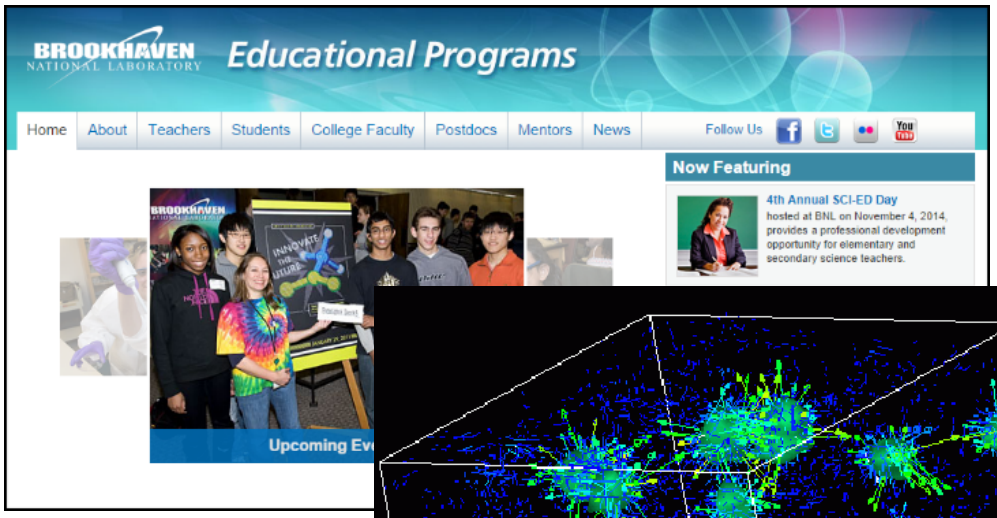
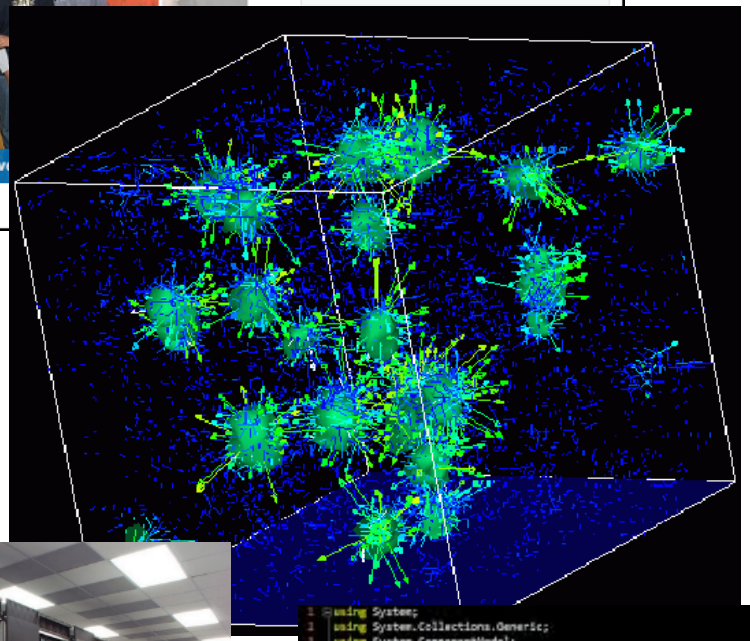
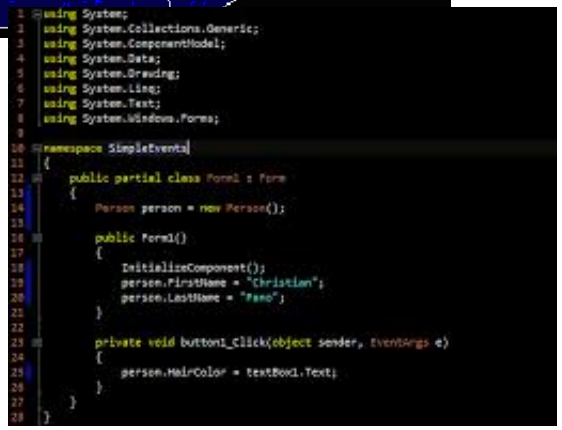






Survey of Scientific Computing (SciComp)

Unit 2

Structure of a Program

Information adapted from cplusplus.com

Goals

- Understand the general program structure
- Begin to identify some code syntax
  - Comments
  - Namespace std
- Modify your 1st and 2nd program “Hello World”

Structure of a Program

The best way to learn a programming language is by writing programs. Typically, the first program beginners write is a program called "Hello World", which simply prints "Hello World" to your computer screen. Although it is very simple, it contains all the fundamental components C++ programs have:

```
1 // my first program in C++
2 #include <iostream>
3
4 int main()
5 {
6   std::cout << "Hello World!";
7 }
```

Copy this program into CodeBlocks

The left panel above shows the C++ code for this program. The right panel shows the result when the program is executed by a computer. The grey numbers to the left of the panels are line numbers to make discussing programs and researching errors easier. They are not part of the program.'

Let's examine this program line by line:

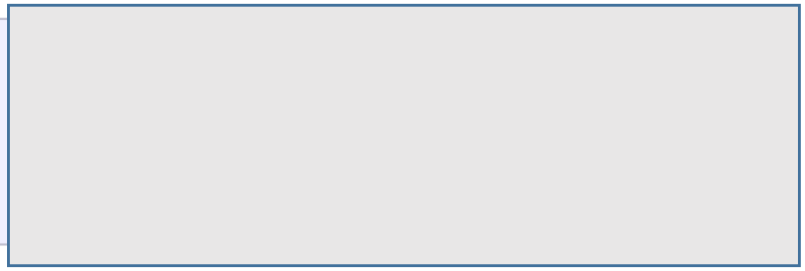
Line 1: // my first program in C++Two slash signs indicate that the rest of the line is a comment inserted by the programmer but which has no effect on the behavior of the program. Programmers use them to include short explanations or observations concerning the code or program. In this case, it is a brief introductory description of the program.

Line 2: #include <iostream>Lines beginning with a hash sign (#) are directives read and interpreted by what is known as the *preprocessor*. They are special lines interpreted before the compilation of the program itself begins. In this case, the directive #include <iostream>, instructs the preprocessor to include a section of standard C++ code, known as *header iostream*, that allows to perform standard input and output operations, such as writing the output of this program (Hello World) to the screen.

Line 3: A blank line. Blank lines have no effect on a program. They simply improve readability of the code.

Structure of a Program..Cont

```
1 // my first program in C++
2 #include <iostream>
3
4 int main()
5 {
6   std::cout << "Hello World!";
7 }
```



Line 4: int main ()This line initiates the declaration of a function. Essentially, a **function is a group of code statements which are given a name**: in this case, this gives the name "main" to the group of code statements that follow. Functions will be discussed in detail in a later section, but essentially, their definition is introduced with a succession of a type (int), a name (main) and a pair of parentheses (()), optionally including parameters.

The function named main is a special function in all C++ programs; it is the function called when the program is run.

The execution of all C++ programs begins with the main function, regardless of where the function is actually located within the code.

Lines 5 and 7: { and }The open brace ({) at line 5 indicates the beginning of main's function definition, and the closing brace (}) at line 7, indicates its end. Everything between these braces is the function's body that defines what happens when main is called. **All functions use braces to indicate the beginning and end of their definitions.**

Structure of a Program..Cont

```
1 // my first program in C++
2 #include <iostream>
3
4 int main()
5 {
6   std::cout << "Hello World!";
7 }
```

Line 6: std::cout << "Hello World!";This line is a C++ statement. **A statement is an expression that can actually produce some effect.** It is the meat of a program, specifying its actual behavior. **Statements are executed in the same order that they appear within a function's body.**

This statement has three parts: First, std::cout, which identifies the **st**andar**d** **c**haracter **out**put device (usually, this is the computer screen). Second, the insertion operator (<<), which indicates that what follows is inserted into std::cout. Finally, a sentence within quotes ("Hello world!"), is the content inserted into the standard output.

*Notice that the statement ends with a semicolon (;). This character marks the end of the statement, just as the period ends a sentence in English. **All C++ statements must end with a semicolon character**. One of the most common syntax errors in C++ is forgetting to end a statement with a semicolon.

Structure of a Program..Cont

You may have noticed that not all the lines of this program perform actions when the code is executed. There is a line containing a comment (beginning with //). There is a line with a directive for the preprocessor (beginning with #). There is a line that defines a function (in this case, the main function). And, finally, a line with a statements ending with a semicolon (the insertion into cout), which was within the block delimited by the braces ({ }) of the main function.

The program has been structured in different lines and properly indented, in order to make it easier to understand for the humans reading it. But C++ does not have strict rules on indentation or on how to split instructions in different lines. For example, instead of

```
1 int main ()
2 {
3   std::cout << " Hello World!";
4 }
```

We could have written:

```
int main () { std::cout << "Hello World!"; }
```

all in a single line, and this would have had exactly the same meaning as the preceding code.

In C++, the separation between statements is specified with an ending semicolon (;), with the separation into different lines not mattering at all for this purpose. Many statements can be written in a single line, or each statement can be in its own line. The division of code in different lines serves only to make it more legible and schematic for the humans that may read it, but has no effect on the actual behavior of the program.

Structure of a Program..Cont

Now, let's add an additional statement to our first program:

```
1 // my second program in C++
2 #include <iostream>
3
4 int main ()
5 {
6   std::cout << "Hello World! ";
7   std::cout << "I'm a C++ program";
8 }
```

Copy this program into CodeBlocks

In this case, the program performed two insertions into std::cout in two different statements. Once again, the separation in different lines of code simply gives greater readability to the program, since main could have been perfectly valid defined in this way:

```
int main () { std::cout << " Hello World! "; std::cout << " I'm a C++ program "; }
```

The source code could have also been divided into more code lines instead:

```
1 int main ()
2 {
3   std::cout <<
4     "Hello World!";
5   std::cout
6     << "I'm a C++ program";
7 }
```

Structure of a Program..Cont

```
1 int main ()
2 {
3   std::cout <<
4     "Hello World!";
5   std::cout
6     << "I'm a C++ program";
7 }
```

And the result would again have been exactly the same as in the previous examples.

Preprocessor directives (those that begin by #) are out of this general rule since they are not statements. They are lines read and processed by the preprocessor before proper compilation begins. Preprocessor directives must be specified in their own line and, because they are not statements, do not have to end with a semicolon (;).

Comments

As noted, comments do not affect the operation of the program; however, they provide an important tool to document directly within the source code what the program does and how it operates.

C++ supports two ways of commenting code:

```
1 // line comment
2 /* block comment */
```

The first of them, known as *line comment*, discards everything from where the pair of slash signs (//) are found up to the end of that same line. The second one, known as *block comment*, discards everything between the /* characters and the first appearance of the */ characters, with the possibility of including multiple lines.

Let's add comments to our second program:

Comments

Let's add comments to our second program:

```
1  /* my second program in C++
2     with more comments */
3
4  #include <iostream>
5
6  int main ()
7  {
8    std::cout << "Hello World! ";     // prints Hello World!
9    std::cout << "I'm a C++ program"; // prints I'm a C++ program
10 }
```

Modify 2nd Program

If comments are included within the source code of a program without using the comment characters combinations //, /* or */, the compiler takes them as if they were C++ expressions, most likely causing the compilation to fail with one, or several, error messages.

Using namespace std

If you have seen C++ code before, you may have seen cout being used instead of std::cout. Both name the same object: the first one uses its ***unqualified name*** (cout), while the second qualifies it directly within the ***namespace*** **std** (as std::cout).

cout is part of the standard library, and all the elements in the standard C++ library are declared within what is called a *namespace*: the namespace std.

In order to refer to the elements in the std namespace a program shall either qualify each and every use of elements of the library (as we have done by prefixing cout with std::), or introduce visibility of its components. The most typical way to introduce visibility of these components is by means of *using declarations*:

```
using namespace std;
```

Using namespace std

```
using namespace std;
```

The above declaration allows all elements in the std namespace to be accessed in an *unqualified* manner (without the std:: prefix).

With this in mind, the last example can be rewritten to make unqualified uses of cout as:

```
1 // my second program in C++
2 #include <iostream>
3 using namespace std;
4
5 int main ()
6 {
7   cout << "Hello World! ";
8   cout << "I'm a C++ program";
9 }
```

Modify 2nd Program

Both ways of accessing the elements of the std namespace (explicit qualification and *using* declarations) are valid in C++ and produce the exact same behavior. For simplicity, and to improve readability, in these examples we will more often use this latter approach with *using* declarations, although note that *explicit qualification* is the only way to guarantee that name collisions never happen.

Now you know…

- Understand the general program structure
- Begin to identify some code syntax
  - Comments
  - Namespace std
- Modify your 1st + 2nd program “Hello World”