CSCI 123 Introduction to Programming Concepts in C++

Brad Rippe

C++ Basics
C++ layout

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

int main() {
    statement1;
    ...
    statement;
    return 0;
}
```

Include directive

Every program must have `main`

Statements are declared on each line in sequential order

Terminates the program
Storage Containers
Variables

int someInt = 0;
double someDub = 0.0;
cout << someInt << endl;
cout << someDub << endl;

0
0.0
cin/cout

• **cin**
  - Sends data from the keyboard
  - `>>` - extraction operator
  - **Extraction skips blanks and line breaks looking for data**

• **cout**
  - Sends output to the screen
  - `<<` - insertion operator

• `\n` – newline - used with literal text
• `endl;` - newline – used when `\n` can’t be used
Undefined Variables

```cpp
int someInt;
double someDub;

cout << someInt << endl;
cout << someDub << endl;
```
Undefined Variables

Microsoft Visual C++ Debug Library

Debug Error!

Program: d:\projects\csci123\Oreilly\debug\undefinedVars.exe
Module: d:\projects\csci123\Oreilly\debug\undefinedVars.exe
File:

Run-Time Check Failure #3 - The variable 'someInt' is being used without being defined.

(Press Retry to debug the application)
Assignments - Memory

Some variables are assigned an address:

```plaintext
someInt = 0;
someDub = 0.0;
```

The memory table shows the assigned addresses:

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0.0</td>
</tr>
<tr>
<td>as@#$%^%$asf</td>
</tr>
</tbody>
</table>

The memory table for the variables is:

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>%$*&amp;^$dsasd</td>
</tr>
<tr>
<td><em>&amp;(*</em>&amp;(_(*&amp;</td>
</tr>
<tr>
<td>as@#$%^%$asf</td>
</tr>
</tbody>
</table>
Identifiers

- Names for a variable
  - Part of the human readability
- Must start with alpha character or “_”
- Other characters must be alpha, numeric, or an “_”
- Case-sensitive
  - someNum is not somenum
- Should be descriptive of the data they contain
- CANNOT BE A KEYWORD (Reserved Word)

- Course uses the following style
  - Lowercase letter for the first word
  - All subsequent words are capitalized
  - Can’t be a C++ reserved word

asteroid Location
# Keywords

<table>
<thead>
<tr>
<th>asm</th>
<th>for</th>
<th>static</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>float</td>
<td>static_cast</td>
</tr>
<tr>
<td>bool</td>
<td>friend</td>
<td>struct</td>
</tr>
<tr>
<td>break</td>
<td>goto</td>
<td>switch</td>
</tr>
<tr>
<td>case</td>
<td>if</td>
<td>template</td>
</tr>
<tr>
<td>catch</td>
<td>inline</td>
<td>this</td>
</tr>
<tr>
<td>char</td>
<td>int</td>
<td>throw</td>
</tr>
<tr>
<td>class</td>
<td>long</td>
<td>true</td>
</tr>
<tr>
<td>const</td>
<td>mutable</td>
<td>try</td>
</tr>
<tr>
<td>const_cast</td>
<td>namespace</td>
<td>typedef</td>
</tr>
</tbody>
</table>

Check Page 1002
Appendix 1 for more
Declarations are usually declared right before their use or at the beginning of `main`
Assignment

• Literal Values
  
someInt = 27;
someDub = 33.33;

• Expressions
  
someInt = 27 + 23;
someDub = 5.55 * 1.0;
someDub = 5.55 * someInt;

• Values of other Variables
  
someInt = someOtherInt;
someDub = someOtherDub;
someInt = someInt2 + someInt3;
someDub = someDub2 / someDub3;
Initialization in Declarations

```c
int someInt = 0, someOtherInt = 2, myInt = 87;
double myDub = 2.2, otherDub = 4.55;
```

Not required by C++, just my own personal taste
## Assignments/Definitions

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>someInt</code></td>
<td><code>0</code></td>
</tr>
<tr>
<td>int</td>
<td><code>someOtherInt</code></td>
<td><code>22</code></td>
</tr>
<tr>
<td>int</td>
<td><code>myInt</code></td>
<td><code>87</code></td>
</tr>
<tr>
<td>double</td>
<td><code>myDub</code></td>
<td><code>2.2</code></td>
</tr>
<tr>
<td>double</td>
<td><code>otherDub</code></td>
<td><code>4.55</code></td>
</tr>
</tbody>
</table>
Input and Output Streams

Program

```cpp
cin >>
```

Program

```cpp
cout <<
```
Predefined Input and Output

- Provided by iostream library
- `#include` directive

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

This is the include directive
Output String Literals

- Enclosed in double quotes
- Uses the insertion operator `<<`

```cpp
cout << "Hello Class\n"
    << "Do you prefer Coke or Pespi?\n"
    << "What was the score of the Lakers Game? "
    << "Heat Game?\n";
```
Escape Sequences

Text is defined in double quotes “TEXT”
Special Characters
“\n” – newline
“\v” – vertical tab
“\r” – carriage return
“\a” – alert bell
“\”” – double quote
“\’” – single quote
“\t” – tab
“\b” – backspace
“\\” – back slash
**Newlines**

- “\n” - is usually used when string literals are output

  ```
  cout << "I love this stuff!\nIsn’t this cool?\n";
  cout << "The current lotto numbers are:\n"
  << lottoNums << endl;
  ```

- `endl` – sends a newline to the output stream and flushes the output buffer.
• Assigning values from the keyboard

```cpp
cin >> numOfHomesOnMarket
   >> avgSellingPrice;
```

>283 634000.99[ENTER]

or

>283[ENTER]
>634000.33[ENTER]
Integer Data Types

- **int**
  - 32 bits
  - Contains whole numbers
  - Ranges from -2,147,483,648 to 2,147,483,647

- **double**
  - 64 bits
  - Fraction – usually around 14 decimal places
  - Ranges from $10^{-308}$ to $10^{308}$

- **float**
  - 32 bits
  - Fraction
  - Ranges from $10^{-38}$ to $10^{38}$

*Depends on implementation*
double radius = 6.66666667;
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout << radius << endl;
## Integer Data Types

- **char**
  - 8 bits
  - ASCII character
  - Alphanumeric and punctuation

- **short**
  - 16 bits
  - Ranges from -32767 to 32767

- **long**
  - 32 bits
  - Fraction
  - Ranges from ~ -2 billion to ~ 2 billion

<table>
<thead>
<tr>
<th>addr</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>10011100</td>
</tr>
<tr>
<td>68</td>
<td>11000000</td>
</tr>
<tr>
<td>69</td>
<td>10000000</td>
</tr>
<tr>
<td>70</td>
<td>00011100</td>
</tr>
</tbody>
</table>
More Data Types

- **bool**
  - 8 bits
  - either *true* or *false*

- **string**
  - Class type
  - Defined in double quotes
  - Must include the `<string>` library
  - This is **not** a primitive type
The following actions are possible but generally not recommended!

Values of type bool can be assigned to int variables
- `true` is stored as 1 (any positive integer)
- `false` is stored as 0

Values of type int can be assigned to bool variables
- Any non-zero integer is stored as `true`
- Zero is stored as `false`
Variable Declarations & Assignments

• Declarations
  type nameOfTheVar;

• Assignments
  nameOfTheVar = someValue;

• One Statement
  type nameOfTheVar = someValue;
Arithmetic Operators

- Can be used for literal values and variables (in order of precedence)
  - Parentheses ( )
  - Multiplication *
  - Division /
  - Modulus %
  - Addition +
  - Subtraction –

- Same precedence as algebraic operations
  - Please Excuse My Dear Aunt Sally

- Operands can be number or variable

- Calculating Right Circular Cylinder \( \pi r^2 h \)
  - PIE*(r * r)*h
Operator Precedence

• If you were a computer, what would you give as a result for:

\[
y = 2 \times 5 \times 5 + 3 \times 5 + 7;
\]
Step 1. $y = 2 \times 5 \times 5 + 3 \times 5 + 7$:

2 * 5 is 10

(Leftmost multiplication)

Step 2. $y = 10 \times 5 + 3 \times 5 + 7$:

10 * 5 is 50

(Leftmost multiplication)

Step 3. $y = 50 + 3 \times 5 + 7$:

3 * 5 is 15

(Multiplication before addition)

Step 4. $y = 50 + 15 + 7$:

50 + 15 is 65

(Leftmost addition)

Step 5. $y = 65 + 7$:

65 + 7 is 72

(Last addition)

Step 6. $y = 72$:

(Last operation—assignment)
# Short Hand Operators

foo = foo op bar  \[\rightarrow\]  foo op= bar

<table>
<thead>
<tr>
<th>Operator</th>
<th>Expression</th>
<th>Shorthand</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>count = count + 2;</td>
<td>count += 2;</td>
</tr>
<tr>
<td>-=</td>
<td>count = count - 2;</td>
<td>count -= 2;</td>
</tr>
<tr>
<td>*=</td>
<td>count = count * 2;</td>
<td>count *= 2;</td>
</tr>
<tr>
<td>/=</td>
<td>count = count / 2;</td>
<td>count /= 2;</td>
</tr>
<tr>
<td>%=</td>
<td>count = count % 2;</td>
<td>count %= 2;</td>
</tr>
</tbody>
</table>
Some Rules

• If both operands are type int
  ○ Then the result is an int type

  23 * 38 = integer result
  23 / 38 = integer result

• If one of the operands is a double
  ○ Then the result is a double type

  23.56 * 38 = double result
  23.56 / 38 = double result
  5 / 2.0 = double result
Flow of Control

cout << "We can output "
    << "variables values\n";
cout << "12/4 = " << 12 / 4 << endl;
cout << "4+12 = " << 4 + 12 << endl;
cout << "18%4 = " << 18 % 4 << endl;
Sequential Execution

1. Add studentGrade to total
2. Add 1 to counter
If statements

```java
if (Boolean Expression) {
    statement1;
    statement2;
    ...
}
```

```java
if (Boolean Expression) {
    statement1;
}
```

Note: My style of brackets is different from your book. Use my style.
if structure diagram

stuGrade >= 60

true

false

Print “passed”
### Comparisons Operators

<table>
<thead>
<tr>
<th>Math Symbol</th>
<th>English</th>
<th>C++ Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
<td>$x == y$</td>
</tr>
<tr>
<td>≠</td>
<td>Not equal to</td>
<td>$x != y$</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>$x &lt; y$</td>
</tr>
<tr>
<td>≤</td>
<td>Less than equal</td>
<td>$x \leq y$</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>$x &gt; y$</td>
</tr>
<tr>
<td>≥</td>
<td>Greater than equal</td>
<td>$x \geq y$</td>
</tr>
<tr>
<td>!</td>
<td>Negation</td>
<td>$(x &lt; y)$</td>
</tr>
</tbody>
</table>
“And” and “Or” Operators

• The “and” operator &&
  o Allow multiple boolean expressions

    if ((fuel >= 30) && (milesTraveled > 5000)) {
        thrust; // assume this is valid C++
    }

• The “or” operator ||
  o Allow multiple boolean expressions

    if ((fuel >= 30) || (milesTraveled < 5000)) {
        thrust;
    }
if / else structure

Syntax:

```c
if (boolean expression) {
  statement1;
  statement2;
  ...
} else {
  statement1;
  statement2;
  ...
}
```
if / else structure diagram

```
if stuGrade >= 60
    Print "Passed"
else
    Print "Failed"
```
Loops

• Repetition statement (*iteration statement*)
• Set of statements is executed as long as the condition remains true

• Pseudocode
  1. While there are more items on my shopping list
  2. Purchase next item and cross it off my list
  3. countNumItems++
  4. cout << itemRemoved << endl;
while structure

- Syntax:

```cpp
int someVar = 234;
while (boolean expression) {
    ... loop body ...
    statement1;
    statement2;
}
cout << someVar << endl;
```
while structure diagram

false

product < 1000

true

loop body

...  

product *= 2;
Do-while structure

- Syntax:

```java
do {
    ... loop body ...
    statement1;
    statement2;
} while (boolean expression);
```
do-while structure diagram

```
loop body
...
product *= 2;
```

```
product < 1000
```

false → true

```
do-
```
Infinite loops

• Try to avoid this situation at all costs
• The end result is abnormal termination of your application (control+c)
• THIS IS A BAD THING!!!
### Increment and Decrement Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>Prefix increment</td>
<td>++a</td>
</tr>
<tr>
<td>++</td>
<td>Postfix increment</td>
<td>a++</td>
</tr>
<tr>
<td>--</td>
<td>Prefix decrement</td>
<td>--b</td>
</tr>
<tr>
<td>--</td>
<td>Postfix decrement</td>
<td>b--</td>
</tr>
</tbody>
</table>
Comments

• Single line Comments
  o // - single line comment
    // Name: Brad Rippe
    // File: Assignment23.cpp
    // Date: 1/21/07

• Multiple line Comments
  o /* ...*/
    /*
     Name: Brad Rippe
     File: Assignment23.cpp
     Date: 1/21/07
    */

    /**
     * @assignment 1
     * Multi-line comment
     */
Constants

- Program cannot modify the variable
- Uses the keyword “const” in the declaration
- Known as the const modifier
- Identifier is capitalized
- Words are separated by “_”

```cpp
const int NUMBER_OF_ITEMS = 500;
const double PIE = 3.14159;
const string CON_DRIVER = "SQLServerDriver";
```
Programming Style

Code conventions are important to programmers for a number of reasons:

• 80% of the lifetime cost of a piece of software goes to maintenance.

• Hardly any software is maintained for its whole life by the original author.

• Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

• If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.
Course Style Guide


Following the course style guide is a MUST.

CSCI 123 – Introduction to Programming Concepts in C++ - Style Guide
Resources

• Course Web Site
  o http://staffwww.fullcoll.edu/brippe/csci123

• Course Style Guide
  o http://staffwww.fullcoll.edu/brippe/csci123/style.aspx

• Course Syllabus
  o http://staffwww.fullcoll.edu/brippe/csci123/syllabus.aspx

• Course Schedule (tentative)
  o http://staffwww.fullcoll.edu/brippe/csci123/schedule.aspx