Mute yourself after joining. We will start at 15:10.

If you haven’t give me the 實體加簽單, please hand-in on time

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 01: Flow of Control

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

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

Course Schedule

Due to national holidays, there will be no class on Sept. 20, Oct. 11, and Dec. 31.

W	Date	Lecture	Method	Readings	Homework
1	Sept. 20, 24	Lec01: Moon Festival / Introduction & Environment Setup		Chapter 1	HW00
2	Sept. 27, Oct.1	Lec02: Flow of Control		Chapter 2	
3	Oct. 4, 8	Lec03: Function Basics / Exception Handling			
4	Oct. 11, 15	Lec04: National Day / String	(15) BOPPPS & IRS		
5	Oct. 18, 22	Lec05: Pointer	(18) BOPPPS & IRS		
6	Oct. 25, 29	Lec06: Parameters and Overloading			
7	Nov. 1, 5	Lec07: Structures and Classes	(1) BOPPPS & IRS		
8	Nov. 8, 12	Lec08: Constructors and Other Tools			
9	Nov. 15, 19	Lec09: Operator Overloading, Friends, and References	(19) BOPPPS & IRS		
10	Nov. 22, 26	Midterm -> 50M Hand-written on Nov. 22 and 3H Computer-based on Nov. 26		No Class	
11	Nov. 29, Dec. 3	Lec10: Inheritance			
12	Dec. 6, 10	Lec11: Inheritance			
13	Dec. 13, 17	Lec12: Polymorphism and Virtual Functions	(12) BOPPPS & IRS		
14	Dec. 20, 24	Lec13: Polymorphism and Virtual Functions			
15	Dec. 27, 31	Lec14: Standard Template Library / New Year Holiday			
16	Jan. 3, 7	Lec15: Templates			
17	Jan. 10, 14	Lec16: Streams and File I/O / Namespace	(14) BOPPPS & IRS		
18	Jan. 17, 21	Final -> 50M Hand-written on Jan. 17 and 3H Computer-based on Jan. 21		No Class	

Console I/O (Text book 57 – 64)

- In C++, the printf and scanf are replaced with cin & cout
- Part of the iostream library and std namespace
- When you do output
  - cout <<
- When you do input
  - cin >>
  - Direction : To where you want the content to be
- You can put variable in cin/cout directly, without
  - scanf(“%i”, number);
  - printf(“%i”, number);

Display 1.1 A Sample C++ Program

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

int main( )
{
    int numberOfLanguages;

    cout << "Hello reader.\n"
         << "Welcome to C++.\n";

    cout << "How many programming languages have you used? ";
    cin >> numberOfLanguages;

    if (numberOfLanguages < 1)
        cout << "Read the preface. You may prefer\n"
             << "a more elementary book by the same author.\n";
    else
        cout << "Enjoy the book.\n";

    return 0;
}
```

Namespace (Text book 521 – 540)

- Namespace is a collection of name definition
  - ex. Function / class / variable
- Namespace is used to isolate function/class with the same name but has different functions
- Std contains all definition from all standard library
- The way we use it
  - Includes entire standard library of name definitions
    - #include <iostream>
      using namespace std;
  - Can specify just the objects we want
    - #include <iostream>
    - using std::cin;
      using std::cout;

Namespace (Text book 521 – 540)

Display 11.7 Placing a Class in a Namespace (Implementation File)

```
//This is the implementation file dtime.cpp.
#include <iostream>
#include <cctype>
#include <cstdlib>
using std::istream;
using std::ostream;
using std::cout;
using std::cin;
#include "dtime.h"

namespace DTimeSavitch
{

    <All the function definitions from Display 11.2 go here.>

}// DTimeSavitch
```

You can use the single `using` *directive*
`using namespace std;`
in place of these four `using` *declarations. However, the four* `using` *declarations are a preferable style.*

Namespace (Text book 521 – 540)

- Multiple namespace in one code project is allowed
- You can also define namespaces of your own

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

namespace Space1
{
    void greeting( );
}

namespace Space2
{
    void greeting( );
}

void bigGreeting( );
```

```cpp
int main( )
{
    {
        using namespace Space2;
        greeting( );
    }

    {
        using namespace Space1;
        greeting( );
    }

    bigGreeting( );

    return 0;
}
```

```cpp
namespace Space1
{
    void greeting( )
    {
        cout << "Hello from namespace Space1.\n";
    }
}

namespace Space2
{
    void greeting( )
    {
        cout << "Greetings from namespace Space2.\n";
    }
}

void bigGreeting( )
{
    cout << "A Big Global Hello!\n";
}
```

String

- We have less trouble now when dealing with stream
- Part of the string library and std namespace
- You do cout and cin directly
- We can also mess the string with some operators / methods
  - +=
  - +
  - ==
  - .length() / .compare()

Display 1.5 Using cin and cout with a string (part 1 of 2)

```
//Program to demonstrate cin and cout with strings
#include <iostream>
#include <string>

using namespace std;
int main( )
{
  string dogName;
  int actualAge;
  int humanAge;

  cout << "How many years old is your dog?" << endl;
  cin >> actualAge;
  humanAge = actualAge * 7;

  cout << "What is your dog's name?" << endl;
  cin >> dogName;

  cout << dogName << "'s age is approximately " <<
         "equivalent to a " << humanAge << " year old human."
         << endl;

  return 0;
}
```

Needed to access the **string** *class.*

How does that become possible ?

- As we already discussed, C++ enable objects to have
  - State (Data)
  - Behavior (Functions)
  - See the implementation of string here https://gcc.gnu.org/onlinedocs/gcc-4.6.2/libstdc++/api/a00259.html
  - https://gcc.gnu.org/onlinedocs/gcc-4.6.2/libstdc++/api/a00770_source.html
- Data

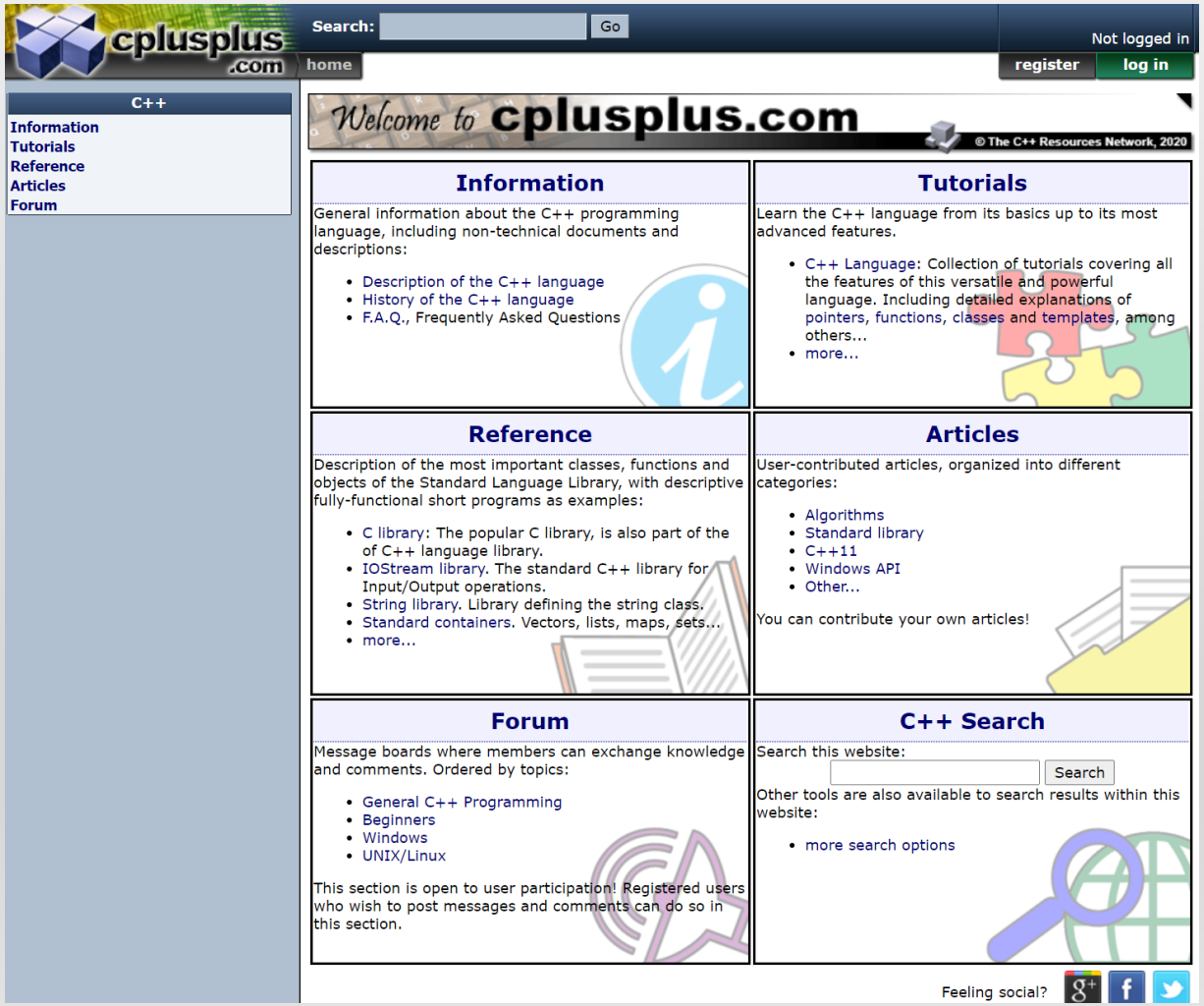
```
• typedef _Alloc allocator_type
• typedef
  __gnu_cxx::__normal_iterator
  < const_pointer, basic_string > const_iterator
• typedef
  _CharT_alloc_type::const_pointer const_pointer
• typedef
  _CharT_alloc_type::const_reference const_reference
• typedef std::reverse_iterator
  < const_iterator > const_reverse_iterator
• typedef
  _CharT_alloc_type::difference_type difference_type
• typedef
  __gnu_cxx::__normal_iterator
  < pointer, basic_string > iterator
• typedef _CharT_alloc_type::pointer pointer
• typedef
  _CharT_alloc_type::reference reference
• typedef std::reverse_iterator
  < iterator > reverse_iterator
• typedef
  _CharT_alloc_type::size_type size_type
• typedef _Traits traits_type
• typedef _Traits::char_type value_type
```

- Functions

```
• basic_string & operator+= (const basic_string &__str)
• basic_string & operator+= (const _CharT *__s)
• basic_string & operator+= (_CharT __c)
• basic_string & operator+= (initializer_list< _CharT > __l)
• basic_string & operator= (const basic_string &__str)
• basic_string & operator= (const _CharT *__s)
• basic_string & operator= (_CharT __c)
• basic_string & operator= (basic_string &&__str)
• basic_string & operator= (initializer_list< _CharT > __l)
• const_reference operator[] (size_type __pos) const
• reference operator[] (size_type __pos)
• void push_back (_CharT __c)
• reverse_iterator rbegin ()
• const_reverse_iterator rbegin () const
• reverse_iterator rend ()
```

Try this out

- http://cplusplus.com/
  - Will be very helpful in your future homework



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)

Assuming you're familiar with the following

- Comparison Operators

Display 2.1 Comparison Operators

MATH SYMBOL	ENGLISH	C++ NOTATION	C++ SAMPLE	MATH EQUIVALENT
=	Equal to	==	`x + 7 == 2*y`	$x + 7 = 2y$
≠	Not equal to	!=	`ans != 'n'`	*ans* ≠ *'n'*
<	Less than	<	`count < m + 3`	*count* < m + 3
≤	Less than or equal to	<=	`time <= limit`	*time* ≤ *limit*
>	Greater than	>	`time > limit`	*time* > *limit*
≥	Greater than or equal to	>=	`age >= 21`	*age* ≥ 21

- Logical Operators
  - Logical AND (&&)
  - Logical OR (||)

Assuming you're familiar with the following

- Truth Table

Display 2.2 Truth Tables

AND

Exp_1	*Exp_2*	*Exp_1* && *Exp_2*
true	true	true
true	false	false
false	true	false
false	false	false

OR

Exp_1	*Exp_2*	*Exp_1* || *Exp_2*
true	true	true
true	false	true
false	true	true
false	false	false

NOT

Exp	!(*Exp*)
true	false
false	true

Precedence of Operators

Display 2.3 Precedence of Operators

::	Scope resolution operator
.	Dot operator
->	Member selection
[]	Array indexing
( )	Function call
++	Postfix increment operator (placed after the variable)
--	Postfix decrement operator (placed after the variable)
++	Prefix increment operator (placed before the variable)
--	Prefix decrement operator (placed before the variable)
!	Not
-	Unary minus
+	Unary plus
*	Dereference
&	Address of
new	Create (allocate memory)
delete	Destroy (deallocate)
delete[]	Destroy array (deallocate)
sizeof	Size of object
( )	Type cast

Highest precedence (done first)

Precedence of Operators (Cont'd)

*	Multiply
/	Divide
%	Remainder (modulo)
+	Addition
–	Subtraction
<<	Insertion operator (console output)
>>	Extraction operator (console input)

Lower precedence (done later)

Precedence of Operators (Cont'd)

Display 2.3 Precedence of Operators

All operators in part 2 are of lower precedence than those in part 1.

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal
!=	Not equal
&&	And
||	Or

Precedence of Operators (Cont'd)

=	Assignment
+=	Add and assign
−=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulo and assign
? :	Conditional operator
throw	Throw an exception
,	Comma operator

Lowest precedence (done last)

Precedence Examples

- Arithmetic before logical
  - x + 1 > 2 || x + 1 < -3 means:
    - (x + 1) > 2 || (x + 1) < -3
- Short-circuit evaluation
  - (x >= 0) && (y > 1)
  - Be careful with increment operators!
    - (x > 1) && (y++)
- Integers as boolean values
  - All non-zero values → true
  - Zero value → false

Precedence Examples

- Some cases
  - cout<<(6 != 5) <<"\n";
  - cout<<(5 + 3 > 4)<<"\n";
  - cout<<(5 + (3 > 4))<< "\n";
  - cout<<(0 || 1 && 0) <<"\n";
  - cout<<(!0 || 0 && 0) <<"\n";
  - cout<<(3 + 4 * 7 / 12 % 5) <<"\n";

Strong Enum

- C++11 introduces **strong enums** or **enum classes**
  - Does not act like an integer
  - Avoid using enums as integer
  - Avoid name confliction
  - Can be assigned to types other than integer

```cpp
enum class EColor : char
{
    RED,
    GREEN,
    BLUE
};
```

Strong Enum

```
1  #include <iostream>
2
3  enum EColor
4  {
5    RED,
6    GREEN,
7    BLUE
8  };
9
10 enum EFruit
11 {
12   APPLE,
13   BANANA
14 };
15
16 int main()
17 {
18   EColor eColor = RED;
19   EFruit eFruit = APPLE;
20
21   if (eColor == eFruit)
22   {
23     std::cout << "color and fruit are equal" << std::endl;
24   }
25   else
26   {
27     std::cout << "color and NOT fruit are equal" << std::endl;
28   }
29 }
```

Legal

Strong Enum

```
1  #include <iostream>
2
3  enum class EColor
4  {
5    RED,
6    GREEN,
7    BLUE
8  };
9
10 enum class EFruit
11 {
12   APPLE,
13   BANANA
14 };
15
16 int main()
17 {
18   EColor eColor = EColor::RED;
19   EFruit eFruit = EFruit::APPLE;
20
21   if (eColor == eFruit)
22   {
23     std::cout << "color and fruit are equal" << std::endl;
24   }
25   else
26   {
27     std::cout << "color and NOT fruit are equal" << std::endl;
28   }
29 }
```

Illegal

Simple File I/O

- We can use cin to read from a file in a manner very similar to reading from the keyboard
- Add at the top

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

- You can then declare an input stream

```
ifstream inputStream;
```

- Next you must connect the inputStream variable to a text file on the disk.

```
inputStream.open("filename.txt");
```

- The “filename.txt” is the pathname to a file in the current directory

Simple File I/O

- Use

  inputStream >> var;

- The result is the same as using cin >> var except the input is coming from the text file and not the keyboard
- When done with the file close it with

  inputStream.close();

- Example : Consider a text file named player.txt with the following text

Display 2.10 Sample Text File, **player.txt**, to Store a Player's High Score and Name

```
100510
Gordon Freeman
```

Simple File I/O

Display 2.11 Program to Read the Text File in Display 2.10

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;
int main( )
{
    string firstName, lastName;
    int score;
    fstream inputStream;

    inputStream.open("player.txt");
    inputStream >> score;
    inputStream >> firstName >> lastName;

    cout << "Name: " << firstName << " "
         << lastName << endl;
    cout << "Score: " << score << endl;
    inputStream.close();

    return 0;
}
```

Sample Dialogue

```
Name: Gordon Freeman
Score: 100510
```

Recap : if - else

Multiway if–else Statement

SYNTAX

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
        .
        .
        .
else if (Boolean_Expression_n)
    Statement_n
else
    Statement_For_All_Other_Possibilities
```

Recap : if - else

EXAMPLE

```
if ((temperature < -10) && (day == SUNDAY))
    cout << "Stay home.";
else if (temperature < -10) //and day != SUNDAY
    cout << "Stay home, but call work.";
else if (temperature <= 0) //and temperature >= -10
    cout << "Dress warm.";
else //temperature > 0
    cout << "Work hard and play hard.";
```

The Boolean expressions are checked in order until the first `true` Boolean expression is encountered, and then the corresponding statement is executed. If none of the Boolean expressions is *true*, then the *Statement_For_All_Other_Possibilities* is executed.

Recap : switch

switch Statement

SYNTAX

```
switch (Controlling_Expression)
{
    case Constant_1:
        Statement_Sequence_1
        break;
    case Constant_2:
        Statement_Sequence_2
        break;
                .
                .
                .
    case Constant_n:
        Statement_Sequence_n
        break;
    default:
        Default_Statement_Sequence
}
```

You need not place a **break** *statement in each case. If you omit a* **break**, *that case continues until a* **break** *(or the end of the* **switch** *statement) is reached.*

Recap : switch

EXAMPLE

```
int vehicleClass;
double toll;
cout << "Enter vehicle class: ";
cin >> vehicleClass;

switch (vehicleClass)
{
    case 1:
        cout << "Passenger car.";
        toll = 0.50;
        break;
    case 2:
        cout << "Bus.";
        toll = 1.50;
        break;
    case 3:
        cout << "Truck.";
        toll = 2.00;
        break;
    default:
        cout << "Unknown vehicle class!";
  }
```

If you forget this **break**, *then passenger cars will pay $1.50.*

Recap : Loops

- 3 Types of loops in C++
- while
  - Most flexible
  - No "restrictions"
- do-while
  - Least flexible
  - Always executes loop body at least once
- for
  - Natural "counting" loop

Recap : Loops

Syntax for while and do-while Statements

A while STATEMENT WITH A SINGLE STATEMENT BODY

```
while (Boolean_Expression)
    Statement
```

A while STATEMENT WITH A MULTISTATEMENT BODY

```
while (Boolean_Expression)
{
    Statement_1
    Statement_2
        .
        .
        .
    Statement_Last
}
```

Recap : Loops

A do-while STATEMENT WITH A SINGLE-STATEMENT BODY

```
do
    Statement
while (Boolean_Expression);
```

A do-while STATEMENT WITH A MULTISTATEMENT BODY

```
do
{
    Statement_1
    Statement_2
      .
      .
      .
    Statement_Last
} while (Boolean_Expression);
```

Do not forget the final semicolon.

Recap : Loops

- for (count=0;count<3;count++)

```
{
    cout << "Hi ";   // Loop Body
}
```

- How many times does loop body execute?
- Initialization, loop condition and update all "built into" the for-loop structure!
- A natural "counting" loop

Summary

- Console I/O
- Namespace in C++
- Char array become String Class
- 4 steps in solving programming problem
- Simple File I/O
- Recap
  - Comparator
  - If-else
  - switch
  - Loops

Q & A

Thank you for your attention.

Part 2 of HW 00 is going to due on 10/1 23:59