C++ 기본 과정

프로그래밍과 프로그래밍 언어

- 다양한 응용 프로그램(애플리케이션 혹은 앱)
  - 스마트폰: 전화, 문자, 사진 촬영 등
  - 컴퓨터: 오피스 프로그램, 웹 브라우저 등
- 응용 프로그램은 프로그래밍 언어를 이용해 제작
- 프로그래밍: 컴퓨팅 기기(컴퓨터나 스마트폰 등)가 어떤 일을 하도록 명령을 내리는 작업
- 프로그래밍 언어: 프로그래밍을 하기 위한 언어
- 코딩(Coding): 프로그래밍 언어를 활용해 특정 목적의 프로그램을 만드는 것

프로그래밍 언어란?

- 컴퓨터에 명령을 내리려고 만든 언어
- 초기 프로그래밍
  - 컴퓨터의 중앙처리장치(CPU) 같은 하드웨어에 전기 신호를 직접 주기 위한 0과 1로 이뤄진 명령의 나열
  - 기계어 (Machine language): 2진 숫자(0과 1)로만 이뤄진 명령
  - 어셈블리어(Assembly language)

고급어와 저급어

- 고급어(High-level language)
  - 하드웨어에 대한 지식이 없는 사람이 좀 더 잘 이해할 수 있고 작성할 수 있는 프로그래밍 언어
  - 베이직(BASIC), 포트란(FORTRAN), C, C++, C#, 자바(Java), 파이썬(Python), 루비(Ruby),펄(Perl), 루아(Lua), R 등
  - 고급어로 작성된 코드도 바로 실행될 수 없음: 컴퓨터가 해석할 수 있는 기계어로 바꿔야 함

- 저급어(Low-level language): 어셈블리어

컴파일드 언어와 인터프리티드 언어

- 컴파일드 언어(Compiled Language)
  - 다수의 명령어로 이뤄진 소스코드를 한 번에 기계어로 번역해서 실행 파일을 만듦
- 인터프리티드 언어(Interpreted Language)
  - 소스코드를 한 줄씩 기계어로 번역해서 실행 결과를 보여줌
  - 스크립트(Script)언어라고도 함

C++ 개발 환경

통합 개발 환경(IDE)

- Integrated Development Environment의 약자로 코딩, 디버그, 컴파일, 배포 등 프로그램 개발에 관련된 모든 작업을 하나의 프로그램 안에서 처리하는 환경을 제공하는 소프트웨어이다.

- 종래의 소프트웨어 개발에서는 컴파일러, 텍스트 편집기, 디버거 등을 따로 사용했다. 이러한 프로그램들을 하나로 묶어 대화형 인터페이스를 제공한 것이 통합 개발 환경이다.

hello, world

hello.cpp

```
#include <iostream>

int main() {
  std::cout << "hello, world\n";
  return 0;
}
```

데이터 타입

- 정수와 실수
  - 소수점 이하 자릿수를 다루지 않는 숫자: 정수형 ex) 10, 0, -12
  - 소수점 이하 자릿수를 다루는 숫자: 실수형(단정도 실수형, 배정도 실수형) ex) 3.14
- 문자와 문자열
  - 하나의 문자 ex) 'A'
  - 문자가 여러 개가 모여 있는 형태: 문자열 ex) "hello"
- 논리
  - 참과 거짓을 표현하기 위한 데이터 타입 ex) true, false

Category	Type	Minimum Size	Note
boolean	bool	1 byte	
character	char	1 byte	May be signed or unsigned Always exactly 1 byte
	wchar_t	1 byte	
	char16_t	2 bytes	C++11 type
	char32_t	4 bytes	C++11 type
integer	short	2 bytes	
	int	2 bytes	
	long	4 bytes	
	long long	8 bytes	C99/C++11 type
floating point	float	4 bytes	
	double	8 bytes	
	long double	8 bytes	

문자열

- 문자열 타입은 기본(built-in) 타입이 아니라 사용자 정의(user-define) 타입임
- 문자열은 객체이므로 문자열 조작 등 여러 기능을 제공함

산술 연산자

연산	연산자	예
곱하기	*	5 * 7 → 35
나누기	/	**7 / 5 → 1, 7 / 5.0 → 1.4**
더하기	+	7 + 5 → 12
빼기	-	7 - 5 → 2
나머지	%	7 % 5 → 2

실수에 대하여도 동일한 연산자 사용 가능
곱하기, 나누기, 나머지 연산자가 더하기와 빼기 연산자보다 우선 순위가 높음
우선 순위를 변경하려면 소괄호(())를 사용하면 됨

비교 연산자

연산	연산자	설명
크다, 크거나 같다	>, >=	10>3 → true, 20>=60 → false
작다, 작거나 같다	<, <=	10<10 → false, 13<=10 → false
같다	==	20 == 20 → true
다르다	!=	20 != 30 → true

논리 연산자

연산	연산자	설명
논리곱(AND)	&&	true && false ➔ false
논리합(OR)	||	true && false ➔ true
논리부정(NOT)	!	!true → false, !false → true

변수

- 변수란?
  - 값을 저장하는 상자의 개념
- 변수 생성 방법
  - 타입 변수명;
- 변수 작명 규칙
  - 변수 이름은 알파벳 대소문자 및 밑줄 문자 그리고 숫자를 사용할 수 있습니다.
  - 변수 이름은 숫자로 시작할 수 없습니다.
  - 변수 이름에는 공백이 포함될 수 없습니다.
  - 예약어를 사용할 수 없습니다.
  - 대소문자를 구분합니다.

변수

- 변수를 생성한 후, 값을 설정하지 않으면 쓰레기 값으로 초기화 됨

- 값의 설정
  - 변수에 값을 설정하려면 = 연산자를 사용해야 함

- 값의 설정 방법
  - 타입 변수명; 변수명 = 값
  - 타입 변수명1, 변수명2;
  - 타입 변수명 = 값;
  - 타입 변수명1 = 값1, 변수명2 = 값2;

변수와 연산자

- 변수는 연산자와 함께 사용 가능
  - 덧셈, 뺄셈, 곱셈, 나눗셈, 나머지 연산자 사용 가능 ex) x + y

- 복합 대입 연산자
  - 변수의 누적을 위해 사용

연산	연산자	설명
덧셈 누적	+=	a=10; a+=3; 변수 a는 13이 저장됨
뺄셈 누적	-=	a=10; a-=3; 변수 a는 7이 저장됨
곱셈 누적	*=	a=10; a*=3; 변수 a는 30이 저장됨
나누기 누적	/=	a=10; a/=3; 변수 a는 3이 저장됨
나머지 누적	%=	a=10; a%=3; 변수 a는 1이 저장됨

  - **누적 시, 변수를 초기화하고 사용해야 함**

증감 연산자

- 변수의 값을 1 증가 또는 감소시키는 연산자
- 단항 연산자
- 변수의 앞 또는 뒤에 올 수 있음
  - ++변수명; -> 변수의 값을 1 증가시킨 후, 변수 사용
  - 변수명++; -> 변수 사용 후, 값을 1 증가시킴
  - --변수명; -> 변수의 값을 1 감소시킨 후, 변수 사용
  - 변수명--; -> 변수 사용 후, 값을 1 감소시킴

const

- 변수를 상수화 하는 키워드
- 사용 방법:
  - `const 타입 변수명;`
- 주의! 선언과 동시에 초기화해야 함

표준 출력

- 기본적으로 값을 모니터로 전송하는 개념
- 사용 방법: `std::cout << 값 << std::endl;`
- 다수의 값을 출력할 경우, <<을 추가적으로 사용하면 됨
  - `std::cout << 값1 << 값2 << 값3 << std::endl;`

표준 입력

- 기본적으로 키보드로부터 값을 입력 받는 개념
- 사용 방법: `std::cin << 변수명;`

네임스페이스

- 프로그램을 구성하는 요소들을 논리적으로 묶을 수 있도록 네임스페이스라는 기능을 제공
- 네임스페이스 정의 방법 - namespace NAMESPACE_NAME { }
- 네임스페이스 안의 식별자에 접근을 위해서는 범위 지정 연산자인 '::'을 사용해야 함

네임스페이스

- Using 지시자(Using Directive)
  - 네임스페이스 안의 모든 이름들이 현재의 유효 범위 내에서 접근할 수 있도록 함
  - usage: using namespace 네임스페이스명;

- Using 선언(Using Declaration)
  - 네임스페이스 안의 특정 이름만 현재의 유효 범위 내에서 접근할 수 있도록 함
  - usage: using 네임스페이스명::식별자;

제어문 1-1. if

- 조건식이 참일 때, 특정 코드를 수행하는 구문

```
if (조건)
{
	조건이 참일 때 수행할 문장;
}
```

제어문 1-2. if~else

```
if (조건)
{
    조건이 참일 때 수행할 문장;
}
else
{
    조건이 거짓일 때 수행할 문장;
}
```

제어문 1-3. if~else if ... else

```
if (조건1) {
  조건1이 참일 때 실행;
}
else if (조건2) {
  조건2가 참일 때 실행;
}
else if (조건3) {
  조건3이 참일 때 실행;
}
...
else {
  위의 조건이 모두 거짓일 때 실행;
}
```

참고! 모든 if문 중첩 가능

제어문 2. switch

- 값에 따라 특정 코드를 수행하는 구문

```
switch (조건문) {
case 값1:
  값1일 때 수행할 내용;
  break;
case 값2:
  값2일 때 수행할 내용;
  break;
...
  default :
    어떤 값에도 일치하지 않을 때 수행할 내용;
}
```

제어문 3. while

- 특정 코드를 반복하기 위해 사용되는 구문

```
while (조건식)
   반복할 내용;
```

- do ~ while: 항상 반복할 내용을 1번 수행한 다음 반복을 수행하는 구문

```
do {
   반복할 내용;
} while (조건식) ;
```

제어문 4. for

- 특정 코드를 반복하기 위해 사용되는 구문

```
for (초기식; 조건식; 증감식)
  반복할 내용;
```

반복의 흐름 제어

- 반복의 흐름을 제어하기 위해 다음의 키워드를 제공
  - `break`: 반복을 즉시 중단하고 반복분을 탈출하는 키워드
  - `continue`: 반복을 즉시 중단하고 그 다음 반복을 수행하는 키워드

함수

- 함수란 프로그램에서 특정 동작을 수행하는 코드를 의미한다. 어떤 함수가 특정 동작을 수행하기 위해서는 반드시 그 함수의 동작을 정의(구현)해야 한다. 그리고 실제 함수를 호출(사용)하기 위해서는 미리 그 함수를 선언해야 한다.

- 함수 선언 방법
  - 반환타입 함수명([타입, ...])

- 함수 정의 방법
  - 반환타입 함수명([타입, ...]) {}

함수 오버로딩

- 다른 인자 타입 또는 개수에 대해 같은 함수 이름을 사용하는 것을 함수 오버로딩(overloading)이라고 한다.

- 함수 오버로딩을 사용하려면 다음의 조건을 만족해야 함
  - 함수 이름은 동일해야 함
  - 함수의 선언에 사용된 매개 변수의 타입 또는 개수가 달라야 함

디폴트 파라미터

- 함수의 형식 인자에 기본 값을 설정할 수 있도록 디폴트 파라미터라는 기능을 제공

- 디폴트 파라미터가 설정된 함수에서 호출 시, 실제 인자를 전달하지 않을 경우 기본 값이 전달됨

포인터

- 포인터(pointer)란 다른 변수(another variable)의 주소(address)를 그 값으로 하는 변수를 의미

- 포인터 변수 선언 방법
  - 타입* 변수명;

- 메모리에 존재하는 값의 주소를 얻을 수 있도록 & 연산자를 제공
  - &변수명

포인터

- 간접 참조
  - 포인터에 저장된 변수를 사용하여 그 위치의 메모리를 참조할 수 있는 메커니즘
- 간접 참조 방법
  - *포인터명;
- 포인터의 활용 예
  - 일반적으로 호출된 함수(callee)에서 호출한 함수(caller)에서 선언된 변수나 배열 등을 참조하기 위해 사용됨

포인터

- 주의! 초기화되지 않은 포인터에 대하여 간접 참조를 수행하면 잘못된 메모리 참조로 오류 발생

- 포인터는 반드시 초기화 후, 사용해야 함

레퍼런스

- 포인터 처리를 보다 쉽게 할 수 있도록 레퍼런스라는 기능을 제공
- 레퍼런스는 기존 객체에 대한 새로운 이름, 즉 별칭을 부여하는 개념
- 레퍼런스 선언 방법 - 타입& 변수명;
- 레퍼런스는 선언만 할 수 없음

배열

- 배열(array)은 번호(index)와 번호에 대응하는 데이터들로 이루어진 자료 구조를 나타낸다.

- 배열 선언 방법
  - 타입 배열명[크기];
  - 타입 배열명[] = {값1, 값2, 값3, ...};

- 인덱싱
  - 배열의 특정 원소를 참조하는 문법
  - 배열명[인덱스];

배열

- 배열을 함수의 인자로 전달하는 방법
  - 반환타입 함수명(타입 배열명[]);

- 배열은 객체가 아니므로 크기 정보가 없음, 따라서 함수 호출 시, 크기 정보도 같이 전달해야 함

구조체

- 여러 가지 타입의 값을 하나의 집합으로 묶을 수 있는 개념
- 단일 데이터 유형만을 저장할 수 있는 배열과는 달리, 구조체는 정수, 실수, 문자열, 또 다른 구조체 등 다양한 데이터 유형을 하나로 묶을 수 있음
- **구조체 안에 함수를 포함할 수 있음**
- 구조체 정의 방법

```
struct 구조체명
{
  구조체 멤버 선언;
};

구조체명 변수;
```

- 구조체는 사용자 정의 자료형이므로 변수를 선언할 수 있음

구조체 포인터

- 구조체 타입의 포인터 선언 가능
  - usage: 구조체명* 변수명;

- 구조체 포인터의 멤버 참조
  - (*구조체형 포인터 변수).멤버
  - 구조체형포인터변수->멤버

자기 참조 구조체 포인터

- 구조체의 멤버로 해당 구조체 자신의 포인터 타입을 사용할 수 있음
- 주로 연결 리스트, 트리, 그래프와 같은 고급 자료 구조를 구현할 때 사용됨

열거 타입

- 정수형 상수들의 집합을 정의하는 사용되는 타입

- 선언 방법
  - `enum identifier { value1, value2, value3, ... };`
  - 상수는 기본적으로 0부터 시작하여 1씩 증가하는 정수 값을 가짐

- 열거 타입으로 생성된 변수는 그 타입의 열거자만 대입 가능

동적 메모리 할당

- 배열에 대한 동적 메모리 할당을 하기 위해 new[]라는 연산자를 제공함
- 배열 동적 할당 방법 - 타입* 변수명 = new 타입[길이];
- 동적으로 할당된 배열 메모리를 해제하기 위해 delete[]라는 연산자를 제공함

배열 동적 할당

- 실행 시간(runtime)에 필요한 만큼 메모리를 생성하는 것을 동적 메모리 할당(dynamic memory allocation)
- 동적 메모리 할당을 하기 위해 new라는 연산자를 제공함
- 동적 메모리 할당 방법 - 타입* 변수명 = new 타입;
- 동적으로 할당된 메모리를 해제하기 위해 delete라는 연산자를 제공함

클래스

- 클래스는 구조체와 동일한 개념
- 클래스 선언 방법

```
class 클래스명
{
   클래스 멤버; //멤버 변수와 멤버 함수
};

클래스명 객체명;
```

- 다음의 접근 지정자를 제공함
  - public: 접근/호출에 제한이 없음
  - private: 오직 구조체 또는 클래스 안에서만 접근/호출 가능
  - protected: 구조체 또는 클래스와 파생 구조체 또는 클래스 안에서만 접근/호출 가능

생성자

- 객체가 생성될 때 자동으로 호출되는 멤버 함수
- 생성자의 이름은 클래스의 이름과 동일해야 함
- 반환 타입을 명기하지 않으며 return 키워드를 사용할 수 없음
- 생성자도 일종의 함수이므로 오버로딩이 가능함
- 객체가 생성될 때, 자동으로 호출되며 객체를 초기화할 때 사용함

기본 생성자

- 사용자가 생성자를 추가하지 않으면 컴파일러가 자동으로 추가하는데 이를 기본 생성자라고 함
- 기본 생성자는 형식 인자가 하나도 없으며 생성자 내부에도 하는 일이 없음
- 사용자가 생성자를 추가하면 컴파일러는 기본 생성자를 추가하지 않음

멤버 초기화

- 생성자 내에서 멤버 변수의 할당은 초기화가 아닌 대입
- 멤버 변수를 초기화하려면 멤버 이니셜라이저를 사용해야 함
- 사용 방법:
  - `생성자() : 멤버변수1(값1), 멤버변수2(값2), ... {}`
- 기본 생성자가 없는 멤버의 경우, 반드시 이를 사용하여 초기화해야 함

소멸자

- 소멸자의 이름은 클래스의 이름과 같고 생성자와의 구분을 위해 이름 앞에 ~ 기호를 사용함
- 소멸자는 반환 타입이 없으며 return 키워드를 사용할 수 없음
- 소멸자는 클래스에서 오직 하나만 존재하며 오버로딩될 수 없음
- 객체가 메모리에서 삭제될 때, 자동으로 호출함
- 소멸자를 추가하지 않으면 컴파일러가 자동으로 추가함

복사 생성자

- 자신의 타입을 형식 인자로 하는 생성자를 복사 생성자라고 함
- 객체의 복사를 지원하기 위해 제공됨
- 사용자가 복사 생성자를 정의하지 않을 경우, 컴파일러가 자동으로 정의하는데 이를 기본 복사 생성자라고 함

this

- 객체 자신을 가리키는 포인터
- 객체가 생성되면 생성된 객체는 this 포인터를 가진다.
- 멤버 함수 내에서 매개변수와 멤버 변수의 이름이 동일할 경우 객체의 멤버 변수임을 명시하기 위해 사용

정적 멤버 함수

- static 키워드를 사용하여 선언된 멤버 함수를 의미
- 정적 멤버 함수는 객체의 생성 없이 호출되는 함수로 일반적인 전역 함수와 동일한 개념임
- 객체의 정보가 없으므로 일반 멤버는 접근할 수 없음

정적 멤버 변수

- static 키워드를 사용하여 선언된 멤버 변수를 의미
- 정적 멤버 변수는 객체의 생성 없이 접근할 수 있는 변수로 일반적이 전역 변수와 동일한 개념임
- 클래스 내부에 정적 멤버 변수는 선언이므로 반드시 외부에 정의를 해야 함

friend

- 클래스 내부에서 `friend`로 선언한 일반 함수 또는 클래스는 아무 제약 없이 클래스 멤버에 접근 가능

연산자 오버로딩

- 연산자 오버로딩은 연산자를 함수의 개념으로 취급하여 사용자가 연산자를 재정의하는 개념
- 연산자 오버로딩 정의 방법 - 반환형 operator연산자기호([매개변수,...]);
- 연산자 오버로딩은 멤버 함수 뿐만 아니라 비 멤버 함수에 의해서도 구현 가능

객체 지향 프로그래밍

- 독립된 단위인 객체를 사용하여 구현하는 프로그래밍 방법론
- 프로그램을 유연하고 변경이 용이하게 만들기 때문에 대규모 소프트웨어 개발에 많이 사용됨
- 객체는 자신만의 고유한 상태와 기능을 가짐
- 캡슐화(encapsulation)를 지원하여 프로그램이 간결해지고 유지보수가 용이함
- 정보 은닉(information hiding)을 지원

상속

- 기존 클래스의 특성을 재사용하여 새로운 클래스를 작성하는 것을 의미
- 상속을 하는 주체를 Base 또는 Super 클래스라고 하며 상속을 받는 주체를 Derived 또는 Sub 클래스라고 함
- 생성자, 소멸자, 대입연산자, 정적 멤버, friend 함수는 상속되지 않음
- 파생(derived) 클래스 정의 방법
  - class 클래스명 : 접근지정자 기본클래스명

함수 오버라이딩

- 자식 클래스가 부모 클래스의 함수를 재정의하는 것을 의미
- 오버라디이딩의 조건은 부모 함수의 시그니처와 완벽하게 동일해야 함

업캐스팅

- 상속 관계에서 자식 클래스의 주소를 부모 타입을 포인터에 저장하는 것을 말함
- 서로 다른 타입을 하나의 타입으로 처리하기 위해 사용
- 부모 포인터를 사용하여 자식의 고유 멤버에는 접근할 수 없음

가상 함수

- 동적 바인딩은 바인딩이 실행시간 중에 일어나거나 프로그램 실행 과정에서 변경될 수 있음을 의미

- C++ 언어에서 동적 바인딩을 수행하려면 virtual 키워드를 사용해야 하며 이를 가상 함수라고 함

- 사용 방법:
  - virtual 반환타입 함수명([매개변수, ...]) {}

순수 가상 함수

- 순수 가상 함수란 함수의 정의가 없는 함수를 의미
- 순수 가상 함수는 클래스 또는 구조체 안에서만 사용 가능
- 선언 방법 - `virtual 리턴타입 함수명([매개변수,...]) = 0;`

추상 클래스

- 추상 클래스는 순수 가상 함수를 하나 이상 포함하고 있는 클래스를 의미

- 추상 클래스는 객체를 생성할 수 없으며 반드시 자식이 추상 클래스의 순수 가상 함수를 구현해야만 객체 생성 가능

예외 처리

- 프로그램 실행에서 예외가 발생한 경우에 대한 처리 과정
- 예외 처리가 없는 경우 정상적인 프로그램 실행이 안될 수도 있음
- try~catch 키워드를 사용하여 예외를 반드시 처리해야 함

```
try
{
    예외가 발생될 수 있는 코드
}
catch (throw에서 전달받은 인수)
{
        예외가 발생했을 때 수행할 내용
}
```

템플릿

- C++에서 템플릿은 클래스나 함수 코드를 자동으로 생성하는 기능
- 함수 템플릿은 타입에 따른 함수를 자동으로 생성하는 기능을 의미
- 클래스 템플릿은 타입에 따른 클래스를 자동으로 생성하는 기능
  - 클래스 템플릿은 타입이 아니라 틀(template)이므로 객체를 생성할 수 없음
  - 객체를 생성할 때, 반드시 타입을 명시적으로 사용해야 함

파일 출력

- #include <fstream>
- ofstream 클래스를 사용하여 객체 생성
- open 멤버 함수를 사용하여 파일 열기
- close 멤버 함수를 사용하여 파일 닫기
- 출력 연산자를 사용하여 파일 쓰기 수행

파일 입력

- #include <fstream>
- ofstream 클래스를 사용하여 객체 생성
- open 멤버 함수를 사용하여 파일 열기
- close 멤버 함수를 사용하여 파일 닫기
- 입력 연산자를 사용하여 파일 입력 수행

파일 모드

파일모드	설명	예
ios::in	파일에서 읽어오기	파일객체.open(파일이름, ios::in);
ios::out	파일에 출력하기	파일객체.open(파일이름, ios::out);
ios::app	파일에 추가하여 출력하기	파일객체.open(파일이름, ios::app);
ios::trunc	파일이 이미 존재하는 경우 삭제하고 새로운 파일로 생성하여 출력	파일객체.open(파일이름, ios::trunc);
ios::binary	이진 파일로 처리하기	파일객체.open(파일이름, ios::in | ios::binary); 파일객체.open(파일이름, ios::out | ios::binary);

STL

- STL은 "Standard Template Library"의 두문자어
- C++ 표준 라이브러리의 일부로서 알고리즘, 컨테이너, 함수자, 반복자 등의 템플릿 기반의 컴포넌트를 제공
- 컨테이너(Container): 컨테이너는 다양한 데이터 구조를 나타냄
  - 주로 사용되는 STL 컨테이너에는 vector, list, map 등이 있음