More on Basic C++ Programming

Hsuan-Tien Lin

Dept. of CSIE, NTU

March 6, 2012

What We Have Done (Chapter 1)

- C++ (in class): I/O, new/delete, class, operator overloading, access control, variable declaration, scope
- C++ (in homework): constructor
- C++ (in reading): constant, typedef, namespace, expression, casting, control flow, functions, inline, C++ Programming

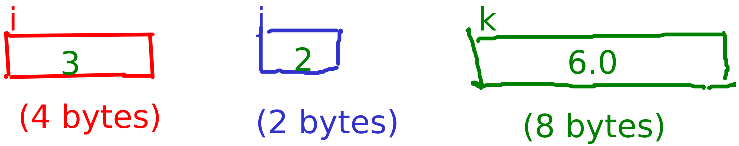
What We Will Do (Chapter 1)

- pointers and references
- template and STL

What We Will Not Do (Chapter 2)

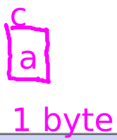
- Object Oriented Programming

Fun Time (1)



What happens in memory?

```
int i;
short j;
double k;
char c = 'a';
i = 3; j = 2;
k = i * j;
```



Life Cycle of a (Primitive) Variable

- declared and created

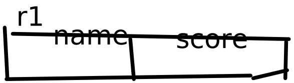
```
int count;
```

- used and modified

```
count += 1;
```

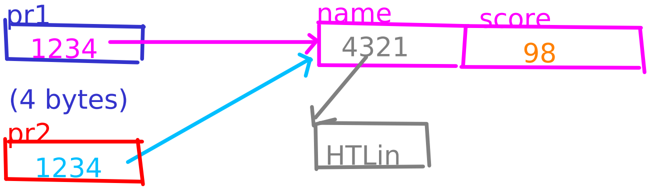
- destroyed
  –automatically (when out of scope)

Fun Time (2)



What happens in memory?

```
class Record{ public: char* name; int score; }    //public

Record* pr1;                          Record r1;
Record* pr2;
pr1 = new Record();
pr2 = pr1;  //how many records are there?
pr1->name = "HTLin";
pr2->score = 98;
```



Fun Time (3)

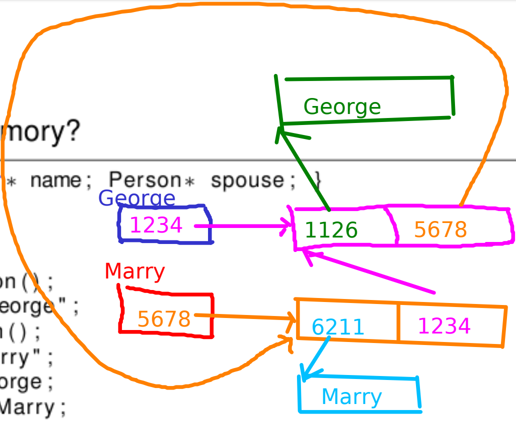
What happens in memory?

```
class Record{ char* name; int score; }

Record r1;
Record r2;
r2 = r1;  //how many records are there?
r1.name = "HTLin";
r2.score = 98;
```

Fun Time (4)

What happens in memory?

```
class Person{ char* name; Person* spouse; }

Person* George;
Person* Marry;
George = new Person();
George->name = "George";
Marry = new Person();
Marry->name = "Marry";
Mary->spouse = George;
George->spouse = Marry;
```



Fun Time (5)

What happens in memory?

```
class Person{ char* name; Person* spouse; }

Person* George;
George = new Person();
George->name = "George";
George->spouse = new Person();
George->spouse->name = "Marry";
George->spouse = new Person();
George->spouse->name = "Lisa";
```

Life Cycle of an Object Instance (C++)

- pointer declared

```
Record* pr;
```

- instance created

```
pr = new Record();
```

- used and modified

```
cout << pr->name;
```

- destroyed

```
delete pr;
```

Pointer: Key Point

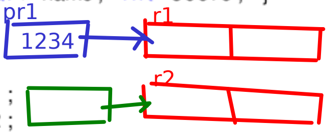
- a instance occupies a space in the memory;
  老太太住在屏東一個房子裡面
- pointer: the “address” to the instance;
  用"海角七號"就可以找到老太太
- pointer variable: holds the pointer;
  一個"信封"，上面寫著海角七號
- any operation on the instance goes thru the pointer;
  要請老太太"回憶"時，拿個信封上寫"海角七號"，接著寫"回憶"，就會使命必達了

```
老人*  信封= new 老人(老太太資料);
信封->回憶();
```

Fun Time (6)

What happens in memory?

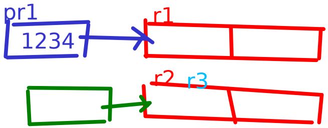
```
class Record{ char* name; int score; }

Record r1;
Record r2;
Record* pr1 = &r1;
Record* pr2 = &r2;
r2 = r1;
pr2 = pr1;
```



Fun Time (7)

What happens in memory?

```
 1  class Record{ char* name; int score; }
 2
 3  Record r1;
 4  Record r2;
 5  Record* pr1 = &r1;
 6  Record* pr2 = &r2;
 7  Record& r3; //reference
 8  r2 = r1;
 9  pr2 = pr1;
10  r3 = r2;
```



note: line 7 and 10 needs to be one line
--- reference cannot be uninitialized

if (r3 = r1) is added afterwards, it means copy the contents of r1 to r3 (alias of r2).

Fun Time (7)

http://yosefk.com/c++fqa/ctors.html#fqa-10.2

What happens in memory?

```
class Record{ char* name; int score; }

  Record r1; //declare a record instance r1
                 //(constructor called if there is one)

  Record myfunc(int param1, double param2);
                 //declare a function prototype with two parameters
                 //and returning a Record
```

```
Record r2(); //declare a function prototype with zero parameters
r2.score++; //and returning a Record
//wrong code because r2 is a function, not an instance
//r2 is a function, hence no constructor call
```

Pointer/Reference: Key Point

- pointer: the “address” to the instance;
  用"海角七號"就可以找到老太太
- reference: the “other name” of the instance;
  用"小島友子"也可以稱呼老太太
- the “original” variable: holds the reference;
  一張"身份證"，上面寫著(住海角七號)
- pointer variable: holds the pointer;
  一個"信封"，上面寫著海角七號
- reference variable: holds the reference;
  一張"名片"，上面寫著(住海角七號)

```
老人  老太太;
老人*  信封= &老太太; //存放住址
老人& 小島友子= 老太太; //用別名
      老太太.回憶();
      信封->回憶();
      小島友子.回憶();
```

Argument/Parameter

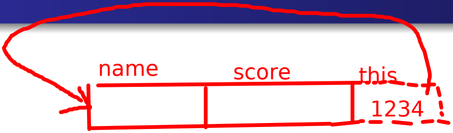
```
Record htlin;
htlin.score = 59;
change1(htlin, 100);
change2(&htlin, 100);
change3(htlin, 100);

void change1(Record r1, int score){
  r1.score = score;
}
void change2(Record* pr1, int score){
  pr1->score = score;
}
void change3(Record& r1, int score){
  r1.score = score;
}
```

Take home study:
which change(s) would work correctly?

Reference Argument/Parameter: Key Point

argument ⇒ parameter: by copying
(unless specifying reference), **same for return value**

this (1/1)



```
class Record{
  int score;
  void set_to(int score){ this->score = score; }
  void adjust_score{ this->set_to(score+10); }
}
```

- which score? which set_to?
- `this`: my (the object's)

this: Key Point

`this`: the pointer pointing to the object itself

template in C++

integer polynomials, double polynomials, fraction polynomials, ...

```
class intArray{
  int arr[100];

  //...
};

class DoubleArray{
  double arr[100];

  //..
};
```

```
template<typename T, typename S>
class pair{
  T first;
  S second;
};

pair<int, double> p;
```

"typename" can also be replaced by "class"

Standard Template Library (STL)

- e.g. `vector`: dynamically growing array (will discuss more soon)

```
#include <vector>
using namespace std;
vector<int> intarray;

intarray.resize(20); intarray[3] = 5;
```

Standard Template Library (STL)

- e.g. `vector`: dynamically growing array (will discuss more soon)

```
#include <vector>
using namespace std;
vector<int> intarray;

intarray.resize(20); intarray[3] = 5;
```