Structured Programming

1.1 Introduction

- In this course you will learn
  - C and C++
  - Structured programming and object oriented programming

1.6 Machine Languages, Assembly Languages, and High-level Languages

- Three types of programming languages
  - Machine languages
    - Strings of numbers giving machine specific instructions
    - Example:

```
+1300042774
+1400593419
+1200274027
```

  - Assembly languages
    - English-like abbreviations representing elementary computer operations (translated via assemblers)
    - Example:

```
LOAD BASEPAY
ADD OVERPAY
STORE GROSSPAY
```


1.6 Machine Languages, Assembly Languages, and High-level Languages

– High-level languages
  - Similar to everyday English, use mathematical notations (translated via compilers)
  - Example:

```
grossPay = basePay + overTimePay
```


1.7 History of C and C++

- C++ evolved from C
  - C evolved from two other programming languages, BCPL and B
- ANSI C
  - Established worldwide standards for C programming
- C++ “spruces up” C
  - Provides capabilities for object-oriented programming
    - Objects are reusable software components that model things in the real world
    - Object-oriented programs are easy to understand, correct and modify

1.8 C++ Standard Library

- C++ programs
  - Built from pieces called classes and functions
- C++ standard library
  - Provides rich collections of existing classes and functions for all programmers to use

1.11 Structured Programming

- Structured programming
  - Disciplined approach to writing programs
  - Clear, easy to test and debug, and easy to modify
- Multitasking
  - Many activities to run in parallel

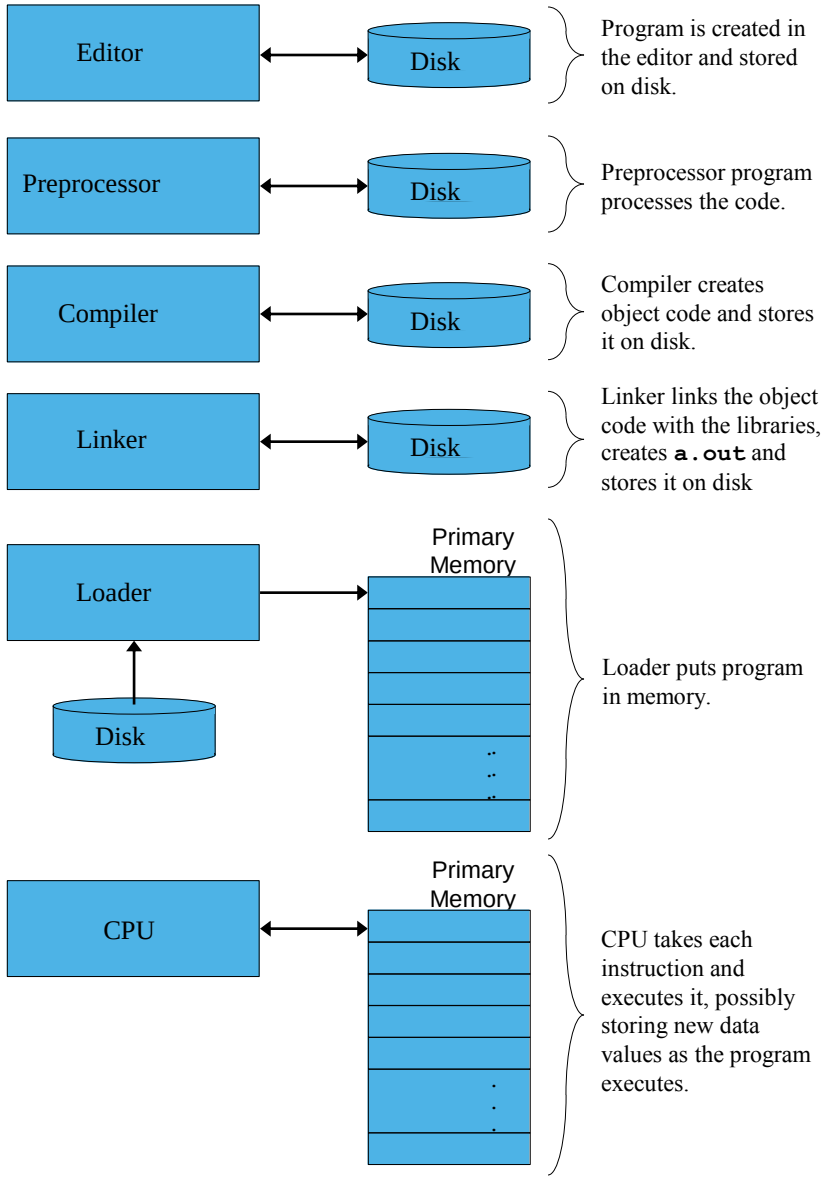
1.12 The Key Software Trend: Object Technology

- Objects
  - Reusable software components that model real world items
  - Meaningful software units
    - Date objects, time objects, paycheck objects, invoice objects, audio objects, video objects, file objects, record objects, etc.
    - Any noun can be represented as an object
  - More understandable, better organized and easier to maintain than procedural programming
  - Favor modularity

1.13 Basics of a Typical C++ Environment

Phases of C++ Programs:

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



1.14 Hardware Trends

- Every year or two computers approximately double
  - The amount of memory they contain
    - Memory used to execute programs
  - The amount of secondary storage they contain
    - Secondary storage (such as disk storage) is used to to hold programs and data over time
  - Their processor speeds
    - The speed at which computers execute their programs

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

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

Comments

Written between **/*** and ***/ block of multiple lines**

or following a **// for single line**.

Improve program readability and do not cause the computer to perform any action.

Message to the C++ preprocessor.

Lines beginning with **#** are preprocessor directives.

#include <iostream> tells the preprocessor to include the contents of the file **<iostream>**, which

C++ programs contain one or more functions, one of which must be **main**

Parenthesis are used to indicate a function

integer value.

Prints the *string* of characters contained between the

ncluding **std::cout**, the **<<**
ng **"Welcome to C++!\n"** and
), is called a *statement*.

All statements must end with a semicolon.

return is a way to exit a function from a function.

return 0, in this case, means that the program terminated normally.

very function

1.19 A Simple Program: Printing a Line of Text

- **`std::cout`**
  - Standard output stream object
  - "Connected" to the screen
  - **`std::`** specifies the "namespace" which **`cout`** belongs to
    - **`std::`** can be removed through the use of **`using`** statements
- **`<<`**
  - Stream insertion operator
  - Value to the right of the operator (right operand) inserted into output stream (which is connected to the screen)
  - **`std::cout << "Welcome to C++!\n";`**
- **`\`**
  - Escape character
  - Indicates that a "special" character is to be output

1.19 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.

- There are multiple ways to print text
  - Following are more examples

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

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

Unless new line **'\n'** is specified, the text continues on the same line.

Outline

1. Load `<iostream>`
2. `main`
   - 2.1 Print `"Welcome"`
   - 2.2 Print `"to C++!"`
   - 2.3 newline
   - 2.4 exit (`return 0`)

Program Output

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

```
Welcome
to

C++!
```

Multiple lines can be printed with one statement.

Outline

1. Load `<iostream>`
2. `main`
2.1 Print `"Welcome"`
2.2 newline
2.3 Print `"to"`
2.4 newline
2.5 newline
2.6 Print `"C++!"`
2.7 newline
2.8 exit (`return 0`)

Program Output

1.20 Another Simple Program: Adding Two Integers

- Variables
  - Location in memory where a value can be stored for use by a program
  - Must be declared with a name and a data type before they can be used
  - Some common data types are:
    - **`int`** - integer numbers
    - **`char`** - characters
    - **`double`** - floating point numbers
  - Example: **`int myvariable;`**
    - Declares a variable named **`myvariable`** of type **`int`**
  - Example: **`int variable1, variable2;`**
    - Declares two variables, each of type **`int`**

1.20 Another Simple Program: Adding Two Integers

- **>>** (stream extraction operator)
  - When used with **`std::cin`**, waits for the user to input a value and stores the value in the variable to the right of the operator
  - The user types a value, then presses the *Enter* (Return) key to send the data to the computer
  - Example:

```
int myVariable;
std::cin >> myVariable;
```

    - Waits for user input, then stores input in **`myVariable`**
- **=** (assignment operator)
  - Assigns value to a variable
  - Binary operator (has two operands)
  - Example:

```
sum = variable1 + variable2;
```


```
1  // Fig. 1.6: fig01_06.cpp
2  // Addition program
3  #include <iostream>
4
5  int main()
6  {
7     int integer1, integer2, sum;           // declaration
8
9     std::cout << "Enter first integer\n";  // p
10    std::cin >> integer1;                  //
11    std::cout << "Enter second integer\n"; // p
12    std::cin >> integer2;                  // read an integer
13    sum = integer1 + integer2;             // assignment of sum
14    std::cout << "Sum is " << sum << std::endl; // print sum
15
16    return 0;   // indicate that program ended success
17 }
```

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

Notice how **std::cin** is used to get user input.

std::endl flushes the buffer and prints a newline.

Variables can be output using **std::cout << variableName**.

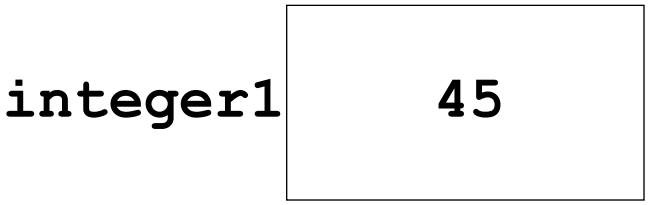
Outline

1. Load `<iostream>`
2. `main`
2.1 Initialize variables `integer1`,
`first integer"`
2.2.1 Get input
2.3 Print `"Enter`
2.4 Add variables and
2.5 Print `"Sum is"`
2.5.1 Output `sum`
2.6 exit (`return 0`)

Program Output

1.21 Memory Concepts

- Variable names
  - Correspond to locations in the computer's memory
  - Every variable has a name, a type, a size and a value
  - Whenever a new value is placed into a variable, it replaces the previous value - it is destroyed
  - Reading variables from memory does not change them
- A visual representation



1.22 Arithmetic

- Arithmetic calculations
  - Use `*` for multiplication and `/` for division
  - Integer division truncates remainder
    - `7 / 5` evaluates to 1
  - Modulus operator returns the remainder
    - `7 % 5` evaluates to 2
- Operator precedence
  - Some arithmetic operators act before others (i.e., multiplication before addition)
    - Be sure to use parenthesis when needed
  - Example: Find the average of three variables a, b and c
    - Do not use: `a + b + c / 3`
    - Use: `(a + b + c ) / 3`

1.22 Arithmetic

- Arithmetic operators:

C++ operation	Arithmetic operator	Algebraic expression	C++ expression
Addition	`+`	*f* + 7	`f + 7`
Subtraction	`-`	*p* – *c*	`p - c`
Multiplication	`*`	*bm*	`b * m`
Division	`/`	*x* / *y*	`x / y`
Modulus	`%`	*r mod s*	`r % s`

- Rules of operator precedence:

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.

1.23 Decision Making: Equality and Relational Operators

- `if` structure
  - Test conditions truth or falsity. If condition met execute, otherwise ignore
- Equality and relational operators
  - Lower precedence than arithmetic operators
- Table of relational operators on next slide

1.23 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`

using statements

- **`using`** statements
  - Eliminate the need to use the **`std::`** prefix
  - Allow us to write cout instead of **`std::cout`**
  - To use the following functions without the **`std::`** prefix, write the following at the top of the program

```
using std::cout;
using std::cin;
using std::endl;
```


```
1  // Fig. 1.14: fig01 14.cpp
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 int main()
11 {
12    int num1, num2;
13
14    cout << "Enter two integers, and I will tell you\n"
15         << "the relationships they satisfy: ";
16    cin >> num1 >> num2;    // read
17
18    if ( num1 == num2 )
19       cout << num1 << " is equal to " << num2 << endl;
20
21    if ( num1 != num2 )
22       cout << num1 << " is not equal to " << num2 << endl;
23
24    if ( num1 < num2 )
25       cout << num1 << " is less than " << num2 << endl;
26
27    if ( num1 > num2 )
28       cout << num1 << " is greater than " << num2 << endl;
29
30    if ( num1 <= num2 )
31       cout << num1 << " is less than or equal to "
32            << num2 << endl;
33
```

Notice the **using** statements.

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
```

The **if** statements test the truth of the condition. If it is **true**, [illegible] is executed. If not, body is skipped.

To include multiple statements in a body, delineate them with braces **{ }**.

```
3 is not equal to 7
```

```
3 is less than 7
```

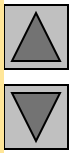
```
3 is less than or equal to 7
```

Outline

1. Load **<iostream>**

2.1 Initialize **num1** and **num2**

2.1.1 Input data



Outline

2.3 exit (`return 0`)

```
34    if ( num1 >= num2 )
35       cout << num1 << " is greater than or equal to "
36            << num2 << endl;
37
38    return 0;   // indicate that program ended successfully
39 }
```

Program Output

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

```
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
```

Home work

- Write a program to print your ID, Name and Level in three different lines using single cout statement.
- Write a program to calculate the area of a square
- Write a program to calculate the area of a rectangle
- Write a program to calculate the area of a circle
- Write a program to calculate the area of a triangle.