

● ● Lecture 03: Function Basics (Ch3)

09/20/2021

Taesoo Kwon

tskwon@seoultech.ac.kr

Course Schedule (Tentative)

No	Topics	Note	Date
1	Introduction to course & C++ basics (Ch1)	Practice: Upload	09/06
2	Flow of control (Ch2)	Practice: Zoom	09/13
3	Function basics (Ch3)	Practice: Upload	09/20
4	Parameters and overloading (Ch4)	Practice: Zoom	09/27
5	Structures and classes (Ch6)	Practice: Upload	10/04
6	Constructors and other tools (Ch7)	Practice: Upload	10/11
7	Operator overloading, friends, and references (Ch8)	Practice: Zoom, Quiz	10/18
8	Midterm exam	9am-12pm	10/25
9	Strings (Ch9)	TBD	11/01
10	Pointers and dynamic array (Ch10)	TBD	11/08
11	Separate compilation and namespaces (Ch11)	TBD	11/15
12	Inheritance (Ch14)	TBD	11/22
13	Polymorphism and virtual functions (Ch15)	TBD	11/29
14	Templates (Ch16) & Standard template library (Ch19)	TBD, Quiz	12/06
15	Final exam	9am-12pm	12/13

OUTLINES

- ❑ Function basics (Ch 3)
 - Predefined functions, programmer-defined functions, scope rules (local vs. global variables)
- ❑ Summary & Next class
- ❑ Practice





Function Basics

: Predefined functions, programmer-defined functions, scope rules

- ☐ Function basics (Ch 3)
- ☐ Summary & Next class
- ☐ Practice

Scope of Chapter 3

❑ Predefined Functions

- How to use predefined C++ functions

❑ Programmer-defined Functions

- Defining, Declaring, Calling
- Recursive Functions

❑ Scope Rules

- Local variables
- Global constants and global variables
- Blocks, nested scopes

Introduction to Functions

❑ Building Blocks of Programs

❑ Other terminology in other languages:

- Procedures, subprograms, methods
- In C++: functions

❑ I-P-O

- Input – Process – Output
- Basic subparts to any program
- Use functions for these "pieces"

Predefined Functions

- ❑ Libraries full of functions for our use!
- ❑ Two types:
 - Those that **return a value**
 - Those that do not (**void**)
- ❑ Must "**#include**" appropriate library
 - e.g.,
 - `<cmath>`: e.g., for `sqrt`
 - `<cstdlib>`: e.g., for `abs`, `labs`
 - `<iostream>`: e.g., for `cout`, `cin`

Using Predefined Functions

❑ Math functions very plentiful

- Found in library `<cmath.h>`
- Most return a value (the "answer")

❑ Example: `theRoot = sqrt(9.0);`

▪ Components:

<code>sqrt :</code>	name of library function
<code>theRoot :</code>	variable used to assign "answer" to
<code>9.0 :</code>	argument or "starting input" for function

▪ In I-P-O:

- I = 9.0
- P = "compute the square root"
- O = 3, which is returned & assigned to `theRoot`

The Function Call

❑ Back to this assignment:

```
theRoot = sqrt(9.0);
```

- The expression "`sqrt(9.0)`" is known as a **function call**, or **function invocation**
- The **argument** in a function call (`9.0`) can be a **literal**, a **variable**, or an **expression**
- The call itself can be **part of an expression**:
 - `bonus = sqrt(sales)/10;`
 - A function call is allowed wherever it's legal to use an expression of the function's return type

[QP] D3.1 A Predefined Function

```
//Computes the size of a dog house that can be purchased
//given the user's budget.
#include <iostream>
#include <cmath>
using namespace std;

int main( )
{
    const double COST_PER_SQ_FT = 10.50;
    double budget, area, lengthSide;

    cout << "Enter the amount budgeted for your dog house $";
    cin >> budget;

    area = budget/COST_PER_SQ_FT;
    lengthSide = sqrt(area);

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "For a price of $" << budget << endl
        << "I can build you a luxurious square dog house\n"
        << "that is " << lengthSide
        << " feet on each side.\n";

    return 0;
}
```

[QP] A Larger Example (2/2)

```
Enter the amount budgeted for your doghouse $25.00
For a price of $25.00
I can build you a luxurious square doghouse
that is 1.54 feet on each side.
```

More Predefined Functions

❏ `#include <cstdlib>`

▪ Library contains functions like:

- `abs()` // Returns absolute value of an int
- `labs()` // Returns absolute value of a long int
- `fabs()` // Returns absolute value of a float

▪ `fabs()` is actually in library `<cmath>`!

- Can be confusing
- Remember: libraries were added after C++ was "born," in incremental phases
- Refer to appendices/manuals for details

More Math Functions

❑ `pow(x, y)`

- Returns x to the power y

```
double result, x = 3.0, y = 2.0;
```

```
result = pow(x, y);
```

```
cout << result;
```

- Here 9.0 is displayed since $3.0^{2.0} = 9.0$

❑ Notice this function receives two arguments

- A function can have **any number of arguments, of varying data types**

Even More Math Functions (1/2)

Display 3.2 Some Predefined Functions

NAME	DESCRIPTION	TYPE OF ARGUMENTS	TYPE OF VALUE RETURNED	EXAMPLE	VALUE	LIBRARY HEADER
sqrt	Square root	double	double	sqrt(4.0)	2.0	cmath
pow	Powers	double	double	pow(2.0, 3.0)	8.0	cmath
abs	Absolute value for int	int	int	abs(-7) abs(7)	7 7	cstdlib
labs	Absolute value for long	long	long	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	Absolute value for double	double	double	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath

Even More Math Functions (2/2)

ceil <i>Pyuh</i>	Ceiling (round up)	double	double	ceil(3.2) ceil(3.9)	4.0 4.0	cmath
floor <i>urh</i>	Floor (round down)	double	double	floor(3.2) floor(3.9)	3.0 3.0	cmath
exit	End program	int	void	exit(1);	None	cstdlib
rand	Random number	None	int	rand()	Varies	cstdlib
srand	Set seed for rand	unsigned int	void	srand(42);	None	cstdlib

Predefined Void Functions

- ❑ No returned value
- ❑ Performs an action, but sends no "answer"
- ❑ When called, it's a statement itself
 - `exit(1);` // No return value, so not assigned
 - This call terminates program
 - void functions can still have arguments
- ❑ All aspects same as functions that "return a value"
 - They just don't return a value!

Random Number Generator

❑ Return "randomly chosen" number

❑ Used for simulations, games

- `rand()`

- Takes no arguments
- Returns value **between 0 & RAND_MAX**

❑ Scaling

- Squeezes random number into smaller range
`rand() % 6`
- Returns random value **between 0 & 5**

❑ Shifting

`rand() % 6 + 1`

- Shifts range **between 1 & 6** (e.g., die roll)

Random Number Seed

❑ Pseudorandom numbers

- Calls to `rand()` produce given "sequence" of random numbers

❑ Use "seed" to alter sequence `srand(seed_value);`

- `void` function
- Receives one argument, the "seed"
- Can use any seed value, including system time:
`srand(time(0));`
- `time()` returns system time as numeric value
- Library `<time>` contains `time()` functions

Random Examples

❑ Random double between 0.0 & 1.0:

`(RAND_MAX - rand()) / static_cast<double>(RAND_MAX)`

- Type cast used to force double-precision division

❑ Random int between 1 & 6:

`rand() % 6 + 1`

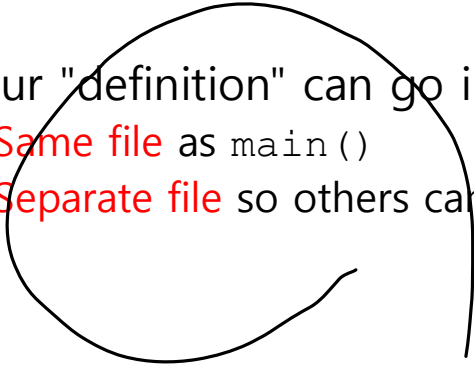
- "%" is modulus operator (remainder)

❑ Random int between 10 & 19:

`rand() % 10 + 10`

0 ~ 9
10 ~ 19

Programmer-Defined Functions

- ❑ Write your own functions!
 - ❑ Building blocks of programs
 - Divide & Conquer
 - Readability
 - Re-use
 - ❑ Your "definition" can go in either:
 - Same file as `main()`
 - Separate file so others can use it, too
- 

Components of Function Use

□ 3 Pieces to using functions:

- **Function Declaration/prototype**
 - Information for compiler
 - To properly interpret calls
- **Function Definition**
 - Actual implementation/code for what function does
- **Function Call**
 - Transfer control to function

Function Declaration

- ❑ Also called **function prototype**
- ❑ An **"informational" declaration** for compiler
- ❑ Tells compiler how to interpret calls
 - **Syntax:**
`<return_type> FnName(<formal-parameter-list>);`
 - **Example:**
`double totalCost(int numberParameter, double priceParameter);`
- ❑ **Placed before any calls**
 - In declaration space of `main()`
 - Or above `main()` in global space

Function Definition

❑ Implementation of function

❑ Just like implementing function `main()`

❑ Example:

```
double totalCost(int numberParameter, double priceParameter)
{
    const double TAXRATE = 0.05;
    double subTotal;
    subtotal = priceParameter * numberParameter;
    return (subtotal + subtotal * TAXRATE);
}
```

❑ Notice proper indenting

Function Definition Placement

- ❑ Placed after function `main()`
 - NOT "inside" function `main()`!
- ❑ Functions are "equals"; no function is ever "part" of another
- ❑ Formal **parameters** in definition
 - "Placeholders" for data sent in
 - "Variable name" used to refer to data in definition
- ❑ **return** statement
 - Sends data back to caller

Function Call

- ❑ Just like calling predefined function

```
bill = totalCost(number, price);
```

- ❑ Recall: `totalCost` **returns** double value

- Assigned to variable named "bill"

- ❑ **Arguments** here: `number, price`

- Recall arguments can be **literals, variables, expressions, or combination**
- In function call, arguments often called "**actual arguments**"
 - Because they contain the "actual data" being sent

[QP] D3.5 A Function to Calculate Total Cost

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

double totalCost(int numberParameter, double priceParameter);
//Computes the total cost, including 5% sales tax,
//on numberPar items at a cost of pricePar each.

int main( )
{
    double price, bill;
    int number;

    cout << "Enter the number of items purchased: ";
    cin >> number;
    cout << "Enter the price per item $";
    cin >> price;

    bill = totalCost(number, price);

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << number << " items at "
         << "$" << price << " each.\n"
         << "Final bill, including tax, is $" << bill
         << endl;

    return 0;
}

double totalCost(int numberParameter, double priceParameter)
{
    const double TAXRATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = priceParameter * numberParameter;
    return (subtotal + subtotal*TAXRATE);
}
```

arguments

[QP] D3.5 A Function to Calculate Total Cost

```
Enter the number of items purchased: 2
Enter the price per item: $10.10
2 items at $10.10 each.
Final bill, including tax, is $21.21
```

[Remind] Formatting Numbers

- ❑ "Magic Formula" to force decimal sizes:

```
cout.setf(ios::fixed);  
cout.setf(ios::showpoint);  
cout.precision(2);
```

- ❑ These statements force all future cout'ed values:

- To have exactly **two digits after the decimal place**

- Example:

```
cout << "The price is $" << price << endl;
```

- Now results in the following:

```
The price is $78.50
```

- ❑ Can modify precision "as you go" as well!

Alternative Function Declaration

❑ Recall: **Function declaration** is "information" for compiler

❑ Compiler only needs to know:

- Return type
- Function name
- Parameter list

parameter name not needed

❑ **Formal parameter names not needed:**

`double totalCost(int, double);`

- Still "should" put in formal parameter names
 - Improves readability

Parameter vs. Argument

□ Terms often **used interchangeably**

□ **Formal** parameters/arguments

- In function declaration
- In function definition's header

□ **Actual** parameters/arguments

- In function call

□ Technically parameter is "formal" piece while argument is "actual" piece*

- *Terms not always used this way

함수 선언, 호출
C/C++
호출하는 값은 실제 → parameter
호출하는 값은 실제
호출하는 값 → argument

Functions Calling Functions

- ❑ We're already doing this!
 - `main()` is a function!
- ❑ Only requirement:
 - Function's declaration must appear first
- ❑ Function's definition typically elsewhere
 - After `main()`'s definition
 - Or in separate file
- ❑ Common for functions to call many other functions
- ❑ Function can even call itself → "Recursion"

Boolean Return-Type Functions

❑ Return-type can be any valid type

- Given function declaration/prototype:

```
bool appropriate(int rate);
```

- And function's definition:

```
bool appropriate (int rate)  
{  
    return (((rate>=10)&&(rate<20)) || (rate==0));  
}
```

- Returns "true" or "false" *ገጽ*
- Function call, from some other function:

```
if (appropriate(entered_rate))  
    cout << "Rate is valid\n";
```


Declaring Void Functions

- ❑ Similar to functions returning a value

- ❑ Return type specified as "void"

- ❑ Example:

- Function declaration/prototype:

- ```
void showResults(double fDegrees, double cDegrees);
```

- Return-type is "void"

- Nothing is returned

# Declaring Void Functions

## ❑ Function definition:

```
void showResults(double fDegrees, double cDegrees)
{
 cout.setf(ios::fixed);
 cout.setf(ios::showpoint);
 cout.precision(1);
 cout << fDegrees
 << " degrees fahrenheit equals \n"
 << cDegrees << " degrees celsius.\n";
}
```

## ❑ Notice: **no return** statement

- Optional for void functions

# Calling Void Functions

- ❑ Same as calling predefined void functions
- ❑ From some other function, like `main()`:
  - `showResults(degreesF, degreesC);`
  - `showResults(32.5, 0.3);`
- ❑ Notice **no assignment**, since **no value returned**
- ❑ Actual arguments (`degreesF, degreesC`)
  - **Passed to function**
  - Function is called to "do its job" with the data passed in

# `main()`: "Special"

- ❑ Recall: `main()` is a function
- ❑ "Special" in that:
  - One and only one function called `main()` will exist in a program
- ❑ Who calls `main()`?
  - Operating system
  - Tradition holds it should have return statement
    - Value returned to "caller" → Here: operating system
  - Should return `"int"` or `"void"`

# Scope Rules

## ❑ Local variables

- Declared inside body of given function
- Available **only within that function**

## ❑ Can have variables with **same names declared in different functions**

- Scope is local: "that function is it's scope"

## ❑ Local variables preferred

- **Maintain individual control** over data
- Need to know basis
- Functions should declare whatever local data needed to "do their job"

# Procedural Abstraction

- ❑ Need to know "what" function does, not "how" it does it!
- ❑ Think "black box"
  - Device you know how to use, but not it's method of operation
- ❑ Implement functions like black box
  - User of function only needs: declaration
  - Does NOT need function definition
    - Called Information Hiding
    - Hide details of "how" function does it's job

# Global Constants and Global Variables

- ❑ Declared "outside" function body
  - Global to all functions in that file
  
- ❑ Declared "inside" function body
  - Local to that function
  
- ❑ Global declarations typical for constants:
  - `const double TAXRATE = 0.05;`
  - Declare globally so all functions have scope
  
- ❑ Global variables?
  - Possible, but SELDOM-USED
  - Dangerous: no control over usage!

# Blocks

- ❑ Declare data **inside compound statement**

- Called a "block"
- Has "**block-scope**"

- ❑ Note: all function definitions are blocks!

- This provides local "function-scope"

- ❑ Loop blocks:

```
for (int ctr=0;ctr<10;ctr++)
{
 sum+=ctr;
}
```

 Variable `ctr` has **scope in loop body block only**



• •

# Summary & Next Class

- ☐ Function basics (Ch 3)
- ☐ Summary & Next class
- ☐ Practice

# Summary

- ❑ Two kinds of functions:
  - "Return-a-value" and void functions
- ❑ Functions should be "black boxes"
  - Hide "how" details
  - Declare own local data
- ❑ Local data
  - Declared in function definition
- ❑ Global data
  - Declared above function definitions
  - OK for constants, not for variables
- ❑ Parameters/Arguments
  - Formal: In function declaration and definition
    - Placeholder for incoming data
  - Actual: In function call
    - Actual data passed to function

# Assignment

- ❑ **Solve Assignment # problems posted on eClass website exercise**
  - eClass → Introduction to Programming(2) → Assignment
- ❑ **Upload your answer sheet on eClass until the deadline**
  - **Firm deadline!!**: **late** submission is **not accepted**
  - **Only docx, hwp, pdf** format allowed (**NOT any figure format including jpg, bmp, png** etc.)
  - eClass → Introduction to Programming(2) → Assignment
  - Don't forget to write your **name, student ID number**.
  - It is not important whether or not your answers are correct. That is, if you **just try to write an answer**, you can **get the perfect scores**.
  - Quiz and exams will rigorously check your efforts on solving the assignment and practice problems by yourself.
- ❑ In order to inquire about the assignment (problem or scoring), please contact to the teaching assistant

# Course Schedule (Tentative)

| No       | Topics                                              | Note                 | Date         |
|----------|-----------------------------------------------------|----------------------|--------------|
| 1        | Introduction to course & C++ basics (Ch1)           | Practice: Upload     | 09/06        |
| 2        | Flow of control (Ch2)                               | Practice: Zoom       | 09/13        |
| 3        | Function basics (Ch3)                               | Practice: Upload     | 09/20        |
| <b>4</b> | Parameters and overloading (Ch4)                    | Practice: Zoom       | 09/27        |
| 5        | Structures and classes (Ch6)                        | Practice: Upload     | 10/04        |
| 6        | Constructors and other tools (Ch7)                  | Practice: Upload     | 10/11        |
| 7        | Operator overloading, friends, and references (Ch8) | Practice: Zoom, Quiz | 10/18        |
| 8        | <b>Midterm exam</b>                                 | <b>9am-12pm</b>      | <b>10/25</b> |
| 9        | Strings (Ch9)                                       | TBD                  | 11/01        |
| 10       | Pointers and dynamic array (Ch10)                   | TBD                  | 11/08        |
| 11       | Separate compilation and namespaces (Ch11)          | TBD                  | 11/15        |
| 12       | Inheritance (Ch14)                                  | TBD                  | 11/22        |
| 13       | Polymorphism and virtual functions (Ch15)           | TBD                  | 11/29        |
| 14       | Templates (Ch16) & Standard template library (Ch19) | TBD, Quiz            | 12/06        |
| 15       | <b>Final exam</b>                                   | <b>9am-12pm</b>      | <b>12/13</b> |



# Practice

- ☐ Function basics (Ch 3)
- ☐ Assignment & Next class
- ☐ Practice

# Practice

- ❑ Chapter 3 Programming Project 5 (p 169)
- ❑ Write a program that asks for the user's **height**, **weight**, and **age**, and then computes **clothing sizes** according to the following formulas.
  - **Hat size** = weight in pounds divided by height in inches and all that multiplied by 2.9.
  - **Jacket size** (chest in inches) = height times weight divided by 288 and then adjusted by adding one-eighth of an inch for each 10 years over age 30. (Note that the adjustment only takes place after a full 10 years. So, there is no adjustment for ages 30 through 39, but one-eighth of an inch is added for age 40.)
  - **Waist** in inches = weight divided by 5.7 and then adjusted by adding one-tenth of an inch for each 2 years over age 28. (Note that the adjustment only takes place after a full 2 years. So, there is no adjustment for age 29, but one-tenth of an inch is added for age 30.)
- ❑ Use **functions for each calculation**.
- ❑ Your program should allow the user to **repeat** this calculation **as often as he or she wishes**.

# Hint

□ Given height, weight, age, compute clothes sizes:

- $\text{hatSize} = \text{weight (lbs.)} / \text{height (in.)} * 2.9$
- $\text{jacketSize (chest size, in.)} = \text{height} * \text{weight} / 288 + (1/8) * (\text{age} - 30) / 10$ 
  - Note carefully that the adjustment only occurs for complete 10 year interval after age 30, e.g.,
    - If  $\text{age} < 40$ , there is no adjustment!
    - $40 \leq \text{age} < 49$  gets 1/8 in. adjustment, etc.
- $\text{waist (in.)} = \text{weight} / 5.7 + (1/10) * (\text{age} - 28) / 2$ 
  - adjustment only occurs for complete 2-year interval after 28, e.g.,
    - $\text{age} = 29$ , no adjustment
    - $30 \leq \text{age} < 32$ , 1/10 inch adjustment.

# Sample Results

Give me your height in inches, weight in pounds, and age in years  
and I will give you your hat size, jacket size(inches at chest)  
and your waist size in inches.

68 143 22

hat size = 6.10

jacket size = 33.76

waist size = 25.09

enter Y or y to repeat,any other character ends.

y

Give me your height in inches, weight in pounds, and age in years  
and I will give you your hat size, jacket size(inches at chest)  
and your waist size in inches.

63 108 21

hat size = 4.97

jacket size = 23.63

waist size = 18.95

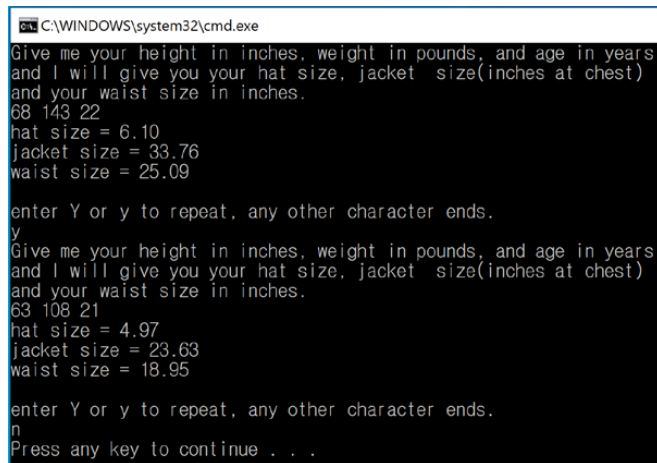
enter Y or y to repeat,any other character ends.

n



# Write Your Program

- ❑ Upload your **code (c++ code file)** and the **capture image (only jpg format)**.
  - eClass → Introduction to Programming(2) → Assignment menu → **Practice #**
  - Due: **eClass announced** (firm deadline)
  - Use functions for each calculation.
  - Your program should allow the user to repeat this calculation as often as he or she wishes.
  - Input number should become your own number
    - Your number needs to be different from those in Sample Results and other students'



```
C:\WINDOWS\system32\cmd.exe
Give me your height in inches, weight in pounds, and age in years
and I will give you your hat size, jacket size(inches at chest)
and your waist size in inches.
68 143 22
hat size = 6.10
jacket size = 33.76
waist size = 25.09

enter Y or y to repeat, any other character ends.
y
Give me your height in inches, weight in pounds, and age in years
and I will give you your hat size, jacket size(inches at chest)
and your waist size in inches.
63 108 21
hat size = 4.97
jacket size = 23.63
waist size = 18.95

enter Y or y to repeat, any other character ends.
n
Press any key to continue . . .
```

- Upload **2 files**: **filename** has to be **your student ID**
  - E.g., if your student ID is 20202456, you have to upload [20202456.cpp](#) and [20202456.jpg](#)
  - Example of jpg file