제8장 부 록

1. C++에서의 형변환 (casting)

◈ static_cast

- 변환된 값으로 초기화되어있는 임시 객체를 생성해 냄
- 형변환 연산자가 정의되어 있는 객체에 대해서만 형변환 가능
- 형식 : static_cast<type>(객체)

[예제] static_cast 예제

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

class Time
{
  private:
        int hour;
        int min;
  public:
        Time(int h=0, int m=0);     // 시, 분을 받는 오버로디드 생성자
        Time(double t);             // 시간을 실수값으로 초기화하는 생성자
        operator double() const;    // 형변환함수
        void show();                // 멤버 출력함수
};

Time::Time(int h, int m)
{
        cout << "int  생성자 호출..." << endl;
        hour=h;
        min=m;
}

Time::Time(double t)
{
        cout << "double 생성자 호출..." << endl;
        hour=int(t);
        min=(int)((t-hour)*60.0);
}

Time::operator double() const
{
        cout << "형변환함수 호출..." << endl;
        double temp;
```

```
        temp=hour+min/60.0;
        return temp;
}

void Time::show()
{
        cout << this->hour << "시간 " << this->min << "분" << endl;
}

int main()
{
        Time a;
        a=3.5;  // 생성자 함수 호출
        a.show();

        double res;
        res=1.5 + static_cast<double>(a);  // operator double() 형변환 멤버함수 호출
        cout << res << endl;

        return 0;
 }
```

◈ const_cast

- 상수화 되어있는 포인터 변수나 참조 변수의 상수성을 일시적으로 제거 함
- 일반 변수와 일반 객체에는 사용할 수 없음
- 형식 : const_cast<type>(객체포인터 변수 또는 참조 객체명)

[예제2] const_cast 예제

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

class A
{
  private:
        int num;
  public:
        A(int n) { num = n; }
        void show();
 };
void A::show()
{
        cout << "num = " << num << endl;
}

void sub(const A &r);
int main()
{
```

```
        A ob(10);
        sub(ob);

        return 0;
}
void sub(const A &r)
{
        const_cast<A &>(r).show();
}
```

◈ dynamic_cast

- 다형성을 띄고 있는 타입을 실제 타입으로 변환(down-casting) 가능
- 변환이 불가능한 경우에는 NULL pointer 반환
- 형식 : dynamic_cast<type>(객체포인터 변수 또는 참조 객체명)

[예제3] dynamic_cast 예제

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

class A{  // 기본 클래스
        int va;
public:
        A(int n = 0) : va(n) {}
        virtual ~A() {}
        virtual void view() { cout << "A class view..." <<va<< endl; }
};
class B : public A{  // 파생 클래스
        int vb;
public:
        B(int n = 0) : vb(n) {}
        void view()  // view함수의 재정의
        { cout << "B class view..." << vb << endl; }
        virtual void prn()
        { cout << "B class prn()... " << vb << endl; }
};

class C : public B{  // 파생 클래스
        int vc;
public :
        C(int n = 0) : vc(n) {}
        void view()  // view함수의 재정의
        { cout << "C class view..." << vc << endl; }
        void prn()  // prn함수의 재정의
        { cout << "C class prn() : vc = " << vc << endl; }
};
```

```
A *GetObject();

int main()
{
	srand(time(0));
	A *ap;
	B *bp;
	int i;

	for(i=0; i<10; i++)
	{
		ap = GetObject();
		ap->view();
		if(bp = dynamic_cast<B *>(ap) )
		{
			bp->prn();
		}
		delete ap;
	}

	return 0;
}

A * GetObject()
{
	A *p;

	switch(rand()%3)
	{
	case 0 : p = new A(10); break;
	case 1 : p = new B(20); break;
	case 2 : p = new C(30); break;
	}
	return p;
}
```

---

* RTTI : Runtime type identification(실행시간 데이터형 정보)
  프로그램 실행도중에 객체의 데이터형을 결정하는 표준 방법

  이 기능은 옵션이 설정되어있어야 사용가능

  ((옵션 설정 하기))

  [솔루션 탐색기] – project명을 마우스 우측버튼 클릭 – [속성] 선택 – [구성 속성] 중 [C/C++]선택 – [언어] 선택 – [런타임 형식 정보 사용]을 [예]로 설정