Goals for This Lecture:

• Introduction to C++ part II
Constants

- Constants, the C++ equivalent of FORTRAN parameters can be declared by adding the constant modifier to the declaration statement.

- Examples:

  ```
  const int COUNTER = 2 ;
  const char KEY= 'A' ;
  const float VELOCITY=2.0 ;
  const double GRAVITY=6.672e-8;
  ```

- Programming style tip: Use uppercase names to distinguish constants in C++ code.
Arithmetic Operators

- Most arithmetic operators in FORTRAN are also the same in C++

<table>
<thead>
<tr>
<th>Purpose</th>
<th>C++ Operator</th>
<th>FORTRAN Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>pow(,)</td>
<td>**</td>
</tr>
<tr>
<td>Modulo or remainder</td>
<td>%</td>
<td>mod(,)</td>
</tr>
</tbody>
</table>
More Assignment Statements

• A shorthand notation exists for combining the assignment with an arithmetic operator

• Form:

  Variable operator = Expression;

  Which is equivalent to

  Variable = Variable operator Expression ;

• Examples:

  i += 2
  Which is equivalent to
  i = i+2;

  i *= 2
  Which is equivalent to
  i = i*2;
# Integer and Floating-point division

- Integer and floating point division works like it does in FORTRAN

<table>
<thead>
<tr>
<th>Expression Form</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>int/double</td>
<td>double</td>
</tr>
<tr>
<td>double/int</td>
<td>double</td>
</tr>
<tr>
<td>double/double</td>
<td>double</td>
</tr>
<tr>
<td>int/int</td>
<td>int</td>
</tr>
</tbody>
</table>
Type Casting

• A Type cast is a way to change on kind of a variable to another
• A cast is like a function for converting one type of variable o another
• We will focus on what are called static_casts
  – These are casts that return a value but which do not change the value of the variable or expression
• Examples:
  ```cpp
  int counter = static_cast<int>(3.14);
  double two = static_cast<double>(2);
  char alpha_numeric = static_cast<char>(1) ;
  ```
• There are other types of casts (const_cast, dynamic_cast, reinterpret_cast) which we will not discuss at this time.
Using Type Casting

• The form for type casting is:
  \[ \text{static\_cast}<\text{type}>(\text{expression}); \]

• The \textit{type} is what you want the expression to become

• How the conversion is carried out is not always obvious

• Example:
  \[ \text{bool prop = static\_cast}<\text{bool}>('M'); \]

• Is ‘M’ converted to \texttt{true} or \texttt{false} (the two permissible values of a bool variable)?

• You’ll have to write a program or go look this up in the documentation to find out…
Old Style Type Casting

- An older form of static casting is:
  \( \text{type(expression)} \)

- The \textit{type} is a function that converts the expression to a specific type

- Examples:
  \( \text{bool(‘M’) } \)
  \( \text{int(3.14)} \)
  \( \text{double(2)} \)

- We will use the new style static\_cast to avoid confusion with declarations
Increment and Decrement operators

- Both C++ and C possess increment (++) and decrement (--) operators.
- Example:
  
```
  i++;
  is equivalent to
  i = i+1 ;
  while
  i--;
  is equivalent to
  i = i-1 ;
```

- Essentially these operators are a shorthand notation for a commonly used combination of assignment and numeric operations.
More on ++ and --

• Both the increment and decrement operators can be used before or after a variable

• Example:

```c
int i=2, j;
j = i++;  // equivalent to i = (j = i)+1;
```

(j has a value of 2, i has a value of 3)

while

```c
int i=2, j;
j = ++i;  // equivalent to j = (i = i+1);
```

(j has a value of 3, i has a value of 3)

• If the operator precedes the variable the variable is incremented before its value is returned, if it follows the variable the value of the variable is returned before it is incremented.

• The increment and decrement operators cannot be used on anything other than a variable
The dangers of increment and decrement operators

• The order of evaluation of many operators is not guaranteed

• This presents a danger of side effects when increment and decrement operators are employed in expressions where the variable appears in multiple places.

• Example:
  ```
  int i=2, j;
  j = i*i + i++;
  ```

• The result is undetermined and may vary from compiler to compiler

• Avoid using these increment and decrement operators in this way at all costs
Input & Output to STDIN & STDOUT

• I/O to STDIN & STDOUT is performed with the objects `cin` & `cout`.

• These are defined in the library iostream.

• Your code can access this library by adding the following two lines at the top of the file:

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

- Values of variables and text can be sent to STDOUT using the `cout` object

- `cout` can output any number of items as long as they are each preceded by the `<<` operator

- Example:
  ```cpp
  cout << "x, y = " << x << y << "\n" ;
  ```

- Unlike FORTRAN, unless you explicitly send a newline character output continues on the same line

- You can also use `endl` in place of `"\n"`
Programming Style Tip

• Output a final `endl` or `"\n"` at the end of your program (if you did any output)

• This guarantees that output buffers are emptied when the program terminates

• If you don’t send a newline through `cout` the output from your program may not appear until the next program executes
Formatting floating point output using **setf** and **precision**

- Formatting for floating point values can be accomplished using the **setf** and **precision** methods on the **cout** object.

  **Example:**
  ```cpp
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(3);
  cout << " x, y = " << x << y << "\n;"
  ```

  - This turns on fixed point format, turns on the decimal point, and sets the number of digits trailing the decimal point to three.
Input using `cin`

- Values of variables can be read in from STDIN using the `cin` object.

- `cin` can input any number of items as long as they are each preceded by the `>>` operator.

- Example:
  ```
  cin >> x >> y ;
  ```

- Input is not read until the return key is typed
  - This allows user to correct input.

- Values must be separated by spaces or line break signaled by returns.

- You can also use `endl` in place of ``
``
Reading Assignment

– Do the “self-test” exercises in Chapter 1 of Savitch

– Read Section 1.5 of Savitch

– Read Chapter 2 of Savitch