Pointers in C

What is a Pointer in C?

C pointer is the derived data type that is used to store the address of another variable and can also be used to access and manipulate the variable's data stored at that location. The pointers are considered as derived data types.

With pointers, you can access and modify the data located in the memory, pass the data efficiently between the functions, and create dynamic data structures like linked lists, trees, and graphs.

Pointer Declaration

To declare a pointer, use the **dereferencing operator (*)** followed by the data type.

Syntax

The general form of a pointer variable declaration is –

```
type *var-name;
```

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer.

Example of Valid Pointer Variable Declarations

Take a look at some of the 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 a 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.

Pointer Initialization

After declaring a pointer variable, you need to initialize it with the address of another variable using the **address of (&) operator**. This process is known as **referencing a pointer**.

Syntax

The following is the syntax to initialize a pointer variable –

```
pointer_variable = &variable;
```

Example

Here is an example of pointer initialization –

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

Here, **x** is an integer variable, **ptr** is an integer pointer. The pointer **ptr** is being initialized with **x**.

Referencing and Dereferencing Pointers

A pointer references a location in memory. Obtaining the value stored at that location is known as **dereferencing the pointer**.

In C, it is important to understand the purpose of the following two operators in the context of pointer mechanism –

- **The & Operator** – It is also known as the "Address-of operator". It is used for Referencing which means taking the address of an existing variable (using **&**) to set a pointer variable.
- **The * Operator** – It is also known as the "dereference operator". **Dereferencing** a pointer is carried out using the *** operator** to get the value from the memory address that is pointed by the pointer.

Pointers are used to pass parameters by reference. This is useful if a programmer wants a function's modifications to a parameter to be visible to the function's caller. This is also

useful for returning multiple values from a function.

Access and Manipulate Values using Pointer

The value of the variable which is pointed by a pointer can be accessed and manipulated by using the pointer variable. You need to use the asterisk (*) sign with the pointer variable to access and manipulate the variable's value.

Example

In the below example, we are taking an integer variable with its initial value and changing it with the new value.

</> Open Compiler

```
#include <stdio.h>

int main() {
  int x = 10;

  // Pointer declaration and initialization
  int * ptr = & x;

  // Printing the current value
  printf("Value of x = %d\n", * ptr);

  // Changing the value
  * ptr = 20;

  // Printing the updated value
  printf("Value of x = %d\n", * ptr);

  return 0;
}
```

Output

```
Value of x = 10
Value of x = 20
```

How to Use Pointers?

To use the pointers in C language, you need to declare a pointer variable, then initialize it with the address of another variable, and then you can use it by dereferencing to get and change the value of the variables pointed by the pointer.

You can use pointers with any type of variable such as integer, float, string, etc. You can also use pointers with derived data types such as array, structure, union, etc.

Example

In the below example, we are using pointers for getting values of different types of variables.

</> Open Compiler

```
#include <stdio.h>

int main() {
  int x = 10;
  float y = 1.3f;
  char z = 'p';

  // Pointer declaration and initialization
  int * ptr_x = & x;
  float * ptr_y = & y;
  char * ptr_z = & z;

  // Printing the values
  printf("Value of x = %d\n", * ptr_x);
  printf("Value of y = %f\n", * ptr_y);
  printf("Value of z = %c\n", * ptr_z);

  return 0;
}
```

Output

```
Value of x = 10
Value of y = 1.300000
```

```
Value of z = p
```

Size of a Pointer Variable

The memory (or, size) occupied by a pointer variable does not depend on the type of the variable it is pointing to. The size of a pointer depends on the system architecture.

Example

In the below example, we are printing the size of different types of pointers:

</> Open Compiler

```c
#include <stdio.h>

int main() {
  int x = 10;
  float y = 1.3f;
  char z = 'p';

  // Pointer declaration and initialization
  int * ptr_x = & x;
  float * ptr_y = & y;
  char * ptr_z = & z;

  // Printing the size of pointer variables
  printf("Size of integer pointer : %lu\n", sizeof(ptr_x));
  printf("Size of float pointer : %lu\n", sizeof(ptr_y));
  printf("Size of char pointer : %lu\n", sizeof(ptr_z));

  return 0;
}
```

Output

```
Size of integer pointer : 8
Size of float pointer : 8
Size of char pointer : 8
```

Examples of C Pointers

Practice the following examples to learn the concept of pointers –

Example 1: Using Pointers in C

The following example shows how you can use the **&** and ***** operators to carry out pointer-related opeartions in C –

</> Open Compiler

```c
#include <stdio.h>

int main(){

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;   /* store address of var in pointer variable*/

   printf("Address of var variable: %p\n", &var);

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %p\n", ip);

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

Output

Execute the code and check its output –

```
Address of var variable: 0x7ffea76fc63c
Address stored in ip variable: 0x7ffea76fc63c
Value of *ip variable: 20
```

Example: Print Value and Address of an Integer

We will declare an int variable and display its value and address –

</> Open Compiler

```
#include <stdio.h>

int main(){

   int var = 100;

   printf("Variable: %d \t Address: %p", var, &var);

   return 0;
}
```

Output

Run the code and check its output –

```
Variable: 100   Address: 0x7ffc62a7b844
```

Example: Integer Pointer

In this example, the address of **var** is stored in the **intptr** variable with & operator

</> Open Compiler

```
#include <stdio.h>

int main(){

   int var = 100;
   int *intptr = &var;

   printf("Variable: %d \nAddress of Variable: %p \n\n", var, &var);
   printf("intptr: %p \nAddress of intptr: %p \n\n", intptr, &intptr);

   return 0;
}
```

Output

Run the code and check its output –

```
Variable: 100
Address of Variable: 0x7ffdcc25860c

intptr: 0x7ffdcc25860c
Address of intptr: 0x7ffdcc258610
```

Example 5

Now let's take an example of a float variable and find its address –

</> Open Compiler

```c
#include <stdio.h>

int main(){

   float var1 = 10.55;

   printf("var1: %f \n", var1);
   printf("Address of var1: %d", &var1);
}
```

Output

Run the code and check its output –

```
var1: 10.550000
Address of var1: 1512452612
```

We can see that the address of this variable (any type of variable for that matter) is an integer. So, if we try to store it in a pointer variable of "float" type, see what happens –

```c
float var1 = 10.55;
int *intptr = &var1;
```

The compiler doesn't accept this, and reports the following **error** −

```
initialization of 'int *' from incompatible pointer type
'float *' [-Wincompatible-pointer-types]
```

Note: The type of a variable and the type of its pointer must be same.

In C, variables have specific data types that define their size and how they store values. Declaring a pointer with a matching type (e.g., float *) enforces "type compatibility" between the pointer and the data it points to.

Different data types occupy different amounts of memory space in C. For example, an "int" typically takes 4 bytes, while a "float" might take 4 or 8 bytes depending on the system. Adding or subtracting integers from pointers moves them in memory based on the size of the data they point to.

Example: Float Pointer

In this example, we declare a variable "floatptr" of "float *" type.

</> Open Compiler

```
#include <stdio.h>

int main(){

   float var1 = 10.55;
   float *floatptr = &var1;

   printf("var1: %f \nAddress of var1: %p \n\n",var1, &var1);
   printf("floatptr: %p \nAddress of floatptr: %p \n\n", floatptr, &floatptr);
   printf("var1: %f \nValue at floatptr: %f", var1, *floatptr);

   return 0;
}
```

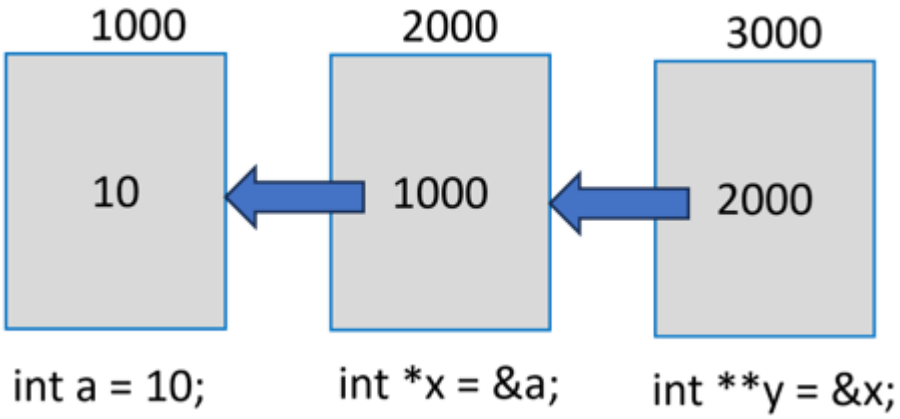
Output

```
var1: 10.550000
Address of var1: 0x7ffc6daeb46c
```

```
floatptr: 0x7ffc6daeb46c
Address of floatptr: 0x7ffc6daeb470
```

Pointer to Pointer

We may have a pointer variable that stores the address of another pointer itself.



In the above figure, "a" is a normal "int" variable, whose pointer is "x". In turn, the variable stores the address of "x".

Note that "y" is declared as "int **" to indicate that it is a pointer to another pointer variable. Obviously, "y" will return the address of "x" and "*y" is the value in "x" (which is the address of "a").

To obtain the value of "a" from "y", we need to use the expression "**y". Usually, "y" will be called as the **pointer to a pointer**.

Example

Take a look at the following example –

</> Open Compiler

```
#include <stdio.h>

int main(){

   int var = 10;
   int *intptr = &var;
   int **ptrptr = &intptr;

   printf("var: %d \nAddress of var: %d \n\n",var, &var);
```

```
   printf("inttptr: %d \nAddress of inttptr: %d \n\n", intptr, &intptr);
   printf("var: %d \nValue at intptr: %d \n\n", var, *intptr);
   printf("ptrptr: %d \nAddress of ptrtptr: %d \n\n", ptrptr, &ptrptr);
   printf("intptr: %d \nValue at ptrptr: %d \n\n", intptr, *ptrptr);
   printf("var: %d \n*intptr: %d \n**ptrptr: %d", var, *intptr, **ptrptr);

   return 0;
}
```

Output

Run the code and check its output –

```
var: 10
Address of var: 951734452

inttptr: 951734452
Address of inttptr: 951734456

var: 10
Value at intptr: 10

ptrptr: 951734456
Address of ptrtptr: 951734464
intptr: 951734452
Value at ptrptr: 951734452

var: 10
*intptr: 10
**ptrptr: 10
```

You can have a **pointer to an array** as well as a derived type defined with **struct**. Pointers have important applications. They are used while calling a function by passing the reference. Pointers also help in overcoming the limitation of a function's ability to return only a single value. With pointers, you can get the effect of returning multiple values or arrays.

NULL Pointers

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.

The NULL pointer is a constant with a value of "0" defined in several standard libraries.

Example

Consider the following program –

```
</>                                                    Open Compiler

#include <stdio.h>

int main(){

   int *ptr = NULL;

   printf("The value of ptr is : %x\n", ptr);

   return 0;
}
```

Output

When the above code is compiled and executed, it produces the following result –

```
The value of ptr is 0
```

In most operating systems, programs are not permitted to access memory at address "0" because that memory is reserved by the operating system.

The memory address "0" has a special significance; it signals that the pointer is not intended to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.

To check for a null pointer, you can use an **if** statement as follows –

```
if(ptr)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

Address of the Variables

As you know, every variable is a memory location and every memory location has its address defined which can be accessed using the ampersand (&) operator, which denotes

an address in memory.

Example

Consider the following example, which prints the address of the variables defined –

```
#include <stdio.h>

int main(){

   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1);
   printf("Address of var2 variable: %x\n", &var2);

   return 0;
}
```

Output

When the above code is compiled and executed, it will print the address of the variables –

```
Address of var1 variable: 61e11508
Address of var2 variable: 61e1150e
```

Pointers in Detail

Pointers have many but easy concepts and they are very important to C programming. The following important pointer concepts should be clear to any C programmer –

Sr.No	Concept & Description
1	**Pointer arithmetic** There are four arithmetic operators that can be used in pointers: ++, --, +, -
2	**Array of pointers** You can define arrays to hold a number of pointers.

3	**Pointer to pointer** C allows you to have pointer on a pointer and so on.
4	**Passing pointers to functions in C** Passing an argument by reference or by address enable the passed argument to be changed in the calling function by the called function.
5	**Return pointer from functions in C** C allows a function to return a pointer to the local variable, static variable, and dynamically allocated memory as well.