261102
Computer Programming

Lecture 1: Introduction

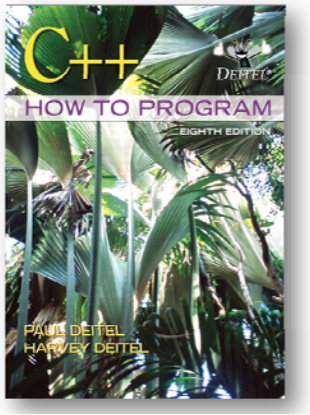
Syllabus

- **Instructor:**

  ผศ.ดร.กานต์ ปทานุคม
  ผศ.ดร.สันติ พิทักษ์กิจนุกูร
  อ.ดร. ปฏิเวธ วุฒิสารวัฒนา

- **Lecture:** Tu-F 14.00-15.00
- **Lab:** Tu-F 15.00-18.00
- **Suggested textbook:**

  Paul Deitel and Harvey Deitel (2012), C++ How to Program (Eighth Edition), Prentice Hall.

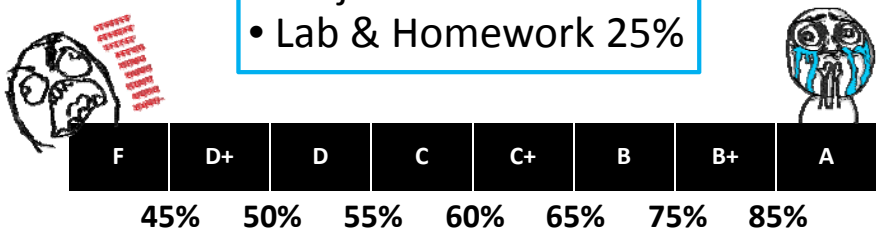


- **Course content:**

  Basic computer programming (C++)

Grading

• Paper Quiz	15%
• Midterm Exam	15%
• Final Exam	15%
• Practical Exam	20%
• Project	10%
• Lab & Homework	25%



What is a Computer?

- **Computer**
  - Device capable of performing computations and making logical decisions
- **Computer programs**
  - Sets of instructions that control computer's processing of data
- **Hardware**
  - Various devices comprising computer
    - Keyboard, screen, mouse, disks, memory, CD-ROM, processing units, …
- **Software**
  - Programs that run on computer

Computer Organization

1. Input unit

- This is the "receiving" section of the computer. It obtains information (data and computer programs) from input devices and places this information at the disposal of the other units for processing.
- Most information is entered into computers through keyboards, touch screens and mouse devices.

Computer Organization

2. Output unit

- This is the "shipping" section of the computer. It takes information that has been processed by the computer and places it on various output devices to make the information available for use outside the computer.
- Most information that's output from computers today is displayed on screens, printed on paper, played as audio or video on portable media players, and transmitted over the Internet.

Computer Organization

3. Memory unit

- The is the rapid access, relatively low-capacity "warehouse" section of the computer.
- It is the area in a computer in which data is stored for quick access by the computer processor (CPU)
- The memory unit is often called either *memory* or *primary memory*. The term Random Access Memory (RAM) is also associated with it.

Computer Organization

4. Arithmetic and logic unit (ALU)

- This is the "manufacturing" section of the computer. It is responsible for performing calculations such as addition, subtraction, multiplication, and division.
- It contains the decision mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether or not they are equal.

Computer Organization

5. Central processing unit (CPU)

- This is the "administrative" section of the computer.
- It is the computer's coordinator and is responsible for supervising the operation of the other sections.

Computer Organization

6. Secondary storage unit

- This is the long-term, high-capacity "warehousing" section of the computer.
- Programs or data not actively being used by the other units are normally placed on secondary storage devices (such disks) until they are again needed later.
- Information in secondary storage takes much longer to access than information in primary memory.
- Examples of secondary storage devices include CD drives, DVD drives and flash drives.
- Less expensive per unit than primary memory

Computer Languages

- Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate *translation* steps.
- Hundreds of such languages are in use today. These may be divided into three general types:
  1. Machine languages
  2. Assembly languages
  3. High-level languages

Machine Languages

- Any computer can directly understand only its own "machine language", defined by its hardware design.
- Machine languages (or machine codes) generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time.
- For example, here's a section of an early machine-language program that adds *overtime pay* to *base pay* and stores the result in *gross pay*:

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

Assembly Languages

- Programming in machine language was simply too slow and tedious for most programmers.
- Instead of using the machine code that computers could directly understand, programmers began using English-like abbreviations to represent elementary operations. These abbreviations formed the basis of "assembly languages".
- Translator programs called "assemblers" were developed to convert early assembly-language programs to machine language.
- The following section of an assembly-language program also adds overtime pay to base pay and stores the result in gross pay:

```
load    basepay
add     overpay
store   grosspay
```

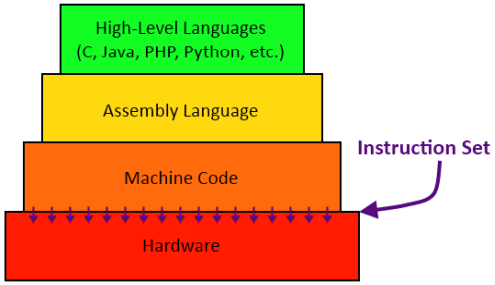

High-level Languages

- The assembly language is still difficult to write instructions to accomplish an easy task.
- To speed up the programming process, "high-level languages" were developed in which single statements could be written to accomplish substantial tasks.
- High-level languages are similar to everyday English and use common mathematical notations
- Need "compiler" to convert high-level language to machine language before execution or "interpreter" for translating and executing an instruction of high-level language programs directly
- A payroll program written in a high-level language might contain a single statement such as

```
grossPay = basePay + overTimePay
```


Computer Languages

- From the programmer's standpoint, high-level languages are more preferable than machine and assembly languages.
- C++, C, Python, PHP, and Java are among the most widely used high-level programming languages.

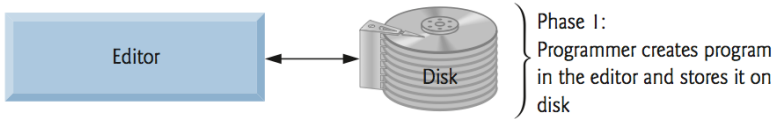


C++ Languages

- Extension of C
- Early 1980s: Bjarne Stroustrup (Bell Laboratories)
- Provides capabilities for object-oriented programming
  - Objects: reusable software components
    - Model items in real world
  - Object-oriented programs
    - Easy to understand, correct and modify
- Hybrid language
  - C-like style
  - Object-oriented style
  - Both
- C++ programs
  - Built from pieces called classes and functions
- C++ standard library
  - Rich collections of existing classes and functions

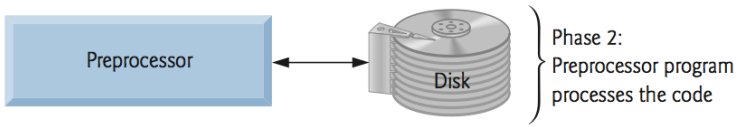
Typical C++ Environment

- **Phase 1: Creating a program**
  - Phase 1 consists of editing a file with an editor program, normally known simply as an "editor".
  - You write a C++ program (typically referred to as "source code") using the editor, make any necessary corrections and save the program on a secondary storage device, such as your hard drive.
  - C++ source code filenames often end with **.cpp**



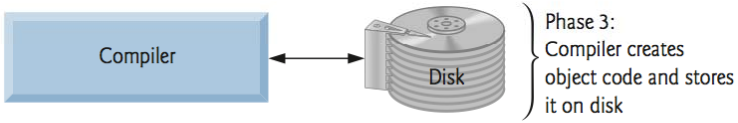
Typical C++ Environment

- **Phase 2: Preprocessing a C++ program**
  - In Phase 2, you give the command to compile the program.
  - In a C++ system, a preprocessor program executes automatically before the compiler's translation phase begins (so we call preprocessing Phase 2 and compiling Phase 3).
  - The C++ preprocessor obeys commands called "preprocessor directives", which indicate that certain manipulations are to be performed on the program before compilation.
  - These manipulations usually include other text files to be compiled, and perform various text replacements.



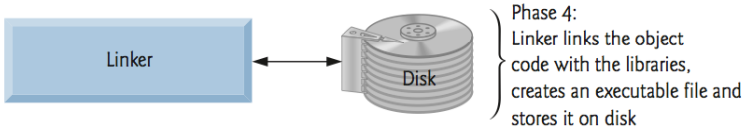
Typical C++ Environment

- **Phase 3: Compiling a C++ program**
  - In Phase 3, the compiler translates the C++ program into machine language, also referred to as "object code".



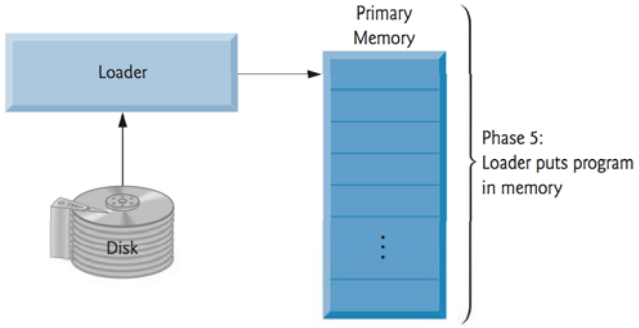
Typical C++ Environment

- **Phase 4: Linking**
  - C++ programs typically contain references to functions and data defined elsewhere, such as in the standard libraries or in the private libraries of groups of programmers working on a particular project.
  - The object code produced by the C++ compiler typically contains "holes" due to these missing parts. A "linker" links the object code with the code for the missing functions to produce an "executable program".



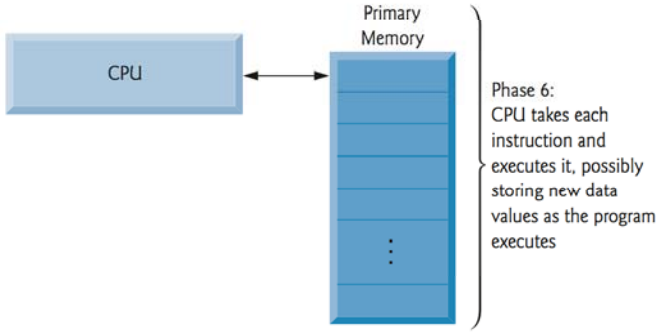
Typical C++ Environment

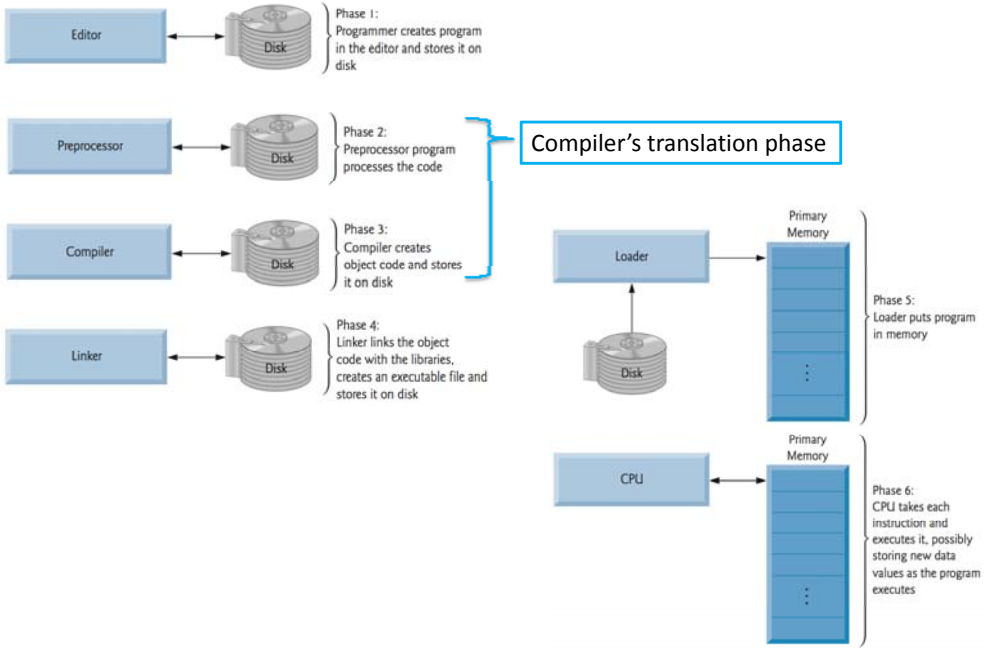
- **Phase 5: Loading**
  - Before a program can be executed, it must first be placed in memory.
  - This is done by the "loader", which takes the executable image from disk and transfers it to memory.
  - Additional components from shared libraries that support the program are also loaded.



Typical C++ Environment

- **Phase 6: Execution**
  - Finally, the computer, under the control of its CPU, executes the program one instruction at a time.





Programming Tools

- **Text Editor**
  - Used for writing source codes and saving in text formats (.c, .cpp, .java, .py, …)
  - Notepad (windows), TextEdit (mac), gedit (linux)
- **Compiler**
  - Convert high-level language into machine code (binaries)
  - cl (VC++), gcc (GNU C), g++ (GNU C++), java (Oracle, OpenJDK), Python
- **Debugger**
  - Used for finding and resolving bug (error or fault) in a program
  - running and examine result of a program step by step
  - pausing the program (breakpoint)
  - tracking the values of variables

Integrated Development Environment (IDE)

- **IDE** is a software application that provides comprehensive facilities to computer programmers for software development.
- An **IDE** normally consists of a source code editor, compiler/interpreter and a debugger.
- Some IDEs support more than one languages.
- Visual Studio, Eclipse , Code::Blocks, MonoDevelop, NetBeans, Dev-C++

Programming Errors

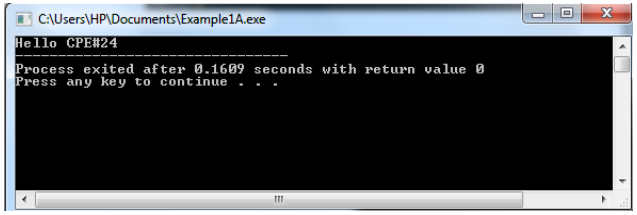
- **Syntax Errors**
  - Source code does not meet the requirements of the language grammar.
  - A program will not compile until all syntax errors are corrected.
- **Logical Errors (Bugs)**
  - Valid program in the language grammar but the program operates incorrectly.

Example 1-A: Printing a Line of Text

Source Code

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello CPE#24";
6
7      return 0;
8  }
```

Output
(on Console)



Example 1-A: Printing a Line of Text

- **Preprocessor directives**
  - Processed by preprocessor before compiling
  - Begin with #
- **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

Example 1-A: Printing a Line of Text

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello CPE#24";
6
7      return 0;
8  }
```

Line 1: `#include <iostream>`

- is a preprocessor directive – a message to the C++ preprocessor.
- Lines that begin with # are processed by the preprocessor before the program is compiled.
- This line notifies the preprocessor to include in the program the contents of the input/output stream header <iostream>.
- This header must be included for any program that outputs data to the screen or inputs data from the keyboard using C++'s stream input/output.

Example 1-A: Printing a Line of Text

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello CPE#24";
6
7      return 0;
8  }
```

Line 3: `int main()`

- is part of every C++ program.
- The parentheses after main indicate that main is a program building block called a function.
- C++ programs typically consist of one or more functions and classes. Exactly one function in every program must be named main.
- The keyword int indicates that main "returns" an integer (whole number) value.
- The left brace, {, (line 4) must begin the body of every function. A corresponding right brace, }, (line 8) must end each function's body.

Example 1-A: Printing a Line of Text

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello CPE#24";
6
7      return 0;
8  }
```

Line 5: `std::cout << "Hello CPE#24";`

- instructs the computer to print the string of characters contained between the double quotation marks (Hello CPE#24).
- The entire line is called "statement". Every C++ statement must end with a semicolon (;)
- Output and input in C++ are accomplished with streams of characters. Thus, when this statement is executed, it sends the stream of characters "Hello World!" to the standard output stream object std::cout— which is normally "connected" to the screen.

Example 1-A: Printing a Line of Text

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello CPE#24";
6
7      return 0;
8  }
```

Line 5: `std::cout << "Hello CPE#24";`

- The std:: before cout is required when we use names that we've brought into the program by the preprocessor directive #include <iostream>.
- The notation std::cout specifies that we are using a name, in this case cout, that belongs to "namespace" std.
- For now, you should simply remember to include std:: before each mention of cout and cin.

Example 1-A: Printing a Line of Text

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello CPE#24";
6
7      return 0;
8  }
```

Line 7: `return 0;`

- is one of several means we'll use to exit a function. When the return statement is used at the end of main, the value 0 indicates that the program has terminated successfully.

Example 1-A: Printing a Line of Text

- **Escape characters**
  - \
  - Indicates "special" character output

Escape code	Description
\n	newline
\r	carriage return
\t	tab
\v	vertical tab
\b	backspace
\f	form feed (page feed)
\a	alert (beep)
\'	single quote (')
\"	double quote (")
\?	question mark (?)
\\	backslash (\)

Example 1-A: Printing a Line of Text

- What if you want to print "Chiang Mai University" on the next line?

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello CPE#24";
6
7      return 0;
8  }
```


Example 1-B: Adding Two Integers

```
1  // This program displays the sum of two integers.
2  #include <iostream> // allows program to perform input and output
3
4  // function main begins program execution
5  int main(){
6      // variable declarations
7      int number1; // first interger to add
8      int number2; // second interger to add
9      int sum; // sum of number1 and number2
10
11     std::cout << "Enter first integer: "; // prompt user for data
12     std::cin >> number1; // read first integer from user into number1
13
14     std::cout << "Enter second integer: "; // prompt user for data
15     std::cin >> number2; // read second integer from user into number2
16
17     sum = number1 + number2; // add the numbers; store result in sum
18
19     std::cout << "Sum is " << sum << std::endl; // display sum; end line
20 } // end function main
```

Example 1-B: Adding Two Integers

- This program uses the input stream object std::cin and the stream extraction operator, >>, to obtain two integers typed by a user at the keyboard, computes the sum of these values and outputs the result using std::cout.

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


Example 1-B: Adding Two Integers

- **Comments**
  - Document programs
  - Improve program readability
  - Ignored by compiler
  - Single-line comment
    - Begin with //
  - Multiple-line comment
    - Begin with /*
    - End with */

Variables

```
1  // This program displays the sum of two integers.
2  #include <iostream> // allows program to perform input and output
3
4  // function main begins program execution
5  int main(){
6      // variable declarations
7      int number1; // first interger to add
8      int number2; // second interger to add
9      int sum; // sum of number1 and number2
10
11     std::cout << "Enter first integer: "; // prompt user for data
12     std::cin >> number1; // read first integer from user into number1
13
14     std::cout << "Enter second integer: "; // prompt user for data
15     std::cin >> number2; // read second integer from user into number2
16
17     sum = number1 + number2; // add the numbers; store result in sum
18
19     std::cout << "Sum is " << sum << std::endl; // display sum; end line
20 } // end function main
```

number1, number2 and sum are the names of variables

Variables

```
7      int number1; // first interger to add
8      int number2; // second interger to add
9      int sum; // sum of number1 and number2
```

Declare

```
12     std::cin >> number1; // read first integer from user into number1
15     std::cin >> number2; // read second integer from user into number2
17     sum = number1 + number2; // add the numbers; store result in sum
```

Assign
Value

```
17     sum = number1 + number2; // add the numbers; store result in sum
19     std::cout << "Sum is " << sum << std::endl; // display sum; end line
```

Use
Stored Value

Variable Declaration

```
type variable_name;
```

Common data types

- `int` - integer numbers
- `double` - floating point numbers
- `bool` – logical type (true, false)
- `char` – characters

Variable's name

- Series of characters (letters, digits, underscores "_")
- Cannot begin with digit.
- Not include "__" (double underscore).
- C++ is case sensitive i.e., a1 and A1 represent different variables.
- Must NOT be a keyword.

```
alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t,
char32_t, class, compl, const, constexpr, const_cast, continue, decltype, default, delete, do,
double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if,
inline, int, long, mutable, namespace, new, noexcept, not, not_eq, nullptr, operator, or, or_eq,
private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static,
static_assert, static_cast, struct, switch, template, this, thread_local, throw, true, try, typedef,
typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while, xor, xor_eq
```


Variable Declaration

- Location in memory (RAM) where value can be stored
- Declare variables with name and data type before use

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

Declarations of variables can be placed almost anywhere in a program, but they must appear before their corresponding variables are used in the program.

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

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


Value Assignment

- **Input stream object**
  - `std::cin`
  - `>>` (stream extraction operator)
    - Waits for user to input value, then press *Enter* (Return) key
    - Stores value in variable to right of operator
    - Converts value to variable data type
- **`=` (Assignment operator)**
  - Assigns value to variable
  - Copy value on the right side to the variable on the left side
  - Example:

```
PI = 3.1416;
sum = variable1 + variable2;
int x = 5;
```

Initialization (Assign while declaring)

Example 1-B: Adding Two Integers

- **Variable declaration:**

```
int number1;
int number2;
int sum;
```

- A variable is a location in the computer's memory where a value can be stored for use by a program.
- These declarations specify that the variables number1, number2, and sum are data of type int, meaning that these variables will hold integer values, i.e., whole numbers such as 7, –11, 0 and 31,914.

Example 1-B: Adding Two Integers

- **Obtaining the First Value from the User**

```
std::cout << "Enter first integer: ";
```

– displays Enter first integer: followed by a space. This message is called a prompt because it directs the user to take a specific action.

```
std::cin >> number1;
```

– uses the standard input stream object cin (of namespace std) and the stream extraction operator, >>, to obtain a value from the keyboard.

– Using the stream extraction operator with std::cin takes character input from the standard input stream, which is usually the keyboard.

Example 1-B: Adding Two Integers

- **Obtaining the Second Value from the User**

```
std::cout << "Enter second integer: ";
std::cin >> number2;
```


Example 1-B: Adding Two Integers

- **Calculating the Sum of the Values Input by the User**

```
sum = number1 + number2;
```

– This assignment statement adds the values of variables number1 and number2 and assigns the result to variable sum using the assignment operator =

Example 1-B: Adding Two Integers

- **Displaying the Result**

```
std::cout << "Sum is " << sum << std::endl;
```

– displays the character string Sum is followed by the numerical value of variable sum followed by std::endl — 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.

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

- C++ provides the modulus operator, %, that yields the remainder after integer division. The modulus operator can be used only with integer operands. The expression x % y yields the remainder after x is divided by y. Thus, 7 % 4 yields 3 and 17 % 5 yields 2.

Arithmetic Operators

- **Examples,**

Algebra: $m = \frac{a + b + c + d + e}{5}$

C++:
```
m = ( a + b + c + d + e ) / 5;
```

Algebra: $y = mx + b$

C++:
```
y = m * x + b;
```