C++ Class and STL Review

임종우 (Jongwoo Lim)

What we have learned so far...

- C++ struct and class:
    - Member variables and functions.
    - Access control - public and private.
- Memory management.
- Pointer, reference, and const.
- C++ STL:
    - vector, set, map, string, etc.
- Multi-file project.
    - Compilation and linking.
    - Header and source files.

Declaration vs. Definition

```
// Function declarations.
int MyFunction(int a, int b);

void DoEverything(void);


// Class declarations.
struct StudentInfo;

class StringVector;
```

```
// Function definitions.
int MyFunction(int a, int b) {
  return a + b;
}

void DoEverything(void) {
  std::string str;
  std::cin >> str;
  ...
}


// Class (and its member function) definitions.
struct StudentInfo {
  int id;
  std::string name;
};

class StringVector {
 public:
  StringVector() {}              // Def.
  int MemberFunctionDecl();   // Decl.
  int MemberFunctionDef() { return 10; }
};

int StringVector::MemberFunctionDecl() {
  ...
}
```

Declaration vs. Definition

- Declaration only provides the name and type info.
- Definition gives the content of the function or class.

- Header files can have any declarations, and class definitions.
  - `#ifndef + #define` to ensure unique definitions.
- Source files can have both declarations and definitions.
  - `#include` statement is just replaced with the file's content.

Structures and Classes

- Members of struct : 'has-a' relation.
  - Member variable : 'has-a-property'
  - Member function : 'has-a-functionality'

```
struct StudentInfo {
  int id;
  std::string name;
  std::vector<int> homework_scores;
};

class StringVector {
 public:
  StringVector() {}
  int AddString(const std::string& str);
  int RemoveString(const std::string& str);
  int GetNumString() const;

 private:
  std::vector<std::string> strings_;
};
```

Structures and Classes

- Instantiation : making a memory instance of the class.
  - Member functions are called on class instances.
  - Constructor : the function executed when instantiated.
  - Destructor : the function executed when destroyed.

```
class StringVector {  // A class type.
 public:
  StringVector() {}
  int AddString(const std::string& str);
  int RemoveString(const std::string& str);
  int GetNumString() const;

 private:
  std::vector<std::string> strings_;
};

int main() {
  StringVector vec;  // An instance of the class StringVector.
  vec.AddString("hello world");
  return 0;
}
```

C++ Class

- Information hiding : hide unnecessary information from users.
  - Data integrity.
  - Interface vs. Implementation.
- private vs. public
  - Public members are visible to everyone.
  - Private members are only visible to its member functions.

```
class StringVector {  // A class type.
 public:
  StringVector() {}
  int AddString(const std::string& str);
  int RemoveString(const std::string& str);
  int GetNumString() const;

 private:
  std::vector<std::string> strings_;
};
```

Memory Management

- Allocate and deallocate memory (in C).
    - `malloc() / free()`
- Create an instance of a class and destroy it.
    - `new / delete`
- Create an array of instances of a class and destroy it.
    - `new [] / delete[]`

```
class MyClass { ... };

int* int_array = (int*) malloc(sizeof(int) * 10);
for (int i = 0; i < 10; ++i) int_array[i] = i;
free(int_array);

MyClass *ptr = new MyClass;
MyClass *array = new MyClass[10];
for (int i = 0; i < 10; ++i) array[i] = *ptr;
delete ptr;
delete[] array;
```

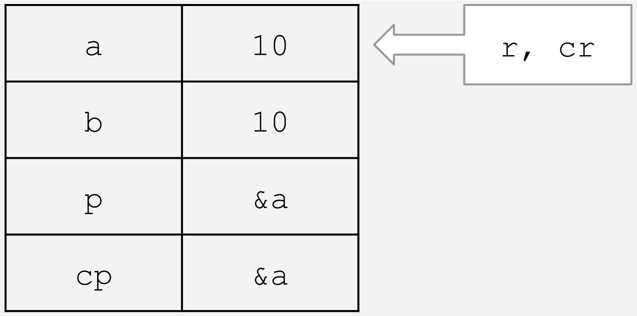
Pointer and Reference

- Pointer : represents a memory location.
- Reference : represents an object (instance of a class).
- Const-ness : the content does not change by operations.
- Const reference : used often in parameter passing.

```
class MyClass { ... };

int MyFunction(const MyClass& arg, int i);

int* int_array = (int*) malloc(sizeof(int) * 10);
// ... Initialize int_array.
const int* min_ptr = NULL;
for (int* p = int_array; p != int_array + 10; ++p) {
  if (!min_ptr || *min_ptr > *p) min_ptr = p;
}
if (min_ptr) cout << "min found: " << *min_ptr << endl;
const int& min_ref = *min_ptr;

MyClass *my_array = new MyClass[10];
MyClass& my_first = my_array[0];
int ret = MyFunction(*(my_array + 5), int_array[0]);
```

Local Variable, Pointer, Reference

```
int a = 10;
int b = a;

int* p = &a;
const int* cp = &a;

int& r = a;
const int& cr = a;


a = 20;     // a: 20, b: 10, p: &a, *p: 20, cp: &a, *cp: 20, r: 20 ,cr: 20.
b = 30;     // a: 20, b: 30, p: &a, *p: 20, cp: &a, *cp: 20, r: 20 ,cr: 20.

*p = 10;    // a: 10, b: 30, p: &a, *p: 10, cp: &a, *cp: 10, r: 10 ,cr: 10.
*cp = 0;    // Error!
r = 40;     // a: 40, b: 30, p: &a, *p: 40, cp: &a, *cp: 40, r: 40 ,cr: 40.
cr = 0;     // Error!

p = &b;     // a: 40, b: 30, p: &b, *p: 30, cp: &a, *cp: 40, r: 40 ,cr: 40.
*p = 50;    // a: 40, b: 50, p: &b, *p: 50, cp: &a, *cp: 40, r: 40 ,cr: 40.

int** pp = &p;
*pp = &a;   // pp: &p, p: &a, *p: 40
*pp = &b;   // pp: &p, p: &b, *p: 50
```



a	10
b	10
p	&a
cp	&a

C++ Standard Template Library

- `namespace std`
- `cin, cout` : streaming input / output.
- `string` : a string class.
- `vector` : an array of a class.
- `set` : an unordered set of elements.
- `map` : a key-value pair mapping.
- Iterator : represents a position in the container, like a pointer.
  - Most containers have `begin`(), `end`().
  - Usually two types, `iterator` and `const_iterator`.

cin, cout	**operator<<**, **operator>>**, **endl**
string	**string**(const char*) string& **operator=**(const string& s) const char* **c_str**() const size_t **size**() const, size_t **length**() const bool **empty**() const size_t **find**(const string& s, size_t pos = 0) const string **substr**(size_t pos = 0, size_t n = npos) const char& **operator[]**(size_t pos), const char& **operator[]**(size_t pos) const [global] string **operator+**(const string& lhs, const string& rhs) string& **operator+=**(const string& s) void **resize**(size_t n) [global] bool **operator==**(const strign& l, const string& r), **!=**, **<**, **>**, **<=**, **>=**
vector<T>	**vector**(), **vector**(size_t n, const T& val = T()), **vector**(const vector& x) vector& **operator=**(const vector& x) T& **operator[]**(size_t i), const T& **operator[]**(size_t i) const size_t **size**() const bool **empty**() const void **resize**(size_t n, T c = T()) void **reserve**(size_t n) void **push_back**(const T& x) void **pop_back**() iterator **begin**(), const_iterator **begin**() const, **rbegin**() iterator **end**(), const_iterator **end**() const, **rend**() iterator **insert**(iterator pos, const T& x) iterator **erase**(iterator pos), iterator **erase**(iterator first, iterator last) T& **front**(), const T& **front**() const T& **back**(), const T& **back**() const void **clear**() void **swap**(vector& x) [global] bool **operator==**(const strign& l, const string& r), **!=**, **<**, **>**, **<=**, **>=**

set<T>	**set**(), **set**(const set& x)
	set& **operator=**(const set& s)
	size_t **size**() const
	bool **empty**() const
	size_t **count**(const T& x) const
	iterator **begin**(), const_iterator **begin**() const, **rbegin**()
	iterator **end**(), const_iterator **end**() const, **rend**()
	iterator **find**(const T& x), const_iterator **find**(const T& x) const
	pair<iterator, bool> **insert**(const T& x)
	size_t **erase**(const T& x)
	void **erase**(iterator pos), void **erase**(iterator first, iterator end)
	void **clear**()
	void **swap**(set& x)
	[global] bool **operator==**(const strign& l, const string& r), **!=**, **<**, **>**, **<=**, **>=**
map<K,V>	**map**(), **map**(const map& x)
	map& **operator=**(const map& s)
	size_t **size**() const
	bool **empty**() const
	size_t **count**(const K& x) const
	iterator **begin**(), const_iterator **begin**() const, **rbegin**()
	iterator **end**(), const_iterator **end**() const, **rend**()
	iterator **find**(const K& x), const_iterator **find**(const T& x) const
	pair<iterator, bool> **insert**(const pair<const K, V>& x)
	V& **operator[]**(const K& x)
	size_t **erase**(const K& x)
	void **erase**(iterator pos), void **erase**(iterator first, iterator end)
	void **clear**()
	void **swap**(map& x)
	[global] bool **operator==**(const strign& l, const string& r), **!=**, **<**, **>**, **<=**, **>=**