내용정리

- C++은 객체지향 / C 는 절차적 프로그래밍 언어
- C++은 객체중심 / C 는 함수기반
- C++에 추가된 레퍼런스 연산자(&)는 초기화한 뒤 값을 바꿀 수 없다.
- static 이 함수 밖에서 선언되면 C, C++ 차이가 없다.

static Member Function

- Is a service of the class, not of a specific object of the class
- static applied to an item at file scope
  - That item becomes known only in that file
  - The static members of the class need to be available from any client code that accesses the file
    - We cannot declare them static in the .cpp file—we declare them static only in the .h file

static Member Function Cont'd

- Declare a member function static
  - If it does not access non-static data members or non-static member functions of the class
- Does not have a this pointer
- Static data members and static member functions exist independently of any objects of a class
  - When a static member function is called, there might not be any objects of its class in memory
- Sometimes it is recommended that all calls to static member functions be made using the class name
  - not an object handle
- A const static member function is a compilation error

Static Local Objects

- Constructor is called only once
  - When execution first reaches where the object is defined
- Destructor is called when main terminates or the program calls function exit
  - Destructor is not called if the program terminates with a call to function abort
- Global and static objects are destroyed in the reverse order of their creation

```
#include <string>
using std::string;

#ifndef CREATE_H
#define CREATE_H

class CreateAndDestroy
{
 public:
   CreateAndDestroy( int, string );
   ~CreateAndDestroy();
 private:
   int objectID;
   string message;
};

#endif
```

```
#include <iostream>
using std::cout;
using std::endl;

#include "CreateAndDestroy.h"

CreateAndDestroy::CreateAndDestroy( int
     ID, string messageString )
{
   objectID = ID;
   message = messageString;

   cout << "Object " << objectID;
   cout << "   constructor runs   ";
   cout << message << endl;
}

CreateAndDestroy::~CreateAndDestroy()
{
   cout << "Object " << objectID;
   cout << "   destructor runs   ";
   cout << message << endl;
}
```

```
#include <iostream>
using std::cout;
using std::endl;
#include "CreateAndDestroy.h"

void create( void );
CreateAndDestroy first( 1,
    "(global before main)" );

int main()
{
   cout << "EXECUTION BEGINS"
    << endl;
   CreateAndDestroy second( 2,
    "(local automatic in main)" );
   static CreateAndDestroy
    third( 3, "(local static in
    main)" );
   create();
   cout << "EXECUTION RESUMES"
    << endl;
   CreateAndDestroy fourth( 4,
    "(local automatic in main)" );
   cout << "EXECUTION ENDS"
    << endl;
   return 0;
}

void create( void )
{
   cout << "CREATE BEGINS"
    << endl;
   CreateAndDestroy fifth( 5,
    "(local automatic in
    create)" );
   static CreateAndDestroy
    sixth( 6, "(local static in
    create)" );
   CreateAndDestroy seventh( 7,
    "(local automatic in
    create)" );
   cout << "CREATE ENDS" << endl;
}
```

- static 이 함수 안에서 선언되면, visibility 에는 차이가 없고 function 내에서만 호출할 수 있다. keep its seed value between invocations, 수명 = program's.
- 이어서, function 이 아닌 data segment 에 저장된다. 한번 init 되면 다시 되지 않고 저장되는 장소가 바뀌니 scope/duration 에 영향을 받는다.
- C++이 다른 점은 static within a class 일 때, becomes a single class variable that's common across all objects of the class 의 기능.
- 함수 호출시 주로 사용되며 함수의 인자처럼 메모리를 차지하지 않는다.

- Function prototype = function declaration

- return type, **Name, n of param, type of params, order of params.**

- 굵은 글씨가 signature, signature 가 같으면 같은 func 취급되니 return type 만 다른 두개 선언하면 compile error.

- int(4), char(1), short(2), long(4), float(4), double(8)

상속 접근 지정자	기반 클래스	파생 클래스로의 상속형태
public	public	public
	private	접근 불가
	protected	protected
private	public	private
	private	접근 불가
	protected	private
protected	public	protected
	private	접근 불가
	protected	protected

- Exception :

1. Indicate problems that occur during a program's execution
2. Exception handling ; can resolve exceptions, allow a program to continue executing or notify the user of the problem and terminate the program in a controlled manner.
Make programs robust and fault-tolerant
3. 협업에서 중요하다.
4. try{}문 : Statements that should be skipped in case of an exception
5. 1 or more catch handlers for each try block
6. Each catch can have only a single parameter, comma 로 이어지면 error.
7. #include <stdexcept>

- Exception specification :

1. Comma-separated list of exception classes in parentheses.
2. int someFunc(double value) throw (ExceptionA, ExceptionB, ExceptionC)
3. Indicates someFunction can throw exceptions of types ExceptionA, B, C.
4. A Function can throw "ONLY" exceptions of types in its specification or types derived from those types.
5. If a function throws a non-specification exception, function unexpected is called, normally terminates the program.

6. No exception specification ; function can throw any exceptions.
7. Empty exception specification(throw()) ; function cannot throw any.
8. 함수에 exception spec 에 없는 exception 을 throw 해도 컴파일 에러는 일어나지 않는다. 실행하는 중에 에러가 일어난다. 그러니 list 에 없는 error 를 미리 예방.

Storage Class
1. Storage class specifiers = { auto, register, **extern**, **mutable**, **static** }
2. Identifier's storage class; the **period** which that identifier exists in memory.
3. Identifier's scope; determines where can be referenced.
4. Identifier's linkage; determines to be known single or multiple files.
5. C++ 변수 선언의 컨텍스트에 있는 저장소 클래스는 개체의 수명, 연결 및 메모리 위치를 제어하는 형식 지정자입니다. 주어진 개체에는 스토리지 클래스가 하나만 있을 수 있습니다. 블록 내에 정의된 변수는 외각, 정적또는 thread_local 지정기를 사용하여 달리 지정하지 않는 한 자동 저장소를 갖습니다. 자동 개체 및 변수는 링크가 없으며 블록 외부의 코드에 표시되지 않습니다. 실행이 블록에 들어갈 때 메모리가 자동으로 할당되고 블록이 종료될 때 할당이 취소됩니다.
6. static: 함수에서 쓰이면 call 간에 공유되도록
class member 에 쓰이면 instance 간에 공유되도록
7. extern; const 되지 않은 global variable 에 대한 사용

```
//fileA.cpp
int i = 42; // declaration and definition

//fileB.cpp
extern int i;   // declaration only. same as i in FileA

//fileC.cpp
extern int i;   // declaration only. same as i in FileA

//fileD.cpp
int i = 43; // LNK2005! 'i' already has a definition.
extern int i = 43; // same error (extern is ignored on definitions)
```

7-2. const global 에 대하여

```
//fileA.cpp
extern const int i = 42; // extern const definition

//fileB.cpp
extern const int i;   // declaration only. same as i in FileA
```

- **mutable** : 데이터 멤버가 변경가능한 것으로 선언하는 것, const 멤버 함수에서이 데이터 멤버에 값을 할당할 수 있습니다. 다음 코드는 m_accessCount 변경가능 하도록 선언 되었으므로 오류가 발생 하지 않고 컴파일되고 따라서 GetFlag 가 const 멤버 함수 이더라도 GetFlag 에서 수정할 수 있습니다.

```
class X
{
public:
    bool GetFlag() const
    {
        m_accessCount++;
        return m_flag;
    }
private:
    bool m_flag;
    mutable int m_accessCount;
};
```

- Upcast; it is safe to cast a pointer up the class hierarchy to any of these base classes. (Since a derived class completely contains the definitions of all the base classes from which it is derived.)

잘못된 예와 잘못된 예)

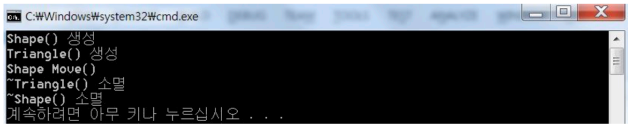
```
void Test1(){
    Triangle* t = new Triangle();
    play(t);

    delete t;
}

void play(Shape* s){
    s->Move();
}
```

이 코드의 결과는 무엇일까?

Triangle 객체를 생성해서 넣었기에 출력 결과는 Triange Move()가 나올 것 같지만, 결과는 다음과 같다.



객체는 Triangle이지만 함수의 파라미터 타입으로 인해 함수는 Triangle의 부모 클래스인 Shape 객체로 인식 된다.

```
class Shape{

public:
    Shape(){
        cout << "Shape() 생성" << endl;
    }
    ~Shape(){
        cout << "~Shape() 소멸" << endl;
    }
    virtual void Move(){
        cout << "Shape Move()" << endl;
    }
};
```

수정된 클래스의 출력 화면이다.

의도대로 Shape Move()가 아닌 Triangle Move()가 출력되었다.

하지만 여전히 Triangle 객체가 해제되진 않고 있어 메모리 누수 현상이 계속된다.

이를 어떻게 해결해야할까?

이 같이 부모의 소멸자 앞에 virtual 키워드를 삽입하면된다.

면 소멸자도 똑같이 동적 바인딩이 이루어지면서 원하는 의도대로 프로그램이 진행된다.

C++언어는 virtual 키워드를 통해 객체와 메서드간의 연결을 동적 바인딩을 명시해주지 않으면 위의 코드에서는 Triangle 객체를 생성했지만 실제로 Move() 메서드의 결과는 "Shape Move()"이다. 그 이유는 virtual 키워드를 입력하지 않았기 때문에 컴파일러가 부모와 자식의 오버라이딩 관계를 정적으로 바인딩했기 때문에 발생한다. 그래서 동적 생성인 new 연산자로 생성했다고 해서 동적 바인딩이

이루어지지 않고, 이미 정적 바인딩이 진행되었기에 함수 호출 시 원하는 출력이 나타나지 않는다. 만약 정상적인 출력을 원한다면 virtual 키워드를 부모 클래스의 오버라이딩 함수 앞에 넣어주면 된다.

이처럼 Upcasting 을 활용하면 파라미터로 부모 클래스 타입을 가지지만, 자식 객체들이 접근하여 그 함수를 활용할 수 있다.

1. Dynamic_cast operator 를 이용해서 it is possible to check whether a pointer actually points to a complete object and can be safely cast to point to another object in its hierarchy.
2. It also run-time check necessary to make the operation safe.

```
// dynamic_cast_1.cpp
// compile with: /c
class B { };
class C : public B { };
class D : public C { };

void f(D* pd) {
    C* pc = dynamic_cast<C*>(pd);     // ok: C is a direct base class
                                      // pc points to C subobject of pd
    B* pb = dynamic_cast<B*>(pd);     // ok: B is an indirect base class
                                      // pb points to B subobject of pd
} // D 를 받아서 C 를 만들고, 다시 C 로 B 를 만들고 // OK
```

This type of conversion is called an "upcast" because it moves a pointer up a class hierarchy, from a derived class to a class it is derived from. An upcast is an implicit conversion.

- Downcast

```
// dynamic_cast_3.cpp
// compile with: /c /GR
class Parent {virtual void f();};
class Child : public Parent {virtual void f();};

void f() {
    Parent* pb = new Child;     // unclear but ok
    Parent* pb2 = new Parent;
```

```
    Child* pd = dynamic_cast<Child*>(pb);     // ok: pb points to a Child
    Child* pd2 = dynamic_cast<Child*>(pb2);     // pb2 points to a Parent
}
```

- classpath; filesystem 라이브러리를 통해서 파일과 디렉토리를 다룰 수 있다. path 객체를 통해 파일과 폴더의 경로에 대한 작업을 손쉽게 할 수 있다.
1. directory_iterator 을 통해 디렉토리 안에 존재하는 파일들을 살펴볼 수 있고, recursive_directory_iterator 을 사용해서 디렉토리 안에 있는 모든 파일/폴더들을 탐색할 수 있다.
2. create_directory 를 통해 디렉토리를 생성하고 copy, remove 함수를 통해 파일/폴더를 복사/삭제 할 수 있다.

- friendly; : 이렇게 private 변수에 접근해야 될 필요성이 있을 때에는 friend 키워드를 사용해서 다른 클래스에 접근 권한을 부여해줄 수 있지만, 이렇게 변수에 직접 접근하도록 하는 것보다는 상속이나 함수를 통해서 변수에 접근할 수 있는 별도의 인터페이스를 제공하는 것이 객체지향 프로그래밍에서는 조금 더 바람직할수도 있다. 그리고 이를 encapsulation 이라고도 한다.

- Polymorphism; : Polymorphism 은 한글로 다형성이라고 하며, 이는 같은 부모의 클래스를 두고 있는 여러 개의 클래스들이 같은 기능을 하는 함수를 오버라이딩해서 밖에서 봤을 때에는 같지만, 내부적으로 동작하는 로직이 다를 때 Polymorphism 을 사용한다고 이야기한다. ex. Car 를 부모클래스도 사용하는 새로운 함수 Motorcycle 클래스를 추가한 예이다.

- final keyword; 1. To designate virtual functions that cannot be overridden in a derived class. 2. To designate classes that cannot be inherited.

```
class BaseClass
{
    virtual void func() final;
};
class DerivedClass: public BaseClass
{
    virtual void func(); // compiler error: attempting to
                         // override a final function
};
```

```
class BaseClass final
{
};
class DerivedClass: public BaseClass // compiler error: BaseClass is
                                     // marked as non-inheritable
{};
```

7. base class object 를 derived class 로 downcast 하였을 때 발생할 수 있는 potential problem

- Downcasting 이란 Upcasting 의 반대의 의미이며 절대로 생성할 때 사용하면 안된다. 애초에 visual studio 라면 오류를 내보낸다. 왜냐하면 다음과 같은 코드가 있다. Triangle* t = new Shape(); 기본적으로 Triangle 클래스가 Shape 클래스의 자식 클래스이기에 필요로 하는 정보량이 훨씬 많기에 그만큼 생성해야 한다. 하지만 Shape() 객체로 생성하면 Triangle 클래스가 필요로 하는 정보량을 모두 생성할 수 없다. 그렇기에 Downcasting 은 생성할 때, 쓸 수 없다. 단지 Upcating 한 객체를 원래대로 되돌릴 때 사용할 뿐이다.

8. base class constructor, derived class constructor, static 과 관련한 initialization 순서문제 (ppt 와 동일) 맨앞에 있음

- Constructor in derived class invokes the constructor of the base class implicitly or explicitly.
- CommissionEmployee constructor called last, constructor body finishies excecution first.
- Parent class 에 기본 constructor 가 정의되어있지 않으면, Constructor(param)만 있는 경우에 Child class 에서 explicitly call 하지 않으면 error.
- Destructor call; reverse order of constructor chain.

2. class 의 member variable 을 private 으로 설정하고 public method 로 접근하는 것의 장점, 단점을 하나씩 서술하라

Encapusulation, 인터페이스와 구현의 분리는 이렇게 리모컨의 작동 방식을 이해하지 않아도 리모컨을 사용할 수 있으므로 효과적이다.

장점 :, 클래스 재사용성과 유지보수 측면에서 더 많은 이점이 있다. 클래스가 내부적으로 어떻게 구현되었는지 몰라도 사용할 수 있다. 이렇게 캡슐화는 프로그램의 복잡성을 줄여주고, 실수도 줄여준다.

단점 : 클래스의 사용자가 인터페이스를 사용하도록 요구하는 것은 멤버 변수에 대한 직접 접근을 제공하는 것보다 추가적인 memory 가 사용된다.