6.13 — Void pointers

BY ALEX ON JULY 19TH, 2007 | LAST MODIFIED BY ALEX ON JULY 8TH, 2017

The **void pointer**, also known as the generic pointer, is a special type of pointer that can be pointed at objects of any data type! A void pointer is declared like a normal pointer, using the void keyword as the pointer's type:

```
void *ptr; // ptr is a void pointer
```

A void pointer can point to objects of any data type:

```
int nValue;
float fValue;

struct Something
{
    int n;
    float f;
};

Something sValue;

void *ptr;
ptr = &nValue; // valid
ptr = &fValue; // valid
ptr = &sValue; // valid
```

However, because the void pointer does not know what type of object it is pointing to, it cannot be dereferenced directly! Rather, the void pointer must first be explicitly cast to another pointer type before it is dereferenced.

```
int value = 5;
void *voidPtr = &value;

//cout << *voidPtr << endl; // illegal: cannot dereference a void pointer

int *intPtr = static_cast<int*>(voidPtr); // however, if we cast our void pointer to an int pointer...

cout << *intPtr << endl; // then we can dereference it like normal
```

This prints:

5

The next obvious questions is: If a void pointer doesn't know what it's pointing to, how do we know what to cast it to? Ultimately, that is up to you to keep track of.

Here's an example of a void pointer in use:

```
#include <iostream>

enum Type
{
    INT,
    FLOAT,
    CSTRING
};

void printValue(void *ptr, Type type)
{
    switch (type)
    {
        case INT:
            std::cout << *static_cast<int*>(ptr) << '\n'; // cast to int pointer and dereference
```

```
            break;
        case FLOAT:
            std::cout << *static_cast<float*>(ptr) << '\n'; // cast to float pointer and dereference
            break;
        case CSTRING:
            std::cout << static_cast<char*>(ptr) << '\n'; // cast to char pointer (no dereference)
            // std::cout knows to treat char* as a C-style string
            // if we were to dereference the result, then we'd just print the single char that ptr is p
pointing to
            break;
    }
}

int main()
{
    int nValue = 5;
    float fValue = 7.5;
    char szValue[] = "Mollie";

    printValue(&nValue, INT);
    printValue(&fValue, FLOAT);
    printValue(szValue, CSTRING);

    return 0;
}
```

This program prints:

```
5
7.5
Mollie
```

Void pointer miscellany

Void pointers can be set to a null value:

```
void *ptr = 0; // ptr is a void pointer that is currently a null pointer
```

Although some compilers allow deleting a void pointer that points to dynamically allocated memory, doing so should be avoided, as it can result in undefined behavior.

It is not possible to do pointer arithmetic on a void pointer. This is because pointer arithmetic requires the pointer to know what size object it is pointing to, so it can increment or decrement the pointer appropriately.

Note that there is no such thing as a void reference. This is because a void reference would be of type void &, and would not know what type of value it referenced.

Conclusion

In general, it is a good idea to avoid using void pointers unless absolutely necessary, as they effectively allow you to avoid type checking. This allows you to inadvertently do things that make no sense, and the compiler won't complain about it. For example, the following would be valid:

```
    int nValue = 5;
    printValue(&nValue, CSTRING);
```

But who knows what the result would actually be!

Although the above function seems like a neat way to make a single function handle multiple data types, C++ actually offers a much better way to do the same thing (via function overloading) that retains type checking to help prevent misuse. Many other places where void pointers would once be used to handle multiple data types are now better done using templates, which also offer strong type checking.

However, very occasionally, you may still find a reasonable use for the void pointer. Just make sure there isn't a better (safer) way to do the same thing using other language mechanisms first!

Quiz

1) What's the difference between a void pointer and a null pointer?

Quiz answers

1) **Hide Solution**

A void pointer is a pointer that can point to any type of object, but does not know what type of object it points to. A void pointer must be explicitly cast into another type of pointer to be dereferenced. A null pointer is a pointer that does not point to an address. A void pointer can be a null pointer.


6.14 -- Pointers to pointers and dynamic multidimensional arrays


Index


6.12a -- For-each loops

Share this:

C++ TUTORIAL | PRINT THIS POST

64 comments to 6.13 — Void pointers

Ishak
April 16, 2018 at 12:49 pm · Reply

"Although some compilers allow deleting a void pointer that points to dynamically allocated memory, doing so should be avoided, as it can result in undefined behavior."

So I shouldn't return the memory location to memory? or should I assign the void pointer to a normal type pointer then delete the pointer?

"It is not possible to do pointer arithmetic on a void pointer. This is because pointer arithmetic requires the pointer to know what size object it is pointing to, so it can increment or decrement the pointer appropriately."

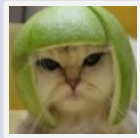
I'm sorry, I might be confused with this one but aren't all pointers like that- unlike array?

also there is this question i had from 6.9a and i didnt get the answer. if you would be kind to answer it.

"A dynamic array starts its life as a pointer that points to the first element of the array. Consequently, it has the same limitations in that it doesn't know its length or size. "

okay? But didn't you say that :"..new[] keeps track of how much memory was allocated to a variable"

thanks again for the outstanding material and the regular help.



Alex
April 16, 2018 at 3:02 pm · Reply

Generally you should try to avoid using void pointers altogether. If you do use them, you should try to use them in a temporary manner (e.g. a function that takes a void pointer), so that the void pointer will go out of scope normally (without deleting its contents). If you do that, you'll never have to delete a void pointer.