**Pointers in C**

Pointers in C are very easy to learn a few tasks in C language are done by using pointers. And some tasks like dynamic memory allocation done only by using pointers. So it is essential to learn pointers. POINTER is a variable that stores the address of the other variable. A pointer is also used to refer to a pointer function. And pointer can be incremented or decremented that is if the pointer is incremented then it points to the next and if the pointer is decremented it points to the previous memory location. The goal of the pointer is to save memory space and perform faster execution. And the size of the pointer in C is 8 bytes but on a 32-bit machine, they take up to 4 bytes.

**How does Pointer Works?**

* If we declare a variable Y of type Integer (Int) then Y will actually store the value.

         Int Y= 1;

         Y is equal to one now.

* But, every variable has both value and address, and that address can be retrieved by putting an ampersand before the variable name like this.

**"&Y"**

* If you are willing to print an address of a variable that address may be a random number and that random number will be different whenever you run your program. which means the same program may give different outputs.

**Example:**

#include<stdio.h>

int main()

{

int Y=1;

printf(“%d”, &Y);

return 0;

}

**Output:** 123756948 and if you run the same code for the second time the output may be different.

In simple terms, variable store values and pointers store the address of the variable.

**Pointer Declaration:**

Like variables, pointers should be declared before using it in the program. We can name pointers anything as long as they obey C’s naming rules.

**Syntax:** Data\_type \* pointer\_variable\_name;

**Example:** int\*a;

**Initializing a pointer:**

After declaring a pointer, we have to initialize the pointer with the standard variable address.

If pointers are not initialized then there may be a problem in the output.

**Syntax:** pointer= &variable;

**Example:** p= &a;

**Advantages of Pointers in C**

* Pointers are useful for accessing memory locations.
* Pointers provide an efficient way for accessing the elements of an array structure.
* Pointers are used for dynamic memory allocation as well as deallocation.
* Pointers are used to form complex data structures such as linked list, graph, tree, etc.

**Disadvantages of Pointers in C**

* Pointers are a little complex to understand.
* Pointers can lead to various errors such as segmentation faults or can access a memory location which is not required at all.
* If an incorrect value is provided to a pointer, it may cause memory corruption.
* Pointers are also responsible for memory leakage.
* Pointers are comparatively slower than that of the variables.
* Programmers find it very difficult to work with the pointers; therefore it is programmer's responsibility to manipulate a pointer carefully.

**Types of Pointers:**

There are eight different types of pointers they are:

* Null pointer
* Void pointer
* Wild pointer
* Dangling pointer
* Complex pointer
* Near pointer
* Far pointer
* Huge pointer

**Null Pointer:**

A pointer that points to nothing is called a Null pointer. Some advantages of Null pointer are:

* We can initialize a pointer variable when that pointer variable is not assigned any actual memory address.
* We can pass a null pointer to a function argument when we are not willing to pass any actual memory address.

**Example 1:**

int \* aInt = NULL;

**Example 2:**

int fun(int \*ptr)

{

      return 15;

}

fun(NULL);

**Example 3:**

if (aINT != NULL)

{ //some code}

else

{//some code}

[C Characteristics](https://tekslate.com/characteristics-of-c-language)

**Void Pointer:**

The void pointer within C is a pointer that is not allied with any data types. This points to some data location within the storage means points to that address of variables. It is also known as a general-purpose pointer. In C, malloc() and calloc() functions return void \* or generic pointers.

**Example:**

int x= 10;

char y= ‘a’;

void \*p= &x //void pointer contains address of int x

p = &y //void pointer holds of char y

**Wild Pointer:**

Pointers that are not initialized are called wild pointers. This pointer may be initialized to a non-NULL garbage value which may not be a valid address.

**Example:**

int main()

{

int \*p; // wild pointer

\*p= 10;

}

Remember if a pointer p points to any known variable, then it is not a wild pointer. In the below program p is a wild pointer until it points to x.

int main()

{

int = \*p; // wild pointer

int x= 20;

p= &x // p is not a wild pointer now

}

### Dangling Pointer:

A pointer that points to a memory location that has been deleted is called a dangling pointer.

**Example 1:**

Deallocation of memory:

#include<stdio.h>

#include<stdlib.h>

int main()

{

   int \*ptr = (int \*)malloc(sizeof(int));

   free(ptr);

   ptr = NULL;

}

**Example 2:**

**Function call:**

#include<stdio.h>

int \*fun()

{

   int y= 15;

   return &y

}

int main()

{

    int \*p = func()

    fflush(stdin)

printf(“%d”, \*p);

return 0;

}

### Complex Pointer:

Before knowing how to read complex pointers then you should first know associativity and precedence.

**Associativity:** Order operators of equal precedence within an expression are employed.

**Precedence:**Operator precedence describes the order in which C reads expressions.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Precedence** | **Associative** |
| (),[] | 1 | Left to Right |
| \*,Identifier | 2 | Right to Left |
| Data Type | 3 | – |

* (): this operator is used to declare and define the function.
* []: this is an array subscript operator.
* \*: this is a pointer operator.
* Identifier: this is the name of a pointer.
* Data type: this is the type of variable.

**Example:**

int (\*p)(int (\*)[3], int (\*)void))

### Near Pointer:

* Near pointer means a pointer that is utilized to bit address of up to 16 bits within a given section of that computer memory which is 16 bit enabled.
* It can only access data of the small size of about 64 kb within a given period, which is the main disadvantage of this type of pointer.

**Example:**

#include<stdio.h>

int main()

{

   int a= 30;

   int near\* ptr;

   ptr= &a;

   printf(“%d”,sizeof ptr);

   return 0;

}

Output: 2

### Far Pointer:

* A far pointer is typically 32 bit which can access memory outside that current segment.
* To utilize the far pointer, the compiler allows a segment register to save segment address, then another register to save offset inside the current segment.

**Example:**

#include<stdio.h>

int main()

{

  int a= 10;

  int far \*ptr;

  ptr=&a;

  print(“%d”, sizeof ptr);

  return 0;

}

### Huge Pointer:

* Same as far pointer huge pointer is also typically 32 bit which can access outside the segment.
* A far pointer that is fixed and hence that part of that sector within which they are located cannot be changed in any way; huge pointers can be.

**Example:**

#include<stdio.h>

int main()

{

   char huge \*far \*a;

   printf(“%d%d%d”, sizeof(a), size(\*a), sizeof(\*\*a));

   return 0;

}

**Output:** 4 4 1.

### Struct pointer:

* Pointers can be utilized to refer to a struct by its address. This helps pass structs to a function.
* The pointer can be dereferenced by the \* operator.
* The -> operator dereferences the pointer to the left operand and later accesses the value of a member of the right operand.

#### **Advantages of pointers in C:**

* Pointers permit the management of structures that are allocated memory dynamically.
* Pointers make it possible to pass the address of the structure rather than the entire structure to the functions.