



고급프로그래밍

고급문법 1

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory
Kwangwoon University
(02-940-5625, mhjeong@kw.ac.kr)

Schedule

주차	주제		과제	퀴즈
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++	C++ 프로그래밍의 기본 (3), 클래스와 객체 (4)	1	1
3		객체생성과 사용 (5)	2	2
4		함수와 참조 (6, 3/26), 복사 생성자와 함수중복(7)	3	3
5		static, friend, 연산자중복 (8, 4/2), 연산자중복 상속(9)	4	4
6		상속 (10, 4/9), 가상함수와 추상클래스 (11)		5
7		템플릿과 STL (12, 4/16), 입출력(13)	5	
8		중간고사		
9		파일 입출력(14), 예외처리 및 C 사용(15)		6
10		람다식(16, 5/7) , 멀티스레딩(17, 5/9)	6	7
11		멀티스레딩(18, 5/14), 고급문법(19, 5/16)		8
12		고급문법		
13	병렬프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15		기말고사		

오늘의 학습내용

- 고급문법
 - Sorting ++
 - 스마트 포인터
 - csv 파일 읽기

Sorting ++

Index with Sort

- STL 컨테이너의 원소를 **sorting** 할 때 인덱스도 함께 **sorting**

```
:
#include <numeric>      // std::iota
using namespace std;


template <class T>
vector<int> index_with_sort(vector<T>& v)
{
    vector<int> index(v.size());
    iota(index.begin(), index.end(), 0);

    sort(index.begin(), index.end(),
          [&v](int t1, int t2) { return v[t1] < v[t2];});
    sort(v.begin(), v.end(),
          [&v](T v1, T v2) { return v1 < v2; });

    return index;
}
```

```
template<class ForwardIt, class T>
void iota(ForwardIt first, ForwardIt last, T value) {
    while(first != last) {
        *first++ = value;
        ++value;
    }
}
```

실습: Sort with Index

```
template <class T>
void sort_with_index(vector<T>& v, vector<int> index)
{

}
```

```
int main()
{
    vector<char>    src;
    vector<double>  des;

    src.push_back('a');
    src.push_back('d');
    src.push_back('b');
    src.push_back('c');

    des.push_back(0.1);
    des.push_back(0.4);
    des.push_back(0.2);
    des.push_back(0.3);

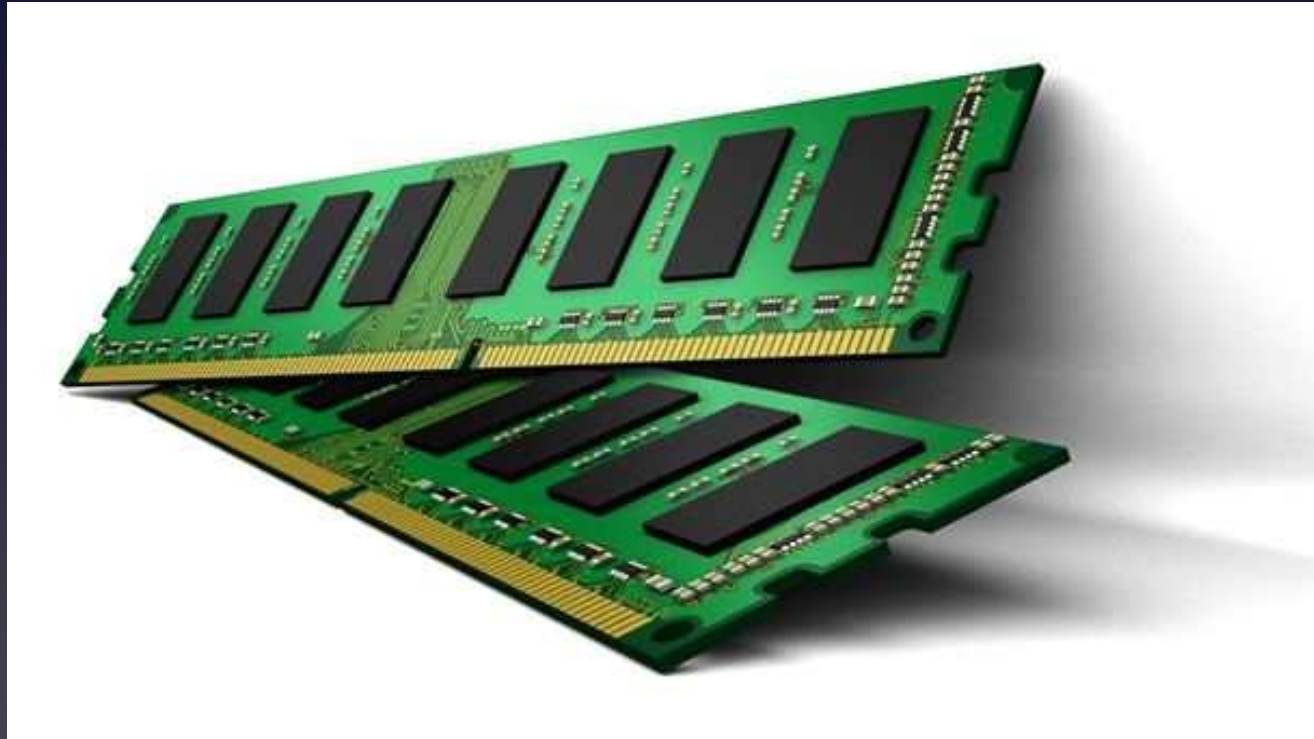
    vector<int> idx = index_with_sort(src);
    sort_with_index(des, idx);

    for(auto& item : src)
        cout << item << endl;
    for(auto& item : des)
        cout << item << endl;
}
```

Smart Pointer

스마트 포인터

- 메모리 관리는 C++ 버그의 원천
- 대부분의 메모리 관련 버그는 메모리 동적 할당과 포인터에서 발생
 - 메모리 누수(memory leak) : 메모리 해제 못함(부주의, 예외처리)
 - 메모리의 중복 해제



예제

- 예외발생으로 인해 동적할당 된 메모리 해제 못함

```
void doSomething( ) {  
    throw 1;  
}  
  
void func( )  
{  
    Simple* ptr = new Simple( );  
    doSomething( );  
    delete ptr;  
}  
  
int main() {  
    try{  
        func( );  
    }  
    catch (int i) {  
        cout << "Exception !" <<endl;  
    }  
}
```

스마트 포인터

■ 스마트 포인터 요건

- STACK에 두어 지역변수로서 관리 - 생성영역(스코프)을 벗어나면 자동 해제
 - ⇒ STACK의 안정성과 HEAP의 유연성 융합
- 포인터를 복사하여 여러 곳에서 사용될 경우, 마지막에 한번 메모리 해제
 - ⇒ 레퍼런스 카운팅
- `auto_ptr` 표준 폐기
- `unique_ptr` 레퍼런스 카운팅 X
- `shared_ptr` 레퍼런스 카운팅 O

shared_ptr

- 동적 할당 객체는 `STACK` 변수로 `share_ptr`에 저장하는 것이 바람직함

```
void leaky( ) //메모리는 해제되지 않음
{
    Simple* ptr = new Simple( );
    ptr->go( );
}
```



```
#include <memory>

void not_leaky( )
{
    shared_ptr<Simple> spSimple( new Simple() );
    spSimple->go( );    //포인터 변수 처럼 사용
}
```

shared_ptr 생성 방법

- shared_ptr 생성방법

```
auto sp1 = make_shared<Simple>("Simple is Good");  
shared_ptr<Simple> sp2(new Simple("Simple is Good"));  
shared_ptr<Simple> sp3(nullptr);  
shared_ptr<Simple> sp4;  
sp4 = make_shared<Simple>("Simple is Good");
```

shared_ptr

- 일반 포인터 변수처럼 '*', '.', '->' 사용 가능

```
class LargeObject {  
    public: void DoSomething(){ }  
};  
void ProcessLargeObject( const LargeObject& lo) { }  
  
void SmartPointerDemo() {  
    // Create the object and pass it to a smart pointer  
    shared_ptr<LargeObject> spLarge(new LargeObject());  
  
    //Call a method on the object  
    spLarge->DoSomething();  
    spLarge.DoSomething();  
  
    // Pass a reference to a method.  
    ProcessLargeObject(*spLarge);  
}  
//spLarge는 자동 소멸
```

C-style 메모리 사용

- `shared_ptr`는 기본적으로 `new`, `delete`를 이용해 메모리 할당 및 해제
- C-style 메모리 사용시 `shared_ptr` 사용 가능

```
int main()
{
    int* p = (int*)malloc(sizeof(int))

    shared_ptr<int> spInt(p, free)
}
```

std::move()

- 함수에서 `shared_ptr` 리턴 해도 복제 오버헤드 없음
→ `std::move()` 자동호출

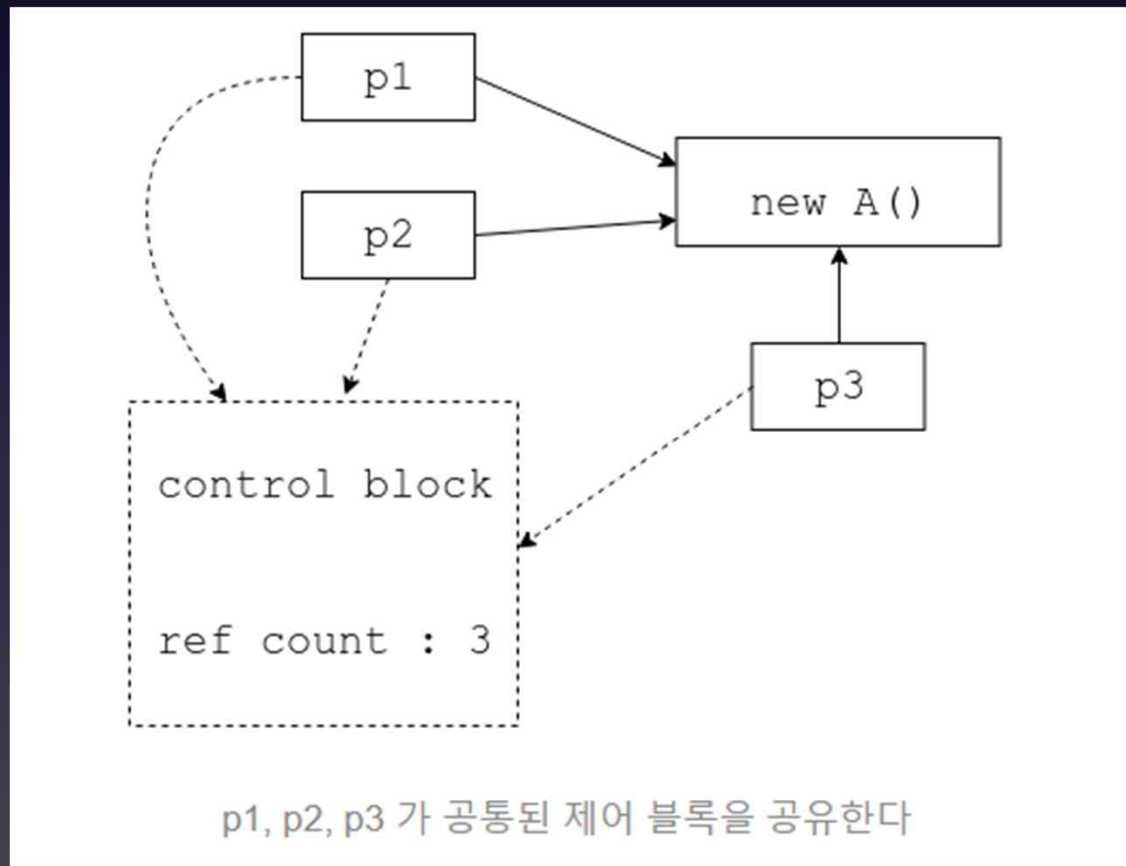
```
shared_ptr<int> func( )
{
    auto sp = make_shared<int>;
    return sp;
}

int main()
{
    shared_ptr<int> spTmp = func( );
    //shared_ptr<int> spTmp(func());
}
```

레퍼런스 카운팅

■ Reference counting

- 특정 포인터에 대해 몇 개의 스마트 포인터가 생성되었는지 계산



레퍼런스 카운팅

■ Reference counting

- 특정 포인터에 대해 몇 개의 스마트 포인터가 생성되었는지 계산

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

int main()
{
    shared_ptr<double> ptr1(new double(123.456));
    cout << ptr1->use_count() << endl;

    auto ptr2(ptr1);
    cout << ptr2->use_count() << endl;

    auto ptr3 = ptr2;
    cout << ptr3->use_count() << endl;
}
```

레퍼런스 카운팅

■ Reference counting

- 특정 포인터에 대해 몇 개의 스마트 포인터가 생성되었는지 계산

```
class Simple
{
    public:
        string stID;
        Simple(st = "") : stID(st) { cout << "Simple::Simple()" << endl; }
        ~ Simple() { cout << "Simple::~~ Simple()" << endl; }
}

void double_delete( )
{
    Simple* ptr = new Simple( );
    shared_ptr<Simple> spT1( ptr );
    shared_ptr<Simple> spT2( ptr );
}

int main
{
    double_delete( ); //소멸자 두 번 호출, 실행오류 발생
}
```

예제

```
#include <iostream>
#include <memory>
#include <vector>
```

```
using namespace std;
```

```
class Simple
```

```
{
```

```
public:
```

```
    string stID;
```

```
    Simple(string st="") : stID(st) { cout << "Simple::Simple()" << endl; }
```

```
    ~Simple() { cout << "Simple::~~Simple()" << endl; }
```

```
};
```

```
int main()
```

```
{
```

```
    vector<shared_ptr<Simple>> vec;
```

```
    vec.push_back(shared_ptr<Simple>(new Simple()));
```

```
    vec.push_back(shared_ptr<Simple>(vec[0]));
```

```
    vec.push_back(shared_ptr<Simple>(vec[1]));
```

```
    cout << "delete the 1st element\n";
```

```
    vec.erase(vec.begin());
```

```
    cout << "delete the 2nd element\n";
```

```
    vec.erase(vec.begin());
```

```
    cout << "delete the last element\n";
```

```
    vec.erase(vec.begin());
```

```
}
```

shared_ptr with vector

```
int main () {  
    vector<shared_ptr<Simple>> ISimple;  
  
    //Create a few new shared_ptr<Simple> instances  
    //and add them to vector using implicit move semantics.  
    ISimple.push_back(make_shared<Simple>("Simple is Good"));  
    ISimple.push_back(make_shared<Simple>("Simple is Better"));  
    ISimple.push_back(make_shared<Simple>("Simple is Best"));  
  
    // Pass by const reference when possible to avoid copying.  
    for (const auto& item : ISimple){  
        cout << item->stID << endl;  
    }  
}
```

실습

- 다음은 스마트 포인터를 이용한 **vector** 객체를 생성하고 출력하는 프로그램이다. 표시된 실행 결과가 나오도록 아래의 빈칸을 채우시오.

```
class Simple
{
public:
    string stID;
    Simple(string st="") : stID(st){ cout << "Simple( )" << endl; }
    ~Simple( ){ cout << "~Simple( )" << endl; }
};

int main( )
{
    vetor<shared_ptr<Symple>> ISimple;
    
    for( auto& item : ISimple)
        cout << item->stID << endl;}
```

```
Simple 1
Simple 2
Simple 3
```

CSV 파일

CSV 파일이란...

- 텍스트 파일
- 테이블 구조로 구성
- 각 행이 콤마로 구분됨
- 인공지능/기계학습의 학습 데이터로 활용
- 엑셀, 한셀, 메모장에서 간단히 읽기/쓰기 가능

	A	B	C	D	E	F
1	No	Korean	Mathemat	English	Total	
2	1	80	90	75	246	
3	2	77	88	80	247	
4	3	55	60	90	208	
5	4	79	89	65	237	
6						

	A	B	C	D	E	F
1		No	Korean	Mathematic	English	Total
2	0	1	80	90	75	246
3	1	2	77	88	80	247
4	2	3	55	60	90	208
5	3	4	79	89	65	237

→ Index

```
grade - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
No,Korean,Mathematics,English,Total
1,80,90,75,246
2,77,88,80,247
3,55,60,90,208
4,79,89,65,237
```

→ Title

→ Data Record

CSV 파일 읽기

```
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <map>
#include <iomanip>

using namespace std;

int main()
{
    //csv 파일 열기
    ifstream file("./grade.csv");

    if(file.is_open() == false)
    {
        cout << "file open error !" << endl;
        return 0;
    }
}
```


CSV 파일 읽기

```
vector<vector<string>> lContentsRow;
istringstream      ssRow;
vector<string>     lRow;
string             stLine, stItem;

while(getline(file, stLine))
{
    //한 줄 읽어서 vector에 저장
    istringstream ssLine(stLine);

    lRow.clear();
    while(getline(ssLine, stItem, ','))
        lRow.push_back(stItem);

    //저장
    lContentsRow.push_back(lRow);
}
file.close();
```

CSV 파일 읽기

```
//출력
for(auto& row : lContentsRow)
{
    for(auto& item : row)
        cout << setw(10) << item << " ";
    cout << endl;
}
}
```

C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe

No	Korean	Mathematics	English	Total
1	80	90	75	246
2	77	88	80	247
3	55	60	90	208
4	79	89	65	237

Press <RETURN> to close this window...

CSV 파일 읽기

- 행 번호 있는 경우
 - 첫 열은 의미 없음 → 삭제
- 빈 칸 있는 경우 → 빈 문자열로 처리됨

	A	B	C	D	E
1		Korean	Mathematic	English	Total
2	0	80	90	75	246
3	1	77	88	80	247
4	2	55	60		208
5	3	79	89	65	237

grade2 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

,Korean,Mathematics,English,Total

0,80,90,75,246

1,77,88,80,247

2,55,60,,208

3,79,89,65,237

CSV 파일 읽기

추가 →

```
file.close();

//행 번호만 있으면 첫 열 삭제
if(lContentsRow[0][0] == "")
    for(auto& item : lContentsRow)
        item.erase(item.begin());

//출력
for(auto& row : lContentsRow)
{
    for(auto& item : row)
        cout << setw(10) << item << " ";
    cout << endl;
}
}
```

