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

2

Requirements

- Lectures and lab sessions
- Labs assignments – roughly 8 across semester
- Quizzes – each 15 minutes, roughly 5 across semester
- Final project
- Exams – 1 midterm, 1 final

- Academic integrity – discuss assignments with your classmates, but you MUST write all your code and all your answers yourself

3

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

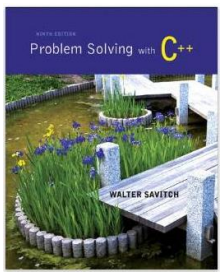
4

Course textbook

Problem Solving With C++

Ninth Edition

Walter Savitch



5

Course website

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

Go online for

- Lecture slides
- Assignments
- Course materials/handouts
- Announcements

6

Instructor

Prof. Daniel Leeds
dleeds@fordham.edu
Office hours: Tues 1-2p, Fri 10:30-11:30a
Office: JMH 328A

7

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

8

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

9

Programs are everywhere

In the dining hall:

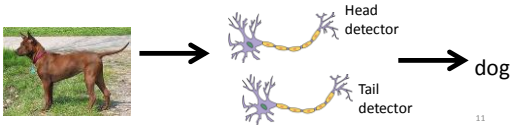
- Cashier
  - Compute price of food purchase, charge payment to account, (if pay cash: compute change)
- HVAC
  - Monitor temperature, adjust A/C or heating
- Electronic signs
  - Display menus and prices, load and display university news

10

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



11

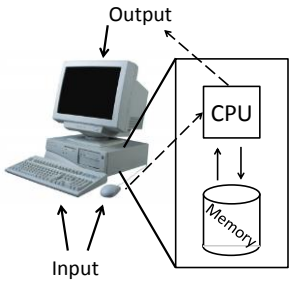
Computer system structure

Central processing unit (CPU) – performs all the instructions

Memory – stores data and instructions for CPU

Input – collects information from the world

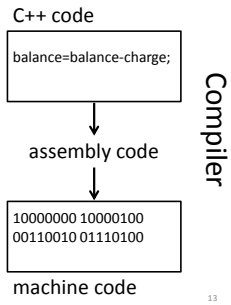
Output – provides information to the world



12

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



13

Why C++?

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

14

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

15

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

16

Programming basics

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

17

Our first program: "Hello world!"

```
// include library of standard input and output commands
#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
```

```
> ./myProgram
Hello world!
>
```

18

The components of "Hello world!"

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

Using comments

```
// include library of standard input and output commands
#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

```
// include library of standard input and output commands
#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

```
// include library of standard input and output commands
#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

```
// include library of standard input and output commands
#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"

```
// include library of standard input and output commands
#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

```
cout << "Hello world!\n";
```

- `cout << "text";` outputs the specified text to the screen
- `cout` is the output stream object
- The text is delimited by double-quotes " "
  - **Only** use simple quotes (") not curly 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! "

25

Output command, part 2

```
cout << "Hello world!\n";
```

```
> ./myProgram
Hello world!
>
```

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

```
cout << "Hello\n world!\n";
```

```
> ./myProgram
```

26

Output command, part 3

```
cout << "Hello world!\n";
```

```
> ./myProgram
Hello world!
>
```

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

```
cout << "Hello world" << "!!!";
cout << "How are \nyou today?";
```

```
> ./myProgram
```

27

User input: “Hello ____!”

```
// include library of standard input and output commands
#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 << "!\n"; // output "<name>!"
    return 0;               // end program
} // End main function
```

```
> ./myProgram
What is your name? Alice
Hello Alice!
>
```

28

Variables

Variables store information

`char`	single character ('a', 'Q')
`int`	integers (-4, 82)
`bool`	logic (true or false)
`float`	real numbers (1.3, -0.45)
`string`	text ("Hello", "reload")

29

Variable declaration

```
// include library of standard input and output commands
#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 << "!\n"; // 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
```

30

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;
```

31

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;`

32

Input command

```
// include library of standard input and output commands
#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 << "!\n";  // output "<name>!"
    return 0;               // end program
} // End main function
```

- `cin >> varName;` receives input from keyboard saves into the `varName`

33

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`

35

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`

36

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

37

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)

38

What does this program do?

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

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

39

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;
```

40

Assignment operators

```
int a = 6;
```

- 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;`

41

Increment and decrement

```
int c = 12;
```

- Increment by 1: `c++` evaluates to `c + 1`

- Decrement by 1: `c--` evaluates to `c - 1`

42

The binary representation

- `int age = 65;` assigns a binary code to memory: 00000000000000000000000001000001
- `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

43

Interpreting binary

Base 10

253 -> 253

2x100+5x10+3x1

Base 2

128 64 32 16 8 4 2 1

\- - - - - - - -

00001001=?

00110000=?

10010010=?

44

From numbers to symbols: the ASCII table

Numeric code	Character	Numeric code	Character	Numeric code	Character	Numeric code	Character
45	-	65	A	85	U	105	i
46	.	66	B	86	V	106	j
47	/	67	C	87	W	107	k
48	0	68	D	88	X	108	l
49	1	69	E	89	Y	109	m
50	2	70	F	90	Z	110	n
51	3	71	G	91	[	111	o
52	4	72	H	92	\	112	p
53	5	73	I	93	]	113	q
54	6	74	J	94	^	114	r
55	7	75	K	95	_	115	s
56	8	76	L	96	`	116	t
57	9	77	M	97	a	117	u
58	:	78	N	98	b	118	v
59	;	79	O	99	c	119	w
60	<	80	P	100	d	120	x
61	=	81	Q	101	e	121	y
62	>	82	R	102	f	122	z
63	?	83	S	103	g	123	{
64	@	84	T	104	h	124	|

Variable types, revisited

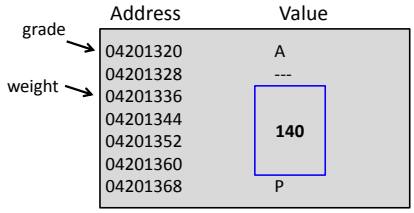
`char`	single character ('a', 'Q')	1 byte
`int`	integers (-4, 82)	4 bytes
`bool`	logic (true or false)	1 byte
`float`	real numbers (1.3, -0.45)	4 bytes
`string`	text ("Hello", "reload")	? bytes

- 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

46

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



47

Assigning between types

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

48

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)

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

49

Type safety

- Ideally, every variable will be used only according to its type
  - A variable will only be used after it has been initialized
  - Only operations defined for the variable's declared type will be applied
  - Every operation defined for a variable leaves the variable with a valid value