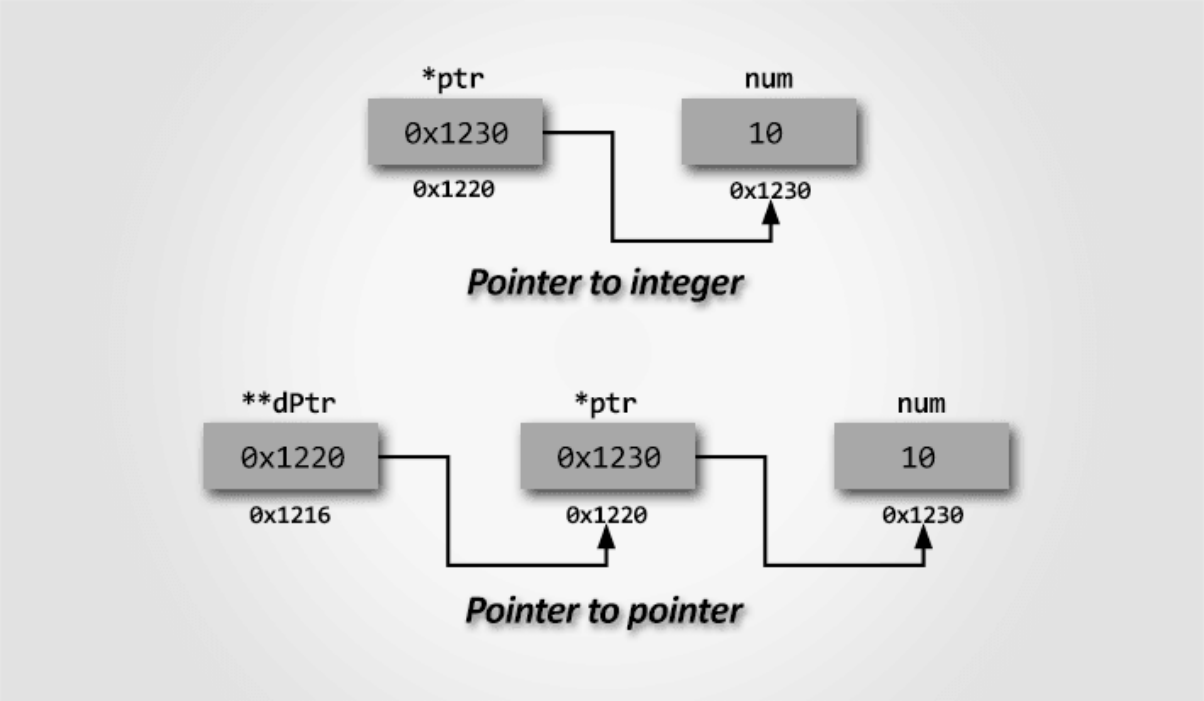
Pointer

- Pointer is a variable that stores memory addresses. Unlike normal variables it does not store user given or processed value, instead it stores valid computer memory address.
- Pointer allows various magical things to be performed in C.
- Pointers are more efficient in handling arrays and structures.
- Pointers are used to return multiple values from a function.
- Pointer allows dynamic memory allocation and deallocation (creation and deletion of variables at runtime) in C. Which undoubtedly is the biggest advantage of pointers.
- Pointer allows to refer and pass a function as a parameter to functions.
- and many more...

For beginners pointers can be a bad dream if not practiced well. However, once mastered you can do anything you want to do in C programming language.

In this exercise I will cover most of the pointer related topics from a beginner level. Always feel free to drop your queries in your Whatsapp group.



Pointers are the heart of C programming. It is the most distinct feature of C, which provides power and flexibility to C. Pointers separates C from other programming languages.

C programmers make extensive use of pointers, because of their numerous benefits. Below are some advantages of pointers.

Pointers are more efficient in handling arrays and structures.

Pointers are used to return multiple values from a function.

We use pointers to get reference of a variable or function.

Pointer allows dynamic memory allocation (creation of variables at runtime) in C. Which undoubtedly is the biggest advantage of pointers.

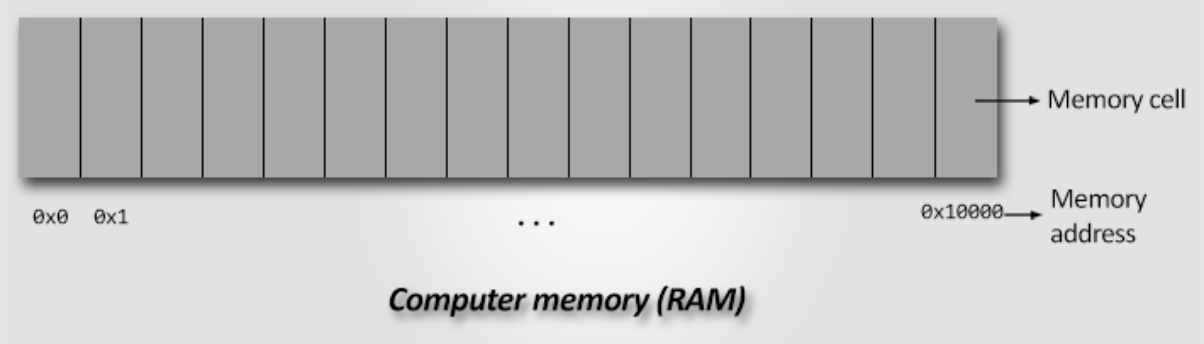
Pointers increases execution speed of program.

Pointers are closely related to low level memory operations. Hence, let us first understand memory in contrast to C programming.

Understanding memory

Computer memory (RAM) is a collection of contiguous block of bytes. Where individual block is called as cell (memory cell). Each cell has a unique numeric address (also known as physical memory address) associated with it. These addresses starts from zero and runs up to maximum memory size (in bytes).

For example, memory location of a 64KB RAM starts from 0 and ends to 65536 (or 0x10000) bytes.



Before I formally introduce pointers let us first see what happens during a variable definition.

Consider the statement int num = 10;

For the above statement, the C compiler allocates memory capable to store an integer. Let say memory is allocated at address 0x1200.

After memory allocation, the C compiler defines a label (variable name) to access the memory location. The label is mapped to the allocated memory.

Finally, the constant 10 is stored at 0x1200. Whenever you refer num inside your program, internally C refers to the memory location of num.

What is a pointer?

A pointer is a variable that stores memory address. If it is a variable, it must have a valid C data type. Yes, every pointer variable has a data type associated with it. Which means an integer pointer can hold only integer variable addresses.

Reference operator &

Because we are dealing with memory addresses, we must know how to get memory address of a variable. We use unary & (reference of) operator to get memory address of a variable. Reference operator is also known as address of operator.

Syntax to use reference of operator

&variable-name;

```
#include <stdio.h>

void main()
{
    int num = 10;

    printf("Value of num   = %d\n", num);

    /* &num gets the address of num. */
    printf("Address of num = %d\n", &num);

      /* &num gets address of num in scientific notation by using %u. */
    printf("Address of num in scientific number = %u\n", &num);


    printf("Address of num in hexadecimal = %x", &num);

    getch();
}
```

Output

```
Value of num   = 10
Address of num = 6487628
Address of num in scientific number = 6487628
Address of num in hexadecimal = 62fe4c
```

Note that Address is always different in different systems. Because it depends upon vacant memory block allocated to your variable from RAM memory of your Computer.

Dereference operator *

Once you have a memory address, you must be willing to get value stored at that memory address, for that we need to dereference the memory address.

Dereferencing is the process of retrieving value at memory location pointed by a pointer. We use unary * dereference operator to get value pointed by a memory address. Dereference operator is also known as indirection operator.

Syntax to use dereference operator

*memory-address-or-pointer-variable;

Example using dereference operator:-

```
#include <stdio.h>

void main()
{
    int num = 10;

    printf("Value of num   = %d\n", num);

    /* &num gets the address of num. */
    printf("Address of num = %d\n", &num);
```

```
    /*
     * &num gets the address of num
     * and (*(&num)) gets value pointed by (&num)
     */
    printf("Value of num   = %d\n", *(&num));

    getch();
}
```

How to declare pointer variable

Once you got basics of memory addresses, reference and dereference operator. Let us declare our first pointer variable.

Pointer variable declaration follows almost similar syntax as of normal variable.

Syntax to declare pointer variable

data-type * pointer-variable-name;

data-type is a valid C data type.

* symbol specifies it is a pointer variable. You must prefix * before variable name to declare it as a pointer.

pointer-variable-name is a valid C identifier i.e. the name of pointer variable.

Example to declare pointer variable

int * ptr;

In above example I declared an integer pointer.

How to initialize pointer variable

There are two ways to initialize a pointer variable. You can use reference operator & to get memory location of a variable or you can also directly assign one pointer variable to other pointer variable.

Examples to initialize pointer variable

int num = 10;

int *ptr = # // Assign address of num to ptr

// You can also assign a pointer variable to another

int *ptr1 = ptr; // Initialize pointer using another pointer

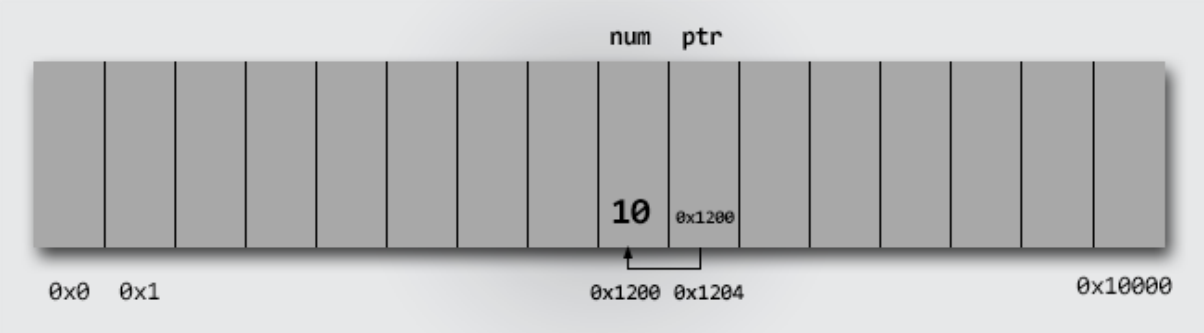
How pointers are stored in memory

You got a basic picture of pointer working. Let us take a closer look on how pointer variables are stored in memory. Consider the following statements

int num = 10;

int *ptr = #

Below is memory representation of above two statements.



Example program to use pointers

```
#include <stdio.h>

void main()
{
    int num = 1;
    int *ptr = &num;   // ptr points to num

    printf("Value of num   = %d \n", num);
    printf("Address of num = %x \n\n", &num);
    printf("Value of ptr         = %x \n", ptr);
    printf("Address of ptr       = %x \n", &ptr);
    printf("Value pointed by ptr = %d \n\n", *ptr);

    /* Change the value of num directly */
    num = 10;
    printf("After changing value of num directly. \n");
    printf("Value of num         = %d \n", num);
    printf("Value pointed by ptr = %d \n\n", *ptr);

    /* Assigns 100 at the address pointed by ptr */
    *ptr = 100;
    printf("After changing value pointed by ptr. \n");
    printf("Value of num         = %d \n", num);
    printf("Value pointed by ptr = %d \n", *ptr);
    getch();
}
```

Output -

```
C:\TC\BIN\test9_a.exe
Value of num   = 1
Address of num = 62fe4c

Value of ptr         = 62fe4c
Address of ptr       = 62fe40
Value pointed by ptr = 1

After changing value of num directly.
Value of num         = 10
Value pointed by ptr = 10

After changing value pointed by ptr.
Value of num         = 100
Value pointed by ptr = 100

--------------------------------
Process exited after 0.09399 seconds with return value 0
Press any key to continue . . .
```

Working of above program

- `int *ptr = &num;` declares an integer pointer that points at `num`.
- The first two `printf()` in line 12 and 13 are straightforward. First prints value of `num` and other prints memory address of `num`.
- `printf("Value of ptr = %x \n", ptr);` prints the value stored at `ptr` i.e. memory address of `num`. Hence, the statement prints memory address of `num`.
- `printf("Address of ptr = %x \n", &ptr);` prints the address of `ptr`.

Don't confuse with address of `ptr` and address pointed by `ptr`. First `ptr` is a variable so it will have a memory address which is retrieved using `&ptr`. And since it is a pointer variable hence it stores memory address which is retrieved using `ptr`.

- `printf("Value pointed by ptr = %d \n\n", *ptr);`, here `*` dereferences value pointed by `ptr` and prints the value at memory location pointed by `ptr`.
- Next, we made some changes to `num` i.e. `num=10`. After changes `printf("Value of num = %d \n", num);` prints 10.
- Since we made changes to our original variable `num`, hence changes are reflected back to pointer that points to the `num`. `*ptr` in line 23, dereferences value pointed by `ptr` i.e. 10.
- `*ptr = 100;` says assign 100 to memory location pointed by `ptr`. Which means, assign 100 to `num` indirectly.
- Since, we again modified the value of `num` using `*ptr = 100`. Hence, `num` and `*ptr` in line 28 and 29 will evaluate to 100.