

THAT PESKY WARNING

- It says something about using deprecated (or antiquated) headers.
- Normally, you do not need to worry about it.
 - But I did find some instances where things do go wrong.
- So, if you see a bunch of errors with no apparent explanation, switch from, e.g.,

```
#include <iostream.h>
```

to

```
#include <iostream>
```

- Then, use the “**standard namespace.**”

NAMESPACES

- A C++ program can be divided into different **namespaces**.
- If you have a variable x defined in namespace n but you are in some other namespace, you can still refer to x as $n::x$.
- Namespaces are a primitive method of encapsulation.
 - They are **arguably** useless (**why?**).
 - But you may need to take them into consideration.
- When compiling with GCC 2.9x or with GCC 3.x/4.x and using those antiquated headers, the library functions (classes, etc.) are in no particular namespace.
- However, when using GCC 3.x/4.x, all the usual functions, variables, and classes (e.g., `cout`, `cin`) are defined in the namespace `std`.
 - Your program isn't.

NAMESPACES (CONT'D)

- So, when using GCC 3.x/4.x you cannot write

```
cout << "Hello.\n";
```

but have to write instead

```
std::cout << "Hello.\n";
```

- So what to do?
 - You could of course prefix each and every library call by `std::`, just to be on the safe side (ugh!).
 - You could also bring to your program entities from the `std` namespace (better).
 - You do this by using the directive `using namespace`, so you do:

```
#include <iostream>
... other header files, code here ...
using namespace std;
```

Also mind the semicolons!

- Declaration construction: `namespace std { ... code ... }`

THE NAMESPACE CONSTRUCTION

- The namespace construction is used to **define** entities in a given namespace

```
#include <iostream>
namespace hello {
    int print() {
        std::cout << "Hello, world.\n"; // cout in other namespace!
    }
}
int main(int argc, char** argv) {
    hello::print(); // print() in other namespace!
}
```

- Separating the declaration from the definition:

```
#include <iostream>
namespace hello {
    void print();
}
void hello::print() { // we are outside 'hello'
    std::cout << "Hello, world.\n"; // cout in other namespace!
}
int main(int argc, char** argv) {
    hello::print(); // print() in other namespace!
}
```

THE USING DIRECTIVE

- The `using` directive is used to **access** entities from other namespace
- Can be used in **any scope** to “import” entities from other namespaces one by one

```
#include <iostream>
```

```
int main(int argc, char** argv) {  
    using std::cout;  
    cout << "Hello, world.\n";  
}
```

```
#include <iostream>
```

```
int main(int argc, char** argv) {  
    std::cout << "Hello, world.\n";  
}
```

- Can also be used to import namespaces as a whole

```
#include <iostream>  
using namespace std;
```

```
int main(int argc, char** argv) {  
    cout << "Hello, world.\n";  
}
```

```
#include <iostream>
```

```
int main(int argc, char** argv) {  
    using std::cout;  
    cout << "Hello, world.\n";  
}
```