TAIPEI TECH 國立臺北科技大學 National Taipei University of Technology

Department of Computer Science and Information Engineering

Object Oriented Programming

Lecture 02: Function Basics

Shuo-Han Chen (陳碩漢),
shchen@csie.ntut.edu.tw

The Sixth Teaching Building 327
M 15:10 - 16:00 & F 10:10 - 12:00

Functions

- Predefined Functions
  - Libraries full of functions for our use!
  - Those that return a value
  - Those that do not (void)
  - Ex. int isupper (char)
- Programmer-defined Functions
  - Building blocks of programs
  - Defining, Declaring, Calling
  - Recursive Functions

Functions (Cont'd)

- 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

Functions (Cont'd)

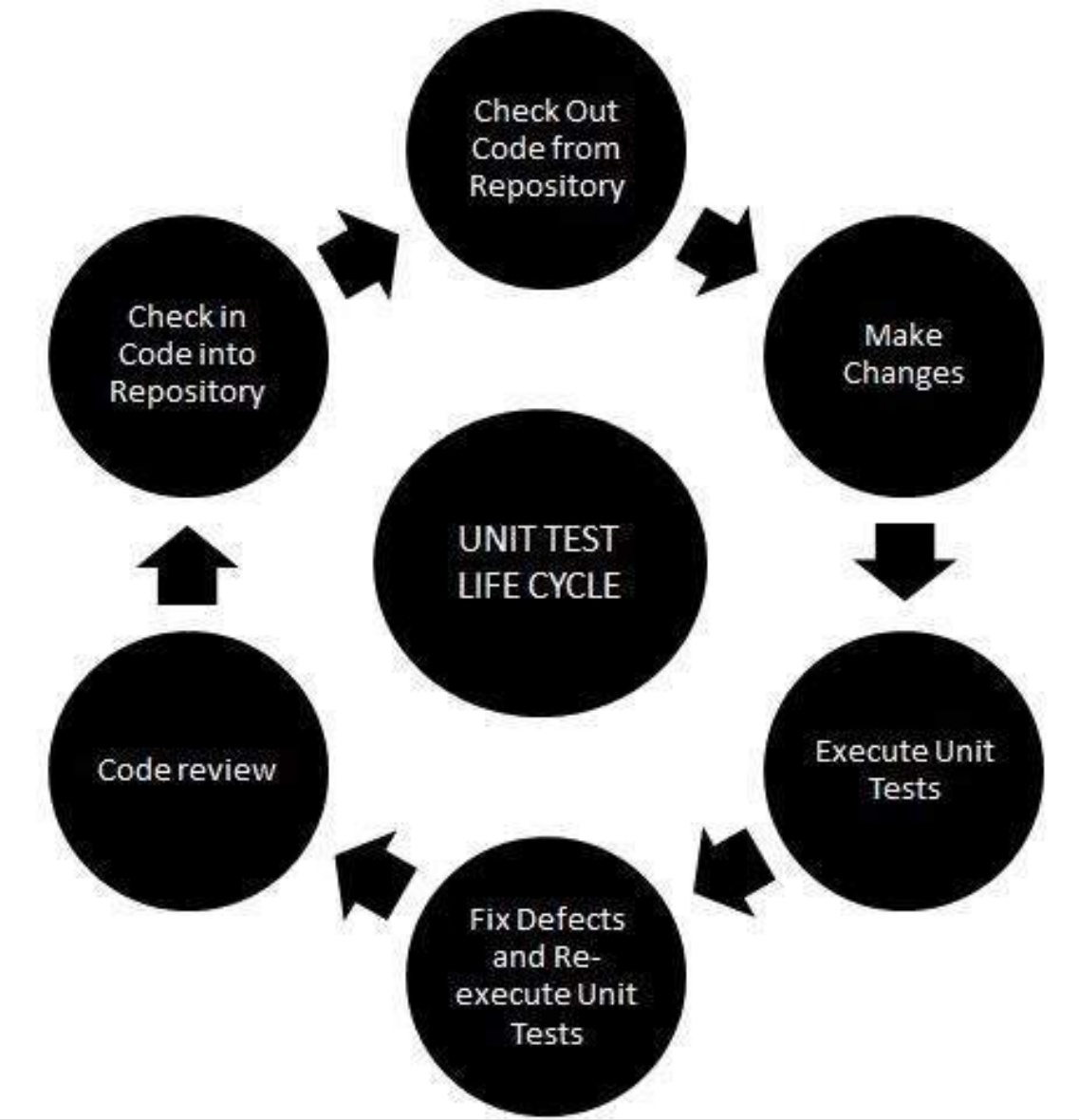
- Recall: main() IS a function
- "Special" in that:
  - One and only one function called main() will exist in a program
- Who calls main()?
  - Operating system
  - Tradition holds it should have return statement
  - Value returned to "caller" -> Here: operating system
  - Should return "int" or "void"

Functions (Cont'd)

- 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

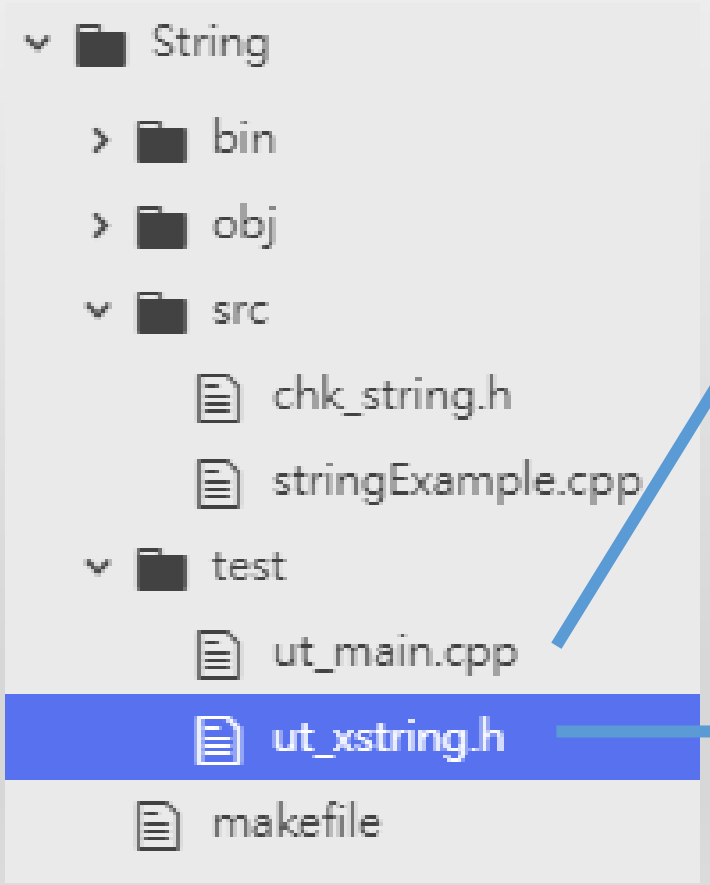
Unit test

- Individual modules of a program are tested individually to determine if there are any issues
- Isolate each unit of the system for identifying, analyzing and fixing issues



The Google Test Framework

- To use Google Test, we need to have following




```cpp
#include <gtest/gtest.h>
#include "ut_xstring.h"

int main(int argc , char **argv){
  testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

```cpp
#include "../src/xstring.h"

TEST(CHKUPPER, case1){
  string test = "AAAbCCCdEE";
  vector<int> result = checkUpper(test);
  ASSERT_EQ(3, result[0]);
  ASSERT_EQ(3, result[1]);
  ASSERT_EQ(2, result[2]);
}
```

The Google Test Framework (Cont'd)

- Checking whether your program works as expceted
  - ASSERT -> Fatal Error, aborting the current function
  - EXPECT -> Continues after the failure

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_TRUE(*condition*);	EXPECT_TRUE(*condition*);	*condition* is true
ASSERT_FALSE(*condition*);	EXPECT_FALSE(*condition*);	*condition* is false

The Google Test Framework (Cont'd)

Fatal assertion	Nonfatal assertion	Verifies
`ASSERT_EQ(val1,val2);`	`EXPECT_EQ(val1,val2);`	*val1* == *val2*
`ASSERT_NE(val1,val2);`	`EXPECT_NE(val1,val2);`	*val1* != *val2*
`ASSERT_LT(val1,val2);`	`EXPECT_LT(val1,val2);`	*val1* < *val2*
`ASSERT_LE(val1,val2);`	`EXPECT_LE(val1,val2);`	*val1* <= *val2*
`ASSERT_GT(val1,val2);`	`EXPECT_GT(val1,val2);`	*val1* > *val2*
`ASSERT_GE(val1,val2);`	`EXPECT_GE(val1,val2);`	*val1* >= *val2*

The Google Test Framework (Cont'd)

Fatal assertion	Nonfatal assertion	Verifies
`ASSERT_STREQ(str1,str2);`	`EXPECT_STREQ(str1,_str_2);`	the two C strings have the same content
`ASSERT_STRNE(str1,str2);`	`EXPECT_STRNE(str1,str2);`	the two C strings have different content
`ASSERT_STRCASEEQ(str1,str2);`	`EXPECT_STRCASEEQ(str1,str2);`	the two C strings have the same content, ignoring case
`ASSERT_STRCASENE(str1,str2);`	`EXPECT_STRCASENE(str1,str2);`	the two C strings have different content, ignoring case

Vectors (Text book 342 – 347)

- Can be thought as an array that can grow and shrink
- Part of standard template library
- Has base type
  - Stores collection of base type values
- Declared differently:
  - Syntax: vector<Base_Type>
    - Indicates template class
    - Any type can be "plugged in" to Base_Type
    - Produces "new" class for vectors with that type
- Example declaration: vector<int> v;

Vectors (Text book 342 – 347) (Cont'd)

- Some basic method
  - size()
  - push_back()
  - capcaity()
- Size <= capacity
  - Size -> The actual number of elements
  - Capacity -> The total number of elements this vector can hold

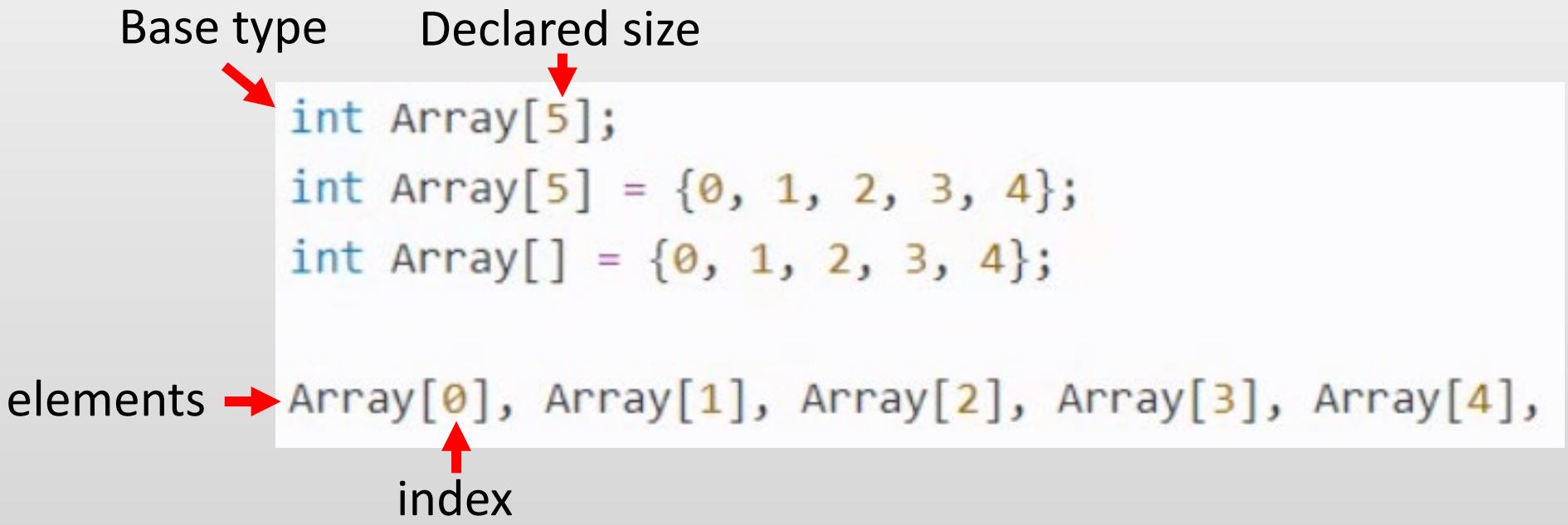
- Find the rest on cplusplus.com

How we are going to solve this problem ?

- In this course, we follow the steps of
  - “How To Solve It” (數學家George Pólya)
  1. **瞭解問題**(understanding the problem)
     - Find the number of continuous upper case in a string
  2. **規劃解法**(devising a plan)
     - 先個別檢查字母的大小寫
     - 用 0 1 代表
     - 找出把所有的 0 的長度印出來
  3. **依規劃解題** (carrying out the plan)
     - 大小寫檢查 isupper()
     - 型態 String, int, vector
  4. **回顧**(looking back) **-> Let’s look back with unit test !**

Array (Text book Chapter 5)

- Similar to C, we can declare fixed size array



- These two are the same during declaration array

```
int Array[];
int* Array;
```

Array (Text book Chapter 5) (cont'd)

- In addition to fix-sized array, we can do dynamically allocated array

```
int* Array = new int[100];
```

- We need to use int*, instead of int Array[], because new return pointer to a newly allocated variable

Array (Text book Chapter 5) (cont'd)

- For loop in a new form
- The C++11 ranged-based for loop makes it easy to iterate over each element in a loop
- Format

```
for (datatype varname : array)
{
        //  varname is set to each successive
        //  element in the array
}
```

```
int arr[] = {20, 30, 40, 50};
for (int x : arr)
  cout << x << " ";
cout << endl;
```

Exception Handling (Text book Chapter 18) (cont'd)

- Typical approach to development:
  - Write programs assuming things go as planned
  - Get "core" working
  - Then take care of "exceptional" cases
- C++ exception-handling facilities
  - Handle "exceptional" situations

- Try-throw-catch

Exception Handling (Text book Chapter 18) (cont'd)

- Typical approach to development:
  - Write programs assuming things go as planned
  - Get "core" working
  - Then take care of "exceptional" cases
- C++ exception-handling facilities
  - Handle "exceptional" situations

- Try-throw-catch

Exception Handling (Text book Chapter 18) (cont'd)

- More different variable type can be cached

```
 9      try
10      {
11          cout << "Enter number of donuts:\n";
12          cin >> donuts;
13          cout << "Enter number of glasses of milk:\n";
14          cin >> milk;
15
16          if (milk <= 0)
17                  throw donuts;

18          dpg = donuts/static_cast<double>(milk);
19          cout << donuts << " donuts.\n"
20               << milk << " glasses of milk.\n"
21               << "You have " << dpg
22               << " donuts for each glass of milk.\n";
23      }
24      catch(int e)
25      {
26          cout << e << " donuts, and No Milk!\n"
27               << "Go buy some milk.\n";
28      }
```

Summary

1. How to write functions
   - Same as you learnt in C
2. Unit test
   - Ideas
   - Google test
   - Identify the problem we didn’t found last time
3. Array
   - Ranged-based for loop
4. Exception Handling
   - Try-throw-catch

Q & A

Thank you for your attention.