example, if we have an array named val, then **val** and **[0]val&** can be used interchangeably! (They mean the same thing) or, for example, **val+5** is not different from **[5]val&** .

```
// C++ program to illustrate Array Name as Pointers in C++
#include <bits/stdc++.h>
using namespace std;

void gholi()
{
    // Declare an array
    int val[3] = { 5, 10, 20 };

    // Declare pointer variable
    int *ptr;

    // Assign address of val[0] to ptr.
    // We can use ptr=&val[0];(both are the same)
    ptr = val ;
    cout << "Elements of the array are: ";
    cout << ptr[0] << " " << ptr[1] << " " << ptr[2];

    return;
}

// Driver program
int main()
{
    gholi();
    return 0;
}
```

Output:

```
Elements of the array are: 5 10 20
```



Now if this "ptr" is passed as an argument to a function, the "val" array will be accessed in a similar way.

Pointers and multidimensional arrays

Consider the following array:

```
int nums[2][3]  =  { {16, 18, 20}, {25, 26, 27} };
```

In general, nums[i][j] is equivalent to ∗(∗(nums+i)+j), see the following table:

Pointer Notation	Array Notation	Value
*(*nums)	nums[0] [0]	16
*(*nums+1)	nums[0] [1]	18
*(*nums+2)	nums[0] [2]	20
((nums + 1))	nums[1] [0]	25
((nums + 1)+1)	nums[1] [1]	26
((nums + 1)+2)	nums[1] [2]	27

Two of the uses of pointers are the production of optimal data structures or dynamic memory allocation, which you will get to know with examples of both cases in the continuation of your class sessions.

As an exercise, write a program that stores some numbers in an array and retrieves the numbers using the pointer title array name. Then, again using pointers, sort the array in ascending order.

POST AN ANSWER TO THIS QUESTION

.The training period is over