

POINTER AND STRUCTURE

A **pointer** is a variable that stores the address of another variable. Unlike other variables that hold values of a certain type, pointer holds the address of a variable. For example, an integer variable holds (or you can say stores) an integer value, however an integer pointer holds the address of a integer variable. In this guide, we will discuss pointers in C programming with the help of examples.

Before we discuss about **pointers in C**, lets take a simple example to understand what do we mean by the address of a variable.

A simple example to understand to access the address of a variable without pointers

In this program, we have a variable num of int type. The value of num is 10 and this value must be stored somewhere in the memory, right? A memory space is allocated for each variable that holds the value of that variable, this memory space has an address. For example we live in a house and our house has an address, which helps other people to find our house. The same way the value of the variable is stored in a memory address, which helps the C program to find that value when it is needed.

So let's say the address assigned to variable num is 0x7fff5694dc58, which means whatever value we would be assigning to num should be stored at the location: 0x7fff5694dc58. See the diagram below.

```
#include <stdio.h>
int main()
{
    int num = 10;
    printf("Value of variable num is: %d", num);
    /* To print the address of a variable we use %p
    * format specifier and ampersand (&) sign just
    * before the variable name like &num.
    */
    printf("\nAddress of variable num is: %p", &num);
    return 0;
}
```

Output:

Value of variable num is: 10

Address of variable num is: 0x7fff5694dc58

A Simple Example of Pointers in C

This program shows how a pointer is declared and used. There are several other things that we can do with pointers, we have discussed them later in this guide. For now, we just need to know how to link a pointer to the address of a variable.

Important point to note is: The data type of pointer and the variable must match, an int pointer can hold the address of int variable, similarly a pointer declared with float data type can hold the address of a float variable. In the example below, the pointer and the variable both are of int type.

```
#include <stdio.h>
int main()
{
    //Variable declaration
```

```

int num = 10;

//Pointer declaration
int *p;

//Assigning address of num to the pointer p
p = #

printf("Address of variable num is: %p", p);
return 0;
}

```

Output:

Address of variable num is: 0x7fff5694dc58

C Example to swap two numbers using pointers

```

/*C program by Chaitanya for beginnersbook.com
 * Program to swap two numbers using pointers*/
#include <stdio.h>

// function to swap the two numbers
void swap(int *x,int *y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
}

int main()
{
    int num1,num2;

    printf("Enter value of num1: ");
    scanf("%d",&num1);
    printf("Enter value of num2: ");
    scanf("%d",&num2);

    //displaying numbers before swapping
    printf("Before Swapping: num1 is: %d, num2 is: %d\n",num1,num2);

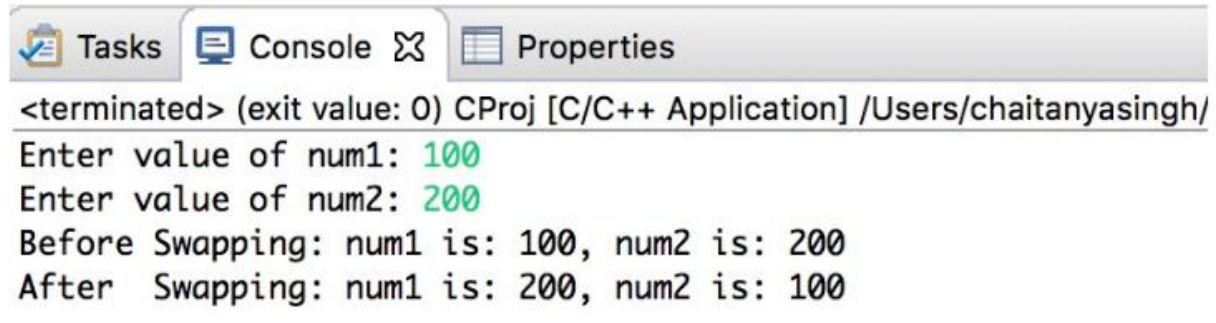
    //calling the user defined function swap()
    swap(&num1,&num2);

    //displaying numbers after swapping
    printf("After Swapping: num1 is: %d, num2 is: %d\n",num1,num2);
}

```

```
return 0;  
}
```

Output:



The screenshot shows a Windows-style application window titled "CProj [C/C++ Application] /Users/chaitanyasingh/". It has three tabs: "Tasks", "Console", and "Properties". The "Console" tab is active, displaying the following text:

```
<terminated> (exit value: 0) CProj [C/C++ Application] /Users/chaitanyasingh/  
Enter value of num1: 100  
Enter value of num2: 200  
Before Swapping: num1 is: 100, num2 is: 200  
After Swapping: num1 is: 200, num2 is: 100
```

Data Structures

Data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.

Depending on your requirement and project, it is important to choose the right data structure for your project. For example, if you want to store data sequentially in the memory, then you can go for the Array data structure.



Types of Data Structure

Basically, data structures are divided into two categories: Linear data structure Non-linear data structure Let's learn about each type in detail.

Linear data structures

In linear data structures, the elements are arranged in sequence one after the other. Since elements are arranged in particular order, they are easy to implement.

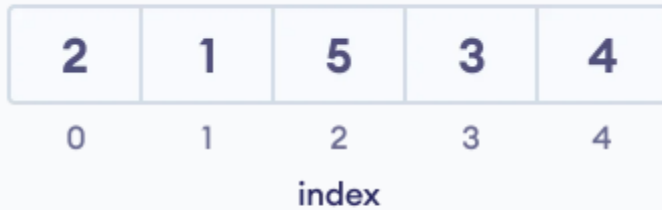
However, when the complexity of the program increases, the linear data structures might not be the best choice because of operational complexities.

Popular linear data structures are:

1. Array Data Structure

In an array, elements in memory are arranged in continuous memory. All the elements of an array are of the same type. And, the type of elements that can be stored in the form of arrays is determined by the programming language.

To learn more, visit [Java Array](#).

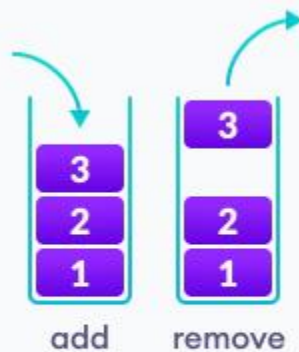


An array with each element represented by an index

2. Stack Data Structure

In stack data structure, elements are stored in the LIFO principle. That is, the last element stored in a stack will be removed first.

It works just like a pile of plates where the last plate kept on the pile will be removed first. To learn more, visit [Stack Data Structure](#).



In a stack, operations can be perform only from one end (top here).

3. Queue Data Structure

Unlike stack, the queue data structure works in the FIFO principle where first element stored in the queue will be removed first.

It works just like a queue of people in the ticket counter where first person on the queue will get the ticket first. To learn more, visit [Queue Data Structure](#).



In a queue, addition and removal are performed from separate ends.

4. Linked List Data Structure

In linked list data structure, data elements are connected through a series of nodes. And, each node contains the data items and address to the next node.

Write a C program to store the information of Students using [Structure](#). The information of each student to be stored is:

Each Student Record should have:

Name

Roll Number

Age

Total Marks

- A structure is a user-defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.
- 'struct' keyword is used to create the student structure as:
- struct Student
- {
- char* name;
- int roll_number;
- int age;
- double total_marks;
- };
- Get the number of Students whose details are to be stored. Here we are taking 5 students for simplicity.
- Create a variable of Student structure to access the records. Here it is taken as 'student'
- Get the data of n students and store it in student's fields with the help of dot (.) operator

Syntax:

- student[i].member = value;
- After all the data is stored, print the records of each students using the dot (.) operator and loop.

Syntax:

student[i].member;

Below is the implementation of the above approach:

```
// C Program to Store Information
```

```
// of Students Using Structure
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
// Create the student structure
struct Student {
    char* name;
    int roll_number;
    int age;
    double total_marks;
};
```

```
// Driver code
```

```
int main()
```

```
{
    int i = 0, n = 5;
```

```
    // Create the student's structure variable
    // with n Student's records
    struct Student student[n];
```

```
    // Get the students data
    student[0].roll_number = 1;
    student[0].name = "Geeks1";
    student[0].age = 12;
    student[0].total_marks = 78.50;
```

```
    student[1].roll_number = 5;
    student[1].name = "Geeks5";
    student[1].age = 10;
    student[1].total_marks = 56.84;
```

```
    student[2].roll_number = 2;
    student[2].name = "Geeks2";
    student[2].age = 11;
    student[2].total_marks = 87.94;
```

```
    student[3].roll_number = 4;
    student[3].name = "Geeks4";
    student[3].age = 12;
    student[3].total_marks = 89.78;
```

```
    student[4].roll_number = 3;
```

```

        student[4].name = "Geeks3";
        student[4].age = 13;
        student[4].total_marks = 78.55;

        // Print the Students information
        printf("Student Records:\n\n");
        for (i = 0; i < n; i++) {
            printf("\tName = %s\n", student[i].name);
            printf("\tRoll Number = %d\n", student[i].roll_number);
            printf("\tAge = %d\n", student[i].age);
            printf("\tTotal          Marks          =          %0.2f\n\n",
student[i].total_marks);
        }

        return 0;
    }

```

Output:

Student Records:

Name = Geeks1
 Roll Number = 1
 Age = 12
 Total Marks = 78.50

Name = Geeks5
 Roll Number = 5
 Age = 10
 Total Marks = 56.84

Name = Geeks2
 Roll Number = 2
 Age = 11
 Total Marks = 87.94

Name = Geeks4
 Roll Number = 4
 Age = 12
 Total Marks = 89.78

Name = Geeks3
 Roll Number = 3
 Age = 13
 Total Marks = 78.55