

1. C ++ 기본

- 1.1 C++ 특징
- 1.2 CPP 에서의 출력과 입력
- 1.3 bool 새로운 자료형
- 1.4 malloc/free & new/delete
- 1.5 new / delete
- 1.6 함수의 오버로딩
- 1.7 함수의 디폴트 value
- 1.8 인라인함수
- 1.9 Namespace
- 1.10 Namespace 함수 선언과 정의의 분리
- 1.11 동일 Namespace의 함수 호출과 다른 네임스페이스의 함수 호출
- 1.12 NameSpace 중첩과 별명
- 1.13 NameSpace using
- 1.14 Refernece 참조자(&)의 새로운 의미 – Alias
- 1.15 매개변수로의 참조자
- 1.16 반환형이 참조자

1.1 C++ 특징

2

특징 - OOP (Object Oriented Programming) 객체지향언어

- 1) 캡슐화 (Encapsulation)
- 2) 정보은폐 (Information Hiding)
- 3) 추상화 (Abstarction)
- 4) 상속 (Inheritance)
- 5) 다형성 (Polymorphism)

1.2 CPP 에서의 출력과 입력

```
#include <iostream>
```

```
int main() {  
    int num = 20;  
    std::cout << "Hello world" << std::endl;  
    std::cout << "Hello " << "world" << std::endl;  
    std::cout << num << ' ' << 'A';  
    std::cout << ' ' << 3.14 << std::endl;  
  
    return 0;  
}
```

```
int main() {  
  
    int val1;  
    std::cout << "첫 번째 숫자 입력:";  
    std::cin >> val1;  
  
    int val2;  
    std::cout << "두 번째 숫자 입력 : " ;  
    std::cin >> val2;  
  
    int result = val1 + val2;  
    std::cout << "덧셈결과 출력" << result << std::endl;  
  
    return 0;  
}
```

std :: 란 무엇일까?

std 는 C++ 표준 라이브러리의 모든 함수, 객체 등이 정의된 이름 공간(namespace) 입니다.

이름 공간은 말그대로 어떤 정의된 객체에 대해 어디 소속인지 지정해주는 것.

코드의 크기가 늘어남에 따라, 혹은 다른 사람들이 쓴 코드를 가져다 쓰는 경우가 많아지면서 중복된 이름을 가진 함수들이 많아졌습니다.

따라서 C++ 에서는 이를 구분하기 위해, 같은 이름이라도, 소속된 이름 공간 이 다르면 다른 것으로 취급 하게 되어 있습니다.

std ::cout / std::endl 란 무엇일까?

- std::cout 는 C 언어의 stdout에 대응하는 표준출력 객체입니다.
- Std::두이 은 C 언어의 'Wn'과 같은 기능을 하며 endl = enter 줄바꿈의 의미입니다.

- 두 개 정수를 입력받아서 그 합을 출력하세요.
- 3과 7이 입력되면 4+5+6

```
int main(void)
{
    int val1, val2;
    int result=0;
    std::cout<<"두 개의 숫자입력: ";
    std::cin>>val1>>val2;

    if(val1<val2)
    {
        for(int i=val1+1; i<val2; i++)
            result+=i;
    }
    else
    {
        for(int i=val2+1; i<val1; i++)
            result+=i;
    }

    std::cout<<"두 수 사이의 정수 합: "
    <<result<<std::endl;
    return 0;
}
```

배열에 문자열 입력받기
이름과 전화번호를 char 배열에 각각 입력받은 후 출력하세요

```
int main() {
    char name[100];
    char phone[100];

    std::cout << "이름 입력: ";
    std::cin >> name;

    std::cout << "전화번호 입력: ";
    std::cin >> phone;

    std::cout << "\n정보 출력\n" << std::endl;

    std::cout << "이름: " << name <<
    "\n전화번호:" << phone << std::endl;

    return 0;
}
```

1. 두 개의 정수를 입력받아 그 합의 결과를 다음과 같이 출력하도록 `std::cout`와 `std::cin`을 이용해 출력

```
첫 번째 숫자 입력:3  
두 번째 숫자 입력 : 4  
연산결과  
  
3+4=7
```

2. 사용자로부터 2부터 9사이의 정수 값을 입력받아 해당 수의 구구단을 출력

1.3 bool 새로운 자료형

* true와 false 두 값을 갖는 자료형

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

```
bool IsPositive(int num)
{
    if(num<0)
        return false;
    else
        return true;
}
```

...

```
int main(void)
{
    bool isPos;
    int num;
    cout<<"Input number: ";
    cin>>num;

    isPos=IsPositive(num);
    if(isPos)
        cout<<"Positive number"<<endl;
    else
        cout<<"Negative number"<<endl;

    return 0;
}
```

1.4.1 malloc/free & new/delete

8

```
#include <iostream>
#include <string.h>
#include <stdlib.h>
using namespace std;

char * MakeStrAdr(int len)
{
    char * str=(char*)malloc(sizeof(char)*len);
    return str;
}

int main(void)
{
    char * str=MakeStrAdr(20);
    strcpy(str, "hello CPP~");
    cout<<str<<endl;
    free(str);
    return 0;
}
```

```
#include <iostream>
#include <string.h>
using namespace std;

char * MakeStrNew(int len)
{
    char * str=new char[len];
    return str;
}

int main(void)
{
    char * str=MakeStrAdr(20);
    strcpy(str, "hello CPP ~");
    cout<<str<<endl;
    delete []str;
    return 0;
}
```


1.4.2 new / delete

기본형 변수 및 배열의 할당과 삭제

```
int * ptr1 = new int;//int 형 변수의 할당
double * ptr2 = new double;//double 형 변수의 할당

int * ptr3 = new int[10];//길이가 10인 int형 배열할당
double * ptr4 = new double[10];//길이가 10인 double형 배열할당

delete ptr1;
delete ptr2;
delete[] ptr3;
delete[] ptr4;
```

구조체 할당과 삭제

```
typedef struct {
    int xpos;
    int ypos;
}Point;

void fucn5() {
    Point *ptr = new Point;
    ptr->xpos = 10;
    ptr->ypos = 20;

    delete ptr;
}
```

1. 길이가 10일 int 형 배열을 new 로 생성하고 1부터 10까지 저장한 후 출력, 함수 종료 전에 배열 삭제
2. 길이가 32인 char 배열을 new로 생성하고 사용자로부터 이름을 입력 받아 저장하고 출력, 함수 종료 전에 삭제
3. 이름, 연락처, 나이를 저장하는 구조체를 만들고 new 로 할당 후 “홍길동“, 010-1234-5678, 26을 저장한 후 출력하고 삭제

1.5 함수의 오버로딩

- 같은 이름을 갖는 함수를 여러 개 정의할 수 있다. 단, 매개변수의 타입이나 개수가 달라야 한다.

```
void MyFunc(void)
{
    std::cout<<"MyFunc(void) called"<<std::endl;
}

void MyFunc(char c)
{
    std::cout<<"MyFunc(char c) called"<<std::endl;
}

void MyFunc(int a, int b)
{
    std::cout<<"MyFunc(int a, int b) called"<<std::endl;
}

int main(void)
{
    MyFunc();
    MyFunc('A');
    MyFunc(12, 13);
    return 0;
}
```

*반환 타입은 오버로딩 조건에 해당되지 않음

C++ 컴파일러에서 함수를 오버로딩하는 과정

1 단계

자신과 타입이 정확히 일치하는 함수를 찾는다.

2 단계

정확히 일치하는 타입이 없는 경우 아래와 같은 형변환을 통해서 일치하는 함수를 찾아본다.

- Char, unsigned char, short 는 int 로 변환된다.
- Unsigned short 는 int 의 크기에 따라 int 혹은 unsigned int 로 변환된다.
- Float 은 double 로 변환된다.
- Enum 은 int 로 변환된다.

3 단계

위와 같이 변환해도 일치하는 것이 없다면 아래의 좀더 포괄적인 형변환을 통해 일치하는 함수를 찾는다.

- 임의의 숫자(numeric) 타입은 다른 숫자 타입으로 변환된다. (예를 들어 float -> int)
- Enum 도 임의의 숫자 타입으로 변환된다 (예를 들어 Enum -> double)
- 0 은 포인터 타입이나 숫자 타입으로 변환된 0 은 포인터 타입이나 숫자 타입으로 변환된다
- 포인터는 void 포인터로 변환된다.

4 단계

유저 정의된 타입 변환으로 일치하는 것을 찾는다.

1. 정수 값 두 개를 입력받아 합을 반환하는 함수 `getSum`을 제작
2. 1번에 이어서 실수 값 두 개를 입력받아 그 합을 반환하는 오버로딩된 `getSum` 함수 제작
3. `getSum(3,4)`와 `getSum(3.1, 4.1)`을 호출하여 오버로딩 결과를 확인
4. 매개변수로 변수의 포인터를 전달받아 저장된 값을 바꿔주는 `swap`를 구현하라.
char, int double 세 가지 자료형에 맞게 오버로딩

1.6.1 함수의 디폴트 value

- 함수 호출 시 매개변수가 전달되지 않으며, 함수 선언부분에 표현된 디폴트 값을 매개변수 값으로 사용
- 함수의 선언과 정의가 분리되어 있는 경우, 선언 부분에 표현

```
int Adder(int num1=1, int num2=2)
{
    return num1+num2;
}

int main(void)
{
    std::cout<<Adder()<<std::endl;
    std::cout<<Adder(5)<<std::endl;
    std::cout<<Adder(3, 5)<<std::endl;
    return 0;
}
```

```
int Adder(int num1=1, int num2=2);

int main(void)
{
    std::cout<<Adder()<<std::endl;
    std::cout<<Adder(5)<<std::endl;
    std::cout<<Adder(3, 5)<<std::endl;
    return 0;
}

int Adder(int num1, int num2)
{
    return num1+num2;
}
```

1.6.2 함수의 디폴트 value

- 매개변수의 일부분만 디폴트 값을 지정할 수 있으며,
- 이 때 디폴트 값을 지정한 매개변수는 디폴트 값이 없는 매개변수 다음에 올 수 있다.

```
int BoxVolume(int length, int width=1, int height=1);

int main()
{
    std::cout<<"[3, 3, 3] : "<<BoxVolume(3, 3, 3)<<std::endl;
    std::cout<<"[5, 5, D] : "<<BoxVolume(5, 5)<<std::endl;
    std::cout<<"[7, D, D] : "<<BoxVolume(7)<<std::endl;
    //std::cout<<"[D, D, D] : "<<BoxVolume()<<std::endl; // 에러
    return 0;
}

int BoxVolume(int length, int width, int height)
{
    return length*width*height;
}
```

- 디폴트 value에 관한 샘플코드 BoxVolume을 함수 오버로딩으로 구현해보자. 실행 결과가 동일해야함

```
int BoxVolume(int length, int width=1, int height=1);

int main()
{
    std::cout<<"[3, 3, 3] : "<<BoxVolume(3, 3, 3)<<std::endl;
    std::cout<<"[5, 5, D] : "<<BoxVolume(5, 5)<<std::endl;
    std::cout<<"[7, D, D] : "<<BoxVolume(7)<<std::endl;
    //std::cout<<"[D, D, D] : "<<BoxVolume()<<std::endl; // 에러
    return 0;
}

int BoxVolume(int length, int width, int height)
{
    return length*width*height;
}
```


- 다음 코드에서 컴파일 에러가 나는 위치와 이유를 설명하시오

```
int SimpleFunc(int a = 10) {  
    return a + 1;  
}  
  
int SimpleFunc() {  
    return 10;  
}  
  
int main() {  
    SimpleFunc();  
    return 0;  
}
```

1.7 인라인함수

- 매크로함수처럼 함수의 몸체부분이 함수호출 문장을 완전히 대체함으로써 속도가 향상
- 매크로 함수는 복잡한 함수의 구현이 매우 어려운 것이 단점이었으나 인라인함수는 복잡한 함수의 구현이 가능
- 하지만, 인라인 함수는 매크로함수처럼 자료형에 독립적이지 않음

```
inline int SQUARE(int x)
{
    return x*x;
}

int main(void)
{
    std::cout<<SQUARE(5)<<std::endl;
    std::cout<<SQUARE(12)<<std::endl;
    return 0;
}
```

인라인함수

```
#define SQUARE (a) a*a

int main(void)
{
    std::cout<<SQUARE(5)<<std::endl;
    std::cout<<SQUARE(1.2)<<std::endl;
    return 0;
}
```

매크로함수

1.8.1 Namespace

```
namespace NameSpace1
{
    void function() {
        std::cout << "NameSpace1의 function" << std::endl;
    }
}

namespace NameSpace2
{
    void function() {
        std::cout << "NameSpace2의 function" << std::endl;
    }
}

int main() {
    NameSpace1::function();
    NameSpace2::function();

    return 0;
}
```

이름 공간(namespace)

이름 공간은 말그대로 어떤 정의된 객체에 대해 **어디 소속인지** 지정해주는 것.

코드의 크기가 늘어남에 따라, 혹은 다른 사람들이 쓴 코드를 가져다 쓰는 경우가 많아지면서 중복된 이름을 가진 함수들이 많아졌습니다. 따라서 C++ 에서는 이를 구분하기 위해, 같은 이름이라도, 소속된 **이름 공간** 이 다르면 다른 것으로 취급하게 되어 있습니다.

1.8.2 Namespace 함수 선언과 정의의 분리

```
namespace NameSpace1
{
    void function();
}

namespace NameSpace2
{
    void function();
}

int main() {
    NameSpace1::function();
    NameSpace2::function();

    return 0;
}

void NameSpace1::function() {
    std::cout << "NameSpace1의 function" << std::endl;
}

void NameSpace2::function() {
    std::cout << "NameSpace2의 function" << std::endl;
}
```

1.8.3 동일 Namespace의 함수 호출과 다른 네임스페이스의 함수 호출

21

```
namespace NameSpace1
{
    void function() {
        std::cout << "NameSpace1의 function" << std::endl;
    }
    void function2() {
        function();//동일 네임스페이스의 함수호출
        NameSpace2::function(); //다른 네임스페이스의 함수
        호출
    }
}

namespace NameSpace2
{
    void function() {
        std::cout << "NameSpace2의 function" << std::endl;
    }
}

int main() {
    NameSpace1::function();
    NameSpace2::function();

    return 0;
}
```

1.8.4 Namespace 중첩과 별명

22

네임스페이스 중첩

```
namespace Parent
{
    int num = 1;

    namespace Child {
        int num = 2;

        namespace GrandChild {
            int num = 3;
        }
    }
}

int main() {

    std::cout << Parent::num << std::endl;
    std::cout << Parent::Child::num << std::endl;
    std::cout << Parent::Child::GrandChild::num << std::endl;

    return 0;
}
```

네임스페이스 별명

```
namespace Parent
{
    int num = 1;

    namespace Child {
        int num = 2;

        namespace GrandChild {
            int num = 3;
        }
    }
}

namespace Gchild = Parent::Child::GrandChild;

int main() {

    std::cout << Parent::Child::GrandChild::num << std::endl;

    std::cout << Gchild::num << std::endl;

    return 0;
}
```

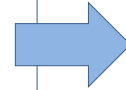
1.8.5 NameSpace using

```
namespace NameSpace1
{
    void function() {
        std::cout << "NameSpace1의 function" << std::endl;
    }
}

using namespace NameSpace1;

int main() {
    function();

    return 0;
}
```



```
using namespace std;

namespace NameSpace1
{
    void function() {
        cout << "NameSpace1의 function" << endl;
    }
}

using namespace NameSpace1;

int main() {
    function();

    return 0;
}
```

1.9.1 Refernece 참조자(&)의 새로운 의미 – Alias

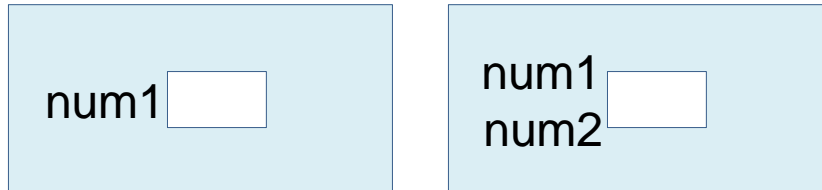
24

- C 에서 &는 변수의 주소값을 반환하는 연산자로 사용

```
int num ;  
int * ptr = &num;
```

- C++에서 변수(메모리 공간에 붙여진 이름)의 별명의 사용가능

```
int num1;  
int &num2 = num1
```



- 참조자의 수에 제한이 없음
- 참조자는 변수에 대해서만 선언이 가능 (int &ref = 20; // X 불가함)
- 리터럴을 참조할 경우 const 를 사용(const int &val = 20; //O 가능)
- 선언과 동시에 변수를 지정해야하며 추후 변경 불가(int &ref ; // X)
- NULL로 초기화 불가 (int &ref = NULL; //X)

```
using namespace std;  
  
int main(void)  
{  
    int num1=1020;  
    int &num2=num1;  
  
    num2=3047;  
    cout<<"VAL: "<<num1<<endl;  
    cout<<"REF: "<<num2<<endl;  
  
    cout<<"VAL: "<<&num1<<endl;  
    cout<<"REF: "<<&num2<<endl;  
    return 0;  
}
```


1.9.2 매개변수로의 참조자

```
using namespace std;

void SwapByRef2(int &ref1, int& ref2)
{
    int temp=ref1;
    ref1=ref2;
    ref2=temp;
}

int main(void)
{
    int val1=10;
    int val2=20;

    SwapByRef2(val1, val2);
    cout<<"val1: "<<val1<<endl;
    cout<<"val2: "<<val2<<endl;
    return 0;
}
```

참조자 reference를 매개변수로 전달하는 것은 외부 변수를 의미함

```
val1: 20
val2: 10
```

1.9.3 반환형이 참조자

26

```
using namespace std;

int & RefRetFuncOne(int &ref)
{
    ref++;
    return ref;
}

int main(void)
{
    int num1=1;
    int &num2=RefRetFuncOne(num1);

    num1++;
    num2++;
    cout<<"num1: "<<num1<<endl;
    cout<<"num2: "<<num2<<endl;

    cout << "num1 주소: " << &num1 << endl;
    cout << "num2 주소: " << &num2 << endl;
    return 0;
}
```

```
num1: 4
num2: 4
num1 주소: 006FFD88
num2 주소: 006FFD88
```

```
using namespace std;

int & RefRetFuncOne(int &ref)
{
    ref++;
    return ref;
}

int main(void)
{
    int num1 = 1;
    int num2 = RefRetFuncOne(num1);

    num1++;
    num2++;
    cout << "num1: " << num1 << endl;
    cout << "num2: " << num2 << endl;

    cout << "num1 주소: " << &num1 << endl;
    cout << "num2 주소: " << &num2 << endl;
    return 0;
}
```

```
num1: 3
num2: 3
num1 주소: 00CFFE54
num2 주소: 00CFFE48
```

1. 참조자를 이용해 인자로 전달한 int형 변수값을 1씩 증가하는 함수
2. 참조자를 이용해 인자로 전달한 int 형 변수의 부호를 바꾸는 함수
3. 다음의 num의 주소값을 저장한 포인터 변수와 이 포인터 변수를 참조하는 참조자를 선언하고 이 포인터변수와 참조자를 이용해 num에 저장된 값을 출력하라
`const int num =12;`
4. 길이를 매개변수로 받아서 해당 길이의 문자열을 저장할 수 있는 배열을 생성하고 그 주소값을 반환하는 함수를 만들고 이를 테스트하는 코드를 작성하라.
5. 다음과 같은 Point 구조체를 정의하고 이를 매개변수로 하는 pntAdder 함수를 정의하라. PntAdder함수는 매개변수로 전달 받은 두 개 점의 x좌표끼리 덧셈, y좌표 끼리 덧셈 연산으로 새로운 Point를 초기화해서 반환하는 함수이다. 이 때 Point 구조체 변수는 new 연산자를 이용해야한다.

```
typedef struct {  
    int xpos;  
    int ypos;  
}Point;  
  
Point& PntAdder(const Point &ptr1, const Point &ptr2);
```


2. 객체지향 프로그래밍

2.1 객체지향 프로그래밍이란

30

- 현실에 존재하는 사물과 대상, 그리고 그에 따른 행동을 있는 그대로 실체화 시키는 형태의 프로그래밍
- 속성은 멤버 변수로 기능은 멤버 함수로 구현
- 좋은 클래스의 기본 조건 : 정보은닉과 캡슐화
- 정보은닉 : 정보은닉은 클래스 외부에서 특정 정보에 접근하는 것을 막기위한 것으로 멤버변수를 `private`으로 선언하고, 해당 변수에 접근하는 함수를 별도로 정의해서, 안전한 형태로 멤버변수의 접근을 유도
- 캡슐화 : 관련있는 함수와 변수를 하나의 클래스 안에 묶는 것

2.2 C++ 구조체

- typedef 없이 선언이 가능
- 구조체가 함수를 가질 수 있음

```
struct Car
{
    char gamerID[ID_LEN];// 소유자ID
    int fuelGauge;// 연료량
    int curSpeed;// 현재속도

    void ShowCarState()
    {
        cout<<"소유자ID: "<<gamerID<<endl;
        cout<<"연료량: "<<fuelGauge<<"%"<<endl;
        cout<<"현재속도: "<<curSpeed<<"km/s"<<endl<<endl;
    }
}

Car mycar;
```

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

```
struct Car
{
    char gamerID[ID_LEN]; // 소유자ID
    int fuelGauge; // 연료량
    int curSpeed; // 현재속도

    void ShowCarState()
    {
        cout<<"소유자ID: "<<gamerID<<endl;
        cout<<"연료량: "<<fuelGauge<<"%"<<endl;
        cout<<"현재속도: "<<curSpeed<<"km/s"<<endl<<endl;
    }
    void Accel()
    {
        if(fuelGauge<=0)
            return;
        else
            fuelGauge-=FUEL_STEP;

        if(curSpeed+ACC_STEP>=MAX_SPD)
        {
            curSpeed=MAX_SPD;
            return;
        }
        curSpeed+=ACC_STEP;
    }
}
```

```
#define ID_LEN 20
#define MAX_SPD 200
#define FUEL_STEP 2
#define ACC_STEP 10
#define BRK_STEP 10
```

```
void Break()
{
    if(curSpeed<BRK_STEP)
    {
        curSpeed=0;
        return;
    }

    curSpeed-=BRK_STEP;
}

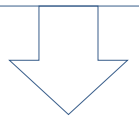
};
```

```
int main(void)
{
    Car run99={"run99", 100, 0};
    run99.Accel();
    run99.Accel();
    run99.ShowCarState();
    run99.Break();
    run99.ShowCarState();

    Car sped77={"sped77", 100, 0};
    sped77.Accel();
    sped77.Break();
    sped77.ShowCarState();
    return 0;
}
```

32

함수는 구조체 외부에
두고 구조체 내부에는
함수 선언만 사용한 예




```

#include <iostream>
using namespace std;
namespace CAR_CONST
{
    enum
    {
        ID_LEN=20,
        MAX_SPD=200,
        FUEL_STEP=2,
        ACC_STEP=10,
        BRK_STEP=10
    };
}
struct Car
{
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;

    void ShowCarState();
    void Accel();
    void Break();
};
void Car::ShowCarState()
{
    cout<<"소유자ID: "<<gamerID<<endl;
    cout<<"연료량: "<<fuelGauge<<"%"<<endl;
    cout<<"현재속도: "<<curSpeed<<"km/s"<<endl<<endl;
}

```

```

void Car::Accel()
{
    if(fuelGauge<=0)
        return;
    else
        fuelGauge-=CAR_CONST::FUEL_STEP;

    if((curSpeed+CAR_CONST::ACC_STEP)>=CAR_CONST::MAX_SPD)
    {
        curSpeed=CAR_CONST::MAX_SPD;
        return;
    }

    curSpeed+=CAR_CONST::ACC_STEP;
}

void Car::Break()
{
    if(curSpeed<CAR_CONST::BRK_STEP)
    {
        curSpeed=0;
        return;
    }

    curSpeed-=CAR_CONST::BRK_STEP;
}

```

2. 클래스와 객체

- ▶ 클래스의 정의 – 클래스란 객체를 정의해 놓은 것이다.
- ▶ 클래스의 용도 – 클래스는 객체를 생성하는데 사용된다.
- ▶ 객체의 정의 – 실제로 존재하는 것. 사물 또는 개념.
- ▶ 객체의 용도 – 객체의 속성과 기능에 따라 다름.

클래스	객체
제품 설계도	제품
TV설계도	TV
붕어빵기계	붕어빵

2.1 클래스의 정의

- ▶ 객체는 속성과 기능으로 이루어져 있다.
 - 객체는 속성과 기능의 집합이며, 속성과 기능을 객체의 멤버(member, 구성 요소)라고 한다.
- ▶ 속성은 변수로, 기능은 메서드로 정의한다.
 - 클래스를 정의할 때 객체의 속성은 변수로, 기능은 메서드로 정의한다.

속성	크기, 길이, 높이, 색상, 볼륨, 채널 등
기능	켜기, 끄기, 볼륨 높이기, 볼륨 낮추기, 채널 높이기 등

변수

메서드

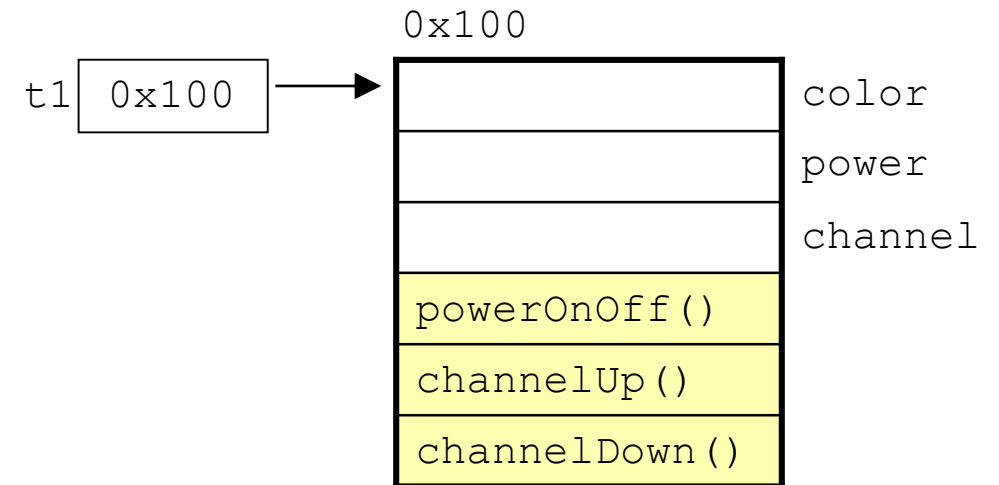
```
class MyTv {  
    public:  
    int channel;  
    bool power;  
    char color[30];  
  
    void powerOnOff() { power = !power; }  
    void channelUp(){ channel++; }  
    void channelDown() { channel--;}  
};
```

2.2 객체의 생성

```
class MyTv {  
    public:  
    int channel;  
    bool power;  
    char color[30];  
  
    void powerOnOff() { power = !power; }  
    void channelUp() { channel++; }  
    void channelDown() { channel--;}  
};
```

```
MyTv t1;
```

MyTv 타입의 참조변수 t1의 선언과 동시에 객체가 생성

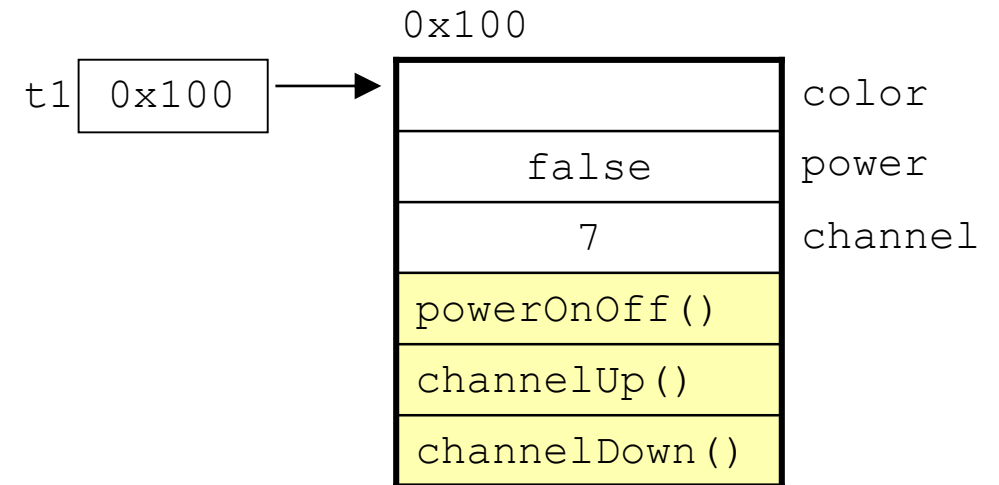


객체의 생성

37

```
class MyTv {  
    public:  
    int channel;  
    bool power;  
    char color[30];  
  
    void powerOnOff() { power = !power; }  
    void channelUp(){ channel++; }  
    void channelDown() { channel--;}  
};
```

```
MyTv t1;  
t1.channel = 7;  
t1.power = false;
```



다음에 명시한 클래스를 만들고 테스트 해봅시다

38

클래스명 : MyTv

멤버변수 :

자료형	변수명	설명
String	color	tv 색상
int	channel	채널
bool	power	전원

메서드 :

반환타입	메서드명	매개변수	기능
void	powerOnOff	없음	power 변수를 반전시켜 저장
void	chanerUp	없음	channel 변수를 하나 증가
void	channelDown	없음	channel 변수를 하나 감소

```
int main() {  
    MyTv t1;  
    t1.channel = 7;  
    strcpy(t1.color, "white");  
    cout << "채널:" << t1.channel << ", 색깔:" << t1.color << endl;  
  
    MyTv t2;  
    t2.channel = 11;  
    strcpy(t2.color, "red");  
    cout << "채널:" << t2.channel << ", 색깔:" << t2.color << endl;  
}
```

다음에 명시한 클래스를 만들고 테스트 해봅시다

39

클래스명 : Student

멤버변수 :

자료형	변수명	설명
char[30]	name	학생이름
int	ban	반
int	no	번호
int	kor	국어점수
int	eng	영어점수
int	math	수학점수

메서드 :

반환타입	메서드명	매개변수	기능
int	getTotal	없음	국, 영, 수 총점을 반환
float	getAverage	없음	국, 영, 수 평균을 반환

클래스를 이용해 두 개의 객체를 생성한 후 getTotal과 getAverage를 이용해서 아래와 같은 결과가 나오도록 코딩하세요

철수 : 총점 - 240점, 평균 - 80점

영희 : 총점 - 270점, 평균 - 90점

다음에 명시한 클래스를 만들고 테스트 해봅시다

40

클래스명 : Card

멤버변수 :

자료형	변수명	설명
char[30]	Shape	카드에 적힌 모양
int	number	카드에 적힌 숫자

메서드 :

반환타입	메서드명	매개변수	기능
void	printCard	없음	카드의 모양과 숫자를 다음과 같은 형태로 화면에 표시 diamond, 7

클래스를 이용해 세 개의 객체를 생성한 후 printCard 함수를 이용해 다음과 같이 출력하도록 코딩하세요

diamond, 7

heart, 3

clover, 11

다음에 명시한 클래스를 만들고 테스트 해봅시다

클래스명 : MyCar

멤버변수 :

자료형	변수명	설명
char[30]	color	차의 색깔
int	dorNum	문의 갯수

메서드 :

반환타입	메서드명	매개변수	기능
없음	showCarInfo	없음	차의 색깔과 문의 개수를 출력

클래스를 이용해 두 개의 객체를 생성한 후 초기화하고 다음과 같이 출력하도록 코딩하세요

문 개수:4, 자동차 색깔:white

문 개수:3, 자동차 색깔:red

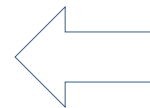
2.3 접근제어지시자 와 정보의 은닉

42

```
class MyCar {  
  
    private:  
    int dorNum;  
    char color[30];  
  
    public:  
    void showCarInfo() {  
        cout << "문 개수: " << dorNum << ", 차 색깔: " << color  
        << endl;  
    }  
    void setCarInfo(int num, const char * str) {  
        dorNum = num;  
        strcpy(color, str);  
    }  
  
};
```

```
void setCarInfo(int num, const char * str) {  
    dorNum = num;  
    strcpy(color, str);  
}
```

```
int main() {  
  
    MyCar car1;  
    car1.dorNum = 4;  
    strcpy(car1.color, "blue");  
    car1.showCarInfo();  
  
    MyCar car2;  
    car2.dorNum = 3;  
    strcpy(car2.color, "white");  
    car2.showCarInfo();  
  
    return 0;  
}
```



MyCar 클래스에 멤버변수의 값을
저장할 수 있는 멤버함수 추가

2.4 생성자

43

- 클래스 이름과 동일한 이름의 함수 형태를 갖고 있음
- 반환형을 갖지 않음
- 여러 개의 생성자를 가질 수 있음
- 하나의 생성자도 갖고 있지 않을 때 컴파일러가 기본 생성자를 호출함 (기본생성자는 매개변수가 없음)

2.4.1 생성자

```
class MyCar {  
  
    private:  
    int dorNum;  
    char color[30];  
  
    public:  
    MyCar(int num, const char *str) {  
        dorNum = num;  
        strcpy(color, str);  
    }  
  
    void showCarInfo() {  
        cout << "문 개수: " << dorNum << ",   차 색깔: " << color  
        << endl;  
    }  
};  
  
int main() {  
    MyCar car1(3, "white");  
    car1.showCarInfo();  
  
    MyCar car2(4, "black");  
    car2.showCarInfo();  
  
    return 0;  
}
```

```
MyCar() {  
    dorNum = 2;  
    strcpy(color, "silver");  
}
```

2.4.2 const 함수

- const 함수 내에서는 멤버 변수에 저장된 값을 변경하지 않는다
- const 함수는 const 함수의 호출만 가능하다

```
class MyCar {  
    private:  
        int dorNum;  
        char color[30];  
  
    public:  
        MyCar(int num, const char * str) {  
            dorNum = num;  
            strcpy(color, str);  
        }  
  
        int getDoorNum() const  
        {  
            dorNum = 3;  
            return dorNum;  
        }  
  
        void showCarInfo() {  
            cout << "문 개수:" << dorNum << ", 자동차 색깔:" << color << endl;  
        }  
  
};
```

2.4.3 하나 이상의 생성자

```
class SimpleClass
{
    int num1;
    int num2;

public:
    SimpleClass()
    {
        num1=0;
        num2=0;
    }
    SimpleClass(int n)
    {
        num1=n;
        num2=0;
    }
    SimpleClass(int n1, int n2)
    {
        num1=n1;
        num2=n2;
    }
    void ShowData() const
    {
        cout<<num1<<' ' <<num2<<endl;
    }
};
```

- 객체가 생성될 때 한 번 호출된다
- 하나 이상의 생성자를 만들 수 있다
- 생성자의 매개변수에 디폴트 값을 지정할 수 있다
- 생성자를 하나도 정의하지 않은 클래스는 컴파일러에 의해 디폴트 생성자가 자동으로 추가된다.
 - 디폴트생성자란 ? 매개변수가 없고 내부적으로 아무 동작도 구현하지 않은 생성자

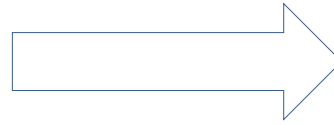
```

class SimpleClass
{
    int num1;
    int num2;

public:
    SimpleClass()
    {
        num1=0;
        num2=0;
    }
    SimpleClass(int n)
    {
        num1=n;
        num2=0;
    }
    SimpleClass(int n1, int n2)
    {
        num1=n1;
        num2=n2;
    }

    void ShowData() const
    {
        cout<<num1<<' ' <<num2<<endl;
    }
};

```



클래스 SimpleClass를 이용해
객체를 생성하는 main 코드

4번째 방법으로 객체를 생성할
수 없음을 확인할 것

```

int main(void)
{
    SimpleClass sc1;
    sc1.ShowData();

    SimpleClass sc2(100);
    sc2.ShowData();

    SimpleClass sc3(100, 200);
    sc3.ShowData();

    //SimpleClass sc4();
    //sc4.ShowData();

    SimpleClass *ptr1 = new SimpleClass;
    ptr1->ShowData();

    SimpleClass *ptr2 = new SimpleClass();
    ptr2->ShowData();

    return 0;
}

```

2.4.4 생성자 매개변수의 기본값(Default Value)

48

```
class SimpleClass
{
    int num1;
    int num2;

public:
    SimpleClass(int n1=0, int n2=0)
    {
        num1=n1;
        num2=n2;
    }

    void ShowData() const
    {
        cout << num1 << ' ' << num2 << endl;
    }
};
```

```
int main(void)
{
    SimpleClass sc1;
    sc1.ShowData();

    SimpleClass sc2(100);
    sc2.ShowData();

    SimpleClass sc3(100, 200);
    sc3.ShowData();
    return 0;
}
```


2.4.5 멤버 초기화 이니셜라이저

49

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

class Seller
{
    int price;
    int numOfProduct;
    int myMoney;
public:
    Seller(int price, int num, int money) : price(price), numOfProduct(num), myMoney(money)
    {
    }

    void ShowSalesResult() const
    {
        cout << "상품가격: " << price << endl;
        cout << "상품개수: " << numOfProduct << endl;
        cout << "현금: " << myMoney << endl << endl;
    }
};
```

- MyTv, Student, Card 클래스의 멤버변수의 접근제어자를 private으로 바꾼다
- 모든 멤버변수의 getter와 setter를 만든다
- 생성자를 만든다

MyTv

1. 기본 생성자
2. 매개변수가 1개인 생성자
: color 만 전달 받는 생성자

Student

1. 매개변수가 3개인 생성자
: 이름과 반 번호가 있는 생성자, 본 생성자 안에서 국영수 점수는 -1로 초기화
2. 매개변수가 6개인 생성자
: 이름, 반, 번호, 국, 영, 수 점수가 있는 생성자

Card

1. 카드 숫자와 모양을 갖는 생성자

클래스명 : Person

멤버변수 :

자료형	변수명	설명
char *	name	이름
int	age	나이

생성자 : 매개변수가 2개인 생성자

- a) 전달받은 myname보다 1이 많은 길이의 char 배열을 new로 생성하고 전달받은 이름을 new로 할당한 메모리에 복사한 후 멤버변수 name을 초기화
- b) 전달받은 myage로 멤버변수를 초기화

자료형	변수명	설명
const char *	myname	이름
int	myage	나이

메서드 :

반환타입	메서드명	매개변수	기능
없음	ShowPersonInfo	없음	사람의 이름과 나이를 콘솔창에 출력

2.5 소멸자

52

- 클래스 이름 앞에 ~가 붙은 형태의 이름
- 반환형을 선언하지 않음
- 객체 소멸과정에서 자동으로 호출되며, 소멸자가 정의되어 있지 않으면 디폴트 소멸자가 자동으로 삽입

2.9 클래스의 생성과 소멸

```
class Person
{
    private:
        char * name;
        int age;
    public:
        Person(const char * myname, int myage)
        {
            int len = strlen(myname) + 1;
            name = new char[len];
            strcpy(name, myname);
            age = myage;
        }

        void ShowPersonInfo() const
        {
            cout << "이름: " << name << endl;
            cout << "나이: " << age << endl;
        }

        ~Person()
        {
            delete[] name;
            cout << "called destructor!" << endl;
        }
};
```

```
int main(void)
{
    Person man1("jennifer", 29);
    Person man2("sunny", 41);
    man1.ShowPersonInfo();
    man2.ShowPersonInfo();
    return 0;
}
```

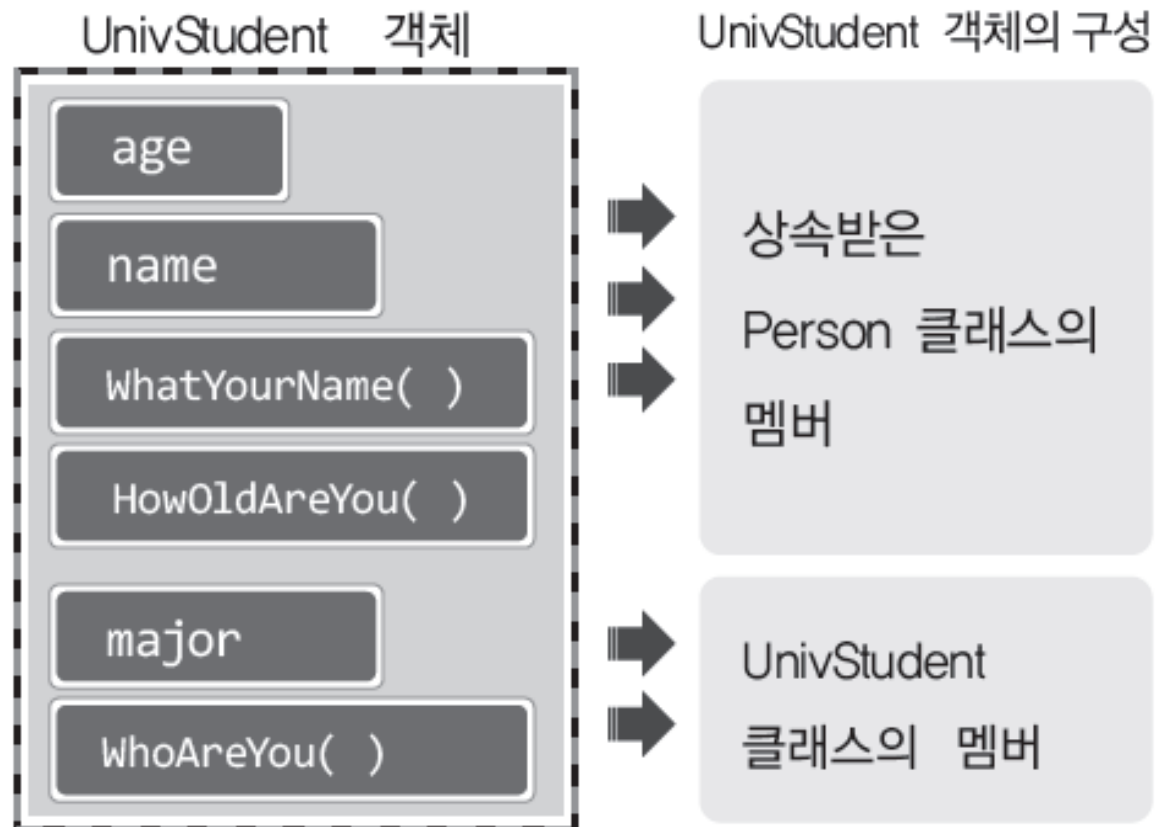
```
이름: jennifer
나이: 29
이름: sunny
나이: 41
called destructor!
called destructor!
```

3. 상속 (INHERITANCE) / 가상함수와 추상클래스

3.1 일반적인 상속의 의미 – 부모로부터 자산이나 재능, DNA를 물려받다

3.1 상속의 기본개념

Person	↔	UnivStudent
상위 클래스	↔	하위 클래스
기초(base) 클래스	↔	유도(derived) 클래스
슈퍼(super) 클래스	↔	서브(sub) 클래스
부모 클래스	↔	자식 클래스



상속 예제 1

56

```
class MyCar {
    private:
        int dorNum;
        char color[30];

    public:
        MyCar() {
            dorNum = 2;
            strcpy(color, "black");
        }

        MyCar(int num, const char * str) {
            dorNum = num;
            strcpy(color, str);
        }

        void showCarInfo() {
            cout << "문 개수:" << dorNum << ", 자동차 색깔:"
                << color << endl;
        }
};
```

```
class MySunRoofCar : public MyCar {

    bool bSunRoof;

};
```

```
int main() {

    MyCar car1(4, "white");
    car1.showCarInfo();

    MyCar car2(3, "red");
    car2.showCarInfo();

    MySunRoofCar car3;
    car3.showCarInfo();

    return 0;
}
```


상속 예제 2

```
class MyCar {
private:
    int dorNum;
    char color[30];

public:
    MyCar() {
        dorNum = 2;
        strcpy(color, "black");
    }

    MyCar(int num, const char* str) {
        dorNum = num;
        strcpy(color, str);
    }

    void showCarInfo() {
        cout << "문 개수:" << dorNum << ", 자동차  
색깔:" << color << endl;
    }
};
```

```
class MySunRoofCar : public MyCar {

private:
    bool bSunRoof;

public:
    bool isSunRoof() {
        return bSunRoof;
    }

};
```

```
int main() {
    MyCar car1(4, "white");
    car1.showCarInfo();

    MyCar car2(3, "red");
    car2.showCarInfo();

    MySunRoofCar car3;
    car3.showCarInfo();
    cout<<car3.isSunRoof();

    return 0;
}
```

상속 예제 3

58

```
class MyCar {  
public:  
    int dorNum;  
    char color[30];  
  
public:  
    MyCar() {  
        dorNum = 2;  
        strcpy(color, "black");  
    }
```

```
    MyCar(int num, const char * str) {  
        dorNum = num;  
        strcpy(color, str);  
    }
```

```
    void showCarInfo() {  
        cout << "문 개수:" << dorNum << ", 자동차 색깔:"  
        << color << endl;  
    }  
};
```

```
class MySunRoofCar : public MyCar {  
  
    public:  
        bool bSunRoof;  
  
    MySunRoofCar(int num, const char * str, bool sunRoof):MyCar(num, str)  
    {  
        bSunRoof = sunRoof;  
    }  
};
```

```
int main() {  
    MyCar car1(4, "white");  
    car1.showCarInfo();  
  
    MyCar car2(3, "red");  
    car2.showCarInfo();  
  
    MySunRoofCar car3(1, "black", true);  
    car3.showCarInfo();  
  
    return 0;  
}
```

```
class MySunRoofCar : public MyCar {  
  
    public:  
    bool bSunRoof;  
  
    MySunRoofCar(int num, const char* str, bool sunRoof):MyCar(num, str) {  
        bSunRoof = sunRoof;  
    }  
  
    void showCarInfo() {  
        cout << "문 개수:" << dorNum << ", 자동차 색깔:" << color;  
        if (bSunRoof) cout << " 썬루프";  
        cout << endl;  
    }  
  
};
```

3.1 상속의 기본개념

60

```
class Person
{
private:
    int age;           // 나이
    char name[50];     // 이름
public:
    Person(int myage, char * myname) : age(myage)
    {
        strcpy(name, myname);
    }
    void WhatYourName() const
    {
        cout<<"My name is "<<name<<endl;
    }
    void HowOldAreYou() const
    {
        cout<<"I'm "<<age<<" years old"<<endl;
    }
};
```

Base Class

```
class UnivStudent : public Person
{
private:
    char major[50];    // 전공과목
public:
    UnivStudent(char * myname, int myage, char * mymajor)
        : Person(myage, myname)
    {
        strcpy(major, mymajor);
    }
    void WhoAreYou() const
    {
        WhatYourName();
        HowOldAreYou();
        cout<<"My major is "<<major<<endl<<endl;
    }
};
```

Derived Class

3.1 상속을 통한 클래스 생성과정

61

```
UnivStudent(char * myname, int myage, char * mymajor)
    : Person(myage, myname)
{
    strcpy(major, mymajor);
}
```

```
int main(void)
{
    UnivStudent ustd1("Lee", 22, "Computer eng.");
    ustd1.WhoAreYou();

    UnivStudent ustd2("Yoon", 21, "Electronic eng.");
    ustd2.WhoAreYou();
    return 0;
};
```

이니셜라이저를 통해서 유도클래스는 기초클래스의 생성자를 명시적으로 호출해야한다.

유도클래스의 생성자는 기초클래스의 멤버를 초기화할 의무를 갖는다.

단! 기초클래스의 생성자를 명시적으로 호출해서 초기화해야한다.

때문에 유도클래스 UnivStudent는 기초클래스의 생성자 호출을 위한 인자까지 함께 전달받아야 한다.

Private 멤버는 유도클래스에서도 접근이 불가능하므로, 생성자의 호출을 통해서

기초클래스의 멤버를 초기화해야 한다

3.2 상속클래스의 생성과 소멸

62

- 생성-> 이니셜라이저
 - Base 클래스의 생성자 명시적 호출

```
class Student: public Person
{
    char major[20]; //전공
public:
    Student(int _age, char* _name, char* _major){
        age=_age;
        strcpy(name, _name);
        strcpy(major, _major);
    }
    const char* GetMajor() const {
        return major;
    }
    void ShowData() const {
        cout<<"이름: "<<GetName()<<endl;
        cout<<"나이: "<<GetAge()<<endl;
        cout<<"전공: "<<GetMajor()<<endl;
    }
};
```

```
BBB(int j) : AAA(j) {
    cout<<"BBB(int j) call!"<<endl;
}
```

```
int main(void)
{
    Student Kim(20, "Hong Gil Dong", "computer");
    Kim.ShowData();

    return 0;
};
```

3.2 상속 클래스 객체의 생성과 소멸

63

•소멸자의 호출 순서

```
class AAA    //Base 클래스
{
public:
    AAA(){
        cout<<"AAA() call!"<<endl;
    }
    ~AAA(){cout<<"~AAA() call!"<<endl;
    }
};
```

- 첫째 : Derived 객체 소멸자 호출
- 둘째 : Base 객체 소멸자 호출
- 셋째 : 메모리 반환

```
class BBB : public AAA    //Derived 클래스
{
public:
    BBB(){
        cout<<"BBB() call!"<<endl;
    }
    ~BBB(){
        cout<<"~BBB() call!"<<endl;
    }
};
```

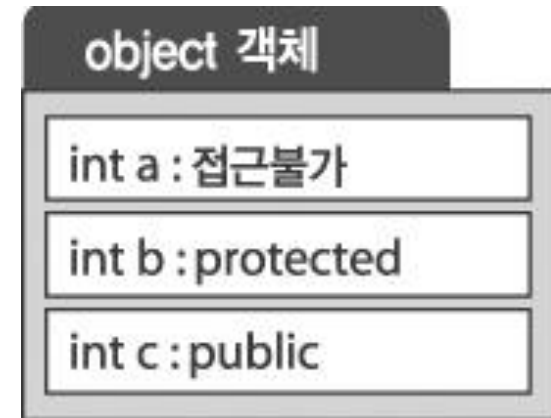
3.3 클래스 상속의 접근 권한

64

```
class Base
{
    Private:
        int a;
    Protected:
        int b;
    Public:
        int c;
};

class Derived : public Base // public 상속
{
    // EMPTY
};
```

```
int main(void)
{
    Derived object;
    return 0;
};
```



- 접근 권한 변경

- Base 클래스의 멤버는 상속되는 과정에서 접근 권한 변경

3.3 클래스 상속과 권한

private < protected < public

```
class Base
{
private:
    int num1;
protected:
    int num2;
public:
    int num3;
    void ShowData()
    {
        cout<<num1<<"", "<<num2<<"", "<<num3;
    }
};
```

```
class Derived : public Base
{
public:
    void ShowBaseMember()
    {
        cout<<num1;        // 컴파일 에러
        cout<<num2;        // 컴파일 OK!
        cout<<num3;        // 컴파일 OK!
    }
};
```

3.4 VIRTUAL의 원리와 추상클래스

다음의 두 개 클래스를 구현하세요

클래스명 : Point

멤버변수 : int형 x, y 좌표를 저장

생성자 : 두 개의 정수형 인자를 받아 멤버변수 초기화

멤버함수 : show() - x, y 좌표값을 콘솔에 출력

클래스명 : Point3D

조상클래스 : Point

멤버변수 : int 형 z 좌표

생성자 : 세 개의 정수를 입력받아 x, y, z를 초기화

멤버함수 : show() - x, y, z 좌표를 콘솔창에 출력

가상함수

클래스명 : Point

멤버변수 : int형 x, y 좌표를 저장

생성자 : 두 개의 정수형 인자를 받아 멤버변수 초기화

멤버함수 : show() - x, y 좌표값을 콘솔에 출력

클래스명 : Point3D

조상클래스 : Point

멤버변수 : int 형 z 좌표

생성자 : 세 개의 정수를 입력받아 x, y, z를 초기화

멤버함수 : show() - x, y, z 좌표를 콘솔창에 출력

```
int main() {  
  
    Point * p = new Point(3, 5);  
    p->show();  
  
    Point3D* p2 = new Point3D(1, 2, 3);  
    p2->show();  
  
    p= (Point*)p2;  
    p->show();  
  
}
```

```
3, 5  
1, 2, 3  
1, 2
```

```
class Point {  
    public:  
    int x;  
    int y;  
  
    Point(int x, int y) :x(x), y(y) {  
    }  
  
    void show() {  
        cout << x << ", " << y << endl;  
    }  
  
};  
  
class Point3D : Point {  
    public:  
    int z;  
  
    Point3D(int x, int y, int z) : Point(x, y), z(z) {  
    }  
  
    void show() {  
        cout << x << ", " << y << ", " << z << endl;  
    }  
  
};
```

가상함수

가상함수

- virtual 키워드를 사용해 선언
- 가상함수를 오버라이딩한 함수도 가상함수가 된다

```
int main() {  
  
    Point * p = new Point(3, 5);  
    p->show();  
  
    Point3D* p2 = new Point3D(1, 2, 3);  
    p2->show();  
  
    Point *p3 = (Point*)p2;  
    p3->show();  
  
}
```

```
3, 5  
1, 2, 3  
1, 2, 3
```

```
class Point {  
    public:  
    int x;  
    int y;  
  
    Point(int x, int y) :x(x), y(y) {  
    }  
  
    virtual void show() {  
        cout << x << ", " << y << endl;  
    }  
  
};  
  
class Point3D : Point {  
    public:  
    int z;  
  
    Point3D(int x, int y, int z) : Point(x, y), z(z) {  
    }  
  
    virtual void show() {  
        cout << x << ", " << y << ", " << z << endl;  
    }  
  
};
```

순수가상함수

70

순수가상함수

함수의 몸체가 구현되지 않음 함수

추상클래스

하나 이상의 멤버함수가 순수 가상함수인 클래스

```
class Student {  
    protected :  
    char name[30];  
    int id;  
  
    public:  
    Student(const char* name, int id):id(id) {  
        strcpy(this->name, name);  
    }  
  
    virtual void show_info() const = 0;  
};
```

```
class Student_Java : Student{  
  
    public:  
    Student_Java(const char* name, int id) : Student(name, id)  
    {  
    }  
    virtual void show_info() const {  
        cout << name << ", " << id << " : 자바기반 수업" << endl;  
    }  
};  
  
class Student_C : Student {  
  
    public :  
    Student_C(const char* name, int id) : Student(name, id) {  
    }  
  
    virtual void show_info() const {  
        cout << name << ", " << id << " : C/C++ 수업" << endl;  
    }  
};
```

```
int main() {  
  
    Student * pS = (Student*) new Student_C("제니퍼", 1);  
    pS->show_info();  
  
    Student* pS2 = (Student*) new Student_Java("제니퍼", 1);  
    pS2->show_info();  
}
```

순수가상함수를 사용할 경우 추상클래스의 자손클래스에서 가상함수를 구현하지 않으면 객체를 생성할 수 없게 된다
Student_Java 클래스의 show_info 멤버함수를 삭제하면 다음과 같은 컴파일 오류가 발생한다.

```
int main() {  
  
    Student * pS = (Student*) new Student_C("제니퍼", 1);  
    pS->show_info();  
  
    Student* pS2 = (Student*) new Student_Java("제니퍼", 1);  
    pS2->show_info();  
}
```

4. 템플릿 (TEMPLATE) / 표준 템플릿 라이브러리

템플릿은 함수를 자동으로 생성하기도 하고 클래스를 자동으로 생성하기도 한다

```
class Data
{
    int data;
public:
    Data(int d) {
        data = d;
    }

    int GetData() {
        return data;
    }
};
```

```
int main(void)
{
    Data d1(10);
    cout << d1.GetData() << endl;

    Data d2(3.14);
    cout << d2.GetData() << endl;

    Data d3('a');
    cout << d3.GetData() << endl;

    return 0;
}
```

```
10
3
97
```

위의 코드에 템플릿을 적용하면?

템플릿 Template - 템플릿클래스

74

템플릿은 함수를 자동으로 생성하기도 하고 클래스를 자동으로 생성하기도 한다

```
template <typename T>
class Data
{
    T data;
public:
    Data(T d) {
        data = d;
    }

    T GetData() {
        return data;
    }
};
```

```
int main(void)
{
    Data<int> d1(10);
    cout << d1.GetData() << endl;

    Data<float> d2(3.14);
    cout << d2.GetData() << endl;

    Data<char> d3('a');
    cout << d3.GetData() << endl;

    return 0;
}
```

```
10
3.14
a
```

이것을 **템플릿 클래스**라고 한다

1. 두 개의 int 형 인자를 전달받아 큰 수를 반환하는 함수를 구현한다.
2. 두 개의 float형 인자를 전달받아 큰 수를 반환하는 함수를 구현한다
3. main에서 위 두 함수를 테스트한다

1. 두 개의 int 형 인자를 전달받아 큰 수를 반환하는 함수를 구현한다.
2. 두 개의 float형 인자를 전달받아 큰 수를 반환하는 함수를 구현한다
3. main에서 위 두 함수를 테스트한다

```
int GetMax(int a, int b) {  
    return (a > b) ? a : b;  
}  
  
double GetMax(double a, double b) {  
    return (a > b) ? a : b;  
}  
  
int main() {  
  
    cout << GetMax(1, 3) << endl;  
    cout << GetMax(3.11, 3.14) << endl;  
  
    return 0;  
}
```

템플릿함수를 적용한다면?

1. 두 개의 int 형 인자를 전달받아 큰 수를 반환하는 함수를 구현한다.
2. 두 개의 float형 인자를 전달받아 큰 수를 반환하는 함수를 구현한다
3. main에서 위 두 함수를 테스트한다

```
template <typename T>
T GetMax(T a, T b) {
    return (a > b) ? a : b;
}

int main() {

    cout << GetMax(1, 3) << endl;
    cout << GetMax(1.1, 3.1) << endl;

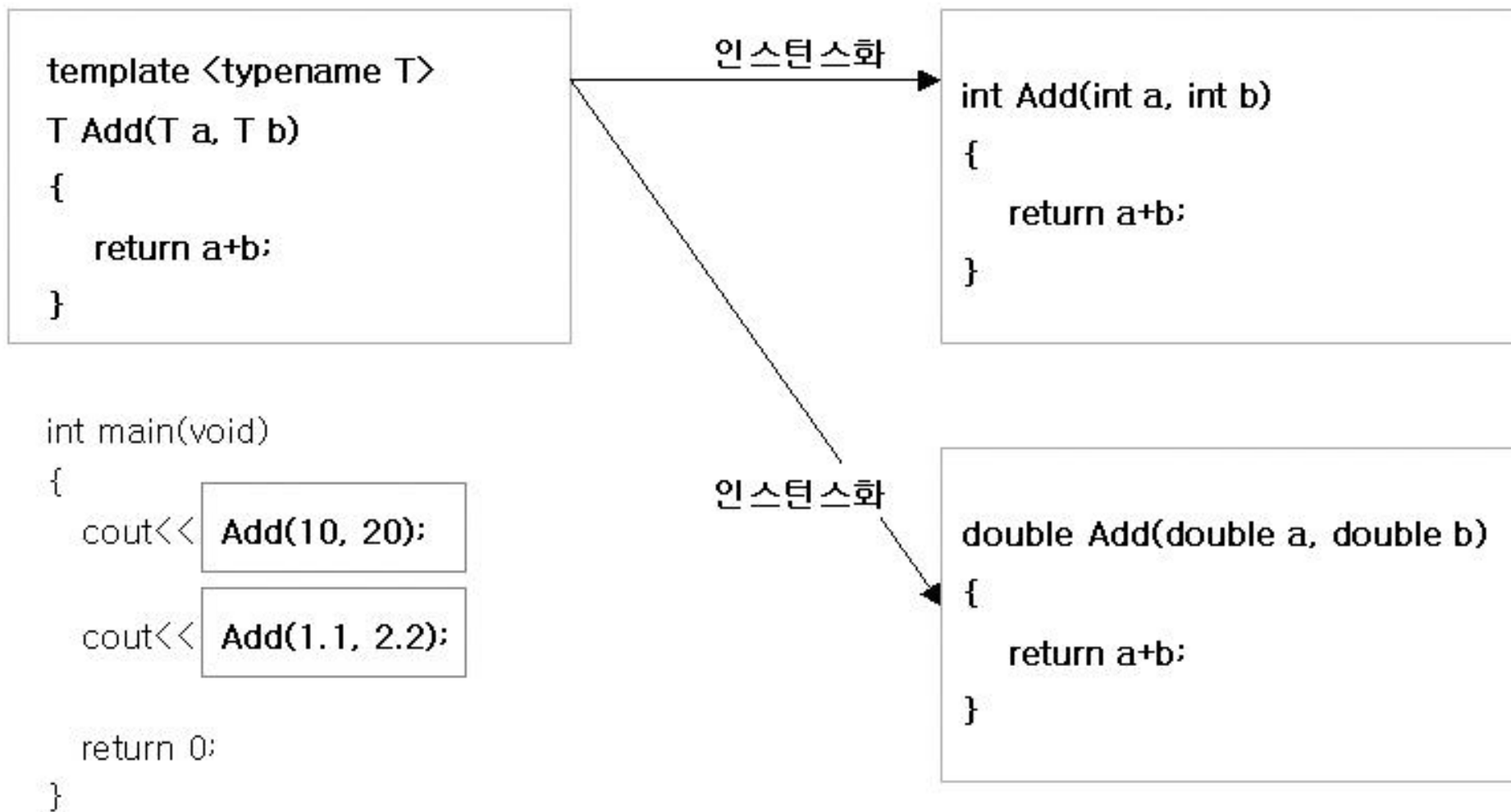
    return 0;
}
```

템플릿 함수는 타입 명시 없이
사용이 가능하다

여러 소스파일로 구성된 프로젝
트에서 템플릿함수는 헤더파일
에 놓아야 한다

4.4 템플릿의 원리 이해

78



4.5 STL (Standard Template Library)

- **표준템플릿 라이브러리의 대표적인 3가지 STL**
 1. 임의 타입의 객체를 보관할 수 있는 컨테이너 (container)
 2. 컨테이너에 보관된 원소에 접근할 수 있는 반복자 (iterator)
 3. 반복자들을 가지고 일련의 작업을 수행하는 알고리즘 (algorithm)

4.5 STL (Standard Template Library)

80

1. 컨테이너 (Container)

컨테이너는 특정 데이터 구조 유형의 개체 모음입니다. STL에는 Array, vector, queue, deque, list, map, set 등과 같은 다양한 유형의 컨테이너 클래스가 있습니다. 이러한 컨테이너는 본질적으로 일반적이며 클래스 템플릿으로 구현됩니다.

컨테이너는 본질적으로 동적이며 다양한 유형의 개체를 보관하는 데 사용할 수 있습니다.

2 반복자 (iterator)

반복자는 STL의 매우 중요하고 구별되는 기능입니다. 반복기는 컨테이너 개체를 통과하는 데 사용되는 구성입니다. 배열을 단계별로 실행하는 데 사용하는 인덱스와 유사하게 반복기는 컨테이너 클래스 개체에 대해 작동하며 데이터를 단계별로 실행하는 데 사용할 수 있습니다.

3 알고리즘 (algorithm)

알고리즘은 컨테이너에서 작동하는 메서드 또는 함수입니다. STL에서 제공하는 알고리즘을 사용하여 컨테이너 클래스 객체의 내용을 검색, 정렬, 수정, 변환 또는 초기화하는 방법을 가질 수 있습니다.

STL에서 제공하는 알고리즘에는 알고리즘을 직접 작성하지 않고도 복잡한 데이터 구조에서 직접 작동 할 수 있는 내장 함수가 있습니다.

STL의 reverse () 함수를 사용하여 연결 목록을 반전시킬 수 있습니다.

1. 컨테이너 (Container)

컨테이너는 개체와 데이터를 저장합니다. 기본적으로 템플릿 기반의 제네릭 클래스입니다.

C++ STL 에서 컨테이너는 크게 두 가지 종류가 있습니다.

1) 시퀀스 컨테이너 (sequence container)

배열 처럼 객체들을 순차적으로 보관 - **vector, list, deque** 대표적으로 정의되어 있습니다.

벡터(std::vector) 의 경우, 쉽게 생각하면 가변길이 배열이라 보시면 됩니다.

벡터에는 원소들이 메모리 상에서 실제로 순차적으로 저장되어 있고,

따라서 임의의 위치에 있는 원소를 접근하는 것을 매우 빠르게 수행할 수 있습니다.

2) 연관 컨테이너 (associative container)

키를 바탕으로 대응되는 값을 찾아주는 컨테이너 - **set, multiset, map, multimap** 대표적입니다.

Vector의 생성

`vector<int> v;`;- 비어있는 vector v를 생성합니다.

`vector<int> v(5);`;- 기본값(0)으로 초기화 된 5개의 원소를 가지는 vector v를 생성

`vector<int> v(5, 2);`;- 2로 초기화된 5개의 원소를 가지는 vector v를 생성

`vector<int> v1(5, 2);vector<int> v2(v1);`;- v2는 v1 vector를 복사해서 생성

Vector 자주 사용하는 함수

`v.at(idx);`;- idx번째 원소를 참조 (`v[idx]` 보다 속도는 느리지만, 범위를 점검하므로 안전)

`v.clear();`;- 모든 원소를 제거.원소만 제거하고 메모리는 남아있음(`size`만 줄어 들고 `capacity`는 그대로)

`v.push_back(T);`;- 마지막 원소 뒤에 전달받은 인자를 삽입

`v.pop_back();`;- 마지막 원소를 제거

`v.begin();`;- 첫번째 원소를 가리킴 (iterator와 사용)

`v.end();`;- 마지막의 "다음"을 가리킬(iterator와 사용)

벡터 사용법

1. 벡터를 사용해 1부터 10까지 저장하라
2. 이렇게 저장한 벡터를 for문을 이용해 모든 요소를 출력하라

```
#include <vector>

void func_5_stl_1() {
    vector<int> vc;

    for (int i = 0; i < 10; i++) {
        vc.push_back(i + 1);
    }

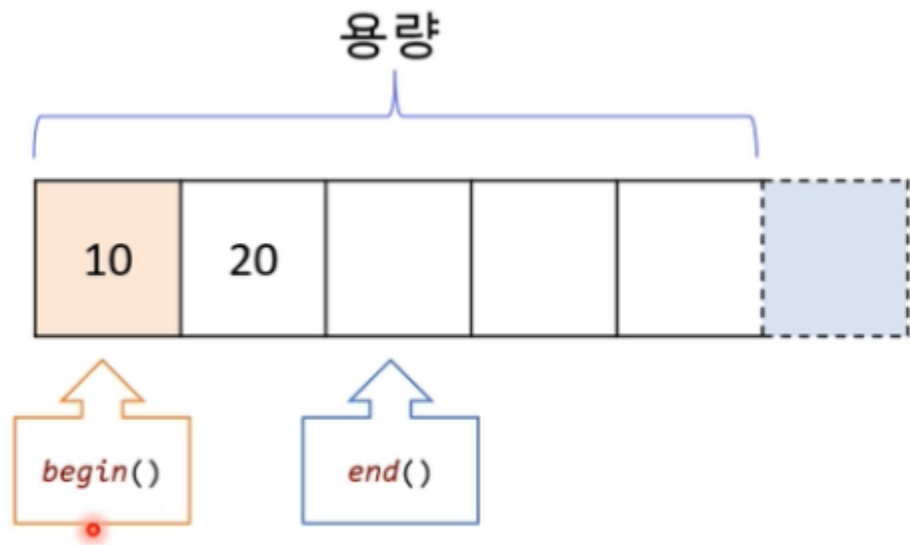
    for (int i = 0; i < vc.size(); i++) {
        cout << vc.at(i) << ", ";
        //cout << vc[i] << ", ";
    }
}
```

- 컨테이너에 저장되어있는 원소들을 공통적인 방법으로 하나씩 접근할 수 있게 해줌.
- 모든 컨테이너들이 다 같은 방법으로 반복자 사용 가능.
- 각 타입에 `::iterator` 또는 `::const_iterator` 를 뒤에 붙여주면 사용이 가능하다.

- `vector` 컨테이너의 반복자 `itr`
- `vector<int>::iterator itr;`
- `vector` 컨테이너의 `const`한 반복자 `citr`
- `vector<int>::const_iterator citr;`

- 포인터와 비슷하게 사용한다.
- 간접 참조 가능
- `itr = v.begin() + 2` 에서 `*itr` 간접 참조를 하면 세번째 원소 값이 리턴된다.

- `iterator` 와 `const_iterator` 의 차이
- `const_iterator` 는 반복자가 가리키는 원소의 값을 변경하지 못한다.
- 반복자 값이 변경되지 못하는게 아니고 반복자가 가리키는 원소의 값이 변경될 수 없는 것!
- 일반 반복자는 포인터와 비슷하고 `const` 반복자는 (간접참조로 값을 변경하지 못하는) `const` 포인터와 비슷하다.



begin();

- vector의 첫 번째 요소를 가리키는 반복자를 반환

end();

- vector의 마지막 요소 바로 뒤의 요소를 가리키는 반복자를 반환

```
vector<int>::iterator bIter = scores.begin();
```

```
vector<int>::iterator eIter = scores.end();
```

1부터 10까지의 정수를 Vector에 저장하고 iterator를 이용해 모든 벡터의 요소를 출력하라

```
void func_5_stl_1() {  
    vector<int> container;  
  
    for (int i = 0; i < 10; ++i)  
        container.push_back(i+1);  
  
    vector<int>::iterator itr;  
    itr = container.begin();  
  
    while (itr != container.end())  
    {  
        cout << *itr << " ";  
        ++itr;  
    }  
}
```

컨테이너에 반복자들을 가지고 정렬, 검색 등의 작업을 쉽게 수행할 수 있도록 도와주는 라이브러리

min_element

- 컨테이너 해당 범위의 가장 작은 원소의 반복자를 리턴한다.
- 컨테이너의 시작위치, 끝위치의 반복자들을 인수로 넘겨 줌 (범위 지정)
- begin 부터 end가 되기전까지의 범위 [begin, end) 중에서의 최소 값의 반복자

```
vector<int> container;  
for (int i = 0; i < 10; ++i)  
    container.push_back(i);  
vector<int>::iterator itr = min_element(container.begin(),  
container.end());  
cout << *itr << endl;
```

max_element

- 컨테이너 해당 범위의 가장 큰 원소의 반복자를 리턴한다.
- 컨테이너의 시작위치, 끝위치의 반복자들을 인수로 넘겨 줌 (범위 지정)
- begin 부터 end가 되기전까지의 범위 [begin, end) 중에서의 최대 값의 반복자

```
vector<int> container;  
  
for (int i = 0; i < 10; ++i)  
    container.push_back(i);  
  
vector<int>::iterator itr = max_element(container.begin(), container.end());  
cout << *itr << endl;
```


find

```
vector<int> container;  
  
for (int i = 0; i < 10; ++i)  
    container.push_back(i);  
  
vector<int>::iterator itr = find(container.begin(), container.end(), 3);
```

sort

```
sort(container.begin(), container.end());
```


5. 파일입출력 / 예외처리

1. 파일 입출력

C++ 의 전체적인 데이터 흐름을 data stream 이라고 하고, Stream에는 크게 두 가지 스트림으로 나누어 집니다.

1. 입출력 스트림(IO Stream) = 키보드입력 /모니터출력
2. 파일 스트림 (fstream)= 저장매체에 저장된 파일을 열고 읽고 쓰는 방법
 - 1) ifstream
 - 2) ofstream
 - 3) fstream

C++의 파일 입출력용 클래스인 ifstream, ofstream, fstream은 전부 fstream이라는 헤더파일 내에 존재합니다
따라서 fstream 헤더파일을 include 시켜야 합니다.

```
#include <fstream>
```

```
ofstream fout;
```

```
ifstream fin;
```

```
fstream fio;
```

[**ifstream**] -> Input file stream

"프로그램에 파일에 있는 어떠한 것들을 스트림 버퍼에 가지고 와서 프로그램에 입력한다."

1. 파일을 열때 사용하는 open 함수

함수원형 : `void open (const char* fileName, ios_base::openmode mode = ios_base::in);`

함수원형 : `void open (const string& fileName, ios_base::openmode mode = ios_base::in);`

첫번째 인자 : open할 파일 이름이 들어가게 됩니다.

두번째 인자로써는 오픈할 모드인데요. 오픈할 파일을 어떤식으로 사용할지?에 따라 모드를 정하면 됩니다.

ios_base::in - 파일을 read할 목적으로 open할 것이다.

ios_base::out - 파일에 write할 목적으로 open할 것이다.

ios_base::binary - 파일을 바이너리 형태로 open할 것이다.

2. 열렸는지 확인하는 `is_open` 함수

함수원형 : `bool is_open() const;`

함수설명 : 파일이 열렸는지 확인하는 함수

3. 파일을 열었으면 꼭 닫아야합니다 `close` 함수

함수원형 : `void close();`

함수설명 : 파일과의 연결을 닫아버리는 함수 입니다.

4. `char` 하나씩 파일에서 프로그램으로 읽어오는 `get` 함수

함수원형 : `istream& get (char& c);`

함수설명 : 읽은 파일에서 한char 단위로 읽어서 매개변수로 넣은 변수 c에 넣어주는 함수입니다.

```
char c;  
while(readFile.get(c))  
{  
    //읽은 char가 c에 들어있습니다.  
    cout << c;  
}
```

5. 한줄씩 라인단위로 읽기 `getline` 함수

함수원형 : `istream& getline(char* str, streamsize len);`

함수설명 : 한줄씩 문자열을 읽어서 `str`에 저장해주는 함수입니다.

한줄의 기준은 '\n' 문자열의 끝을 알리는 개행 문자가 올때 까지, 혹은 파일의 끝을 알리는 EOF를 만날때 까지 입니다

****위 `ifstream::getline()` 함수를 사용할때 주의할점은** 문자열을 받아오는 형태가 `char*` 타입이기 때문에 `string` 타입으로 바로 받을 수 없다는 특징이 있습니다.

6. 파일이 끝날때 까지 읽기 위한 `eof` 함수

함수원형 : `bool eof() const;`

함수설명 : 파일의 끝이 나오면 `true`를 반환하고 아니면 `false`를 반환합니다.

`eof()` 함수가 불리면 커서의 위치를 확인하는 내부 로직에 의해서 파일의 끝에 도착했는지 아닌지를 판단 하게 됩니다. 파일의 끝을 만나게 되면 `true`를 반환.

5.2 ifstream example code

97

```
std::ifstream readFile;           //읽을 목적의 파일 선언
readFile.open("words.txt");      //파일 열기
if(readFile.is_open())           //파일이 열렸는지 확인
{
    while(!readFile.eof())        //파일 끝까지 읽었는지 확인
    {
        char arr[256];
        readFromFile.getline(arr, 256); //한줄씩 읽어오기
    }
}
readFile.close();                //파일 닫기
```

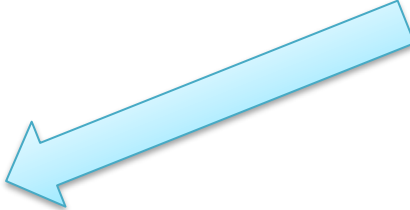
5.2 ifstream - Example

98

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

int main(void)
{
    ifstream readfile;
    readfile.open("test.txt"); //파일 열기

    if (readfile.is_open())
    {
        while (!readfile.eof())
        {
            //2. std::getline.
            string str;
            getline(readfile, str);
            cout << str << endl; //지금은 읽은 문자열 바로 출력.
        }
        readfile.close(); //파일 닫아줍니다.
    }
    return 0;
}
```



```
//1. ifstream의 getline.
/*
char tmp[256];
readfile.getline(tmp, 256);
cout << tmp << endl; //지금은 읽은 문자열 바로 출력
*/
```

프로그램의 출력을 파일에 할 수 있게 돕는 클래스

▽ ofstream 함수 원형과 멤버 변수

헤더파일 : <fstream>

1. 파일을 열때 사용하는 open 함수

함수원형 : `void open (const char* fileName, ios_base::openmode mode = ios_base::out);`

함수원형 : `void open (const string& fileName, ios_base::openmode mode = ios_base::out);`

함수설명 : 첫번째 인자는 파일이름입니다.

두번째 인자는 위에서 파일 읽기에서는 in이었는데 파일에 쓰기는 out으로 매개변수 default 사용법은 파일읽기 ifstream에서 사용한 방법과 동일함.

파일에서 읽어 올거냐(in), 파일에 쓸거냐(out)에 따라 두번째 매개변수가 달라지게 됩니다.

2. 파일쓰기 write 함수

함수원형 : **ostream& write(const char* str, streamsize n);**

함수설명 : 첫번째 매개변수로 받은 캐릭터 포인터 타입의 문자열의 n만큼의 길이만큼 파일에 write하는 함수

>> 파일에 쓰기 간단 예제

```
std::ofstream writeFile;           //쓸 목적의 파일 선언
writeFile.open("words.txt");       //파일 열기
char arr[11] = "BlockDMask";       //파일에 쓸 문자열
```

```
if(writeFile.is_open())           //파일이 열렸는지 확인
{
    writeFile.write(arr, 10);      //파일에 문자열 쓰기
}
writeFile.close();                //파일 닫기
```

****Important :**

c언어 배열로 나타내는 문자열은 문자열 끝에 'w0'이 들어가 있기 때문에 배열의 "총 길이-1"을 write의 두번째 인자로 넣어야 합니다.

따라서 "Love Love w0"는 char[] 배열의 길이는 11이지만, 실제로 문자는 10개 이므로 10을 넣어야 정상적으로 파일에 기록됩니다.

5.4 fstream - Example

101

```
#include<iostream>
#include<string>
#include<fstream>
#include<vector>
using namespace std;

int main(void)
{
    //프로그램에 내장되어있는 단어들
    vector<string> words = { "Cplusplus", "banana", "code", "program" };
    int len = static_cast<int>(words.size());

    //ifstream readFromFile;
    //readFromFile.open("words.txt"); //이 두줄을 한줄로 해결하는 방법은 아래방법!
    ifstream readFromFile("words.txt");

    //만약, 파일이 없다면 파일을 만들고 기본문자들 세팅
    if (readFromFile.is_open())
    {
        //파일이 존재합니다.

        words.clear(); //새 단어들을 읽어오기 전에 예제 단어들을 삭제

        while (!readFromFile.eof()) //단어장이 끝날때까지.
        {
```

```

    //ifstream::getline을 이용해서 char 배열 타입으로 읽어오는 방법
    //char arr[256];
    //readFromFile.getline(arr, 256);

    //std::getline 함수를 이용해서 string 타입으로 읽어오는 방법.
    //이 방법이 string을 사용하기 더 편리합니다.
    string tmp;
    getline(readFromFile, tmp);
    words.push_back(tmp);      //읽어온 단어 저장
}
readFromFile.close(); //반드시 파일 닫아줍니다.
}
else
{
    //is_open이 false 인 경우는 파일이 없거나, 해당 파일에 접근이 불가능한 경우.
    ofstream writeToFile;
    writeToFile.open("words.txt"); //파일을 새로 만들어줍니다.
    for (int i = 0; i < len; ++i)
    {
        string tmp = words[i];
        if (i != len - 1)
            tmp += "\n"; //마지막 단어 빼고 엔터 넣어주기

        //tmp.c_str() : C++ string -> const char* 타입으로 변환
        writeToFile.write(tmp.c_str(), tmp.size());
    }
    writeToFile.close(); //파일 닫기
}
return 0;
}

```

예외처리

일반적이지 않은 프로그램의 흐름의 처리이며 에러가 아니다

```
int main(void)
{
    int a, b;
    cout<<"두개의 숫자 입력 : ";
    cin>>a>>b;

    cout<<"a/b의 몫 : "<<a/b<<endl;
    cout<<"a/b의 나머지 : "<<a%b<<endl;
    return 0;
}
```

```
int main(void)
{
    int a, b;

    cout<<"두개의 숫자 입력 : ";
    cin>>a>>b;
    if(b==0){
        cout<<"입력오류!"<<endl;
    }
    else {
        cout<<"a/b의 몫 : "<<a/b<<endl;
        cout<<"a/b의 나머지 : "<<a%b<<endl;
    }
    return 0;
}
```

5.5 C++의 예외처리 메커니즘 – try..catch

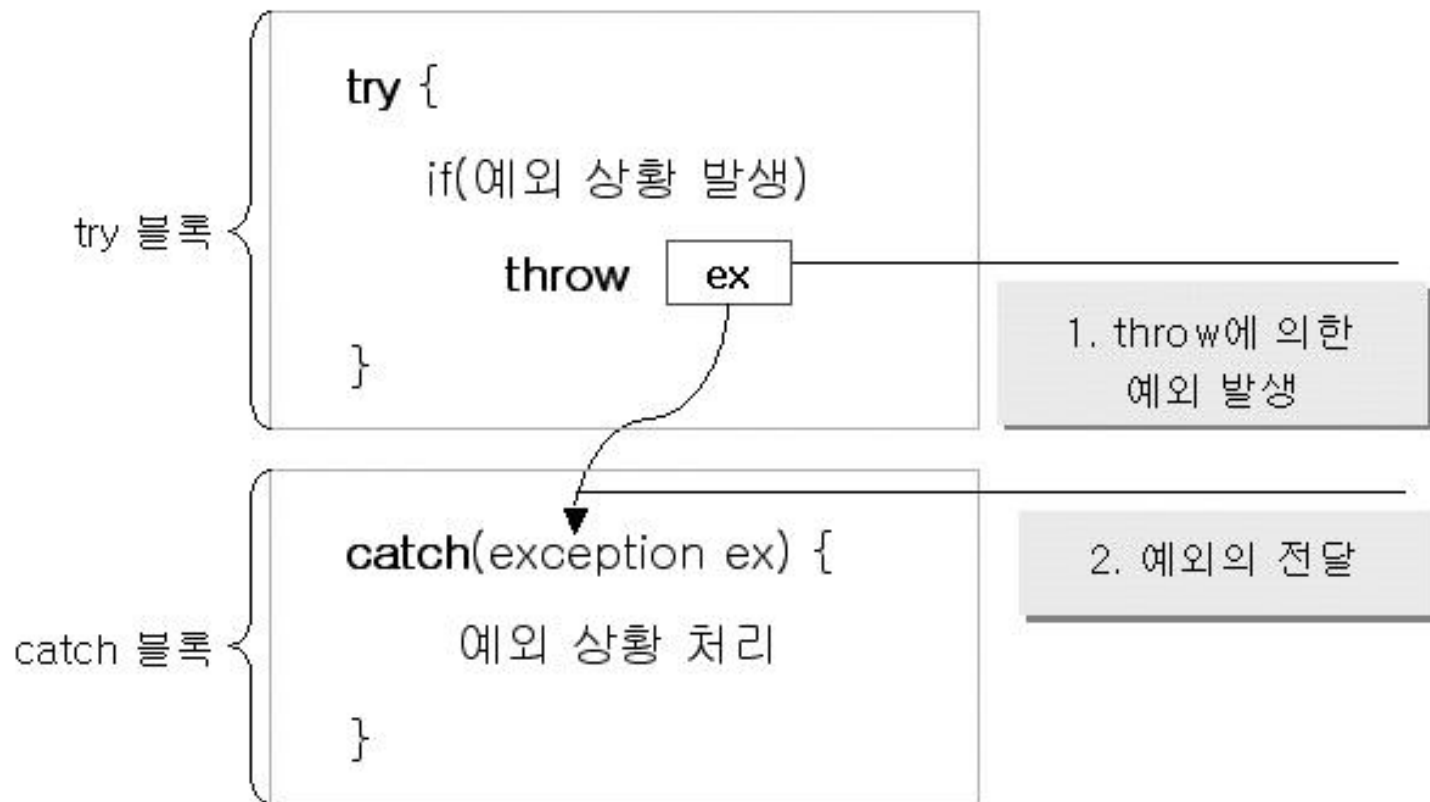
```
try {  
    /* 예외 발생 예상 지역 */  
}
```

```
throw X; // X “발생된 예외를 전달”
```

```
catch(처리 되어야 할 예외의 종류) {  
    /* 예외를 처리하는 코드가 존재할 위치 */  
}
```


5.5 C++의 예외처리 메커니즘 – try..catch

105



5.5 C++의 예외처리 메커니즘 - try..catch

106

```
int main(void)
{
    int a, b;

    cout<<"두개의 숫자 입력 : ";
    cin>>a>>b;

    try{
        if(b==0)
            throw b;
        cout<<"a/b의 몫 : "<<a/b<<endl;
        cout<<"a/b의 나머지 : "<<a%b<<endl;
    }
    catch(int exception){
        cout<<exception<<" 입력."<<endl;
        cout<<"입력오류! 다시 실행 하세요."<<endl;
    }

    return 0;
}
```



5.6 C++의 예외처리 flow

107

```
int main(void)
{
    int a, b;

    cout<<"두개의 숫자 입력 : ";
    cin>>a>>b;

    try{
        cout<<"try block start"<<endl;

        if(b==0)
            throw b;
        cout<<"a/b의 몫 : "<<a/b<<endl;
        cout<<"a/b의 나머지 : "<<a%b<<endl;

        cout<<"try block end"<<endl;
    }
    catch(int exception){
        cout<<"catch block start"<<endl;

        cout<<exception<<" 입력."<<endl;
        cout<<"입력오류! 다시 실행 하세요."<<endl;
    }

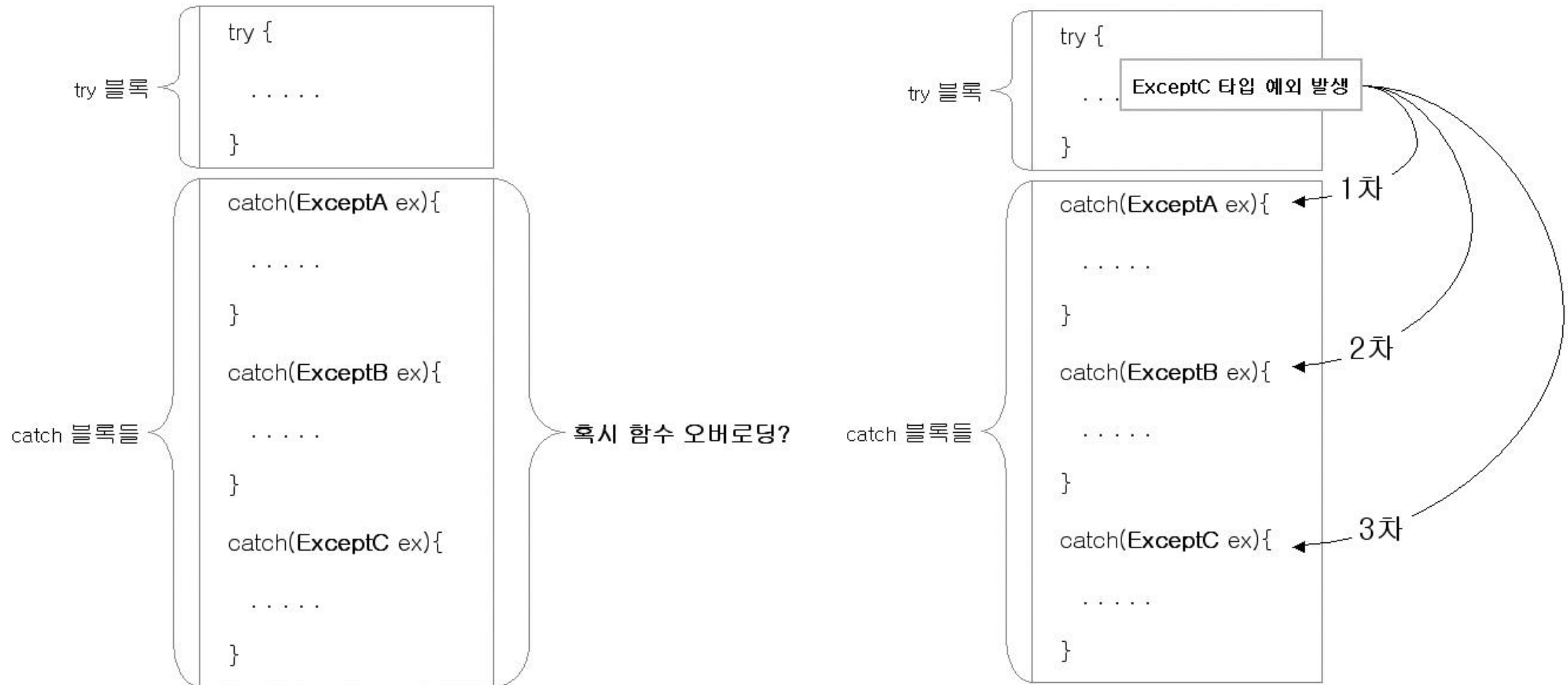
    cout<<"THANK YOU!"<<endl;
    return 0;
}
```

예외가 발생하면 try 블록의 나머지 부분 무시
예외 처리 후 catch 블록 이후부터 실행

5.7 예외처리 flow - 예외가 전달되는 방식 -> 클래스와 상속

108

catch 블록에 예외가 전달되는 방식
inheri_catch1.cpp, inheri_catch2.cpp



하나의 try, 여러 개의 catch
catch_understand.cpp,
catch_over1.cpp, catch_over2.cpp

```
int main(void)
{
    int num;

    cout<<"input number: ";
    cin>>num;

    try{
        if(num>0)
            throw 10; // int형 예외 전달.
        else
            throw 'm'; // char형 예외 전달.
    }
    catch(int exp){
        cout<<"int형 예외 발생"<<endl;
    }
    catch(char exp){
        cout<<"char형 예외 발생"<<endl;
    }
    return 0;
}
```

5.7 new 연산자에 의해 전달되는 예외

110

bad_alloc

new 연산자가 메모리 공간 할당을 실패 했을 때 발생시키는 예외

```
using std::bad_alloc;

int main(void)
{
    try{
        int i=0;
        while(1){
            cout<<i++<<"번째 할당"<<endl;
            double(*arr)[10000]=new double[10000][10000];
        }
    }
    catch(bad_alloc ex){
        ex.what();
        cout<<endl<<"END"<<endl;
    }

    return 0;
}
```

수고 많으셨습니다.