Object-Oriented Programming
(in C++)

Introduction to C++ Programming

Professor Yi-Ping You (游逸平)
Department of Computer Science
http://www.cs.nctu.edu.tw/~ypyou/

Outline

- First Program in C++
  - Comments
  - Preprocessor Directives
  - Output Stream
  - Namespace
  - Development Environments
- Modifying Our First C++ Program
  - Memory Concepts
  - Input Stream
  - Arithmetic

First Program in C++: Printing a Line of Text

```cpp
// Fig. 2.1: fig02_01.cpp
// Text-printing program.
#include <iostream> // allows program to output data to the screen

// function main begins program execution
int main()
{
   std::cout << "Welcome to C++!\n"; // display message

   return 0; // indicate that program ended successfully
} // end function main
```

```
Welcome to C++!
```

Fig. 2.1 | Text-printing program.

Comments

- Explain programs to other programmers
  - Improve program readability
- Ignored by the compiler
- Two ways to insert comments
  - Single-line comments
    - A comment beginning with // is called a single-line comment
  - Block comments
    - C’s style (C90/ANSI C)
    - Begin with /* and end with */

Preprocessor Directive

- Preprocessor directives
  - Processed by preprocessor before compiling
  - Begin with **#**
  - Example
    - `#include <iostream>`
      - Tells preprocessor to include the input/output stream **header file `<iostream>`**
- White spaces
  - Blanks, tabs, and blank lines are used to make programs easier to read
  - Extra spaces are ignored by the compiler

Typical C++ Development Environment

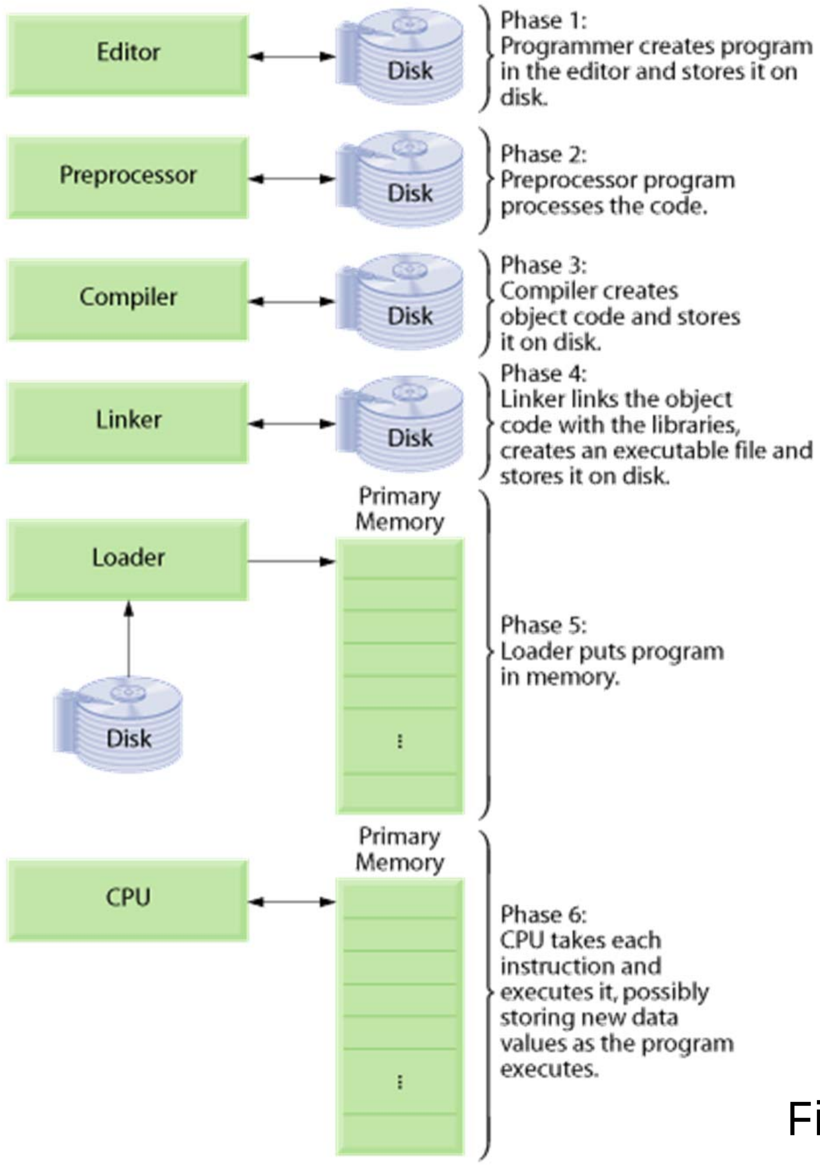

Editor
Disk
Phase 1:
Programmer creates program in the editor and stores it on disk.
Preprocessor
Disk
Phase 2:
Preprocessor program processes the code.
Compiler
Disk
Phase 3:
Compiler creates object code and stores it on disk.
Linker
Disk
Phase 4:
Linker links the object code with the libraries, creates an executable file and stores it on disk.
Loader
Disk
Primary Memory
Phase 5:
Loader puts program in memory.
CPU
Primary Memory
Phase 6:
CPU takes each instruction and executes it, possibly storing new data values as the program executes.

Fig. 1.1

Function `main()`

- A part of every C++ program
  - Exactly **one** function in a program must be **`main`**
  - **`main`** is a `Keyword`
    - Keyword : A word in code that is reserved by C++ for a specific use.
  - It shall have a return type of type `int`
- Header of function main : `int main( )`
- Body is delimited by braces (**`{ }`**)

Statements

- Instruct the program to perform an action
- All statements end with a semicolon (;)
- Examples :

  -
    ```
    std::cout << “Welcome to C++!\n”;
    ```

  -
    ```
    return 0;
    ```

Output Statement (1/2)

```
std::cout << “Welcome to C++!\n”;
```

```
printf(“welcome to C++!\n”);
```

- **std::cout**
  - Standard output stream object
  - Defined in input/output stream **header file** **`<iostream>`**
  - We are using a name (**`cout`**) that belongs to “namespace” **`std`**
  - Normally outputs to computer screen
- Stream insertion operator **<<**
  - Value to right (right operand) inserted into left operand
  - The string of characters contained between “ ” after the operator << shows on computer screen

Namespaces

- Each namespace defines a scope in which identifiers and variables are placed
- Namespaces solve the naming conflicts between different pieces of code

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

namespace Space1 {
  void greeting() {
    cout << “Space1”;
  }
}
namespace Space2 {
 void greeting() {
    cout << “Space2”;
  }
}
```

```
int main() {
  Space1::greeting();
  Space2::greeting();

  {
    using namespace Space1;
    greeting();
  }
  {
    using namespace Space2;
    greeting();
  }
}
```

Output Statement (2/2)

- Escape character : backslash : "\"
- Escape sequence : A character preceded by backslash (\)
  - Indicates “special” character output
    - `\n`: Newline (Position the screen cursor to the beginning of the next line)
    - `\r`: Carriage return (Position the screen cursor to the beginning of the current line)
    - `\t`: Horizontal tab
    - `\\`: Backslash
    - `\’`: Single quote
    - `\”`: Double quote

- UNIX/Linux uses a newline character (\n) as a line terminator
- Windows uses carriage-return newline pairs (\r\n)

Return Statement

- One of several means to exit a function
- When used at the end of `main`
  - The value `0` indicates the program terminated successfully
  - Example

    -
      ```
      return 0;
      ```

Good Programming Practices

- Add comments
  - Every program should begin with a comment that describes the purpose of the program, author, date, and time
- Use good indentation
  - Indent the entire body of each function one level within the braces that delimit the body of the function. This makes a program's functional structure stand out and helps make the program easier to read

Run C++ Program

- Save the program with the right extension
  - `programOne.cpp`
- Compiling the program:
  - `g++ programOne.cpp –o programOne.out`
- Run the executable file
  - `./programOne.out`

Development Environments

- Popular IDEs/Compilers
  - GNU's g++ with a text editor of choice
    - http://gcc.gnu.org/
  - Microsoft's Visual Studio 2015
    - Available at ftp://ca.nctu.edu.tw
  - Microsoft's Visual Studio Express
    - http://www.microsoft.com/express/download/
  - MinGW
    - http://www.mingw.org/
  - Dev-C++ (BloodShed)
    - http://www.bloodshed.net/dev/devcpp.html

Outline

- First Program in C++
  - Comments
  - Preprocessor Directives
  - Output Stream
  - Namespace
  - Development Environments
- Modifying Our First C++ Program
  - Memory Concepts
  - Input Stream
  - Arithmetic

Modifying Our First C++ Program

- Print text on one line using multiple statements
  - Each stream insertion resumes printing where the previous one stopped
  - Statements:

```
Std::cout << “Welcome ”;
Std::cout << “to C++!\n”;
```

```
Welcome to C++!
```

Modifying Our First C++ Program

- Print text on several lines using a single statement.
  - Each newline escape sequence positions the cursor to the beginning of the next line
  - Two newline characters back to back outputs a blank line
  - Example statement :

```
Std::cout << “Welcome\nto\n\nC++!\n”;
```

```
Welcome
to

C++!
```

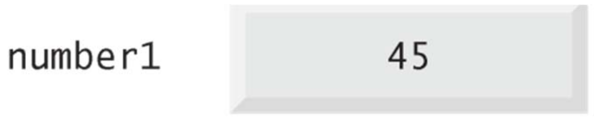
Another C++ Program: Adding Integers

```cpp
// Fig. 2.5: fig02_05.cpp
// Addition program that displays the sum of two integers.
#include <iostream> // allows program to perform input and output

// function main begins program execution
int main()
{
   // variable declarations
   int number1; // first integer to add
   int number2; // second integer to add
   int sum; // sum of number1 and number2

   std::cout << "Enter first integer: "; // prompt user for data
   std::cin >> number1; // read first integer from user into number1

   std::cout << "Enter second integer: "; // prompt user for data
   std::cin >> number2; // read second integer from user into number2

   sum = number1 + number2; // add the numbers; store result in sum

   std::cout << "Sum is " << sum << std::endl; // display sum; end line
} // end function main
```

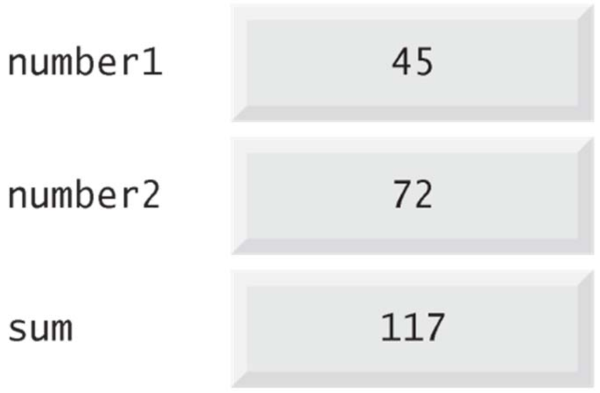
```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Memory Concepts

- Variable names



  - Correspond to actual locations in computer's memory
    - Every variable has name, type, size, and value
  - When a new value placed into variable, the value overwrites the old value
    - Writing to memory is destructive
    - Reading variables from memory nondestructive
  - Example
    - `sum = number1 + number2;`
      - Value of `sum` is overwritten
      - Values of `number1` and `number2` remain intact



Variable

- Location in memory where value can be stored
- Common data types (fundamental, primitive, or built-in)
  - **`int`** – integer numbers : 1, 2, 4,….
  - **`char`** – characters : ‘a’, ‘c’, …
  - **`float, double`** – floating point numbers: 2.5, 4.96
  - **`bool`** – true, false
- The value of a variable could be changed while the program is running

Declaration of Variables (1/3)

- Declare variables with name and data type **before they are used**
- A variable name is any valid identifier that is not a keyword
  - Series of characters - letters, digits, underscores (_)
  - Cannot begin with digit
  - Case sensitive
  - Choosing meaningful identifiers helps make a program self-documenting

Declaration of Variables (2/3)

- Can declare each variable on a separate line.

```
int integer1;
int integer2;
int sum;
```

- Can declare several variables of same type in one declaration.
  - Comma-separated list

```
int integer1, integer2, sum;
```

Declaration of Variables (3/3)

- C++ is much less restrictive than C (C90/ANSI C) in where variables can be declared and initialized
  - In C++, variables can be declared and initialized anywhere, not just at the start of a main program or function block

```
int main() {
  int k=1;  //
initialization
  if (k==1)
  {
    k++;
    int j;  // declaration
    j=k+1;
  }
  return 0;
}
```

```
int main() {
  int j;
  for (int i; i < 10; i++) {
    j++;
  }
  return 0;
}
```

Assign Value to Variables

- Assignment operator =
  - Assigns value on right to variable on left
  - Binary operator (two operands)
- Assign a value after declaration

```
int integer1;   //declaration
integer1 = 10;  //assignment
```

- Declare and assign a value at the same time.

```
int integer2 = 20;
```

Input Stream Object

- **`std::cin`** from **`<iostream>`**
  - Usually connected to keyboard
  - Stream extraction operator **>>**
    - Waits for user to input value, press *Enter* (Return) key
    - Stores value in variable to right of operator
      - Converts value to variable data type
  - Example

```
int number1;
std::cin >> number1; // scanf("%d", &number1);
```

    - Reads an integer typed at the keyboard
    - Stores the integer in variable `number1`

Stream Operators

- Using multiple stream insertion operators (<<) in a single statement is referred to as concatenating, chaining or cascading stream insertion operations
  - Calculations can also be performed in output statements
- `std::endl` is a so-called stream manipulator
  - The name `endl` is an abbreviation for “end line” and belongs to namespace `std`.
  - The `std::endl` stream manipulator outputs a newline, then “flushes the output buffer”

Constant variables

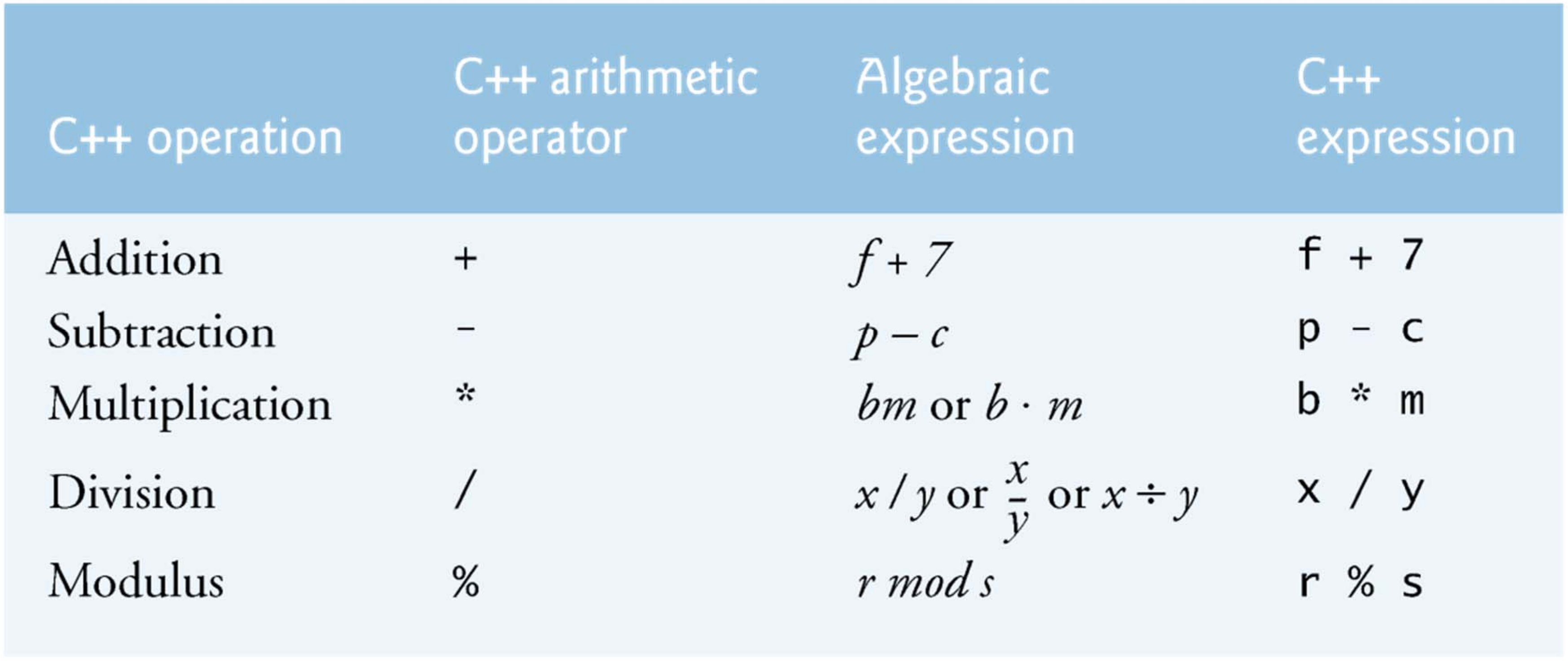
- Declared using the **`const`** qualifier
- Also called named constants or read-only variables
- Must be initialized with a constant expression when they are declared and cannot be modified thereafter
- Example:

```
const int size = 5;
```

Arithmetic (1)

- Arithmetic operators
  - + : Addition
  - - : Subtraction
  - * : Multiplication
  - / : Division
    - Integer division truncates remainder
      - 7 / 5 evaluates to 1
  - % : Modulus operator returns remainder
    - 7 % 5 evaluates to 2
    - Attempting to use the modulus operator (%) with non-integer operands is a compilation error.

Arithmetic Operators

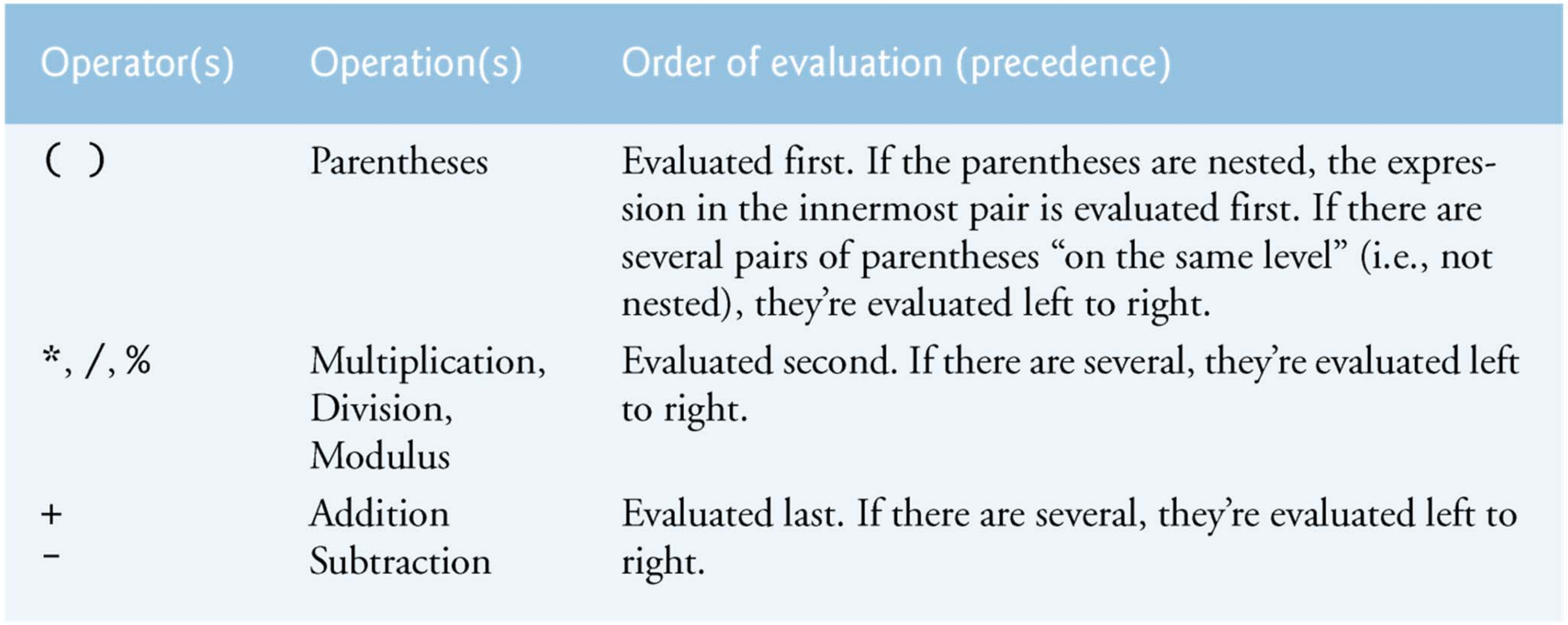


C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`-`	$p - c$	`p - c`
Multiplication	`*`	bm or $b \cdot m$	`b * m`
Division	`/`	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Modulus	`%`	$r \bmod s$	`r % s`

Arithmetic (2)

- Straight-line form
  - Required for arithmetic expressions in C++
  - All constants, variables, and operators appear in a straight line
- Grouping sub-expressions
  - Parentheses are used in C++ expressions to group sub-expressions
    - Same manner as in algebraic expressions
  - Example
    - a * (b + c)
      - Multiple a times the quantity b + c

Precedence of Arithmetic Operators



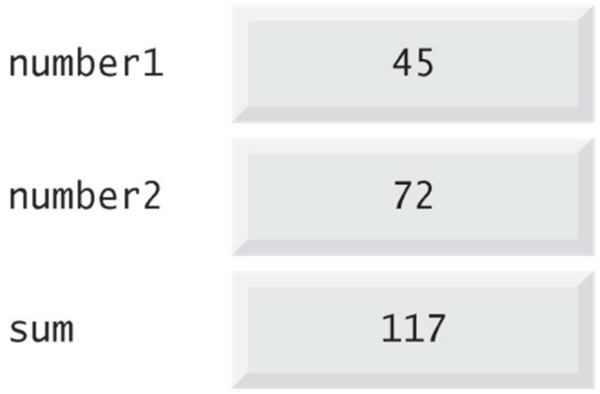
Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
*, /, %	Multiplication, Division, Modulus	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

Performing Arithmetic Calculation

- Adding two integers

```
int number1, number2, sum;
std::cin >> number1;
std::cin >> number2;
sum = number1 + number2;
```

  - Add the values of `number1` and `number2`
  - Store result in `sum`



Type Sizes and Ranges

- The size and range of any data type is compiler and architecture dependent
- Many architectures implement data types of a standard size. ints and floats are often 32-bit, chars 8-bit, and doubles are usually 64-bit