C++언어 기초 및 활용

저급 언어와 고급 언어

- **저급 언어(Low-Level Language)**
  - **기계** 중심 언어
  - 실행 속도가 빠르다
  - 기계 마다 다른 언어를 가진다

- **고급 언어(Hight-Level Language)**
  - **사람** 중심 언어
  - 실행을 위해 번역 과정이 필요
  - 기계 마다 소스의 수정 없이 사용이 가능

	저급 언어	고급 언어
실행 속도	빠름	느림
사용성	어려움	쉬움
호환성	나쁨	좋음

빌드 과정

- **빌드(Build)**
  - 실행 파일(*.exe)을 만드는 과정

- **전처리기(Preprocessor)**
  - 컴파일러를 위해 **코드를 정리**
  - 전처리기 구문(#...)으로 헤더 파일(*.h)을 불러오거나 코드상에 필요한 기호 상수(#define)로 코드를 채워주는 등의 작업
  - *.cpp 파일이 *.i 파일로 변환

- **컴파일러(Compiler)**
  - 고급 언어(**cpp language**)를 저급 언어(**assembly language**)로 **번역**
  - 컴파일 에러 : 문법에 맞지 않아 번역을 할 수 없음
  - *.i 파일이 *.s파일로 변환

- **어셈블러(Assembler)**
  - 저급 언어(**assembly language**)를 저급 언어(**기계어:machine language**)로 **번역**
  - *.s 파일이 *.o파일로 변환
  - 컴파일이 끝난 **목적 파일(*.obj)이 생성**

- **링커(Linker)**
  - **라이브러리(*.lib)나 목적 파일(*.obj)을 하나로 연결(link)**
  - **실행 파일(*.exe)** 또는 라이브러리 파일(*.dll)을 **생성**

Hello World

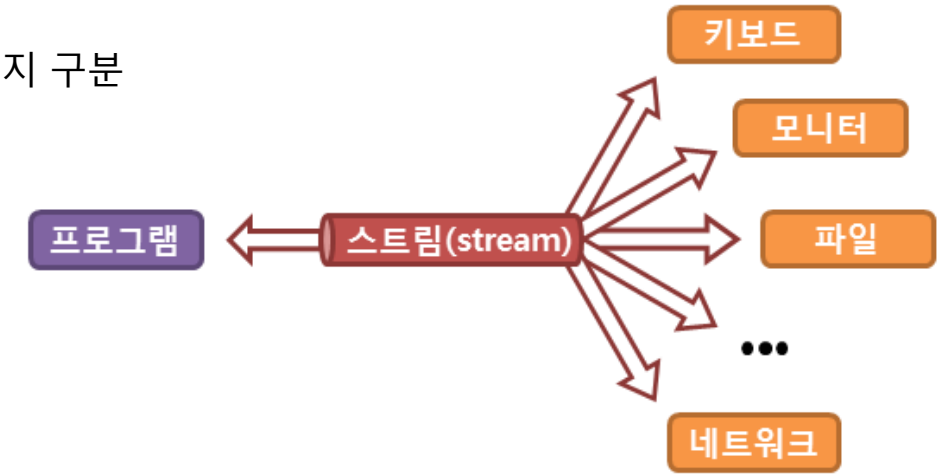
```
#include <iostream>

int main()
{
    std::cout << "Hello World!!" << std::endl;

    return 0;
}
```

Hello World

- **Stream**
  - 입력이나 출력이 표현된 데이터의 이상화된 흐름, 중간 매개체

- **namespace**
  - 하나의 유효 영역(스코프)안에 같은 이름의 식별자가 둘 이상 존재할 경우, 모든 식별자가 고유하도록 보장하는 코드 영역

- **범위 지정 연산자(::)**
  - 어느 namespace의 변수 및 함수인지 구분



변수

- **variable**
  - 데이터를 저장할 수 있는 메모리 공간

이름	크기(바이트)	범위
bool	1	false(0), true(1)
(signed/unsigned) char	1	-2^7 ~ 2^7-1 0 ~ 255
(signed/unsigned) short	2	-2^{15} ~ $2^{15}-1$ 0 ~ 65,535
(signed/unsigned) int	4	-2^{31} ~ $2^{31}-1$ 0 ~ 4,294,967,295
(signed/unsigned) long	4	-2^{31} ~ $2^{31}-1$ 0 ~ 4,294,967,295
(signed/unsigned) long long	8	-2^{63} ~ $2^{63}-1$ 0 ~ 18,446,744,073,709,551,615
float	4	$3.4*10^{-38}$ ~ $3.4*10^{38}$
(long) double	8	$-1.79*10^{308}$ ~ $1.79*10^{308}$
enum	varies(기본 4)	None

상수화

- **const**
  - const 예약어
  - **값을** 변경할 수 없는(constant), **상수로 고정** 시키기 위한 키워드
  - **반드시 변수 선언과 동시에 값이 초기화** 되어야 한다

```
const int value
```

- **포인터 상수**
  - 포인터가 가리키는 주소의 **값을 상수화**

```
const int* ptr
```

- **상수 포인터**
  - 포인터가 가리키는 **주소를 상수화**

```
int* const ptr
```

- **상수 포인터 상수**
  - 포인터가 가리키는 주소와 값 모두를 상수화

```
const int* const ptr
```

클래스(class)

- **클래스(class) 란?**
  - C언어에서 배우는 구조체(structure)의 확장
  - 다양한 타입의 **변수와 함수의 집합**
  - Default 접근 제한 : **private**

- **C++에서의 구조체(structure)**
  - C 언어와 달리 함수를 가질 수 있다
  - Default 접근 제한 : **public**

```
#include <string>
class Person
{
public:
    void Speak()
    {
        std::cout << "Hello World!!" << std::endl;
    }

    std::string name;
    int gender;
    int age;
};
```

클래스(class)

- **생성자(constructor)**
  - 해당 클래스의 객체가 인스턴스화될 때 자동으로 호출되는 특수한 종류의 멤버 함수
  - 멤버 변수를 적절한 기본값 또는 사용자 제공 값으로 초기화

- **소멸자(Destructor)**
  - 객체가 소멸될 때 자동으로 실행되는 클래스의 멤버 함수

```
class Vector2
{
public:
    Vector2() { }
    ~Vector2() { }

    float x, y;
};
```

객체 지향 프로그래밍(OOP)

- **객체 지향 프로그래밍(Object-Oriented Programming)**

- 객체 : 생활에 보이며 사용되는 모든 것
- **객체** : **틀**(class)**을 이용하여 만들어진 실체**(instance)

	절차 지향	객체지향
기준	기능	객체
구현	무엇을 어떤 절차로	누가 어떤 일을

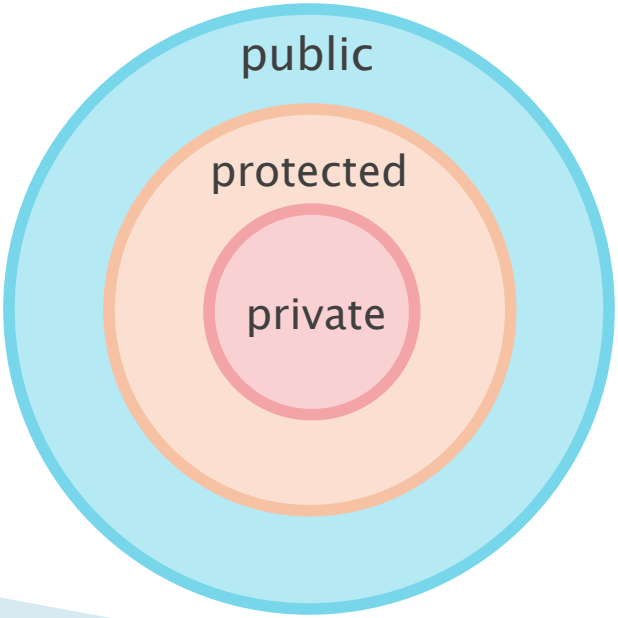
- **정보 은닉**
- **캡슐화**
- **상속성**
- **다형성**

정보 은닉(information hiding)

- **데이터를 외부에 보이지 않도록 정보를 숨긴다**

- **접근 제한자**

- 멤버 속성을 외부에서 접근을 막거나 허용 하도록 한다
- public : 클래스 외부에서 접근/사용 할 수 있도록 허용
- protected : 상속 관계에 있는 자식 클래스까지만 접근/사용을 허용
- private : 클래스 내부에서만 접근/사용을 허용



정보 은닉(information hiding)

```
class Vector2
{
public:
    Vector2() {}
    ~Vector2() {}

private:
    float x, y;
};
```

```
class Person
{
public:
    void Speak()
    {
        std::cout << "Hello World!!" << std::endl;
    }

private:
    std::string name = "홍길동";
    int gender = 0;
    int age = 0;
};
```

캡슐화(encapsulation)

- 숨겨둔 데이터를 함수와 묶어서 관리
- 숨겨둔 데이터를 외부에서 접근할 수 있도록 한다

```
class Vector2
{
public:
    Vector2() {}
    ~Vector2() {}

    void SetVector2(float x, float y)
    {
        this->x = x;
        this->y = y;
    }

    float GetX() const { return x; }
    float GetY() const { return y; }

private:
    float x, y;
};
```

상속성(inheritance)

- class 간에 부모/자식 관계를 가진다
- 부모의 특성/기능을 자식에서 그대로 물려 받는다

```
class Vector3 : public Vector2
{
public:
    void SetVector3(float x, float y, float z)
    {
        SetVector2(x, y);
        this->z = z;
    }

    float GetZ() const { return z; }

private:
    float z;
};
```

다형성(polymorphism)

- **같은 이름의 함수가 다른 동작을 하도록 한다**

- **오버라이딩(overriding)**
  - 부모 클래스의 함수를 재정의

- **오버로딩(overloading)**
  - 매개변수의 수와 타입이 다른, 같은 이름의 함수를 여러 개 정의

- **템플릿(template)**
  - 데이터 타입만 다른 같은 형태의 함수를 개별적으로 작성하지 않고 사용하도록 한다

다형성(polymorphism)

```
class Person
{
public:
    virtual void Speak()
    {
        std::cout << "Hello World!!" << std::두이;
    }
    …
}

class American : public Person
{
public:
    void Speak() override
    {
        std::cout << "Name₩t: " << name << std::endl;
        std::cout << "Gender₩t: " << gender << std::endl;
        std::cout << "Age₩t: " << age << std::endl;
    }
};
```

다형성(polymorphism)

```
class American : public Person
{
public:
    void Speak() override
    {
        std::cout << "Name₩t: " << name << std::endl;
        std::cout << "Gender₩t: " << gender << std::endl;
        std::cout << "Age₩t: " << age << std::endl;
    }

    void Speak(std::string msg)
    {
        std::cout << msg << std::endl;
    }
};
```

다형성(polymorphism)

```
class American : public Person
{
public:
    void Speak() override
    {
        std::cout << "Hello World!!" << std::endl;
        std::cout << "Name₩t: " << name << std::endl;
        std::cout << "Gender₩t: " << gender << std::endl;
        std::cout << "Age₩t: " << age << std::endl;
    }

    template<typename T>
    void Speak(T msg)
    {
        std::cout << msg << std::endl;
    }
};
```

다형성(polymorphism)

```
template<typename T>
class Vector2
{
public:
    void SetVector2(T x, T y)
    {
        this->x = x;
        this->y = y;
    }

    T GetX() const { return x; }
    T GetY() const { return y; }

private:
    T x, y;
};
```

```
template<typename T>
class Vector3 : public Vector2<T>
{
public:
    void SetVector3(T x, Ty, T z)
    {
        SetVector2(x, y);
        this->z = z;
    }

    T GetZ() const { return z; }

private:
    T z;
};
```

업 캐스팅(Upcasting)/다운 캐스팅(Downcasting)

- **업 캐스팅**
  - 자식 형태의 포인터를 부모 형태로 변환

```
int main()
{
    American a;
    Person* p = &a;
    p->Speak();
    return 0;
}
```

- **다운 캐스팅**
  - 부모 형태의 포인터를 자식 형태 포인터로 변환

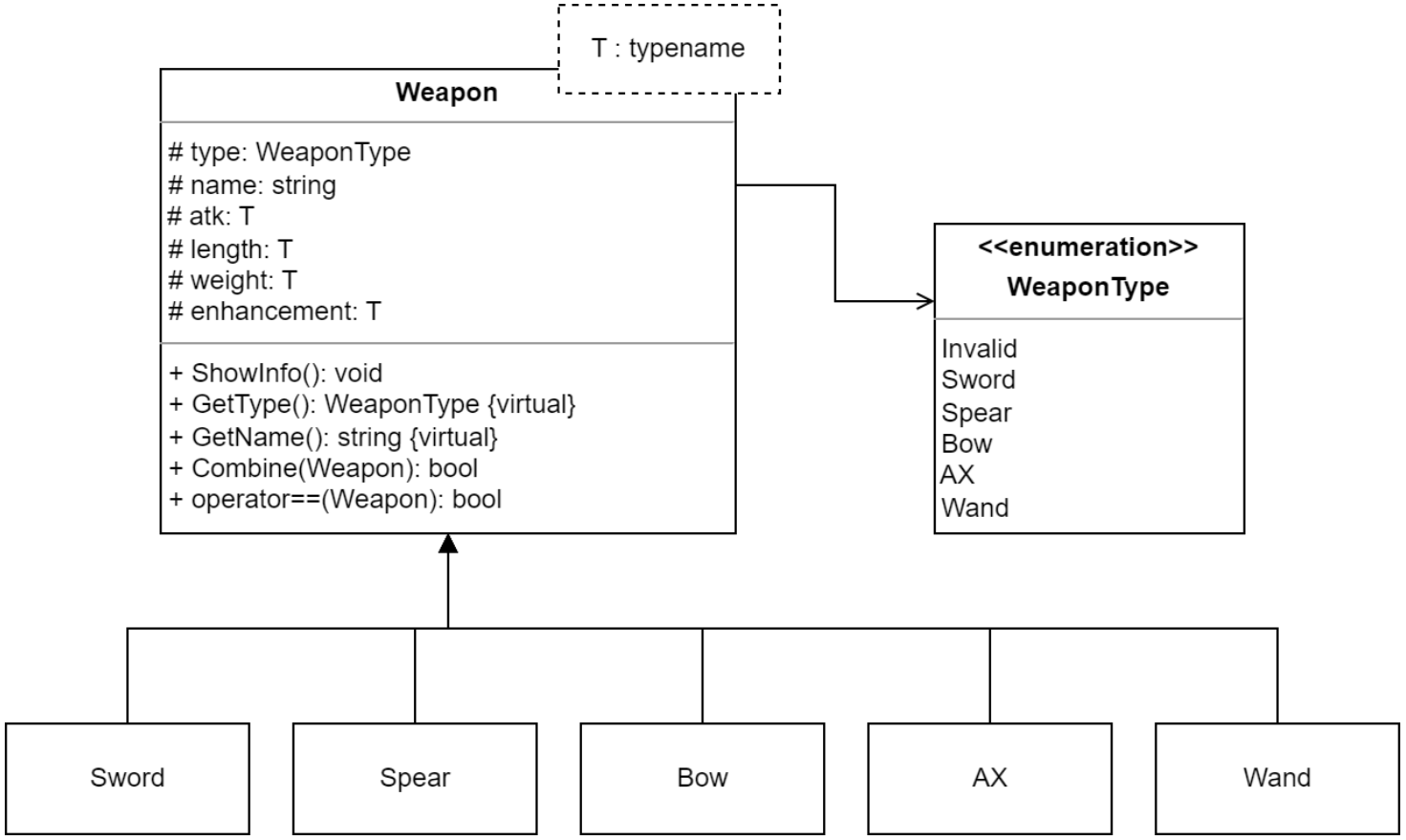
```
int main()
{
    American a;
    Person* p = &a;
    p->Speak();
    American* b = (American*)p;
    b->Speak("₩n0ooooh!!");
    return 0;
}
```

연산자 오버로딩(operator overloading)

- **연산자 오버로딩**
  - 기존에 제공되는 연산자에 대하여 의미를 다시 부여

```
class Vector2
{
public:
    ...
    Vector2 operator+(const Vector2& v)
    {
        Vector2 result;
        result.x = x + v.x;
        result.y = y + v.y;
        return result;
    }
    ...
};
```

실습


T : typename
Weapon
type: WeaponType
name: string
atk: T
length: T
weight: T
enhancement: T
+ ShowInfo(): void
+ GetType(): WeaponType {virtual}
+ GetName(): string {virtual}
+ Combine(Weapon): bool
+ operator==(Weapon): bool
<<enumeration>>
WeaponType
Invalid
Sword
Spear
Bow
AX
Wand
Sword
Spear
Bow
AX
Wand

- **제목**
  - 내용