

Re: show disassembly

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2008-05/msg00638.html>

- *From:* Fil <Fil@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Sat, 31 May 2008 08:15:00 -0700
-

"Alex Blekhman" wrote:

You should get yourself a decent C++ textbook. Getting answers to specific questions on Usenet is OK, but it cannot substitute a proper learning.

Yes I am asking a lot of questions. And I feel somehow that I might abuse of your patience (I recognize that you're rally patient cause I never had in any forum such immedate and exhaustive answers). So thank you for the patience so far.

"Fil" wrote:

I read that the memory of variables is freed once a function is finished.
Is that the same for pointers?

C++ defines three types of storage: static, dynamic and automatic. Each type of storage has different rules reagrding its lifetime and allocation/deallocation.

Static storage type is for constants, global and static variables, which are exist for the lifetime of a program.

Dynamic storage type obtained via `operator new' (and/or `malloc' CRT function) and exists untill explicitly released with `operator delete' (and/or `free' CRT function).

Automatic storage for local variables is allocated on stack by the compiler and released automatically (hence the name "automatic") upon exit from current scope block.

Pointer variable is not any different than any other variable and will cease to exist if allocated on stack. However, the memory that the pointer points to is the subject to its storage type

Re: show disassembly

rules.

For example:

```
{
int a = 42; // regular automatic variable
int* p1 = &a; // pointer to some int
int* p2 = new int[10]; // another pointer
}
```

Here all variables are local and automatic. The memory they occupy will be reclaimed once execution flow leaves the scope block. However, the memory that `p2` points to will outlive the scope block. You should take care of the value that `p2` holds in order to free allocated memory later. Otherwise you'll leak it:

```
int* pOut = NULL;
{
// ...
int* p2 = new int[10];

pOut = p2;

} // p2 is destroyed, but pOut holds the address

// Free allocated memory elsewhere.
delete[] pOut;
pOut = NULL;
```

But I can understand that it is useful to free some memory if we are in the middle of a function and there's a long way to go before the end and we can run short of memory.

The whole point of dynamic storage is that you want it to outlive current function and pass allocated memory to the caller. Otherwise you could happily use local automatic variables for storage.

Dynamic storage is my only way to create an array whose size I can't predict when I write the program because it will depend on a variable. In your above paragraph, are you saying that, since in my case I don't care about having a scope longer than my function, I could use local automatic variables?

Does pret is still pointing to it before you run the next assignment (do we find in pret the address of the first slot of memory which it was pointing to even after delete before the

Re: show disassembly

assignment to NULL – I will try that by myself)?

`pret' itself will not change after `operator delete'. But the memory it pointed to will no longer be valid. Memory manager is free to rearrange memory blocks as it sees fit. Accessing a memory after freeing it leads to undefined behaviour. ("Undefined behaviour" is C++ term that means that the outcome of the operation is undefined. You may get away with it, but may crash your system as likely.)

That's why it's a good habit to zero unused pointer immediately after deallocation in order to prevent accessing freed memory.

let say we also want to free the memory occupied by the variables we're not going to use anymore. Can we undeclare them?

You cannot undeclare them, of course, but you can introduce another scope block:

```
void foo()
{
  int a = 5;

  {
    int b = 10;
  } // variable b is destroyed here

} // variable a is destroyed here
```

At that point I also could think of functions. I am running my main, I used a function and I know I won't use it anymore. It probably takes some space. Can I free it or is it automatically done?

It's done automatically with the end of a program. Function's code lives in static storage and cannot be explicitly deallocated.

HTH
Alex

Thanks a lot

.

Re: show disassembly