Week 1

Programming on Algorithmic Languages

General Notes About C++ and This Course

- Course geared toward novice programmers
  - Stress programming clarity
  - C and C++ are portable languages
- Portability
  - C and C++ programs can run on many different computers
- Compatibility
  - Many features of current versions of C++ not compatible with older implementations

1 General Notes About C++ and This Course

- What do you need?
  - Books:
    - **C++ How to Program, Fifth (fourth) Edition** By H. M. Deitel - Deit & Associates
    - **C++A Beginner's Guide** By Herbert Schildt
    - **Absolute C++** By Walter Savitch
  - IDE:
    - Microsoft Visual C++ 2008 (Express or Professional editions)
  - Sites:
    - http://cplusplus.com/
    - http://e-practice.org
    - http://www.iitu.kz/
  - **Your Mind (Brain)**

Introduction to C++ Programming

- C++ language
  - Facilitates structured and disciplined approach to computer program design
- Following several examples
  - Illustrate many important features of C++
  - Each analyzed one statement at a time
- Structured programming
- Object-oriented programming

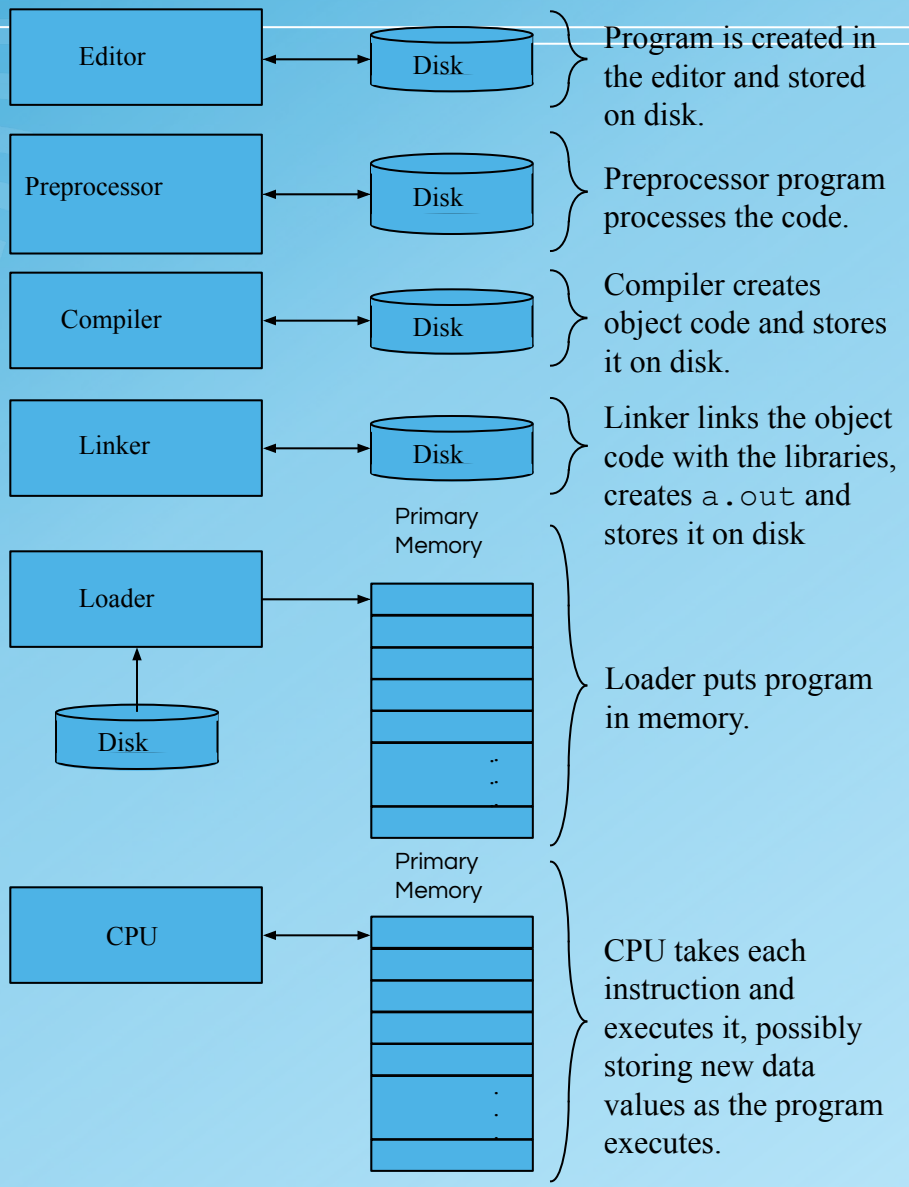
Basics of a Typical C++ Environment

- C++ systems
  - Program-development environment
  - Language
  - C++ Standard Library

Basics of a Typical C++ Environment

Phases of C++ Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



Basics of a Typical C++ Environment

- Input/output
  - `cin`
    - Standard input stream
    - Normally keyboard
  - `cout`
    - Standard output stream
    - Normally computer screen
  - `cerr`
    - Standard error stream
    - Display error messages

1.3 A Simple Program: Printing a Line of Text

- Comments
  - Document programs
  - Improve program readability
  - Ignored by compiler
  - Single-line comment
    - Begin with //
- Preprocessor directives
  - Processed by preprocessor before compiling
  - Begin with #

```
1  // Fig. 1.2: fig01_02.cpp
2  // A first program in C++.
3  #include <iostream>
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n";
9
10    return 0;   // indicate that program ended successfully
11
12 } // end function main
```

Single-line comments.

Function `main` returns an integer value.

Left brace { begins function body.

Preprocessor directive to include input/output stream header file `<iostream>`.

Function `main` appears exactly once in every C++ program.

Statements end with a semicolon ;

Name `cout` belongs to namespace `std`.

Stream insertion operator.

Keyword `return` is one of several means to exit function; value `0` indicates program terminated successfully.

Corresponding right brace } ends function body.

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

1.31 A Simple Program: Printing a Line of Text

- Standard output stream object
  - `std::cout`
  - "Connected" to screen
  - `<<`
    - Stream insertion operator
    - Value to right (right operand) inserted into output stream
- Namespace
  - `std::` specifies using name that belongs to "namespace" `std`
  - `std::` removed through use of `using` statements
- Escape characters
  - `\`
  - Indicates "special" character output

1.31 A Simple Program: Printing a Line of Text

Escape Sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double quote character.

```
1  // Fig. 1.4: fig01_04.cpp
2  // Printing a line with multiple statements.
3  #include <iostream>
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome ";
9     std::cout << "to C++!\n";
10
11    return 0;   // indicate that program ended successfully
12
13 } // end function main
```

Multiple stream insertion statements produce one line of output.

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

```
1  // Fig. 1.5: fig01_05.cpp
2  // Printing multiple lines with a single statement
3  #include <iostream>
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome\nto\n\nC++!\n";
9
10    return 0;   // indicate that program ended successfully
11
12 } // end function main
```

Using newline characters to print on multiple lines.

```
Welcome
to

C++!
```

1.4 Variables

- Variables
  - Location in memory where value can be stored
  - Common data types
    - **`int`** - integer numbers
    - **`char`** - characters
    - **`double`** - floating point numbers
  - Declare variables with name and data type before use

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

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

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

1.4 Variables

- Variables
  - Variable names
    - Valid identifier
      - Series of characters (letters, digits, underscores)
      - Cannot begin with digit
      - Case sensitive

Memory Concepts

- Variable names
  - Correspond to actual locations in computer's memory
  - Every variable has name, type, size and value
  - When new value placed into variable, overwrites previous value
  - Reading variables from memory nondestructive

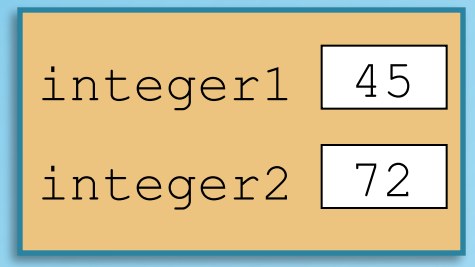
Memory Concepts

```
std::cin >> integer1;
```

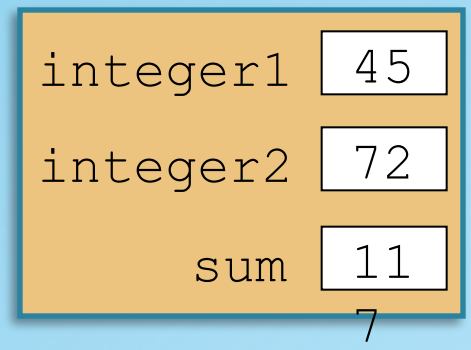
- Assume user entered 45


```
std::cin >> integer2;
```

- Assume user entered 72




```
sum = integer1 + integer2;
```



Data types

C and C++ have four basic built-in data types, described here for binary-based machines.

- **char** is for character storage and uses a minimum of 8 bits (one byte) of storage, although it may be larger.
- **int** stores an integral number and uses a minimum of two bytes of storage.
- The **float** and **double** types store floating-point numbers, usually in IEEE floating-point format. **float** is for **single precision** floating point and **double** is for **double-precision** floating point.

Data types

Integral Types	Floating-Point Types
bool	float
char	double
signed char	long double
unsigned char	
short int	
unsigned short int	
int	
unsigned int	
long int	
unsigned long int	
wchar_t	

Data types

Specifiers

Specifiers modify the meanings of the basic built-in types and expand them to a much larger set. There are four specifiers:

- **Long**
- **Short**
- **Signed**
- **Unsigned**

modify the maximum and minimum values that a data type will hold.

tell the compiler how to use the sign bit with integral types and characters (floating-point numbers always contain a sign).

Data types

- The exact sizes and ranges of values for the fundamental types are implementation dependent.
- The range of values a type supports depends on the number of bytes that are used to represent that type.
  - Consider a system with 4 byte (32 bits) ints.
    - **signed int** type, the nonnegative values are in the range 0 to 2,147,483,647 (2^{31} 1).
    - **signed int** type, the negative values are in the range 1 to 2,147,483,648 (2^{31}).
    - **unsigned int** on the same system would use the same number of bits to represent data, but would not represent any negative values.

This is a total of 2^{32} possible values

values in the range 0 to 4,294,967,295 (2^{32} 1)

C++ Data Types

The guaranteed minimum range for each type as specified by the ANSI/ISO C++ standard

Type	Minimal Range
char	–127 to 127
unsigned char	0 to 255
signed char	–127 to 127
int	–32,767 to 32,767
unsigned int	0 to 65,535
signed int	Same as **int**
short int	–32,767 to 32,767
unsigned short int	0 to 65,535
signed short int	Same as **short int**
long int	–2,147,483,647 to 2,147,483,647
signed long int	Same as **long int**
unsigned long int	0 to 4,294,967,295
float	1E–37 to 1E+37, with six digits of precision
double	1E–37 to 1E+37, with ten digits of precision
long double	1E–37 to 1E+37, with ten digits of precision

C++ Data Types

```
#include <iostream>
/*
This program shows the difference between
signed and unsigned integers.
*/

using namespace std;

int main()
{
    short int i;     // a signed short integer
    short unsigned int j;   // an unsigned short integer
    j=60000;
    i=j;
    cout<<i<<"  "<<j<<endl;
    return 0;
}
```

60,000 is within the range of an unsigned short int, but is typically outside the range of a signed short int . Thus, it will be interpreted as a negative value when assigned to i.

```
-5536  60000
```

Arithmetic

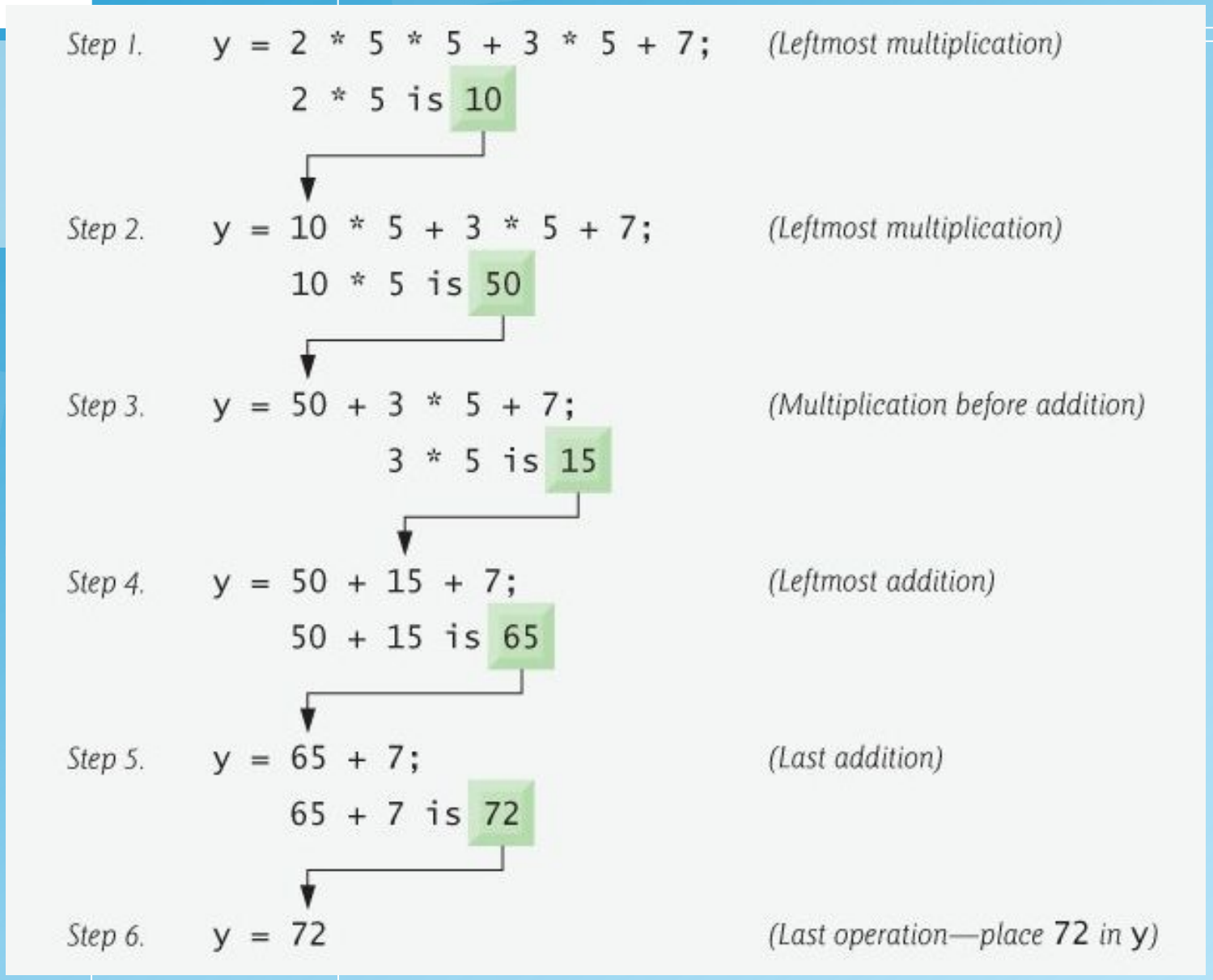
- Arithmetic calculations
  - *
    - Multiplication
  - /
    - Division
    - Integer division truncates remainder
      - 7 / 5 evaluates to 1
  - %
    - Modulus operator returns remainder
      - 7 % 5 evaluates to 2

Arithmetic

- Rules of operator precedence
  - Operators in parentheses evaluated first
    - Nested/embedded parentheses
      - Operators in innermost pair first
  - Multiplication, division, modulus applied next
    - Operators applied from left to right
  - Addition, subtraction applied last
    - Operators applied from left to right

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 are evaluated left to right.
`*`, `/`, or `%`	Multiplication Division Modulus	Evaluated second. If there are several, they re evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Arithmetic


Step 1.
y = 2 * 5 * 5 + 3 * 5 + 7;
(Leftmost multiplication)
2 * 5 is 10
Step 2.
y = 10 * 5 + 3 * 5 + 7;
(Leftmost multiplication)
10 * 5 is 50
Step 3.
y = 50 + 3 * 5 + 7;
(Multiplication before addition)
3 * 5 is 15
Step 4.
y = 50 + 15 + 7;
(Leftmost addition)
50 + 15 is 65
Step 5.
y = 65 + 7;
(Last addition)
65 + 7 is 72
Step 6.
y = 72
(Last operation—place 72 in y)

1.8 Decision Making: Equality and Relational Operators

- `if` structure
  - Make decision based on truth or falsity of condition
    - If condition met, body executed
    - Else, body not executed
- Equality and relational operators
  - Equality operators
    - Same level of precedence
  - Relational operators
    - Same level of precedence
  - Associate left to right

1.8 Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

```
1  //
2  // Using if statements, relational
3  // operators, and equality operators.
4  #include <iostream>
5
6  using std::cout;  // program uses cout
7  using std::cin;   // program uses cin
8  using std::endl;  // program uses endl
9
10 // function main begins program execution
11 int main()
12 {
13    int num1;  // first number to be read from user
14    int num2;  // second number to be read from user
15
16    cout << "Enter two integers, and I will tell you\n"
17         << "the relationships they satisfy: ";
18    cin >> num1 >> num2;   // read two integers
19
20    if ( num1 == num2 )
21       cout << num1 << " is equal to " << num2 << endl;
22
23    if ( num1 != num2 )
24       cout << num1 << " is not equal to " << num2 << endl;
25
```

`using` statements eliminate need for `std::` prefix.

Declare variables.

Can write `cout` and `cin` without `std::` prefix.

`if` structure compares values of `num1` and `num2` to test for equality.

If condition is true (i.e., values are equal), execute this statement.

`if` structure compares values of `num1` and `num2` to test for inequality.

If condition is true (i.e., values are not equal), execute this statement.

```
26    if ( num1 < num2 )
27       cout << num1 << " is less than " << num2 << endl;
28
29    if ( num1 > num2 )
30       cout << num1 << " is greater than " << num2 << endl;
31
32    if ( num1 <= num2 )
33       cout << num1 << " is less than or equal to "
34            << num2 << endl;
35
36    if ( num1 >= num2 )
37       cout << num1 << " is greater than or equal to "
38            << num2 << endl;
39
40    return 0;   // indicate that program ended successfully
41
42 } // end function main
```

Statements may be split over several lines.

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Readings:

- **C++ How to Program,** By H. M. Deitel
  - Chapter 1. Introduction to Computers, the Internet and World Wide Web
  - Chapter 2. Introduction to C++ Programming

Thanks for your attention!