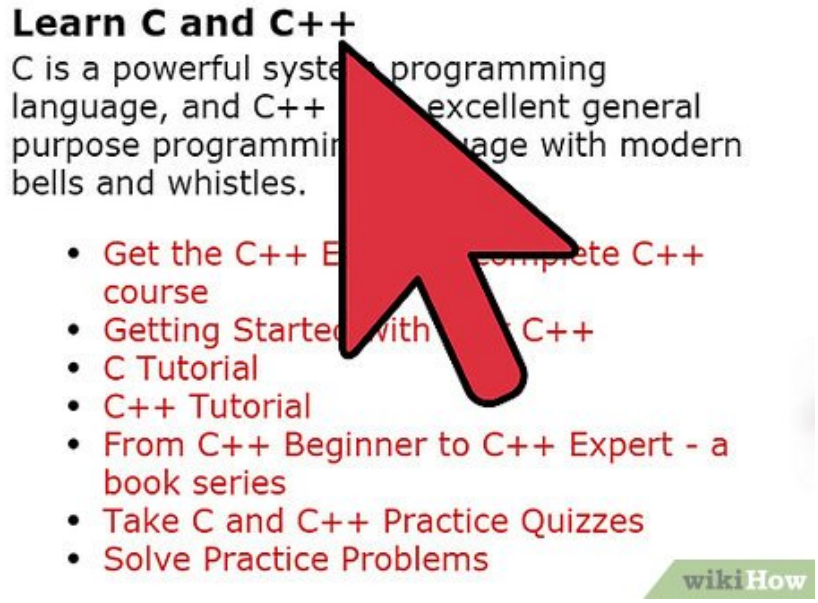
How to Create Pointers in C

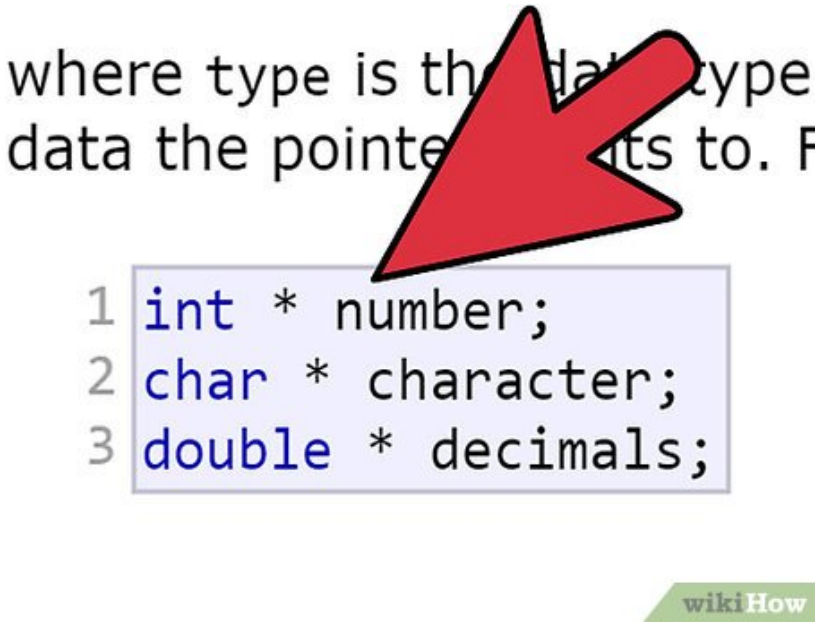
Pointers are the nightmare of every new C programmer. However, they are also the feature that made C the widespread, powerful programming language it is until today. Like many other programming features and constructs, there is a...

Steps

1. 

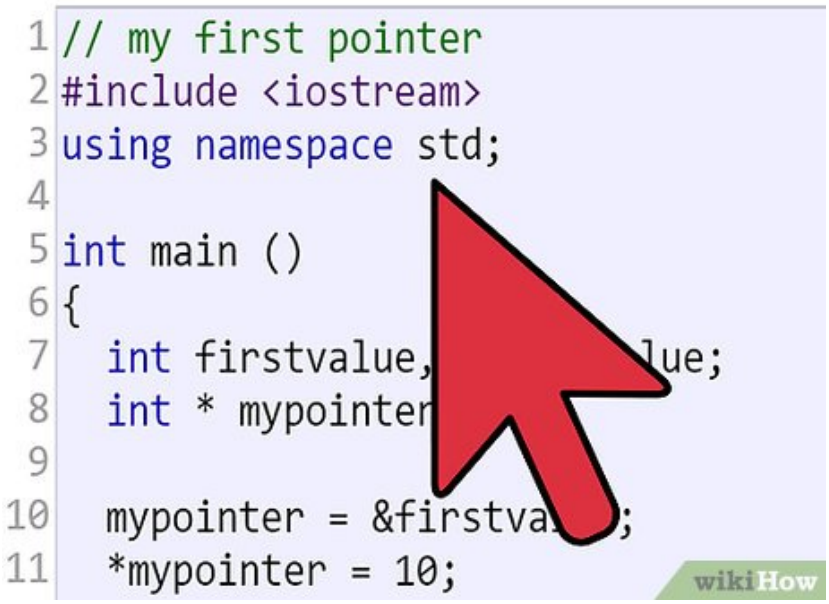
Decide the type of the pointer (that is, the type of data the pointer will be pointing to). The following tips might help:

1. If you are declaring a dynamic array, use the array items' data type.
2. If you are declaring the pointer to access the data of a variable, use the same data type as the variable.
3. If you are declaring the pointer to traverse a list structure, use the list node data type (usually a user created struct).
4. If you are declaring the pointer to traverse a tree, use the data type of the tree node, or a pointer to the tree node type as the type (pointer to a pointer of tree node type!).

2. 

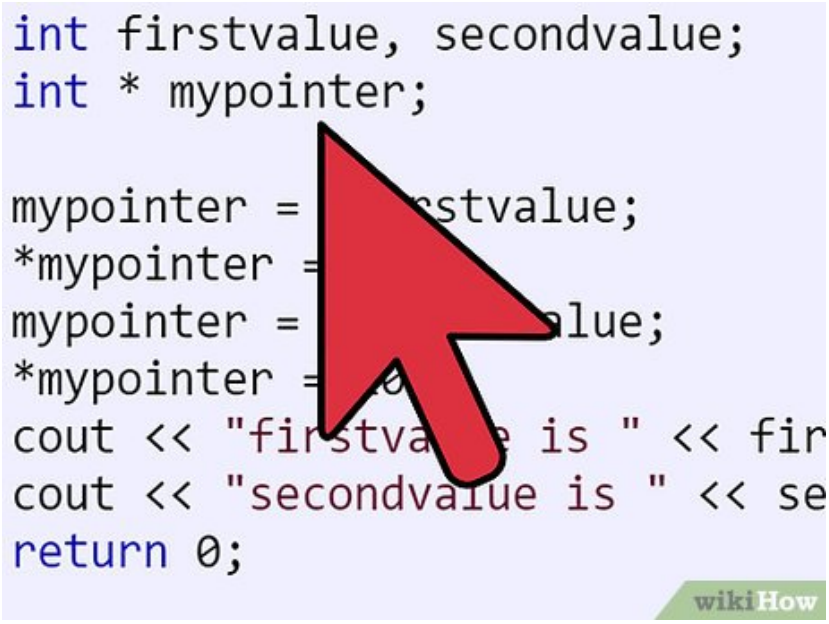
Declare the pointer using a syntax like this: data-type * pointer-identifier; where

1. *data-type* is the type you decided in step 1
2. *pointer-identifier* is the identifier or name of the pointer variable

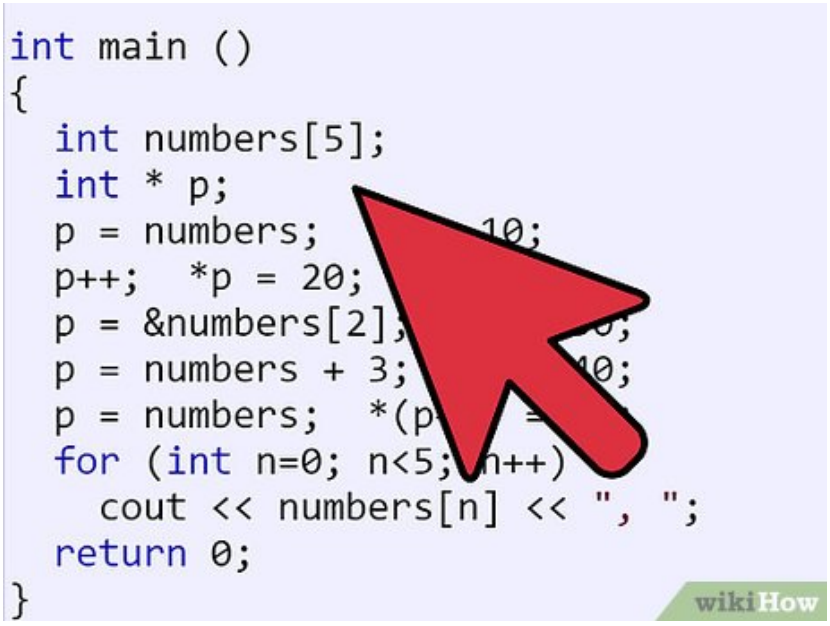
3. 

Assign the pointer to an initial memory location. This can be done using one of the following methods:

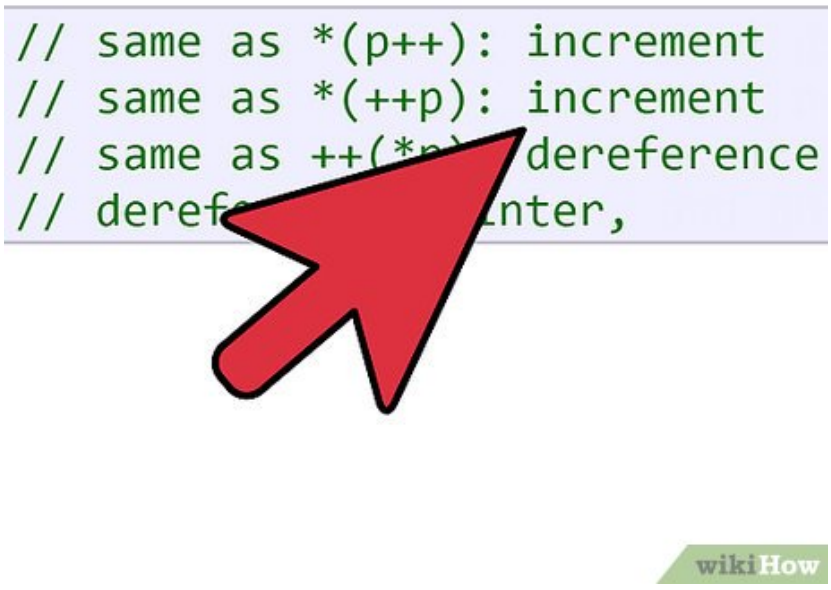
1. Allocating memory and pointing to it by the pointer: int * i = malloc(sizeof(int)*n); where n is the number of memory blocks to assign.
2. Assigning the address of a variable to the pointer: int * i = & x; where "x" is an integer and (&) means address-of.
3. Assigning an array identifier to the pointer: int * i = array1; where array1 is an integer array(int[] array1;).
4. Assigning a reference to the pointer: int * i = a; where "a" is an integer reference (int & a;).
5. Assigning another pointer to the pointer: int * i = z; where "z" is another integer pointer (int * z;)

4. 

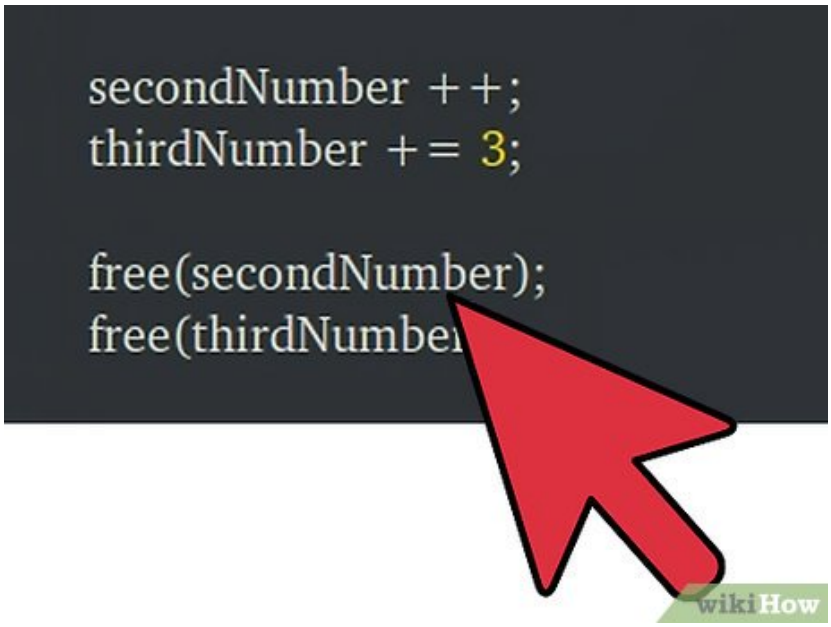
Whenever you need to extract the data item currently pointed to by the pointer, use the value-at-address operator (*): int x = *i; where i is an integer pointer.

5. 

Use the indexing operator on the pointer as if it was an array whenever you want to get a memory location next to the pointer without actually advancing the pointer. For example, if you have an integer pointer i, you can use i[2] which will retrieve the integer that is after the integer immediately after the integer pointed to by the reference (the integer that is 2 integers after the current location). The pointer i will still be pointing to the same memory location. Another alternative to this is getting the value at the pointer 2 steps after this pointer: *(i + 2)

6. 

Use the increment(++), decrement(--), += and -= operators whenever you need to change the current location. i += 5; will advance the integer pointer i 5 integers forward.

7. 

After you finish using the pointer, if you allocated memory to that pointer, make sure you free the allocated memory using the free() function. (free(i); where i is a pointer)