(PRACTICAL)
COMPUTATIONAL PHYSICS

Physics 551
Lecture 3

NOTATION

Extra Reading

Optional Exercise

Recommended

- This lecture slides for this course will attempt to use a uniform notation throughout. A normal paragraph looks like this.

☞ *Italicized paragraphs with pen bullets will indicate definitions, with the defined word or phrase shown in* ***SMALL-CAPS****.*

✏ Pencil bullets will indicate the introduction of **new notation**.

☞ Pointing hand bullets indicate important points that might otherwise be overlooked.

SEARCHING FOR FILES AND DIRECTORIES

`man find`

- The **`find`** utility is used to search the Linux file system for particular files or directories e.g.

```
$ find path -name "name_pattern"
```

☞ The ***`path`*** argument specifies the top level of the hierarchy that **`find`** should search.

☞ The **`name_pattern`** argument supplied to the the **`-name`** flag can be a **string**, or a **wildcard expression**.

☞ *A **WILDCARD EXPRESSION** is a string that uses occurrences of the "*" character to represent **any number** of **arbitrary** characters.*

SEARCHING FOR FILES AND DIRECTORIES

`man find`

- If `find` is invoked using the `-name` flag, then the **final token** of the file or directory's path **must** match `name_pattern`.
- For example:

```
$ find . -name "*file*"
```

will match and return "`./subdirectory/somefile`", but **not** "`./myfiles/examples`"

☞ To match patterns against **other path tokens** you must use the `-wholename` flag e.g.

```
$ find . -wholename "*file*"
```

SEARCHING FOR SPECIFIC STRINGS IN FILES

`man grep`

- The **`grep`** utility is used to search **within** a file for occurrences of a particular string e.g.

```
$ grep "pattern" file_path
```

- The **`"pattern"`** argument can be a **simple string**, or a special pattern specifier called a **regular expression**.

➪ ***REGULAR EXPRESSIONS*** *are specially formatted* ***strings*** *that* ***describe*** *arbitrary patterns of characters to be matched.*

- They use a special syntax to **generically** describe the arrangements of characters that are required to obtain a valid match.

SEARCHING FOR SPECIFIC STRINGS IN FILES

`man grep`

- The **syntax** used to define regular expressions is **rich** and **extensive**, but too complex to cover in detail here.
- See this lecture's **recommended reading** and the demonstration material for more information.
- As an example of the power of regular expressions, the invocation

```
$ grep "^[A-Z]+\.[a-z]{3}" text_file
```

will search *`text_file`* for lines ***starting*** with ***at least one upper case letter***, followed by a ***literal `'.'` character*** and then ***exactly 3 lower case letters***!

AUTOMATIC COMMAND ARGUMENT SUBSTITUTION

- One of the most useful shell commands is the `xargs` command, which **repeatedly executes another** shell command with **different arguments**.
- A typical `xargs` invocation looks like `man xargs`

```
$ cat subList | xargs -isub repeated_command sub
```

- Here *`subList`* is assumed to be a **newline-separated list** of argument values to be substituted.
- The pipe operator "`|`" is used to pass the output of `cat` to the input of `xargs`.

AUTOMATIC COMMAND ARGUMENT SUBSTITUTION

- `xargs` invokes `command` once for **each** argument listed in *`subList`*.
- *`sub`* is an **arbitrary** string of characters. When `xargs` invokes `command` it replaces every instance of *`sub`* with the argument obtained from *`subList`*.
- *`sub`* should be chosen so that it does not conflict with any tokens e.g. flags comprising the invocation of `command`.
- See the **demonstration material** for concrete examples.

DEMONSTRATION

Using `xargs`, `find` and `grep`

Clone the Shell Utility demonstration material from Github:

```
$git clone https://github.com/hughdickinson/CompPhysL3Shell.git
 /home/computationalphysics/Documents/theShellGym/lecture3
```

MORE C++

Be sure to thoroughly review the demonstration material!

- **Make sure** that you clone the **C++ demonstration material** for this lecture from GitHub.
- It contains **a lot** of information that you **will not** find in the slides including:

  scopes, code blocks, functions, arrays, pointers, references, pass-by-value versus pass-by-reference and namespaces.

- It also introduces the `<cmath>` header file, which provides several **mathematical** functions that will be very useful for your homework.

DEMONSTRATION

More C++

Clone the C++ demonstration material from Github:

```
$git clone https://github.com/hughdickinson/CompPhysL3CPP.git
 /home/computationalphysics/Documents/cPlusPlus/lecture3
```

"COMPILING" C++ CODE

- So far, we have used the **cling** interpreter to experiment with the components of the C++ language.
- Cling permits algorithmic **evaluation** and **prototyping** using code snippets that do not constitute a valid C++ program.
- However, all **complete** C++ programs should be **compiled** and executed as **standalone binary executables**.
- Several compiler utilities exist for C++, including **proprietary** compilers like Microsoft's `Visual C++` as well as **open-source** options like the GNU Compiler Collection's `g++` and `cling`'s backend compiler, `clang++`.

THE MAIN FUNCTION

- **All compiled** C++ programs **must** define a function called `main`.
- *The* **`main`** *function is called the* ***ENTRY POINT*** *of the compiled program.*
- When the operating system runs your program it will automatically call the **`main`** function.

☞ The **`main`** function **must** return an **integer** value.

- Several function **signatures** are permitted for the **`main`** function.

THE MAIN FUNCTION

- The **most commonly used** definition of the main function in C++ looks like:

```
int main(int argc, char * argv[]){
    /* …code goes here… */
    return 0;
}
```

- **This** function signature has **two parameters** that relate to the shell command your program was invoked with
  1. The integer parameter **`argc`** (**arg**ument **c**ount) counts the number of command line tokens **including the program name** comprising the invocation.
  2. The second parameter **`argv`** is actually an array of **`argc`** pointers to **arrays** of characters (i.e. strings) that specify the **values** of the command line tokens.

☞The main function should return 0 on success and non-zero values otherwise.

BUILDING OUR PROGRAM

- We will use the `clang++` utility to convert our C++ code into a binary executable. man clang (**not** man clang++!)
- To simplest possible invocation of `clang++` looks like

```
$ clang++ -o pathToExecutable sourceCodeFiles…
```

- This command actually does several things - compilation being one of them!
  1. **Preprocesses** each of the *`sourceCodeFiles`*.The code in included **header files** is merged with your source code at this stage.
  2. **Compiles** each of the **preprocessed** *`sourceCodeFiles`* into intermediate **assembler** files.

INVOKING THE COMPILER

- ☞ Assembler code is expressed in a special language with instructions that are optimized for a **specific computer architecture**.

3. **Links** each of the assembled **binary object files** and any required **static libraries** into the main binary executable.
4. Adds references to any **dynamic** (or **shared**) **libraries** that will provide executable code **at runtime**.
   - ☞ Even the simplest C++ programs will probably use elements of the **C++ standard library** at runtime.
5. Generates the specified binary executable with the correct filesystem **permissions** (check with `ls -l`) to enable its execution.

DEMONSTRATION

Building a simple C++ program

Clone the C++ demonstration material from Github:

```
$git clone https://github.com/hughdickinson/CompPhysL3CPP.git
 /home/computationalphysics/Documents/cPlusPlus/lecture3
```

LECTURE 3 SUMMARY

- After reviewing the material in this lecture **and completing the reading exercises** you should know:

  1. How to search for files and directories in the Linux filesystem using the **`find`** utility.
  2. How to search for literal strings and character patterns within individual files using the **`grep`** utility.
  3. How to repeatedly invoke a shell command with arguments substitutes from a list using the **`xargs`** utility.

LECTURE 3 SUMMARY

4. How to pass the output from one shell command to another using the **pipe operator** "|".
5. What is meant by the terms **scope** and **code block** and how the two are related in C++.
6. How to **declare** and **define** functions in Cling (using `.rawInput`) and C++ in general.
7. The distinction between function **parameters** and function **arguments**.

LECTURE 3 SUMMARY

8. Which tokens in a function declaration constitute the function's **signature**.
9. What is meant by function **overloading**.
10. How to **call** functions in C++.
13. How to declare and initialize **arrays** in C++.
14. That array **elements** are **zero-indexed** in C++.
15. How to **access** and **assign** array elements using the **indexing operator** ("[]").

LECTURE 3 SUMMARY

16. What is meant by the term **pointer** in C++.
17. How to **declare** and **initialize** pointer-type variables in C++
18. How to **dynamically allocate** memory using the **new** operator.
19. How to **dereference** a pointer in order to access the value stored at the memory location it points to.
20. How to declare and initialize **pointers to arrays**.

LECTURE 3 SUMMARY

16. What is meant by the term **pointer** in C++.
17. How to **declare** and **initialize** pointer-type variables in C++
18. How to **dynamically allocate** memory using the **new** operator.
19. How to **dereference** a pointer in order to access the value stored at the memory location it points to.
20. How to declare and initialize **pointers to arrays**.

LECTURE 3 SUMMARY

21. How to **dynamically allocate memory** for **pointer-to-array** types using the **new[]** operator.
22. How to **free dynamically allocated memory** using the appropriate **`delete`** or **`delete[]`** operators.
23. How to **index** the elements of **pointer-to-array types** i.e. in an identical fashion to normal array types.
24. The implications of passing function arguments **by value** for the persistence of in-function modifications.

LECTURE 3 SUMMARY

25. That arrays are **always** passed as **pointers to their first element**.
26. The implications of this for the persistence of in-function modifications to array elements.
27. How to declare and initialize **references** in C++,
28. That **all** references **must** be **initialized when they are declared**.
29. That references define an **alias** for a **preexisting** identifier.

LECTURE 3 SUMMARY

30. How to write functions that specify that their arguments are passed **by reference**.
31. The implications of this for the persistence of in-function modifications to such arguments.
32. That **literal** values **cannot** be passed to functions as (mutable) references.
33. The meaning and purpose of **namespaces** in C++.

LECTURE 3 SUMMARY

34. How to **define** namespaces in C++.
35. How to **explicitly specify** that an identifier belongs to a namespace using the **scope resolution operator** "`::`".
36. That entities provided by the C++ standard library are defined within the "`std`" namespace.
37. That mathematical functions and constants are provided by the "*`cmath`*" header file.

LECTURE 3 SUMMARY

34. How to write complete, **compile-able** C++ programs that contain the **required** `main` function.
35. That the `main` function **must** return an **integer** value upon completion - conventionally, **zero on success**.
36. The form of a commonly used **signature** for the `main` function that gives access to the **shell tokens** used to invoke the **compiled executable**.
37. How to **build** a simple C++ program using `clang++`.

LECTURE 3 HOMEWORK

Read sections:

- Program structure → Statements and flow control
- Program structure → Functions
- Program structure → Name visibility
- Compound data types → Arrays
- Compound data types → Pointers
- Compound data types → Dynamic memory

from the **C++ Reference** language tutorial:

http://www.cplusplus.com/doc/tutorial

Be sure to thoroughly review the C++ demonstration material!

Read Chapters 2.2 and 2.5 from the **Git Pro Book**.

LECTURE 2 HOMEWORK

Investigate the functions that are provided by the **cmath** header file that is provided by the C++ standard library.
http://www.cplusplus.com/reference/cmath/

Regular Expressions Tutorial - See Blackboard Learn.

- Complete the **Lecture 3 Homework Quiz** that you will find on the course Blackboard Learn website.
- This week's quiz has fewer questions but they are **more difficult**.
- You will need to **extend** the concepts we covered in the lecture. Simple **copy-and-paste will not be enough**!