P1.S11

작성일시	@2024년 10월 29일 오후 6:19
강의 번호	C++ 언리얼
유형	강의
복습	☐
날짜	@2024년 10월 29일

함수 포인터#1

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


// 오늘의 주제 : 함수 포인터

int Add(int a, int b)
{
    return a + b;
}

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

class Item
{
public:
    Item() : _itemId(0), _rarity(0), _ownerId(0)
    {

    }

public:
```

```
    int _itemId; // 아이템을 구분하기 위한 ID
    int _rarity; // 희귀도
    int _ownerId; // 소지자 ID
};

typedef bool (ITEM_SELECTOR)(Item* item, int);

Item* FindItem(Item items[], int itemCount, ITEM_SELECTOR* se
{
    // 안전 체크

    for (int i = 0; i < itemCount; i++)
    {
        Item* item = &item[i];
        if (selector(item, value))
            return item;
    }

    return nullptr;
}

bool IsRareItem(Item* item, int value)
{
    return item->_rarity >= value;
}

bool IsOwnerItem(Item* item, int ownerId)
{
    return item->_ownerId == ownerId;
}

int main()
{
    int a = 10;

    // 바구니 주소
    // pointer[ 주소 ] -> 주소[   ]
```

```
    typedef int DATA;

    // 1) 포인터         *
    // 2) 변수 이름      pointer
    // 3) 데이터 타입 int
    DATA* pointer = &a;

    // 함수
    typedef int(FUNC_TYPE)(int, int);

    // 1) 포인터         *
    // 2) 변수 이름      fn
    // 3) 데이터 타입 함수 (인자는 int, int 반환은 int)
    FUNC_TYPE* fn;

    // 시그니처가 동일한 함수를 담아줄수있다(int 두개 받고 int하나 반환ㅎ
    fn = Sub;

    // 함수의 이름은 함수의 시작 주소 (배열과 유사)
    int result = fn(1, 2); // 기본 문법
    cout << result << endl;

    int result2 = (*fn)(1, 2); // 함수 포인터는 *(접근연산자) 붙어도
    cout << result2 << endl;

    Item items[10] = {};
    items[3]._rarity = 2; // 희귀한 아이템
    FindItem(items, 10, IsRareItem, 2);
    Item* rareItem = FindItem(items, 10, IsOwnerItem, 100);

    return 0;
}
```

함수 포인터 #2

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

```
// 오늘의 주제 : 함수 포인터 #2

class Knight
{
public:

    // 정적 함수 : 나이트 객체랑은 연관성없음, 그냥 위치만 나이트 안에있을
    // _hp같이 특정 객체에 의존적인 값 수정 불가능
    /*static void HelloKnight()
    {

    }*/

    // 멤버 함수
    int GetHp(int, int)
    {
        cout << "GetHp()" << endl;
        return _hp;
    }
public:
    int _hp = 100;
};

// typedef의 진실
// typedef 왼쪽 오른쪽 -> 오른쪽(커스텀 타입 정의)
// 정확히는 왼쪽/오른쪽 기준이 아니라.
// [선언 문법]에서 typedef을 앞에다 붙이는 쪽

typedef int INTEGER;
typedef int* POINTER;
// 이제 FUNC라는 타입자체가 어떤 함수타입을 나타낼것이다. 함수의 시그니처
//typedef int FUNC(int, int);
typedef int ARRAY[20];

// 어지간해선 이 버전으로 기억하기. 한방에 함수포인터 만들기
typedef int(*PFUNC)(int, int); // 함수 포인터
```

```
typedef int(Knight::*PMEMFUNC)(int, int); // 멤버 함수 포인터

int Test(int a, int b)
{
    cout << "Test" << endl;
    return 0;
}

class Dog
{
public:
    int GetHp(int, int)
    {
        cout << "GetHp()" << endl;
        return 0;
    }
};

int main()
{
    // 함수 포인터
    // 1) 포인터          *
    // 2) 변수의 이름 fn
    // 3) 타입            함수(인자로 int 2개를 받고, int 1개를 반환)

    // 한번에 만드는 방법(익숙해지자)
    //int (*fn)(int, int);

    // 이런 방법도 있음
    //typedef int(FUNC_TYPE)(int, int);
    //FUNC_TYPE* fn;


    // 엄밀히 말해서 FUNC func; 이런 형태는 활용 안할거임
    // 포인터로만 사용하게 될것
    PFUNC fn;

    fn = &Test; // & 생략 가능 (C언어 호환성 때문)
```

```
    fn(1, 2);
    (*fn)(1, 2);


    // 위 문법으로 [전역 함수 / 정적 함수]만 담을 수 있다 (호출 규약이 동
    //fn = &Knight::GetHp;  // 안되는걸 볼수있음

    Test(1, 2);

    Knight k1;
    k1.GetHp(1, 1);


    // Knight 안에있는 것들만 사용
    PMEMFUNC mfn;
    mfn = &Knight::GetHp; // 멤버함수인경우 & 생략 불가능. 따라서 그
    (k1.*mfn)(1, 1); // 괄호로 묶어주고 중간에 *을 넣는 형태

    Knight* k2 = new Knight();
    (k2->*mfn)(1, 1);

    delete k2;

    return 0;
}
```

함수 객체

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


// 오늘의 주제 : 함수 객체

void HelloWorld()
{
    cout << "Hello World" << endl;
```

```
}

void HelloNumber(int number)
{
    cout << "Hello Number"<< number << endl;
}

class Knight
{
public:
    void AddHp(int addHp)
    {
        _hp += addHp;
    }

private:
    int _hp = 100;
};

class Functor
{
public:
    void operator()()
    {
        cout << "Functor Test" << endl;
        cout << _value << endl;
    }

    bool operator()(int num)
    {
        cout << "Functor Test" << endl;
        _value += num;
        cout << _value << endl;

        return true;
    }

private:
```

```
    int _value = 0;
};

class MoveTask
{
public:
    void operator()()
    {
        // TODO
        cout << "해당 좌표로 플레이어 이동" << endl;
    }
public:
    int _playerId;
    int _posX;
    int _posY;
};

int main()
{
    // 함수 객체 (Functor) : 함수처럼 동작하는 객체
    // 함수 포인터의 단점

    // 함수 포인터 선언

    void(*pfunc)(void);

    // 동작을 넘겨줄 때 유용하다
    pfunc = &HelloWorld;
    (*pfunc)();

    // 함수 포인터 단점
    // 1) 시그니처가 안 맞으면 사용할 수 없다
    // 2) 상태를 가질 수 없다
    //pfunc = &HelloNumber; // 오류남

    // [함수처럼 동작]하는 객체
    // () 연산자 오버로딩
    HelloWorld();
```

```
    Functor functor;

    functor();
    bool ret = functor(3);

    // MMO에서 함수 객체를 사용하는 예시
    // 클라 <-> 서버
    // 서버 : 클라가 보내준 네트워크 패킷을 받아서 처리
    // ex)클라 : 나(5, 0) 좌표로 이동시켜줘!
    MoveTask task;
    task._playerId = 100;
    task._posX = 5;
    task._posY = 0;

    // 나중에 여유 될 때 일감을 실행할 수 있게 된다
    task();

    return 0;
}
```

템플릿 기초 #1

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


// 오늘의 주제 : 템플릿 기초 #1

class Knight
{
public:
    // ...

public:
    int _hp = 100;
};
```

```
template<typename T> // typename 대신 class를 넣어도 됨
void Print(T a)
{
    cout << a << endl;
}

template<typename T1, typename T2>
void print(T1 a, T2 b)
{
    cout << a << " " << b << endl;
}

template<typename T>
T Add(T a, T b)
{
    return a + b;
}

// 연산자 오버로딩 (전역함수 버전)
// knight도 적용시켜주기 위한 시도
ostream& operator<<(ostream& os, const Knight& k)
{
    os << k._hp;
    return os;
}


// 템플릿 특수화
// knight란 예외 경우를 처리해주기 위한 시도
template<>
void Print(Knight a)
{
    cout << "Knight !!!!!!!" << endl;
    cout << a._hp << endl;
}

int main()
```

```
{
    // 템플릿 : 함수나 클래스를 찍어내는 틀
    // 1) 함수 템플릿
    // 2) 클래스 템플릿

    Print<int>(50);
    Print(50.0f);
    Print(50.0);
    Print("Hello World");

    print("Hello ", 100);

    int result1 = Add(1, 2);
    int result2 = Add<float>(1.11f, 2.22f);

    Knight k1;
    Print(k1);

    return 0;
}
```

템플릿 기초 #2

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


// 오늘의 주제 : 템플릿 기초 #2

// typename T를 붙이면 '조커카드' (어떤 타입도 다 넣을 수 있음)
// 그런데 무조건 typename을 붙여야 하는 것은 아니다
// template< > 안에 들어가는건 [골라줘야 하는 목록]이라고 볼 수 있음
template<typename T, int SIZE>
class RandomBox
{
public:
    T GetRandomData()
    {
```

```
        int idx = rand() % SIZE;
        return _data[idx];
    }

public:
    T _data[SIZE];
};


// 템플릿 특수화
template<int SIZE>
class RandomBox<double, SIZE> // 옆에다가 붙여줘서 특수하다는걸 표현
{
public:
    double GetRandomData()
    {
        cout << "RandomBox Double" << endl;
        int idx = rand() % SIZE;
        return _data[idx];
    }

public:
    double _data[SIZE];
};

void Hello(int number)
{

}

int main()
{
    srand(static_cast<unsigned int>(time(nullptr)));

    // 템플릿 : 함수나 클래스를 찍어내는 툴
    // 1) 함수 템플릿
    // 2) 클래스 템플릿
```

```
    RandomBox<int, 10> rb1;

    for (int i = 0; i < 10; i++)
    {
        rb1._data[i] = i;
    }
    int value1 = rb1.GetRandomData();
    cout << value1 << endl;

    RandomBox<int, 20> rb2;
    for (int i = 0; i < 20; i++)
    {
        rb2._data[i] = i + 0.5f;
    }
    float value2 = rb2.GetRandomData();
    cout << value2 << endl;

    // rb1 과 rb2는 아예 다른 클래스로 생성된다
    //rb1 = rb2; // 오류남. SIZE를 10으로 통일해주면 오류 안남


    RandomBox<double, 20> rb3;
    for (int i = 0; i < 20; i++)
    {
        rb3._data[i] = i + 0.5f;
    }
    double value3 = rb3.GetRandomData();
    cout << value3 << endl;

    return 0;
}
```

콜백 함수

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

```
// 오늘의 주제 : 콜백 (Callback)

class Item
{
public:

public:
    int _itemId = 0;
    int _rarity = 0;
    int _ownerId = 0;
};

class FindByOwnerId
{
public:
    bool operator()(const Item* item)
    {
        return (item->_ownerId == _ownerId);
    }
public:
    int _ownerId;
};

class FindByRarity
{
public:
    bool operator()(const Item* item)
    {
        return (item->_rarity >= _rarity);
    }
public:
    int _rarity;
};

template<typename T>
Item* FindItem(Item items[], int itemCount, T selector)
{
    for (int i = 0; i < itemCount; i++)
```

```
    {
        Item* item = &items[i];

        if (selector(item))
            return item;
    }

    return nullptr;
}

int main() {
    // 함수 포인터 + 함수 객체 + 템플릿
    // 콜백 (Callback) : 다시 호출하다? 역으로 호출하다?

    // 게임을 만들 때 이런 콜백의 개념이 자주 등장한다
    // ex) MoveTask 실습 등

    // 어떤 상황이 일어나면 -> 이 기능을 호출해줘
    // ex) UI 스킬 버튼을 누르면 -> 스킬을 쓰는 함수를 호출

    Item items[10];
    items[3]._ownerId = 100;
    items[8]._rarity = 2;

    FindByOwnerId functor1;
    functor1._ownerId = 100;

    FindByRarity functor2;
    functor2._rarity = 1;

    Item* item1 = FindItem(items, 10, functor1);
    Item* item2 = FindItem(items, 10, functor2);

    return 0;
}
```