**Example: Pass Addresses to Functions**

#include <stdio.h>

void swap(int \*n1, int \*n2);

int main()

{

int num1 = 5, num2 = 10;

// address of num1 and num2 is passed

swap( &num1, &num2);

printf("num1 = %d\n", num1);

printf("num2 = %d", num2);

return 0;

}

void swap(int\* n1, int\* n2)

{

int temp;

temp = \*n1;//5

\*n1 = \*n2;//10

\*n2 = temp;//5

}

When you run the program, the output will be:

num1 = 10

num2 = 5

## Example 2: Passing Pointers to Functions

#include <stdio.h>

void addOne(int\* ptr) {

(\*ptr)++; // adding 1 to \*ptr

}

int main()

{

int\* p, i = 10;

p = &i;

addOne(p);

printf("%d", \*p); // 11

return 0;

}

Here, the value stored at p, \*p, is 10 initially.

We then passed the pointer p to the addOne() function. The ptr pointer gets this address in the addOne() function.

Inside the function, we increased the value stored at ptr by 1 using (\*ptr)++;. Since ptr and p pointers both have the same address, \*p inside main() is also 11.

The address of num1 and num2 are passed to the swap() function using swap(&num1, &num2);.

Pointers n1 and n2 accept these arguments in the function definition.

void swap(int\* n1, int\* n2) {

... ..

}

When \*n1 and \*n2 are changed inside the swap() function, num1 and num2 inside the main() function are also changed.

Inside the swap() function, \*n1 and \*n2 swapped. Hence, num1 and num2 are also swapped.

Notice that swap() is not returning anything; its return type is void.

**Example 2: Passing Pointers to Functions**

#include <stdio.h>

void addOne(int\* ptr) {

(\*ptr)++; // adding 1 to \*ptr

}

int main()

{

int\* p, i = 10;

p = &i;

addOne(p);

printf("%d", \*p); // 11

return 0;

}

Here, the value stored at p, \*p, is 10 initially.

We then passed the pointer p to the addOne() function. The ptr pointer gets this address in the addOne() function.

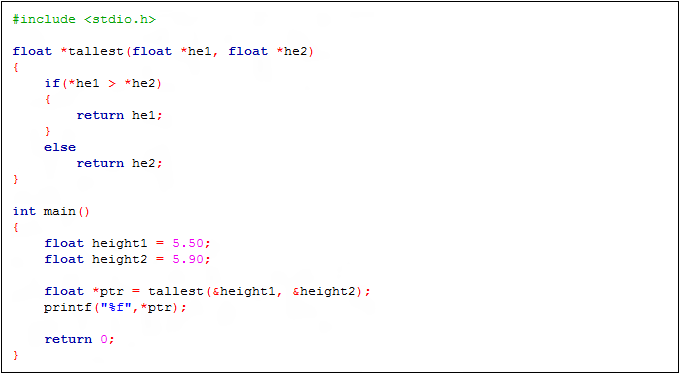
Inside the function, we increased the value stored at ptr by 1 using (\*ptr)++;. Since ptr and p pointers both have the same address, \*p inside main() is also 11.

**Functions Returning Address**

We have seen so far that a function can return a single value by its name or return multiple values through pointer parameters. Since pointers are a data type in C, we can also force a function to return a pointer to the calling function.

For example, in the following program, the function **tallest( )** receives the addresses of the variables **‘height1’** and **‘height2’**, decides which one is larger using the pointers **‘he1’** and **‘he2’** and then returns the address of its location. The returned value is then assigned to the pointer variable **‘ptr’** in the calling function. In this case, the address of **‘height1’** is returned and assigned to **‘ptr’** and therefore the output will be the value of **‘height1’**.

**Program**

[](https://hajsoftutorial.com/c-programing-tutorial/wp-content/uploads/2016/04/2-3.png)

Functions returning a Pointer

[Pointers](https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/) in [C programming language](http://www.geeksforgeeks.org/c/) is a variable which is used to store the memory address of another variable. We can pass pointers to the function as well as return pointer from a function. But it is not recommended to return the address of a local variable outside the function as it goes out of scope after function returns.

**Program 1:**

The below program will give segmentation fault since **‘A’** was local to the function:

// C program to illustrate the concept of

// returning pointer from a function

#include <stdio.h>

// Function returning pointer

int\* fun()

{

int A = 10;

return (&A);

}

// Driver Code

int main()

{

// Declare a pointer

int\* p;

// Function call

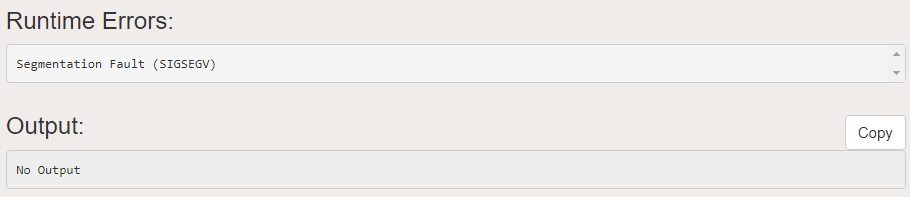
p = fun();

printf("%p\n", p);

printf("%d\n", \*p);

return 0;

}

**Output:**  
Below is the output of the above program:  
[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20200818154000/Screenshot-111.png)  
**Explanation:**

The main reason behind this scenario is that compiler always make a [stack](https://www.geeksforgeeks.org/stack-data-structure-introduction-program/) for a function call. As soon as the function exits the function stack also gets removed which causes the local variables of functions goes out of scope.

[Static Variables](https://www.geeksforgeeks.org/static-variables-in-c/) have a property of preserving their value even after they are out of their scope. So to execute the concept of returning a pointer from function in C you must define the local variable as a static variable.

**Program 2:**

// C program to illustrate the concept of

// returning pointer from a function

#include <stdio.h>

// Function that returns pointer

int\* fun()

{

// Declare a static integer

static int A = 10;

return (&A);

}

// Driver Code

int main()

{

// Declare a pointer

int\* p;

// Function call

p = (int\*) fun();

// Print Address

printf("%p\n", p);

// Print value at the above address

printf("%d\n", \*p);

return 0;

}**Output:**

0x601038

10