CISC 1600/1610
Computer Science I

Programming in C++
Professor Daniel Leeds
dleeds@fordham.edu
JMH 328A

Introduction to programming with C++

Learn
• Fundamental programming concepts
• Key techniques
• Basic C++ facilities

By the end of the course, you will be able to:
• Write small C++ programs
• Read much larger programs
• Learn the basics of many other languages
• Proceed to advanced C++ courses

Requirements

• Lectures and lab sessions
• Labs assignments – roughly 5 across semester
• Final project
• Exams – 1 midterm, 1 final

• Academic integrity – discuss assignments with your classmates, but DO NOT copy assignments

How to succeed in class

Ask questions
• In class
• In office hours, tutor room
• Study together and discuss assignments with each other (without plagiarizing!)

Textbook
• Read and re-read the material
• Complete practice problems

Start coding and studying early

Course textbook

Problem Solving
With C++
Ninth Edition
Walter Savitch

Course website

http://storm.cis.fordham.edu/leeds/cisc1600

Go online for
• Lecture slides
• Assignments
• Course materials/handouts
• Announcements
Instructor

Prof. Daniel Leeds
dleeds@fordham.edu
Office hours: Tues 2-3p, Wed 12-1p
Office: JMH 328A

A program provides a computer with a set of simple instructions to achieve a goal

Programs are everywhere

On your computer:
- Web browser
  – Request and display information from distant sites
- Word processor
  – Record text, change appearance, save to disk
- Music player
  – Organize mp3’s, select time in song, play, stop

Programs are everywhere

In the dining hall:
- Cashier
  – Compute price of food purchase, charge payment to account, (if pay cash: compute change)
- HVAC
  – Monitor temperature, adjust A/C or heating
- Electronic signs
  – Display menus and prices, load and display university news

Programs are everywhere

In humans:
- Sports
  – When to run, where to run; when to pass, who to pass to; when to shoot
- The brain
  – Neurons working together to combine information about an image to recognize a dog or a car

Computer system structure

Central processing unit (CPU) – performs all the instructions
Memory – stores data and instructions for CPU
Input – collects information from the world
Output – provides information to the world
C++ – high-level language

- High-level language
  - Uses words to describe instructions
  - More intuitive to people
  - Computer-independent
- Machine-language
  - Uses binary to describe instructions
  - Less intuitive to people
  - Computer-dependent

Why C++?

Some programming history:
- C++ developed as improvement on C
- C developed as improvement on B
- B developed as improvement on ...
- BCPL – Basic Computer Programming Language
- Various languages before BCPL – ADA, COBOL, FORTRAN

Why C++?

- Popular modern programming language
- In use since 1980’s
- Similar structure to many/most other popular languages (Java, C#, Perl, Python)

Course outline

- Programming basics, input/output, arithmetic
- Conditional statements
- Loops
- Modularity – functions
- Complex data – arrays, vectors strings, and classes

Throughout the semester:
- Proper programming style

Programming basics

- Program structure and components
- Output text
- Variables
- Input information
- Perform arithmetic
- Type safety

Our first program: “Hello world!”

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

int main()
{
    // Begin main function
    cout << "Hello world!\n";  // output "Hello world!"
    return 0;                /* indicate successful
                              program completion */
} // End main function
```

```bash
> ./myProgram
Hello world!
>
```
The components of “Hello world!”

- Comments  //, /* */
- main function
- Preprocessor directives #include

---

Using comments

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

int main()
{
    // Begin main function
    cout << "Hello world!\n"; // output "Hello world!"
    return 0; /* indicate successful program completion */
}

// End main function
```

- Explain programs to other programmers
- Ignored by compiler
- Syntax:
  // single line comment
  /* multi-line comment */

---

Preprocessor directives

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

int main()
{
    // Begin main function
    cout << "Hello world!\n"; // output "Hello world!"
    return 0; /* indicate successful program completion */
}

// End main function
```

- Lines beginning with #
- Executed before compiling the program

---

main function

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

int main()
{
    // Begin main function
    cout << "Hello world!\n"; // output "Hello world!"
    return 0; /* indicate successful program completion */
}

// End main function
```

Every C++ program has the function int main()

- main contains the instructions to be executed by the program
- The instructions included in the “body” of main are placed between curly braces {

---

Statements

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

int main()
{
    // Begin main function
    cout << "Hello world!\n"; // output "Hello world!"
    return 0; /* indicate successful program completion */
}

// End main function
```

- Instructions to be performed when the program is run
- Each statement is completed with a ;

---

Using “white spaces”

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

int main()
{
    // Begin main function
    cout << "Hello world!\n"; // output "Hello world!"
    return 0; /* indicate successful program completion */
}

// End main function
```

- “White spaces” are blank lines, space characters, and tabs
- White spaces are ignored by the compiler
- Use indentation to group pieces of code together
Output command

```cpp
std::cout << "Hello world!\n";
```

- `cout` outputs the specified text to the screen.
- The text is delimited by double-quotes " ".
- `<<` is the "stream insertion operator" directing the text into `cout`.

Terminology:
A "character" is any single letter or symbol. E.g.:
'b', '?', '&'
A collection of characters is called a "string." E.g.:
"Hello world", "afe094n", "C++ is fun! ".

Output command, part 2

```cpp
std::cout << "Hello world!\n";
```

> `./myProgram`

Hello world!

> `./myProgram`

- Escape character: backslash \`
- Escape sequence: backslash followed by another character
- New line: `\n`
- Tab: `\t`

```cpp
std::cout << "Hello\n world!\n";
```

> `./myProgram`

Hello world!

> `./myProgram`

- We can place multiple stream insertion operators in a sequence.

```cpp
std::cout << "Hello" << " world.";
stan::cout << "How are \nyou today?";
```

> `./myProgram`

Hello world!

> `./myProgram`

- We can place multiple stream insertion operators in a sequence.

User input: “Hello _____!”

```cpp
// include library of standard input and output commands
#include <iostream>
using namespace std;

int main()
{
    // Begin main function
    string name; // create variable called name
    cin >> name; // get name from user
    cout << "Hello "; // output "Hello 
    cout << name << "!"; // output <name>!
    return 0; // end program
} // End main function
```

> `./myProgram`

What is your name? Alice
Hello Alice!

> `./myProgram`

Variables

Variables store information

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>single character ('a', 'Q')</td>
</tr>
<tr>
<td>int</td>
<td>integers (-4, 82)</td>
</tr>
<tr>
<td>bool</td>
<td>logic (true or false)</td>
</tr>
<tr>
<td>float</td>
<td>real numbers (1.3, -0.45)</td>
</tr>
<tr>
<td>vector</td>
<td>sequence of values (16,5), (-2.3,3.4,-0.4))</td>
</tr>
<tr>
<td>string</td>
<td>text (&quot;Hello&quot;, &quot;reload&quot;)</td>
</tr>
</tbody>
</table>

Variable declaration

```cpp
// include library of standard input and output commands
#include <iostream>
using namespace std;

int main()
{
    // Begin main function
    string name; // create variable called name
    cin >> name; // get name from user
    cout << "Hello "; // output "Hello 
    cout << name << "!"; // output <name>!
    return 0; // end program
} // End main function
```

"Declare" new variable by writing type followed by variable name.
More examples:

int age, weight; // multiple declarations
Variable declaration and initialization

• All variables must be declared before they are used
  int cost;   // declare variable
• Variables are initialized with the first assignment statement
  cost = 25;  // initialize variable
• Declaration and initialization can be performed in one line
  int weight = 140;

“Constant” variables

• The value of a variable ordinarily can be changed throughout the program
  const fixes variable value after initialization
  const float healthyTemp = 98.6;

Variable names

• A variable name is any valid identifier that is not a keyword
  – Starts with a letter, contains letters, digits, and underscores (_) only
  – Cannot begin with a digit
  – Case sensitive:
    username≠userName≠UserName

Variable names, part 2

Choose meaningful names

• Avoid acronyms
• Avoid lengthy names
• Good:
  age, size, address, count, sumData
  x, y, i – single letters as counting variables
• Bad:
  rbi, lda, xZ25,
  neuron_response_magnitude

Keywords

Also known as: “Reserved names”

• Examples
  – cout, return, string, int
• Must be used as they are defined in the programming language
• Cannot be used as variable names

Variable assignment

• Typically, variables are assigned values with the = operator
  string weather;
  weather = "sunny";
  cout << "The weather today is ";
  cout << weather << endl;
• The variable to be changed is always to the left of the = operator
• The value assigned from the right of the = operator
  – Constants: weight = 140;
  – Variables: ageErica = ageJen;
  – Expressions: balance = balance - cost;
Input command

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

int main()
{
    // Begin main function
    string name;             // create variable called name
    cout << "What is your name?");
    cin >> name;             // get name from user
    cout << "Hello ");       // output "Hello 
    cout << name << ");   // output "<name>!
    return 0;               // end program
} // End main function
```

```
• cin >> varName; receives input from keyboard
  saves into varName
```

Arithmetic in C++

Operators
- Addition: 5 + 2 evaluates to 7
- Subtraction: 5 – 2 evaluates to 3
- Multiplication: 5 * 2 evaluates to 10
- Division: 4 / 2 evaluates to 2
- Modulo: 5 % 2 evaluates to 1 (only integers)

Order of operations
- First: Parentheses
- Second: Multiplication, Division, Modulo
- Third: Add, Subtract
- Evaluate from Left to Right
- Evaluate inner-most parentheses before outer ones

```
int a = ( 4 * ( 5 + 2 ) - 4 ) / 4;
```

Assignment operators
- Standard assignment: `a = 3;
- Other assignments:
  - `a += 3;` // `a = a + 3;
  - `a -= 3;` // `a = a - 3;
  - `a *= 3;` // `a = a * 3;
  - `a /= 3;` // `a = a / 3;
  - `a %= 3;` // `a = a % 3;

Increment and decrement
- `int c = 12;
  - Increment by 1: `c++` evaluates to `c + 1`
  - Decrement by 1: `c--` evaluates to `c - 1`

What does this program do?

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

int main()
{
    // Begin main function
    int dollars, coins;
    cout << "How many dollars do you have? ");
    cin >> dollars;
    coins = dollars*4;
    cout << "I will give you " << coins;
    cout << " coins.
"};
    return 0;
} // End main function
```

```
• int dollars, coins;
  cout << "How many dollars do you have? ");
  cin >> dollars;
  coins = dollars*4;
  cout << "I will give you " << coins;
  cout << " coins."
"};
  return 0;
} // End main function
```
The binary representation

- `int age = 65;` assigns a binary code to memory: 01000001
- `char grade = 'A';` assigns a binary code to memory: 01000001
- Every variable value is a number in binary, C++ interprets the binary number based on the variable type.

From numbers to symbols: the ASCII table

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
<th>ASCII</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>73</td>
<td>'A'</td>
<td>A</td>
</tr>
<tr>
<td>48</td>
<td>72</td>
<td>'B'</td>
<td>B</td>
</tr>
<tr>
<td>47</td>
<td>71</td>
<td>'C'</td>
<td>C</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Variable types, revisited

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>single character ('a', 'Q')</td>
<td>1 byte</td>
</tr>
<tr>
<td>int</td>
<td>integers (-4, 82)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>bool</td>
<td>logic (true or false)</td>
<td>1 byte</td>
</tr>
<tr>
<td>float</td>
<td>real numbers (1.3, -0.45)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>vector</td>
<td>sequence of values</td>
<td>? bytes</td>
</tr>
<tr>
<td>string</td>
<td>text (&quot;Hello&quot;, &quot;reload&quot;)</td>
<td>? bytes</td>
</tr>
</tbody>
</table>

- Each variable is represented by a certain number of 0s and 1s
- Each 0-or-1 is a bit
- 8 bits in a row is a byte

Assigning between types

```c
int x = 5;
float y = -2.5;
float z = x * y;
int g = y - x;
```

Variables – locations in memory

- Each variable indicates a location in memory
- Each location holds a value
- Value can change as program progresses
- Variable value exists before initialization

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>04B0C320</td>
<td>A</td>
</tr>
<tr>
<td>04B0C328</td>
<td>---</td>
</tr>
<tr>
<td>04B0C330</td>
<td>140</td>
</tr>
<tr>
<td>04B0C348</td>
<td>P</td>
</tr>
</tbody>
</table>

Assigning between types

- `int vs float`
  - If compiler permits, floats will be rounded to nearest integer and ints will be expanded to a precision float
- `int vs char`
  - If compiler permits, char will be converted to integer ASCII code and int will be converted to corresponding ASCII character
- `int vs bool`
  - If compiler permits, bool will be converted to 0 (if false) or 1 (if true) and int will be converted to false (of 0) or 1 (if not 0)

```c
int x = 5;
float y = -2.5;
float z = x * y;
int g = y - x;
```