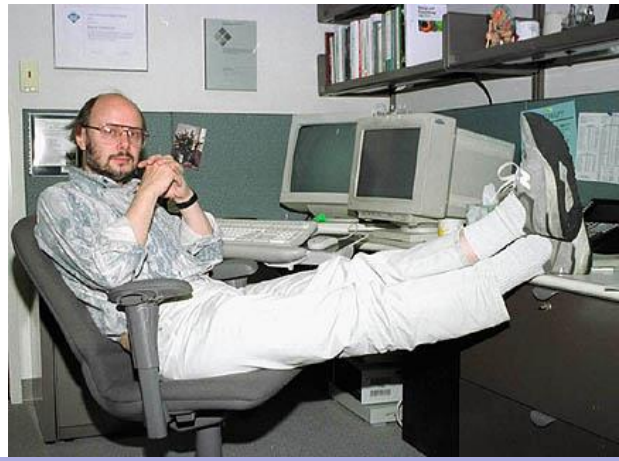
Introduction to Programming and Computing for Scientists

Tutorial-2a: First steps with C++ programming

Quick refresher : C++ in a nuttshell

- Created by Bjarne Stroustrup at AT&T Bell Laboratories in the early 1980's, later maintenance, governance has been taken over by the Standard C++ Foundation
- General purpose, strongly-typed, high-level programming language based on the C language. C++ is an "incremented" version of C
  - "*the name signifies the evolutionary nature of the changes from C*"
  - ++ is the (post) increment operator in C
  - C++ is much more than that, it is a very powerfull & flexible language
- Compiled language
  - Compilers play an important role, standard-compliant and compatible compilers have been a challenge for C++ community for quite a long period
- Multi-paradigm language: **object-oriented**, procedural, functional
- Multi-platform (Windows, Unix, Mac, etc...)
- Standardized by an ISO WG, latest version C++11



Before we start...

"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg."

Bjarne Stroustrup, developer of the C++ programming language

"The problem with using C++ … is that there's already a strong tendency in the language to require you to know everything before you can do anything."

Larry Wall, developer of the Perl language

One of the simpliest c++ programs

- Main function
  - No input parameter ()
  - Returns an integer (zero value for success)
  - Main is mandatory
  - C++ programs begin executing at main()
  - What the function does is described within {}
  - // is the way to add single line comments

```
// simple c++ code

#include <iostream>


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

- The program itself, the body of the main function between the left and right brace {}:
  - Prints a text message to the standard output (screen)
  - Printing is done with the iostream library, therefore the iostream library has to be ”included” into the program
  - The library defines several objects & functions to work with I/O streams
    - std::cout is the standard output, printing to the stdoutput is done via the >> operator
    - std::cout is a c++ way of denoting the namespace of the library
  - The message to be printed is the one in quotes: Hello, world!
  - Note the semicolon (;) at the end of the printout line

From source code to executable

The usual steps for generating a C++ program are:

1. Editing: User writes the source code for a C++ program in the text editor
2. Preprocessing: The C++ preprocessor carries out various source code manipulations before the compiler's translation phase. The manipulations usually include inserting the contents of other source code files, preparing for conditional compilation, and performing various text replacements.
    1. E.g. #include statements are taken care by the preprocessor
3. Compilation: The C++ compiler translates the C++ program into machine code (or object code).
4. Linking: The linker links the object codes (user written code and existing libraries) to produce an executable file.

Note: compilers can combine (hide) steps 2-4 from the user.

Exercise:

- Type in the simple C++ code from the previous slide (save as hello.cpp)
- Generate an executable: g++ -o printhello hello.cpp
- Run the binary file: ./printhello
- Use the // to comment out various lines in the code, compile it and look at the compiler error messages.

basic Input / Output

I/O in C++ is managed via streams of bytes, where a stream is simply a sequence of bytes

- cout << “some text to be printed out”;
  - cout is the output stream object
  - << is the stream insertion operator
  - cout and << are used to display text strings on the screen
- cin >> Variable;
  - cin is the input stream object
  - >> is the stream extraction operator
  - cin and >> are used to take input from the standard input stream (usually the keyboard) and assign it to a variable

Note: more advanced I/O stream operations including character strings and reading/writing to/from files will come later

Exercise:

- Write a simple program that asks for two integers and prints out their sum and multiplication.
- Hint: you can chain output strings
- Hint: \n is the new line “character” or you can use the endl stream object
- Hint: it is possible to insert mathematical formulas into the output stream

basic Input / Output: exercise solution

```
// IO exercise
#include <iostream>

int main() {
    int numberA, numberB;
    std::cout << "Enter the first number:\n";
    std::cin >> numberA;
    std::cout << "Enter the second number:"<< std::endl;
    std::cin >> numberB;
    std::cout << "Sum of " << numberA << " and " << numberB << " is " << numberA+numberB << "\n";
    std::cout << numberA << " times " << numberB << " equals " << numberA * numberB << std::endl;
}
```

Variables

- A variable is a “name” that is associated with memory reserved for storing the variable's value.
  - You store data in a program by assigning values to variables
- Every variable has a name, a type, a value and a scope/lifetime:
  - Name: series of characters consisting of letters, digits, and underscores. Case sensitive. No spaces. May not begin with a number.
  - Type: defines what sort of values the variable can store (e.g. int, float or char)
  - Value: a variable gets its value via the assignment operator (=)
  - Scope: a variable can be global or local:
    - Variables declared outside functions, including main(), are global. They exist for the duration of a program and can be accessed from anywhere in the code.
    - Variables declared inside functions are local to those functions. Local variables may be accessed only inside the block in which they are declared. When a function begins, it allocates space on the stack to hold its local variables. This space exists only while the function is active, after the function returns, it deletes the allocated stack space, including all local variables.

Variables: Data types (further details)

- **Exact ranges and memory sizes** of datatypes **may differ** depending on compilers, compiler options and operating systems
- Basic types: bool, char, int, float, double
- Modifiers: unsigned, signed (default), short, long

Type (keyword)	Size (byte)	Range
char	1 byte	-128 to 127 or 0 to 255
int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
short int	2 bytes	-32,768 to 32,767
long int	4 bytes	-2,147,483,648 to 2,147,483,647
float	4 bytes	(+/-) 1.17E-38 to 3.4E+38 (6 or 7 significant digits)
double	8 bytes	(+/-) 2.2E-308 to 1.79E+308 (15 significant digits)
long double	16 bytes	(+/-) 3.4E-4932 to 1.1E+4932 (appx. 19 significant digits)

Basic operators

- In C++ the basic operators work the same way as in other languages, including the usual operator precedence (first parentheses, then */%, then +-, then left to right rule)
- division (/) applied to integers will truncate the result (e.g. 5/2 equals to 2 in integer division)
- ++: increment operator adds one to its operand
- --: decrement operator substracts one from its operand
- =: assignment operator to assign values to variables

Operation	C++ operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%
Decrement	--
Increment	++
Assignment	=

Relational and logical operators

Operation	C++ operator
Equal to	==
Not equal to	!=
Greater than	>
Less than	<
Greater than or equal	>=
Less than or egual	<=
Negation	!
And	&&
OR	||
XOR	^
Bitwise AND	&
Bitwise OR	|
Bitwise shifts	<< and >>

Variables: exercise

Detemine the actual size of the various data types on your system: g++ compiler of Ubuntu Linux installed on a 32 bit Virtual Machine.

- Hint: use the sizeof() function

```
#include <iostream>
using namespace std;
int main() {
        cout << "Size of int: " << sizeof(int) << endl;
        cout << "Size of short int: " << sizeof(short int) << endl;
        cout << "Size of long int: " << sizeof(long int) << endl;
        cout << "Size of float: " << sizeof(float) <<  endl;
        cout << "Size double: " << sizeof(double) << endl;
        cout << "Size of long double: " << sizeof(long double) <<  endl ;
}
```

Check what happens when

- variable values are outside of variable range
- an operation goes out-of-range value
  - Try various operators

```
#include <iostream>
using namespace std;
int main() {
   unsigned short int small, overflow;
   small = 65535; // allowed range 0 to 65535
   overflow = 2*small;
   cout << "the value of small " << small <<  endl ;
   cout << "the value of overflow " << overflow <<  endl;
}
```

Functions in C++

- Theoretically all the code could be written inside a single main() function ...
- However, for maintainability and manageability reasons, it is better to break it into smaller procedures. These are called functions.
- Implementing a C++ function involves the following elements:
  - Function definition
    - Consists of header and body
    - Body is the source code that makes up the function
    - Header specifies return value, name and parameter list
  - Function prototype
    - Functions must be declared before they are called
    - Prototypes usually specified in header files that are called via the #include statement
  - Function call
    - The statement that executes a function is called a function call
    - Function calls can be specified any time
    - Can be used in assignments
- Advanced topics: pass by reference or pointers

```
int sumup(int x)
{
 int sum, y = 5;
 sum = x + y;
 return sum;
}
```

```
int sumup(int );
```

```
int bignmber, inputnumber;
inputnumber = 12;
bigbumber = sumup ( inputnumber );
```

Functions: exercise

In this exercise, you're required to create a user-defined function to capture the program logic of the main program and call that function from main().

- Rewrite the following simple code by introducing a user-defined function:

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

int main()
{
 int first, second, larger;
 cout<<"enter the first number:" << endl;
 cin>>first;
 cout<<"enter the second number:" << endl ;
 cin>>second;

 // The program logic that can be turned into a function
 larger = second;

 if (first > second){
  larger= first;
 }
 // Printing the result
 cout << "The larger number is " << larger  << endl ;
}
```

Functions: exercise solution

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

int largernumber(int, int);

int main(){

 int first, second, larger;
 cout<<"enter the first number:" << endl;
 cin>>first;
 cout<<"enter the second number:" << endl;
 cin>>second;

 larger = largernumber (first, second);
 cout << "The larger number is " << larger << endl;
}

int largernumber(int l_first, int l_second){
 if (l_first > l_second)  return l_first;
 return l_second;
}
```

Variables scope: exercise

The program below will not compile because of scope errors. Investigate which variables are used out-of-scope and comment out the corresponding code lines.

```
#include <iostream>
using namespace std;
int globalScope = 0; //This is a global variable, visible everywhere.

void foo() {
  int fooScope = 1; //Only visible within foo function
  cout << "fooScope: " << fooScope << endl;
  cout << "localScope: " << localScope << endl;
}
int main() {
  cout << "globalScope: " << globalScope << endl;

  { //Any block declares a scope, even this useless one
    int localScope = 3;
    cout << "localScope: " << localScope << endl;
    foo();
    cout << "fooScope: " << fooScope << endl;
    int globalScope = 100; // variable hiding, very bad practice!
    cout << "globalScope: " << globalScope << endl;
  }
  cout << "localScope: " << localScope << endl;
  cout << "globalScope: " << globalScope << endl;
}
```

Variables scope: exercise solution

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

int globalScope = 0; //This is a global variable, visible everywhere.

void foo() {
  int fooScope = 1; //Only visible within foo function
  cout << "fooScope: " << fooScope << endl;
  //cout << "localScope: " << localScope << endl; //Error! localScope is only visible in main()
}

int main() {
  cout << "globalScope: " << globalScope << endl; //OK! Will print 0

  { //Any block declares a scope, even this useless one
    int localScope = 3;
    cout << "localScope: " << localScope << endl; //OK! Will print 3
    foo();
    //  cout << "fooScope: " << fooScope << endl; //Error! fooScope is out of scope
    int globalScope = 100; // variable hiding, very bad practice!
    cout << "globalScope: " << globalScope << endl; // Will print 100
  }

  //cout << "localScope: " << localScope << endl; //Error! localScope is out of scope
  cout << "globalScope: " << globalScope << endl;
}
```

Further reading

- In case you quickly need a C++ environment to test some simple code:
  - http://cpp.sh
- Standard C++ foundations:
  - https://isocpp.org/about
- About number representation:
  - http://www.cprogramming.com/tutorial/floating_point/understanding_floating_point_representation.html