Scientific Programming With C++

Jerry Ebalunode

http://support.hpedsi.uh.edu/

University of Houston
Houston, TX

UNIVERSITY of HOUSTON
DIVISION OF RESEARCH
HEWLETT PACKARD ENTERPRISE DATA SCIENCE INSTITUTE

Online references

- http://www.cplusplus.com/reference
- http://www.cplusplus.com/doc/tutorial
- And Google…..

First Access Your Account

- UHVPN required if outside UH network
-

```
ssh  username@mreb202.cacds.e.uh.edu
```

- Log into your accounts
  - Username or login = cougarnet ID
  - Password = **cougarnet password**

- You can connect directly from anywhere on UH network
- From Outside network use UH VPN
- For others use either
  - Code chef
    - https://www.codechef.com/
  - Xcode/Visual Studio/Kdevelop

Developing locally

- For others that prefer to run code locally or use either
  - Xcode/Visual Studio/Kdevelop/NetBeans
  - Online code testing
    - Code chef
      - https://www.codechef.com/

Getting Started

- Use the terminal to download intro2c++lab.zip file to your home directory
  - Run the following commands

```
##Upload intro2c++_lab.zip    ## to get tutorial package
cd


 cp   /project/dsi/tutorials/intro2c++_lab.zip      ~

unzip    intro2c++_lab.zip

cd  intro2c++
```

- Now, you can begin working with tutorial files on your terminal

C++ Programming Language

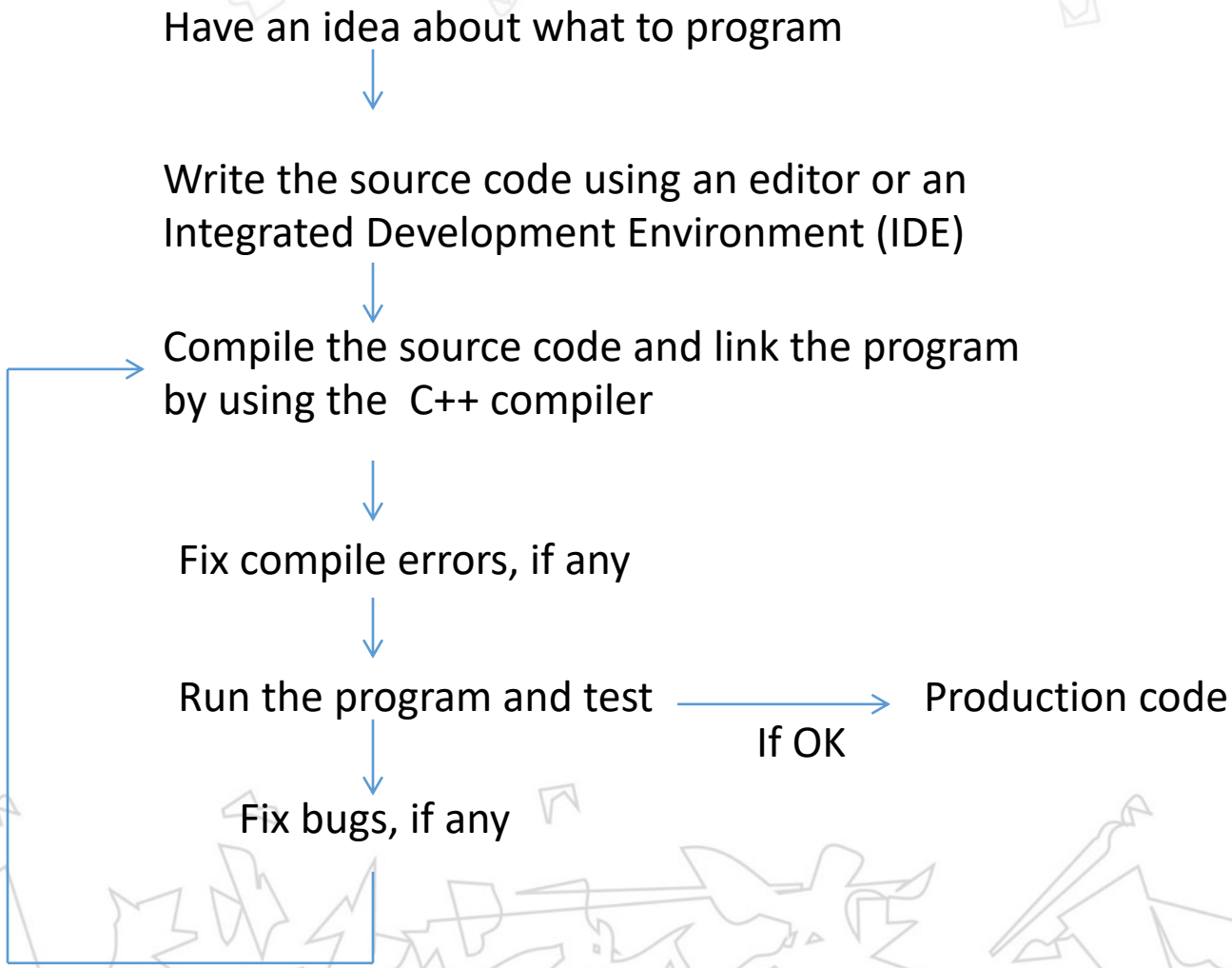
It is a superset of “C” programming language

Developed by Bjarne Stroustrup

C++ is a low-level programming language

- Object-Oriented Programming
  - Classes (algorithm + data)→ driven by data and methods (data)
  - Higher level of abstraction
- Generic Programming
  - Template programming → type independence
  - Reusable code
- Procedural programming
  - Series of computational steps leading to a desired goal

Mechanics of Creating a C++ Program



Writing the Source Code: MyFirstProgram.cpp

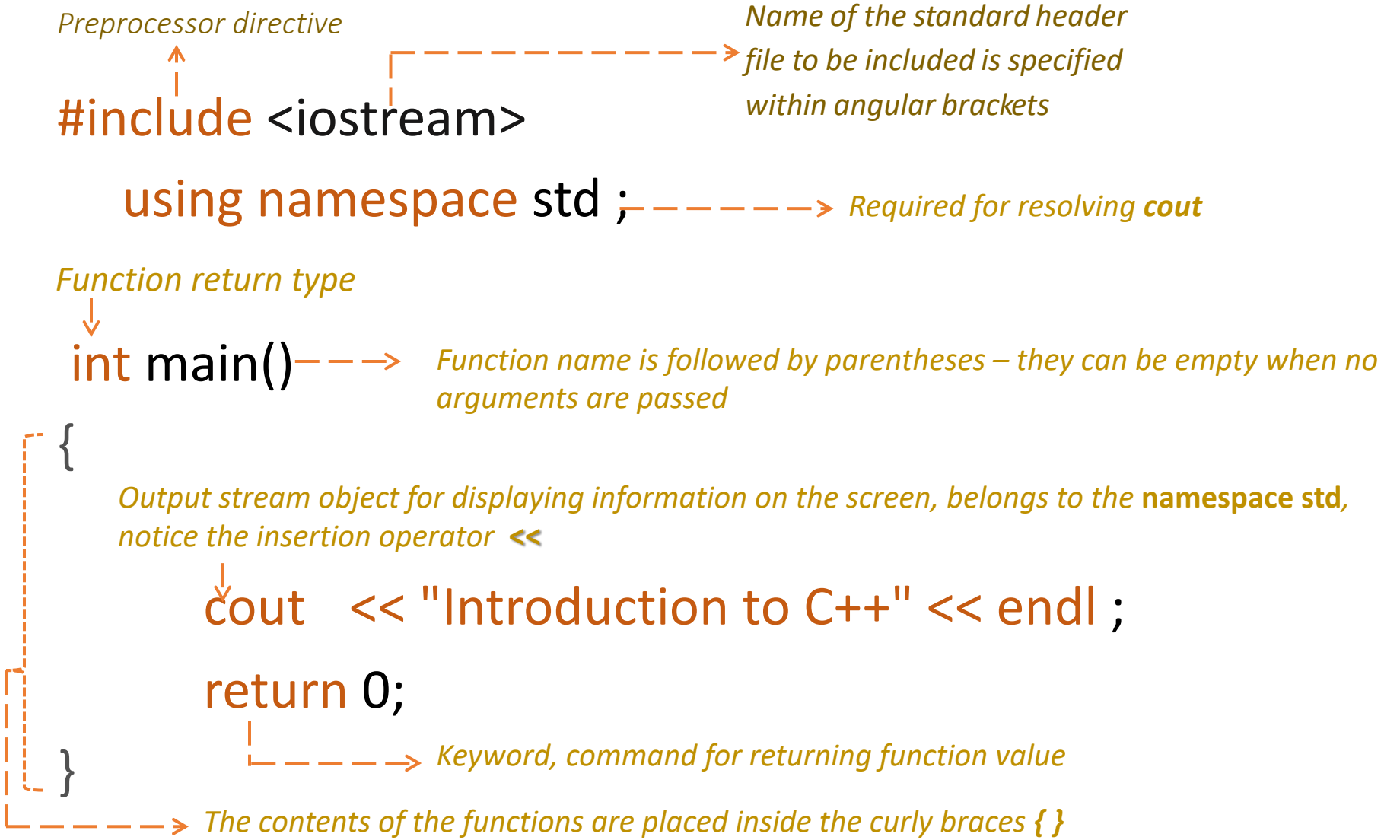
```
#include  <iostream>

using   namespace std ;

int       main()
{
          cout     << "Introduction to C++" << endl ;

          return 0;
}
```

A Closer Look At The Source Code: Myfirstprogram.Cpp



How To Create An Executable From Source Code

- Save source code file → Compile → Link → Run

- Save your program (source code) in a file having a “cpp” extension.
  - Example, `MyFirstProgram.cpp`
    - `note C++ file extensions include`
      - **`cc ,cpp, cxx, C`**
  - Compile and Link your code (linking is done automatically by the c++ compiler)

```
c++  -o  MyFirstProgram  MyFirstProgram.cpp
```

- Execute the program

```
./MyFirstProgram
```

- Repeat the steps above every time you fix an error in code!

Different Compilers

- Different commands for different compilers (*e.g.,* icpx for intel compiler and nvc++ for nvidia compilers)
  - GNU C++ compiler (**most popular and free**) also called g++

    c++ -o MyFirstCpp MyFirstCpp.cpp

    Or

    g++ -o MyFirstCpp MyFirstCpp.cpp

  - Intel C++ compiler

    icpx -o MyFirstCpp MyFirstCpp.cpp
  - PGI/Nvidia C++ compiler

    nvc++ -o MyFirstCpp MyFirstCpp.cpp
- To see a list of compiler options, their syntax, and a terse explanation, execute the compiler command with the

  c++ -help or --help option

  man c++

Note we would be using GNU "c++" compiler

Pop-Quiz 1
(add the missing components)

```
???   <iostream>
using namespace  std;
 int  main ( )
?
       cout  << "Introduction to C++“ <<  endl;

       cout  << "Enjoy the Quiz“         << endl;

        return     0;
?
```

Warnings, Errors and Bugs

- Compile-time warnings
  - Diagnostic messages : unused variables, un/intended type conversions, unknown pragmas etc
  - Compile still gets the code compiled
- Compile-time errors
  - Typographical errors: cuot , $include
  - Syntax errors: unknown data types, functions, undefined types
  - Compiler fails to compile the code into an object
- Link-time errors
  - Missing library files
  - Compiler fails to compile the code into an executable
- Run-time errors
  - Null pointer assignment
- Bugs
  - Unintentional functionality

Find the Error: myError.cpp

```
#include  < iostream >
using  namespace  std;
int  main()
{
        cuot <<"Find the error"<<
endl
      retrun 0;

}
```

Error Message (compile-time error)

```
c++ -o myError myError.cpp
myError.cpp:1:22: iostream :No such
file or directory
myError.cpp: In function `int main()':
myError.cpp:6: error: `cuot' was not
declared in this scope
myError.cpp:7: error: expected `;'
before "retrun"
myError.cpp:7: error: `retrun' was not
declared in this scope
```

Comments and New Line: rules.cpp

```
/*    We  use  comments to describe what the code is doing
*             rules.cpp  should print one statement per line
*             this is a multi-line comment
*/
#include <iostream>
 using      namespace std;
 int main()
{
             cout << "Braces come in pairs.";
             cout << “Comments come in pairs.";
             cout << "All statements end with semicolon.";
             cout << "Every program has a main function.";

             return 0;
}
```

Output of rules.cpp

```
Braces come in pairs.Comments come
in pairs.All statements end with a
semicolon.Every program must have  a
main function.
```

Output looks odd! We want to see a new line of text for every *"cout"* statement.

Comments and New Line: rules2.cpp

```
/*
*   use  comments to describe what the code is doing
*                   rules.cpp  should print one statement per line
•    this is a multi-line comment
*/


#include  <iostream>
 using namespace std;
 int main()
{
        /*notice  the  usage  of  endl
        cout  <<  "Braces          come in pairs." << endl;
        cout  <<  "Comments come in pairs." << endl;
        /*  \n  can  also  be   used                  */
        cout  <<  "All statements end with semicolon.\n";
        cout  <<  "Every program has a main function." << endl;
        return 0;
}
//this  is how single line comments are specified
```

Output of rules2.cpp

```
Braces come in pairs.
Comments come in pairs.
All statements end with a semicolon.
Every program must have a main function.
```

- The output looks better now! with “endl” c++ keyword.
- endl inserts a newline and flushes the stream.
- Also used when writing newline to files

Variables

- Information-storage places
  - A memory location to store data for a program
- Compiler makes room for them in the computer's memory
- Can contain *string, characters, numbers etc.*
- Their values can change during program execution
- All variables should be declared before they are used and should have a data type associated with them

Data Types

- Data types tell about the kind of data that a variable holds
- Categories of data types are:
  - **Built-in:** `char double float long short signed unsigned void int`
  - **User-defined:** `struct class`
  - **Derived:** `array or function pointer`
- We have already seen an example code in which an integer data type was used to return value from a function: `int main()`
- Compiler-dependent range of values associated with each type
  - Example: a **signed integer** can have a value in the range

    -32,768 to **32,767** on a 16-bit computer or

    -2,147,483,647 to **2,147,483,647** on a 32-bit computer

    -2^{63} to $2^{63} - 1$ for 64 bit computer

Data Types: Simple Types (1 of 2)

Display 1.2 **Simple Types**

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
`short` (also called `short int`)	2 bytes	−32,768 to 32,767	Not applicable
`int`	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
`long` (also called `long int`)	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
`float`	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
`double`	8 bytes	approximately 10^{-308} to 10^{308}	15 digits

Data Types: Simple Types (2 of 2)

`long double`	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
`char`	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
`bool`	1 byte	`true`, `false`	Not applicable

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types `float`, `double`, and `long double` are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

Variable Names and Variable Declaration

- Each variable needs a name (or an identifier) that distinguishes it from other variables
- A valid variable name is a sequence of one or more alphabets, digits or underscore characters
- Keywords cannot be used as variable names
- Declaration is a statement that defines a variable
- Variable declaration includes the specification of data type and an identifier. Example:

```
int  number1;
float  number2;
```

- Multiple variables can be declared in the same statement

```
int x, y, z;
```

- Variables can be signed or unsigned
- Signed types can represent both positive and negative values, whereas unsigned types can only represent positive values

```
signed double temperature;
```

New C++11 Types

- auto
  - Deduces the type of the variable based on the expression on the right side of the assignment statement

```
auto x = expression;
```

- decltype
  - Determines the type of the expression. In the example below, x*3.5 is a double so y is declared as a double.

```
decltype(x*3.5) y;
```

Reading Keyboard Input: readInput1.cpp

```
#include <iostream>
using namespace std;
int main()
{

  float temperature1;
  float temperature2;
  float        average;
  cout  <<  "Enter the first temperature reading in Fahrenheit:  ";

    cin >>  temperature1;

  cout  <<  "Enter the second temperature reading in Fahrenheit:  ";
  cin >> temperature2;

  average = (temperature1 + temperature2)/2.0;
  cout  <<  "The average temperature is: " <<  average <<" F"<<  endl
  cout  <<  "The average temperature in Kelvin is: " <<  (5/9.0 * (average - 32)) + 273 <<" K"<<  endl;
  return  0;
}
```

variable declarations. It provides storage for the information you enter or **compute**.

input statement that causes the program to wait till the input is entered

```
Enter the first temperature reading in
Fahrenheit:  49
Enter the second temperature reading in
Fahrenheit:  20
The average temperature is: 34.5 F
The average temperature in Kelvin is: 274.389 K
```

Notes:

cin is the predefined object in C++ that corresponds to the standard input stream and >> operator is extraction operator

Variable Initialization

- A variable can be assigned value at the time of its declaration by using assignment operator or by constructor initialization

```
int       x   = 10;
double pi =3.14159265358979;
char      x   =        'a';
string name =“John Doe”
```

- Variables can also be assigned values using C++ objects as in:

```
cin >> myName;
```

Scope of Variables

- A variable can be either of global or local scope
  - Global variables are defined outside all functions and they can be accessed and used by all functions in a program file
  - A local variable can be accessed only by the function in which it's created
- A local variable can be further qualified as **static**, in which case, it remains in existence rather than coming and going each time a function is called

```
static  double   pi  = 3.14159265358979;
```

- A register type of variable is placed in the machine registers for faster access – compilers can ignore this advice

```
register int x;
```

Assigning Data

- Initializing data in declaration statement
  - Results "undefined" if you don't!

    int myValue = 0;
- Assigning data during execution
  - Lvalues (left-side) & Rvalues (right-side)
    - Lvalues must be variables
    - Rvalues can be any expression
    - Example:

      distance = rate * time;

      Lvalue: "distance"
      Rvalue: "rate * time"

Assigning Data: Shorthand Notations

EXAMPLE	EQUIVALENT TO
`count += 2;`	`count = count + 2;`
`total -= discount;`	`total = total - discount;`
`bonus *= 2;`	`bonus = bonus * 2;`
`time /= rushFactor;`	`time = time/rushFactor;`
`change %= 100;`	`change = change % 100;`
`amount *= cnt1 + cnt2;`	`amount = amount * (cnt1 + cnt2);`

Data Assignment Rules

- Compatibility of Data Assignments
  - Type mismatches
    - General Rule: Cannot place value of one type into variable of another type
  - int intVar = 2.99;// 2 is assigned to intVar!
    - Only integer part "fits", so that's all that goes
    - Called "implicit" or "automatic type conversion"
  - Literals
    - 2, 5.75, "Z", "Hello World"
    - Considered "constants": can't change in program

Literal Data

- Literals
  - Examples:

```
2                   // Literal constant int
5.75                // Literal constant double
"Z"                 // Literal constant char
"Hello World"       // Literal constant string
```

- Cannot change values during execution
- Called "literals" because you "literally typed" them in your program!

Constants and Constant Expressions

- Naming your constants
  - Literal constants are "OK", but provide little meaning
  - e.g., seeing 24 in a program, tells nothing about what it represents
- Use named constants instead
- The value of a constant never changes

```
const   double      e =   2.71828182;
```

- **Useful for protecting the value of a variable like global parameters**

```
const double pi= 3.14159265358979
```

Some Operators Common in C and C++

Arithmetic: +, -, /, *, %, ++, --, =

Relational: a == b, a != b, a > b, a < b, a >= b, a <= b

Logical: !a, a && b, a || b

Member and Pointer: a[], *a, &a, a->b, a.b

Others: sizeof

Bitwise: ~a, a&b, a|b, a^b, a<<b, a>>b

More about operators and precedence:

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

Parentheses and Precedence: checkParentheses.cpp

```
#include <iostream>
using namespace std;
int main()
{
        int total;
        //multiplication has higher precedence than subtraction
        total=100-25*2;
        cout << "The total is: " << total << endl;

        //parentheses make a lot of difference!
        total=(100-25)*2;
        cout << "The total is: " << total << endl;
        return 0;
}
```

Output:
The total is: $50
The total is: $150

Operators in C++ But Not in C

- Scope resolution operator **::**
- Pointer-to-member declarator **::***
- Pointer-to-member operator **->***
- Pointer-to-member operator **.***
- Memory Release operator **delete**
- Line feed operator **endl**
- Memory allocation operator **new**
- Field width operator **setw**
- Insertion operator **<<**
- Extraction operator **>>**

Formatting Output

- Formatting numeric values for output
  - Values may not display as you'd expect!
    cout << "The price is $" << price << endl;
    - If price (declared double) has value 78.5, you might get:
      - The price is $78.500000 or:
      - The price is $78.5
- We must explicitly tell C++ how to output numbers in our programs!

Formatting Numbers

- "Magic Formula" to force decimal sizes:
  ```
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(2);
  ```
- These stmts force all future cout'ed values:
  - To have exactly two digits after the decimal place

  Example:
  ```
  cout << "The price is $" << price << endl;
  ```
  Now results in the following:
  The price is $78.50
- Can modify precision "as you go" as well!

Error Output

- Output with cerr
  - cerr works same as cout
  - Provides mechanism for distinguishing between regular output and error output
- Re-direct output streams
  - Most systems allow cout and cerr to be "redirected" to other devices
    - e.g., line printer, output file, error console, etc.

Libraries

- C++ Standard Libraries
- #include <Library_Name>
  - Directive to "add" contents of library file to your program
  - Called "preprocessor directive"
    - Executes before compiler, and simply "copies" library file into your program file
- C++ has many libraries
  - input/output, math, strings, etc.

Namespaces

- Namespaces defined:
  - Collection of name definitions
- For now: We are interested in namespace "std"
  - Has all standard library definitions we need
- Examples:

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

  - Includes entire standard library of name definitions
-

```
#include <iostream>
using std::cin;
using std::cout;
```

  - Can specify just the objects we want

Summary 1

- C++ is case-sensitive
- Use meaningful names
  - For variables and constants
- Variables must be declared before use
  - Should also be initialized
- Use care in numeric manipulation
  - Precision, parentheses, order of operations
- #include C++ libraries as needed

Summary 2

- Object cout
  - Used for console output
- Object cin
  - Used for console input
- Object cerr
  - Used for error messages
- Use comments to aid understanding of your program
  - Do not overcomment