Lesson 5

Keywords

`const` vs `volatile`

`static`

`private/public/protected` (OO languages like C++/C# only)

`static const` or `const static` (why is this needed?)

`virtual`

A note on C

Don’t write C (unless you have to).

However:

You **should** understand how to write C (although you should never write it).

const vs volatile

In C (compiled lanaguage), const vs volatile refer to the level of optimisation a compiler could do with these declared variables. Const variables can be preloaded into memory, volatile variables need to be left alone.

In C++, const is similar but volatile refers to thread safety (and actually just should not be used!)

const vs volatile

Different in C#:

const:

Does what it says on the tin: declares the variable (or class, in C# all variables are classes) as readonly.

volatile:

Declares that the variable could be modified by multiple threads (out of scope of this teaching for now, but worth knowing)

static

Different between C and C++/C#

C: static functions confine them to that particular file.

OO languages: static refers to a class member/function that can be accessed without an object instantiation.

static const

private/public/protected

Private: access only within the class.

Public: can access from within the class.

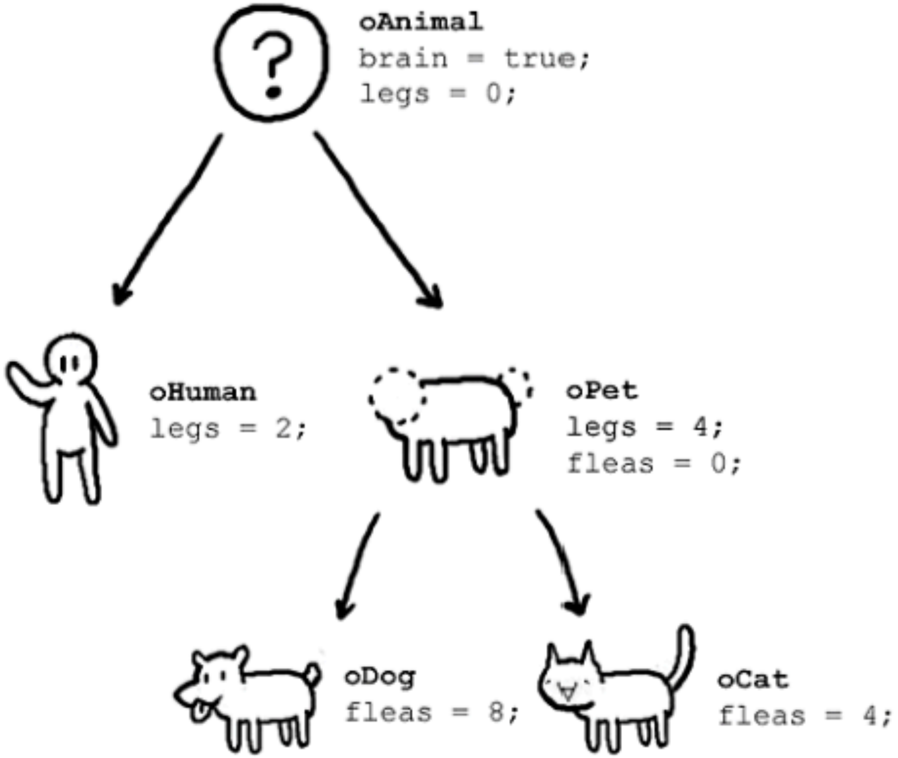
Protected: effectively private, but still modifiable from classes that inherit.

Inherit?

What does that mean?

Inherit?

What does that mean?



virtual

functions described as virtual are functions to be defined in child classes.

Definition: A parent class containing only virtual functions is an ***abstract base class.***

virtual

```
class animal
{
virtual public void makeNoise();
//no definition needed here. animal is thus an abstract base class
}
class dog : animal // dog inherits from animal
{

makeNoise(){
Console.Writeline(“Woof”);
}

}
```

C coding time

Pointers.

Important to understand, because pass by value and pass by reference is a concept universal to all programming languages.

Pointers

“Point” to a piece of memory.
Declared with *. “get the variable address” done with & (ampersand)

UNFORTUNATELY

The designers of C also overloaded the * operator, which declares and also dereferences the pointer (in other words, get what is stored at this piece of memory).

Pointers thus often confuse people when they see the * operator doing multiple things. This is further compounded as ** can mean get a pointer to a pointer, or also dereference a new pointer, depending on the context.

Simple example

```
#include <stdio.h>
int main(int argc, char* argv[]){

    int p = 42;
    int *p_pointer;
    p_pointer = &p;
    printf("%d\n",*p_pointer);


}
```

What’s the point though?

C was the first language that attempted to make life easier.

C only returns: void (nothing), single types (ints, doubles, long ints etc)

That’s it.

What's the point though?

What if I wanted a function that operated on an array and returned an array? Or I wanted to return a char? Or a string?

Well, if we had no pointers, we'd be stuck, because, put simply, you can't.

What’s the point though?

Instead we have pointers. C functions often look a bit like this:

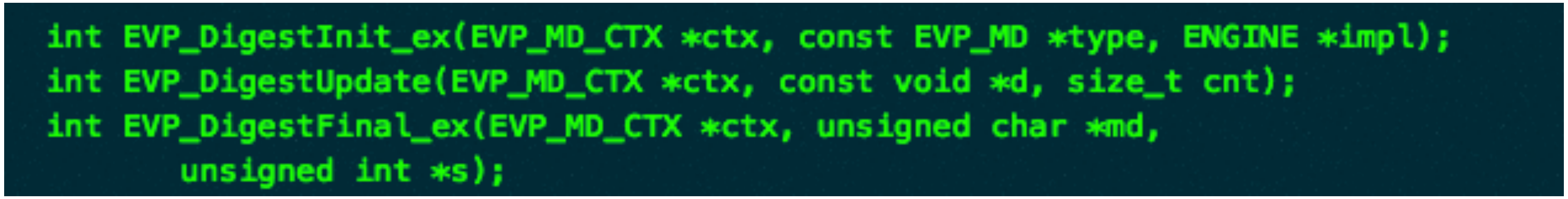
```
int doSomething(int Number, int* inputArray, int* outputArray){
…
}
```

We pass in a pointer to an input array, and a pointer to an output array, and then the function looks at the piece of memory instead.

Note how we’re still restricted to only using single types.

What’s the point though?

Real C code function headers:
Lots of pointers!



```
int EVP_DigestInit_ex(EVP_MD_CTX *ctx, const EVP_MD *type, ENGINE *impl);
int EVP_DigestUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt);
int EVP_DigestFinal_ex(EVP_MD_CTX *ctx, unsigned char *md,
       unsigned int *s);
```

There are no pointers in higher level languages!

Or are there?

Actual C# excerpt from one of the problems we worked though:

```
private void countCharacters(string input,ref int[ ] countArr)
 {
     foreach (char c in input)
     {
         for (int i = 0; i < alphabet.Length; i++)
         {
             if (c == alphabet[i])
             {
                 countArr[i]++;
             }
         }
     }
 }
```

Pointer summary:

*   Points to a piece of memory or dereferences a pointer.

&   Returns the address of a variable.

Next time:

No homework (have a good xmas!)

But next lesson we're going to go through data structures, and this will be written in C.

Make sure you know how to compile a simple C program. On windows you can run from the developer command window, vc "program.c" and it should compile to an exe.

And we **will** be making liberal use of pointers!