C++ Tutorial for C Programmers

Chavalit Srisathapornphat

Summarized from Eric Brasier's Tutorial Page at http://www.4p8.com/eric.brasseur/cppcen.html

วัตถุประสงค์

- แนะนำ C++ สำหรับ C Programmers
  - ใช้ไวยากรณ์/คำสั่ง C++ ใน C code ของคุณได้
  - อ่าน C++ code ได้เข้าใจ (บ้าง)
- ไม่ใช่ tutorial สำหรับ OOP (Object-Oriented Programming)
- http://codepad.org
  - ใช้เว็บไซต์นี้เพื่อทดสอบโปรแกรม หากไม่มีคอมไพเลอร์

A new way to include libraries

- ไม่ใช้การระบุนามสกุล .h แล้ว
  - การ include แบบเดิมยังใช้ได้อยู่ แต่จะมี warning จากคอมไพเลอร์
  - Standard C libraries จะขึ้นต้นชื่อด้วย c
    - ต้องมีบรรทัด **`using namespace std;`** ที่ต้นไฟล์

```
#include <iostream>
// This is a key C++ library
#include <cmath>
// The standard C library math.h
using namespace std;

int main ()
{
   double a;

   a = 1.2;
   a = sin (a);

   cout << a << endl;

   return 0;
}
```

Another short program example

```
#include <cstdio>
#include <iostream>

using namespace std;

int main()
{
   printf("Hello C++\n");
   cout << "New way to print out\n";
   cout << "Another way to print a newline character" << endl;
}
```

// for one-line remarks

- ใช้เครื่องหมาย // เป็นการเริ่มต้น comment ได้
  - ไม่มีปิด
  - ปิด comment เองที่สิ้นบรรทัด

```
#include <iostream>
// The iostream library is often used.
using namespace std;
// Using the standard library namespace.
int main ()
// The program's main routine.
{
   double a;
   // Declaration of variable a.

   a = 456.47;
   a = a + a * 21.5 / 100;
   // A calculation.

   cout << a << endl;
   // Display the content of a.

   return 0;   // Program end.
}
```

Console input and output streams

- input จาก standard input
  - ใช้ cin >>
- output ไปยัง standard input
  - ใช้ cout <<

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

int main()
{
   int a;          // a is an integer variable
   char s [100];
   // s points to a string of max 99 characters

   cout << "This is a sample program." << endl;
   cout << endl;  // Just a line feed (end of line)

   cout << "Type your age : ";
   cin >> a;

   cout << "Type your name: ";
   cin >> s;

   cout << endl;

   cout << "Hello " << s << " you're " << a
        << " old." << endl;
   cout << endl << endl << "Bye!" << endl;
   return 0;
}
```

I/O Formatting - 1

- #include <iostream> --> cin, cout, cerr, clog
- #include <iomanip> --> formatted I/O e.g. setw, setprecision
- #include <fstream> --> file processing
- cin >> variable; -- รับอินพุตจาก stdin ไว้ในตัวแปร
- cout << expression; -- นำค่าของ expression ส่งออก stdout
  - cout << static_cast<void*>(string1) << endl;
    - พิมพ์ address ของ char string1[ ]

I/O Formatting - 2

- cout << setw(10) << left << setfill('*') << "ten" << "four" << "five" << endl;
  - setw - กำหนดความกว้างของพื้นที่แสดงผล (default แสดงชิดขวา) - ไม่คงอยู่
  - left - แสดงผลชิดซ้ายในพื้นที่ที่เตรียมไว้
  - setfill - กำหนดตัวอักษรที่ fill เติมในช่องว่างที่เหลือ (default เป็น space)
  - endl - ขึ้นบรรทัดใหม่
- cout << hex << number << endl;
  - setbase(10) - - 8, 10, 16 หรือ oct, dec, hex และจะคงอยู่จนกว่าจะเปลี่ยน
- cout << setprecision(3) << a << endl;
- cout << showpoint << setprecision(3) << setw(10) << a << endl;
  - setprecision( ) - กำหนดจำนวนตำแหน่งทศนิยม - คงอยู่
  - showpoint - แสดง 0 ทางขวาสุดหลังจุดทศนิยม - คงอยู่

Variables can be declared anywhere inside the code

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

int main ()
{
   double a;

   cout << "Hello, this is a test program." << endl;
   cout << "Type parameter a: ";
   cin >> a;
   a = (a + 1) / 2;

   double c;

   c = a * 5 + 1;
   cout << "c contains      : " << c << endl;

   int i, j;

   i = 0;
   j = i + 1;
   cout << "j contains      : " << j << endl;
   return 0;
}
```

ใช้ทำ ให้ code ดูสะอาดขึ้น

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

int main ()
{
   double a;

   cout << "Type a number: ";
   cin >> a;
   {
      int a = 1;
      a = a * 10 + 4;
      cout << "Local number: " << a << endl;
   }
   cout << "You typed: " << a << endl;

   return 0;
}
```

A variable can be initialised by a calculation involving other variables

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

int main ()
{
   double a = 12 * 3.25;
   double b = a + 1.112;

   cout << "a contains: " << a << endl;
   cout << "b contains: " << b << endl;

   a = a * 2 + b;

   double c = a + b * a;

   cout << "c contains: " << c << endl;

   return 0;
}
```

Declare a variable to be local to a loop

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

int main ()
{
   int i;                          // Simple declaration of i
   i = 487;

   for (int i = 0; i < 4; i++)     // Local declaration of i
   {
      cout << i << endl;           // This outputs 0, 1, 2 and 3
   }

   cout << i << endl;              // This outputs 487

   return 0;
}
```

Global variable accessed inside the scope of local variable with the same name

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

double a = 128;

int main ()
{
   double a = 256;

   cout << "Local a:  " << a   << endl;
   cout << "Global a: " << ::a << endl;

   return 0;
}
```

Reference (1)

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

int main ()
{
   double a = 3.1415927;
   double &b = a;          // b is a reference to a

   b = 89;

   cout << "a contains: " << a << endl;
   // Displays 89.

   return 0;
}
```

Reference (2)

- Reference จะไม่สามารถถูกเปลี่ยนค่าได้
  - กำหนดค่าได้ครั้งแรก ครั้งเดียว
- Reference มักใช้ในการส่งค่าตัวแปรผ่านฟังก์ชัน (call by reference)

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

void change (double &r, double s)
{
   r = 100;
   s = 200;
}

int main ()
{
   double k, m;

   k = 3;
   m = 4;

   change (k, m);
   cout << k << ", " << m << endl;
   // Displays 100, 4
   return 0;
}
```

Variables in other namespaces

- namespace ใช้กำหนดขอบเขตของ code ให้เป็นกลุ่มเดียวกัน
- อ้างอิงตัวแปรที่กำหนดไว้ใน namespace อื่นได้
  - ใช้เครื่องหมาย ::

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

namespace first
{
   int a;
   int b;
}

namespace second
{
   double a;
   double b;
}

int main ()  {
   first::a = 2;
   first::b = 5;
   second::a = 6.453;
   second::b = 4.1e4;
   cout << first::a + second::a << endl;
   cout << first::b + second::b << endl;
   return 0;
}
```

Inline functions

- กรณี function ง่าย ๆ สั้น ๆ แค่คำนวณแล้ว return ค่า ไม่ได้มี for loop ยุ่งยาก การใช้ inline function ทำให้ทำงานเร็วขึ้น
  - ข้อเสีย คือ ได้ executable file ใหญ่ขึ้น

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

inline double hypothenuse (double a, double b)
{
   return sqrt (a * a + b * b);
}

int main ()
{
   double k = 6, m = 9;

   // Next two lines produce exactly
   // the same code:

   cout << hypothenuse (k, m) << endl;
   cout << sqrt (k * k + m * m) << endl;

   return 0;
}
```

Exceptions*

- ใช้ try เพื่อดักจับข้อผิดพลาดที่อาจเกิดขึ้นขณะรันโปรแกรม ด้วยคำสั่ง throw
- แล้ว catch เป็นการระบุ ให้รันคำสั่งหลังจากเกิดการ throw ขึ้น

```
using namespace std;
#include <iostream>
#include <cmath>
int main ()  {
   int a, b;
   cout << "Type a number: ";
   cin >> a;
   cout << endl;
   try  {
      if (a > 100) throw 100;
      if (a < 10)  throw 10;
      throw a / 3;
   }
   catch (int result)  {
      cout << "Result is: " << result << endl;
      b = result + 1;
   }
   cout << "b contains: " << b << endl;
   cout << endl;

   // another example of exception use:
   char zero []     = "zero";
   char pair []     = "pair";
   char notprime [] = "not prime";
   char prime []    = "prime";
   try  {
      if (a == 0) throw zero;
      if ((a / 2) * 2 == a) throw pair;
      for (int i = 3; i <= sqrt (a); i++)   {
         if ((a / i) * i == a) throw notprime;
      }
      throw prime;
   }
   catch (char *conclusion)   {
      cout << "The number you typed is "<< conclusion << endl;
   }
   cout << endl;
   return 0;
}
```

Default parameters

- กำหนดค่าเริ่มต้นให้กับ parameters ได้ กรณีที่ caller ไม่ได้ส่งค่ามาให้ ก็จะได้รับค่านี้

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

double test (double a, double b = 7)
{
   return a - b;
}

int main ()
{
   cout << test (14, 5) << endl;
   // Displays 14 - 5
   cout << test (14) << endl;
   // Displays 14 - 7
   return 0;
}
```

Function overload

- ฟังก์ชันมีชื่อเหมือนกันได้
  - ถ้า list ของ parameters แตกต่างกัน
  - มัก ใช้ในกรณีที่ต้องการสร้าง functions ให้ทำงานอย่างเดียวกันกับ typeห ของ parameters ที่ต่างชนิดกัน

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

double test (double a, double b)
{
   return a + b;
}

int test (int a, int b)
{
   return a - b;
}

int main ()
{
   double   m = 7,  n = 4;
   int      k = 5,  p = 3;

   cout << test(m, n) << " , "
        << test(k, p) << endl;

   return 0;
}
```

Define symbolic operators for new data types

- Operator overload
  - กำหนดให้ operators พื้นฐานที่มีอยู่ ทำหน้าที่ที่ต้องการกับชนิดของข้อมูลใหม่

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

struct vector  {
   double x;
   double y;
};

vector operator * (double a, vector b)  {
   vector r;
   r.x = a * b.x;
   r.y = a * b.y;
   return r;
}

int main ()  {
   vector k, m;  // No need to type "struct vector"
   k.x =  2;     // To be able to write
   k.y = -1;     // k = vector (2, -1)
                 // see chapter 19.

   m = 3.1415927 * k;  // Magic!

   cout << "(" << m.x << ", " << m.y << ")" << endl;
   return 0;
}
```

Operator overload - 1

- Operator overload
  - Define symbolic operators for new data types
  - กำหนดให้ operators พื้นฐานที่มีอยู่ ทำหน้าที่ที่ต้องการกับชนิดของข้อมูลใหม่

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

struct vector  {
   double x;
   double y;
};

vector operator * (double a, vector b)  {
   vector r;
   r.x = a * b.x;
   r.y = a * b.y;
   return r;
}

int main ()  {
   vector k, m;  // No need to type "struct vector"
   k.x =  2;     // To be able to write
   k.y = -1;     // k = vector (2, -1)
                 // see chapter 19.

   m = 3.1415927 * k;  // Magic!

   cout << "(" << m.x << ", " << m.y << ")" << endl;
   return 0;
}
```

Operator overload - 2

- Operator overload
  - << output stream operator ก็สามารถถูก overload ได้
  - เพื่อส่ง output ออกในรูปแบบที่ต้องการ

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

struct vector
{
   double x;
   double y;
};

ostream& operator << (ostream& o, vector a)
{
   o << "(" << a.x << ", " << a.y << ")";
   return o;
}

int main ()  {
   vector a;

   a.x = 35;
   a.y = 23;

   cout << a << endl;     // Displays (35, 23)

   return 0;
}
```

Template functions - 1

- เขียน function ที่เป็น template (ต้นแบบ) ไว้ โดยไม่ต้องกำหนด type ของ parameters (และ return value)
- C++ จะสร้าง (เติมเต็ม) function ที่สมบูรณ์ให้เอง
  - จากตัวอย่าง C++ จะสร้าง
    - **`int minimum (int a, int b)`**
    - **`double minimum (double a, double b)`**
- หากมี operators ใดที่ต้องทำงานกับ type/class ของเราเอง เราจะต้อง overload ไว้ให้ครบถ้วน

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

template <class ttype> ttype minimum (ttype a, ttype b) {
   ttype r;

   r = a;
   if (b < a) r = b;
   return r;
}

int main () {
   int i1, i2, i3;
   i1 = 34;
   i2 = 6;
   i3 = minimum (i1, i2);
   cout << "Most little: " << i3 << endl;

   double d1, d2, d3;
   d1 = 7.9;
   d2 = 32.1;
   d3 = minimum (d1, d2);
   cout << "Most little: " << d3 << endl;
   cout << "Most little: " << minimum (d3, 3.5) << endl;
   return 0;
}
```

Template functions - 2

- จำนวนของชนิดข้อมูล ใน template มีได้มากกว่าหนึ่ง
- และไม่จำเป็นต้องเป็น template ทุกตัว
  - ชนิดข้อมูลพื้นฐานก็ยังคงใช้ได้

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

template <class type1, class type2>
type1 minimum (type1 a, type2 b) {
   type1 r, b_converted;
   r = a;
   b_converted = (type1) b;
   if (b_converted < a) r = b_converted;
   return r;
}

int main (){
   int i;
   double d;

   i = 45;
   d = 7.41;

   cout << "Most little: " << minimum (i, d) << endl;
   cout << "Most little: " << minimum (d, i) << endl;
   cout << "Most little: " << minimum ('A', i) << endl;

   return 0;
}
```

new/delete - malloc/free

- new/delete
  - ใช้แทน malloc และ free ได้
- new[], delete[]
  - ทำงานกับ array

```
#include <iostream>   #include <cstring>   using namespace std;
int main ()  {
   double *d;
   d = new double;
   *d = 45.3;   cout << "Type a number: ";
   cin >> *d;
   *d = *d + 5;
   cout << "Result: " << *d << endl;
   delete d;

   d = new double[15];
   d[0] = 4456;
   d[1] = d[0] + 567;
   cout << "Content of d[1]: " << d[1] << endl;
   delete [] d;

   int n = 30;
   d = new double[n];
   for (int i = 0; i < n; i++)
      d[i] = i;
   delete [] d;

   char *s;
   s = new char[100];
   strcpy (s, "Hello!");
   cout << s << endl;
   delete [] s;
   return 0;
}
```

Add methods to class/struct

- เราสามารถสร้าง function เพื่อให้ทำงานกับข้อมูลใน struct ได้ เรียกว่า method
- เรียกใช้ได้โดยอ้างถึง ชื่อตัวแปรของ struct ดังกล่าว ตามด้วย . ตามด้วยชื่อ method( )
- ในการโปรแกรมแบบ OOP
  - เรียก a ว่า instance คือ เป็น object ที่สร้างจากต้นแบบใน class vector
  - class คือ struct ที่ซ่อน data เอาไว้ภายใน

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

struct vector  {
   double x;
   double y;

   double surface ()  {
      double s;
      s = x * y;
      if (s < 0) s = -s;
      return s;
   }
};

int main ()  {
   vector a;

   a.x = 3;
   a.y = 4;
   cout << "The surface of a: " << a.surface() << endl;
   return 0;
}
```

Class with public data

- data ของ class จะถูกอ้างถึงได้จากโปรแกรมที่อยู่นอก class ก็ต่อเมื่อประกาศภายใต้ public:
  - โดยทั่วไป ไม่ควรใช้ public data

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

class vector {
public:
   double x;
   double y;

   double surface ()  {
      double s;
      s = x * y;
      if (s < 0) s = -s;
      return s;
   }
};

int main () {
   vector a;

   a.x = 3;
   a.y = 4;

   cout << "The surface of a: " << a.surface() << endl;
   return 0;
}
```

Constructor/Destructor-1

- Constructor เป็น method พิเศษที่จะถูกเรียกเมื่อเริ่มสร้าง object
  - มีชื่อเดียวกับ class
- Destructor ถูกเรียกเมื่อ object ถูกทำลาย
  - มีชื่อเดียวกับ class นำหน้าด้วย ~

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

class vector
{
public:

   double x;
   double y;

   vector ()                        // same name as class
   {
      x = 0;
      y = 0;
   }

   vector (double a, double b)
   {
      x = a;
      y = b;
   }

};

int main ()
{
   vector k;                        // vector () is called

   cout << "vector k: " << k.x << ", " << k.y << endl << endl;

   vector m (45, 2);                // vector (double, double) is called

   cout << "vector m: " << m.x << ", " << m.y << endl << endl;

   k = vector (23, 2);              // vector created, copied to k, then erased

   cout << "vector k: " << k.x << ", " << k.y << endl << endl;

   return 0;
}
```

Constructor/Destructor-2

- ไม่ควร overload constructor หากไม่จำเป็น
- ให้ใช้ default parameters แทน
- destructor ปกติไม่จำเป็นต้องมี
- ต้องมี ในกรณีที่มีการ new หรือ malloc เมื่อมีการสร้าง object

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

class vector
{
public:

   double x;
   double y;

   vector (double a = 0, double b = 0)
   {
      x = a;
      y = b;
   }
};

int main ()
{
   vector k;
   cout << "vector k: " << k.x << ", " << k.y << endl << endl;

   vector m (45, 2);
   cout << "vector m: " << m.x << ", " << m.y << endl << endl;

   vector p (3);
   cout << "vector p: " << p.x << ", " << p.y << endl << endl;

   return 0;
}
```

Constructor/Destructor-3

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

class person
{
public:

   char *name;
   int age;

   person (char *n = "no name", int a = 0)
   {
      name = new char [100];                  // better than malloc!
      strcpy (name, n);
      age = a;
      cout << "Instance initialized, 100 bytes allocated" << endl;
   }

   ~person ()                                 // The destructor
   {
      delete name;                            // instead of free!

                                              // delete [] name would be more
                                              // academic but it is not vital
                                              // here since the array contains
                                              // no C++ sub-objects that need
                                              // to be deleted.

      cout << "Instance going to be deleted, 100 bytes freed" << endl;
   }
};
```

```
int main ()
{
    cout << "Hello!" << endl << endl;

    person a;
    cout << a.name << ", age " << a.age << endl << endl;

    person b ("John");
    cout << b.name << ", age " << b.age << endl << endl;

    b.age = 21;
    cout << b.name << ", age " << b.age << endl << endl;

    person c ("Miki", 45);
    cout << c.name << ", age " << c.age << endl << endl;

    cout << "Bye!" << endl << endl;

   return 0;
}
```

Overload operator [ ]

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

class array
{
public:
   int size;
   double *data;

   array (int s)
   {
      size = s;
      data = new double [s];
   }

   ~array ()
   {
      delete [] data;
   }

   double &operator [] (int i)
   {
      if (i < 0 || i >= size)
      {
         cerr << endl << "Out of bounds" << endl;
         exit (EXIT_FAILURE);
      }
      else return data [i];
   }
};
```

```
int main ()
{
   array t (5);

   t[0] = 45;                          // OK
   t[4] = t[0] + 6;                    // OK
   cout << t[4] << endl;               // OK

   t[10] = 7;                          // error!

   return 0;
}
```

Copy constructor

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

class person
{
public:

   char *name;
   int age;

   person (char *n = "no name", int a = 0)
   {
      name = new char[100];
      strcpy (name, n);
      age = a;
   }

   person (const person &s)       // The COPY CONSTRUCTOR
   {
      name = new char[100];
      strcpy (name, s.name);
      age = s.age;
   }

   person& operator= (const person &s)  // overload of =
   {
      strcpy (name, s.name);
      age = s.age;
      return *this;
   }

   ~person ()
   {
      delete [] name;
   }
};
```

```
void modify_person (person& h)
{
   h.age += 7;
}

person compute_person (person h)
{
   h.age += 7;
   return h;
}
```

```
int main ()
{
   person p;
   cout << p.name << ", age " << p.age << endl << endl;
   // output: no name, age 0

   person k ("John", 56);
   cout << k.name << ", age " << k.age << endl << endl;
   // output: John, age 56

   p = k;
   cout << p.name << ", age " << p.age << endl << endl;
   // output: John, age 56

   p = person ("Bob", 10);
   cout << p.name << ", age " << p.age << endl << endl;
   // output: Bob, age 10

   // Neither the copy constructor nor the overload
   // of = are needed for this operation that modifies
   // p since just the reference towards p is passed to
   // the function modify_person:
   modify_person (p);
   cout << p.name << ", age " << p.age << endl << endl;
   // output: Bob, age 17

   // The copy constructor is called to pass a complete
   // copy of p to the function compute_person. The
   // function uses that copy to make its computations
   // then a copy of that modified copy is made to
   // return the result. Finally the overload of = is
   // called to paste that second copy inside k:
   k = compute_person (p);
   cout << p.name << ", age " << p.age << endl << endl;
   // output: Bob, age 17
   cout << k.name << ", age " << k.age << endl << endl;
   // output: Bob, age 24

   return 0;
}
```