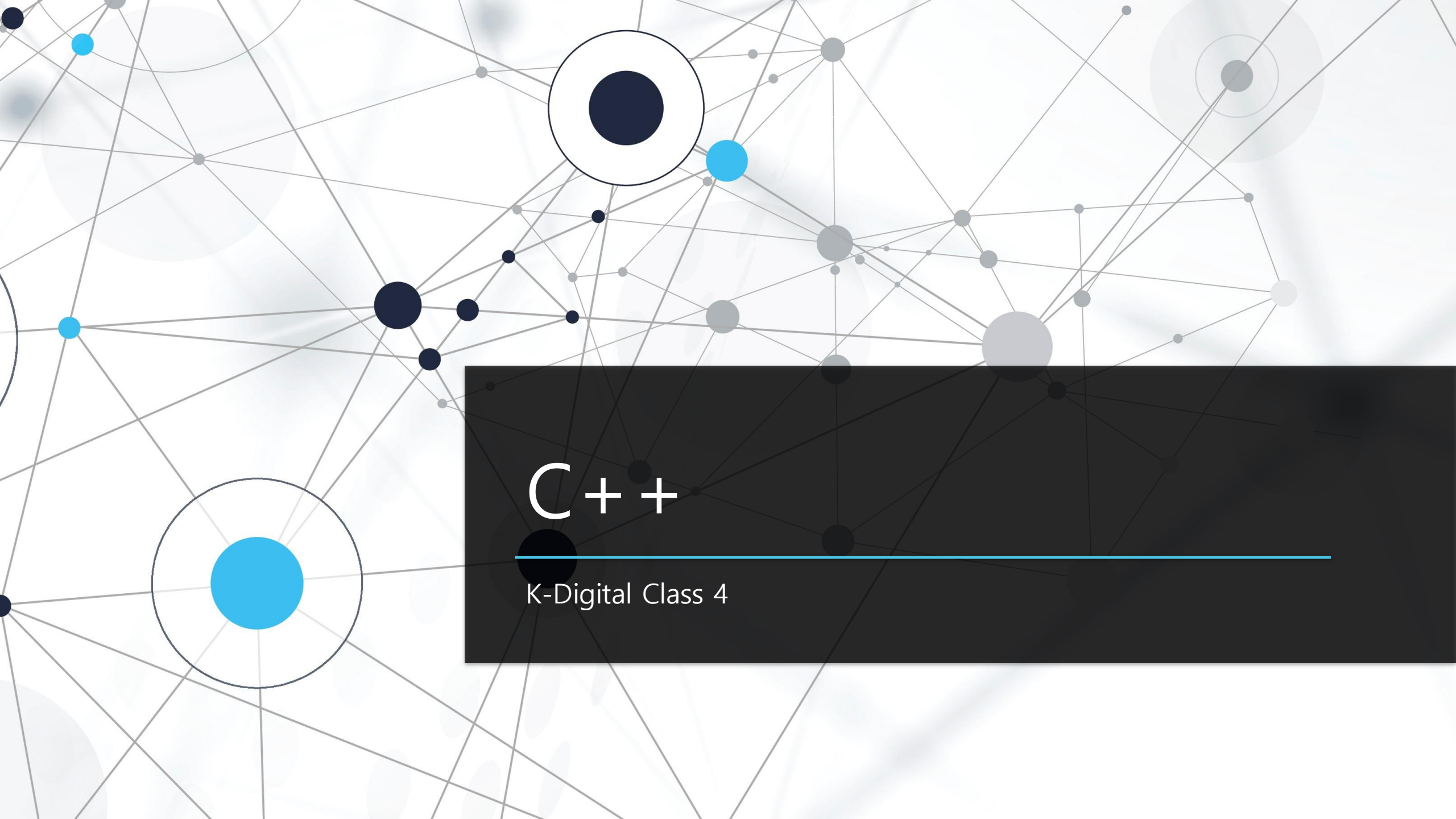


C++

K-Digital Class 4

C++ 개요

- C++는 Bjarne Stroustrup이 1979년 Bell Labs에서 개발한 High-Level Programming Language이다. C++는 Windows, Mac OS 및 다양한 UNIX/LINUX 버전과 같은 다양한 Platform에서 실행된다.
- Stroustrup이 작업할 기회가 있었던 Language 중 하나는 Simula라는 Language 였다. 이름에서 알 수 있듯이 주로 Simulation을 위해 설계된 Language 이다. 여기서 객체 지향(Object Oriented)의 개념을 C Language 에 추가 하여 1983 C++를 발표한다.
- ++ 연산자(Operator)를 추가 하여 C++라고 명명 하게 된다.
  - ++은 +1의 의미를 가진다.

C++의 특징

- C++은 절차 지향적이며 구조적 프로그래밍 언어이다.
- C++은 객체 지향 프로그래밍 언어이다.
- C++은 Compile(번역) 언어이다.
  - Compiler라는 매체를 통한 소스 파일을 기계어로 Compile(번역)하는 언어.

- 이처럼 C++은 세 가지 프로그래밍 방식을 모두 지원하는 언어이며, 따라서 다양한 방식으로 프로그램을 작성할 수 있다.
- 또한, 다양한 시스템에서의 프로그래밍을 지원하는 유용하고도 강력한 클래스 라이브러리들이 아주 많이 제공된다.
- 하지만 C++은 이렇게 언어로서 다양한 방식을 지원하며 강력하게 거듭났지만, 프로그래머의 측면에서 보면 이러한 다양한 기능을 모두 배워야 하는 부담으로 작용하기도 한다.

C++ 표준

- C++에 대한 표준은 ANSI(American National Standards Institute)와 ISO(International Organization for Standardization)가 표준화 작업을 진행한다.
- 1998년에 첫 C++ 국제 표준인 ISO/IEC 14882:1998이 제정되며, 이 표준을 C++98이라고 부르게 된다.
- 이후 C++98에서 단순한 기술적 개정만을 진행한 ISO/IEC 14882:2003(C++03)이 2003년에 공개되었다.
- 2011년에는 많은 언어적 특성이 추가된 ISO/IEC 14882:2011이 발표되며, 이 표준을 C++11이라고 부르게 된다.
- 이후 C++11의 사소한 버그 수정 및 약간의 기술적 개선을 진행한 ISO/IEC 14882:2014(C++14)가 2014년에 공개되었다.
- 2017년 파일 시스템, STL 병렬 알고리즘 처리 등이 추가되어 12월에 C++1z로 알려졌던 C++17 표준이 ISO/IEC 14882:2017이라는 정식 명칭으로 최종 승인되었다.

C++ PROGRAMMING

- 소스 파일(source file)의 작성
- 선행처리기(preprocessor)에 의한 선행처리
- 컴파일러(compiler)에 의한 컴파일
- 링커(linker)에 의한 링크
- 실행 파일(executable file)의 생성

C++ PROGRAMMING

소스 파일(source file)의 작성

- Programming에서 가장 먼저 해야 할 작업은 바로 프로그램을 작성하는 것이다.
- 다양한 에디터를 사용하여 C++ 문법에 맞게 논리적으로 작성된 프로그램을 원시 파일 또는 소스 파일이라고 한다.
- C++을 통해 작성된 소스 파일의 확장자는 대부분 .cpp 가 된다.

C++ PROGRAMMING

전처리기(preprocessor)에 의한 선행처리

- 전처리기(Preprocessor)는 실제 컴파일이 시작되기 전에 컴파일러에게 정보를 사전 처리하도록 지시하는 지시사항이다.
- Source file 중에서도 전 처리 문자(#)로 시작하는 전 처리 지시문의 처리 작업을 의미한다.
- 이러한 선행처리 작업은 전처리기(preprocessor)에 의해 처리된다.
- 전처리기는 코드를 생성하는 것이 아닌, 컴파일하기 전 컴파일러가 작업하기 좋도록 Source code를 재구성해주는 역할만을 한다.

C++ PROGRAMMING

컴파일러(compiler)에 의한 컴파일

- Computer는 0과 1로 이루어진 이진수로 작성된 기계어만을 이해할 수 있다.
- Source file은 개발자에 의해 C++ 언어로 작성되기 때문에 컴퓨터가 그것을 바로 이해할 수는 없다.
- 따라서 Source file 을 Computer가 알아볼 수 있는 기계어로 변환시켜야 하는데, 그 작업을 컴파일(compile)이라고 한다.
- Compile은 C/C++ compiler에 의해 수행되며, compile이 끝나 기계어로 변환된 파일을 Object file 이라고 한다.
- 이러한 Object file의 확장자는 .o 나 .obj 가 된다.

C++ PROGRAMMING

링커(linker)에 의한 링크

- Compiler에 의해 생성된 Object file은 운영체제(Operating System(OS))와의 Interface를 담당하는 시동 코드(Start-up code)를 가지고 있지 않다.
- 또한, 대부분의 C++ 프로그램에서 사용하는 표준 라이브러리 파일도 가지고 있지 않다.
- 하나 이상의 Object file과 Library file, Start-up code 등을 합쳐 하나의 파일로 만드는 작업을 링크(Link)라고 한다.
- Link는 링커(linker)에 의해 수행되며, Link가 끝나면 하나의 새로운 실행 파일이나 Library file이 생성된다.
- 이처럼 여러 개의 Source code file을 작성하여 최종적으로 링크를 통해 하나의 실행 파일로 만드는 것을 분할 컴파일이라고 한다.

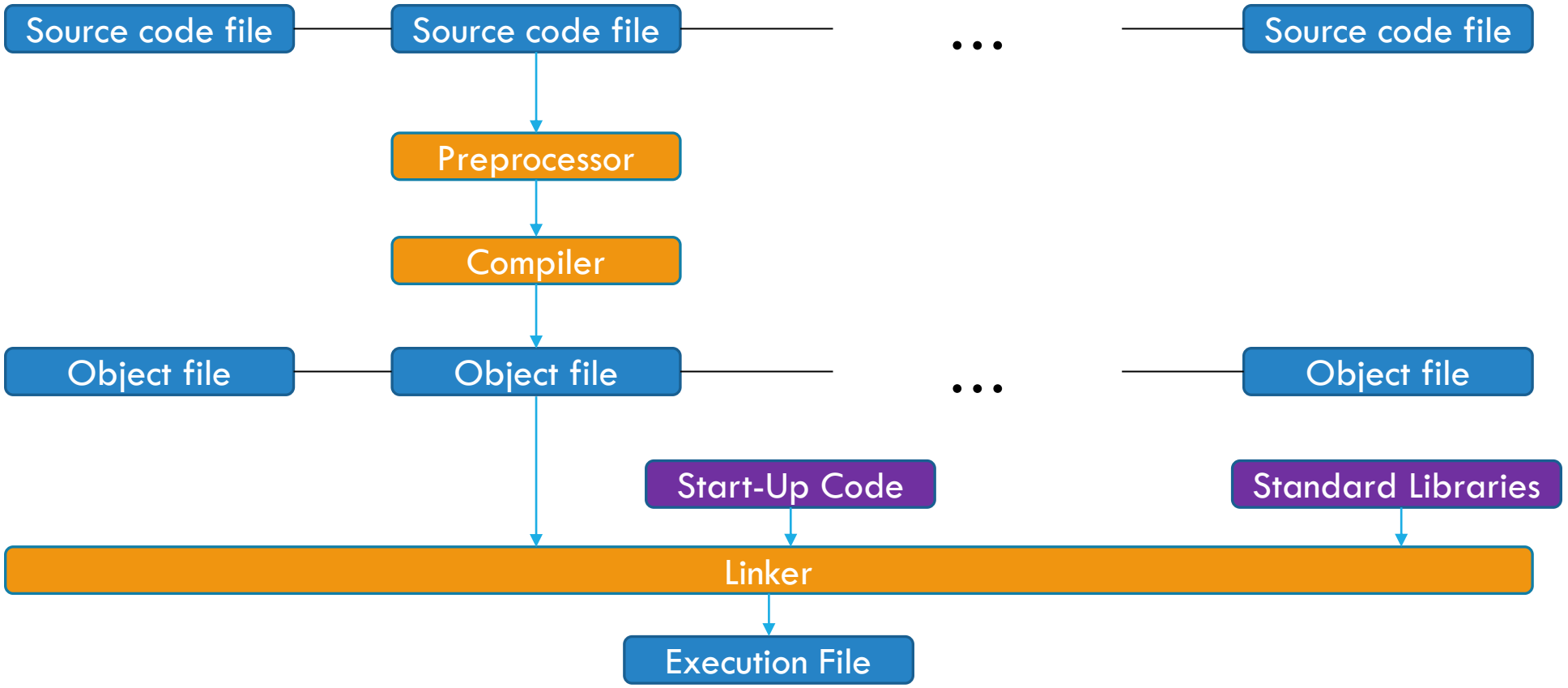
C++ 프로그램은 main함수를 가지고 있을 것이다. 그렇다면 이 main은 누가 호출하는 것이냐? 이것이 바로 스타트업 코드이다. 스타트업 코드의 기능은 간단히 얘기하면, CPU 레지스터 초기화와 정적 변수 초기화를 한 후 main을 호출하는 것이다.

C++ PROGRAMMING

실행 파일(executable file)의 생성

- Source code file은 전처리기(Preprocessor), Compiler 그리고 Linker에 의해 실행 파일로 변환된다.
- 최근 사용되는 개발 툴은 대부분 전처리기(Preprocessor), Compiler, Linker를 모두 내장하고 있으므로 Source code file에서 한 번에 실행 파일을 생성할 수 있다.
- Windows의 경우 이렇게 생성된 실행 파일의 확장자는 .exe 가 된다.
- Linux/Mac/Unix의 경우 실행 권한을 가진 확장자 없이 실행 파일이 생성된다.

C++ PROGRAMMING



C++ PROGRAMMING

Object-Oriented Programming

- C++는 객체 지향 개발의 네 가지를 포함하여 객체 지향 프로그래밍 (Object-Oriented Programming)을 완벽하게 지원한다.
  - Encapsulation(캡슐화)
  - Information hiding(정보의 은닉화)
  - Inheritance(상속)
  - Polymorphism(다형성)

C++ PROGRAMMING

Object-Oriented Programming

- Encapsulation : 객체의 속성(data fields)과 행위(methods)를 하나로 묶고, 실제 구현 내용 일부를 외부에 감추어 은닉한다.
- Information hiding : 은닉이라는 단어 때문에 캡슐화와 은닉화를 혼동하는 분이 많습니다. 은닉화는 캡슐화를 통해 얻어지는 "실제 구현 내용 일부를 외부에 감추는" 효과이다.
- Inheritance : Parent의 Method 혹은 Field를 Child class에게 상속해 주는 것이다. 즉 '기존 class에 method를 추가하거나 재정의하여 새로운 class를 정의한다.'
- Polymorphism : 하나의 객체에 여러 가지 타입을 대입할 수 있다는 것을 의미한다. 반대로, 단형성은 하나의 객체에 하나의 타입만 대응할 수 있다.

C++ PROGRAMMING

Basic syntax of a C++

HelloWorld.cpp > ...

```cpp
// HelloWorld.cpp

#include <iostream>

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

    return 0;
}

```

C++ PROGRAMMING

Structure of a C++ program

- Line1
  - 주석이며 프로그램의 동작에는 영향을 미치지 않는다. 코드나 프로그램에 관한 짧은 설명이나 관찰을 서술 하는데 사용한다.
- Line3
  - #으로 시작하는 행은 전처리기에서 읽고 해석하는 지시문이다. 컴파일이 시작되기 전에 해석되는 특수 라인이다. 이 경우 헤더 iostream으로 알려진 표준 C++ 코드를 포함하도록 전처리기에 지시한다.
- Line5
  - C/C++는 main 함수에서 실행이 시작 되기 때문에 반드시 있어야 하는 함수이다.
- Line6, 10
  - 6행의 여는 중괄호({)는 main 함수 정의의 시작을 나타내고 10행의 닫는 중괄호(})는 끝을 나타낸다.
- Line7
  - 이 줄은 C++ 문장(Statement)이다. 실제 동작을 지정하는 프로그램의 핵심이다. 모든 C/C++ 문장은 세미콜론(;)으로 끝난다.

C++ PROGRAMMING

Comments(주석)

- 프로그램이 수행하는 작업과 작동 방식을 소스 코드 내에서 직접 문서화하는 중요한 도구를 제공한다.
- C++는 코드 주석 처리의 두 가지 방법을 지원한다.
  - //
  - /* comment */
- Source code 설명을 HTML 문서로 저장 할 수 있는 doxygen type 주석을 권장 한다.

HelloWorld.cpp > ...

```cpp
// HelloWorld.cpp
/* This is Hello World C++ File*/
/* C++ comments can also
   * span multiple lines
*/
#include <iostream>

using namespace std;


/**
 * @brief Must need main function in C++ source code
 *
 * @return int
 */
int main()
{
    cout << "Hello World!\n";

    return 0;
}

```

C++ PROGRAMMING

Using namespace std

- cout은 표준 라이브러리의 일부이며 표준 C++ 라이브러리의 모든 요소는 namespace라고 하는 std namespace 내에서 선언되어 있다.
- std namespace에 있는 요소를 참조하기 위해 프로그램은 Library의 모든 요소 사용에 대해 자격을 부여하거나 (cout에 std::를 붙여서 수행한 것처럼) 해당 구성 요소의 가시성을 도입해야 한다. 이러한 구성 요소의 가시성을 도입하는 가장 일반적인 방법은 선언을 사용하는 것이다.

HelloWorld.cpp > ...

```cpp
// HelloWorld.cpp
/* This is Hello World C++ File*/
/* C++ comments can also
   * span multiple lines
*/
#include <iostream>

using namespace std;


/**
 * @brief Must need main function in C++ source code
 *
 * @return int
 */
int main()
{
    cout << "Hello World!\n";

    return 0;
}

```

C++ 변수(VARIABLE)

- 변수(variable)란 데이터(data)를 저장하기 위해 프로그램에 의해 이름을 할당 받은 메모리 공간을 의미한다.
- 즉, 변수란 데이터(data)를 저장할 수 있는 메모리 공간을 의미하며, 이렇게 저장된 값은 변경될 수 있다.
- C++에서 숫자 표현에 관련된 변수는 정수형 변수와 실수형 변수로 구분할 수 있다.
- 또 다시 정수형 변수는 char형, int형, long형, long long형 변수로, 실수형 변수는 float형, double형 변수로 구분된다.
- 관련된 데이터를 한 번에 묶어서 처리하는 사용자 정의 구조체(structure) 변수도 있다.

C++ 변수(VARIABLE)

변수(Variable)의 이름 생성 규칙

- C++에서는 변수의 이름을 비교적 자유롭게 지을 수 있다.
- 변수의 이름은 해당 변수에 저장될 데이터의 의미를 잘 나타내도록 짓는 것이 좋다.
- C++에서 변수의 이름을 생성할 때 반드시 지켜야 하는 규칙은 다음과 같다.
  - 변수의 이름은 영문자(대소문자), 숫자, Underscore(_)로만 구성된다.
  - 변수의 이름은 숫자로 시작될 수 없다.
  - 변수의 이름 사이에는 공백을 포함할 수 없다.
  - 변수의 이름으로 C++에서 미리 정의된 키워드(keyword)는 사용할 수 없다.
  - 변수 이름의 길이에는 제한이 없다.

C++에서는 변수의 이름에 대소문자를 구분하므로 이 점에 주의해야 한다.

C++ 변수(VARIABLE)

비트(bit)와 바이트(byte)

- Computer는 모든 data를 2진수(Binary)로 표현하고 처리한다.
- 비트(bit)란 Computer가 data를 처리하기 위해 사용하는 data의 최소 단위이다.
- 이러한 bit에는 2진수의 값(0과 1)을 단 하나만 저장할 수 있다.
- 바이트(byte)란 비트가 8개 모여서 구성되며, 한 문자(Character)를 표현할 수 있는 최소 단위다.

C++ 변수(VARIABLE)

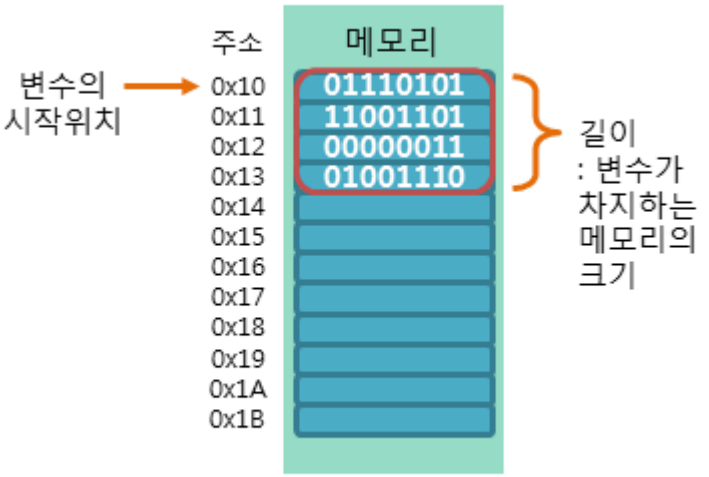
Variable(변수)과 Memory Address

- 변수는 기본적으로 메모리의 주소(address)를 기억하는 역할을 한다.
- 메모리 주소란 물리적인 메모리 공간을 서로 구분하기 위해 사용되는 일종의 식별자(identifier)다.
- 즉, 메모리 주소란 메모리 공간에서의 정확한 위치를 식별하기 위한 고유 주소를 의미한다.
- 변수를 참조할 때는 메모리의 주소를 참조하는 것이 아닌, 해당 주소에 저장된 데이터를 참조하게 된다.
- 따라서 변수는 데이터가 저장된 메모리의 주소 뿐만 아니라, 저장된 데이터의 길이와 형태에 관한 정보도 같이 기억해야 한다.

C++ 변수(VARIABLE)

Variable(변수)과 Memory Address

- 다음 그림은 메모리상에 변수가 어떤 식으로 저장되는지를 보여줍니다.
- 그림에서 하나의 메모리 공간에는 8개의 비트로 이루어진 1바이트의 데이터가 저장되어 있다.
- 해당 변수의 길이는 총 4개의 메모리 공간을 포함하므로, 해당 변수는 총 4바이트의 데이터를 저장하고 있는 것이다.



형태 : 메모리에 저장된 데이터를 해석하는 방법

C++ 변수(VARIABLE)

Declaring (Creating) Variables

- C++에는 다양한 유형의 변수가 있다. 예를 들면 다음과 같다.

- int - 123 또는 -123과 같이 소수 없이 정수(정수)를 저장한다.
- double - 19.99 또는 -19.99와 같은 소수를 사용하여 부동 소수점 숫자를 저장한다.
- char - 'a' 또는 'B'와 같은 단일 문자를 저장합니다. Char 값은 작은따옴표로 묶는다.
- string - "Hello World"와 같은 텍스트를 저장한다. 문자열 값은 큰따옴표(“”)로 묶는다.
- bool - true 또는 false의 두 가지 상태로 값을 저장한다.

C++ 변수(VARIABLE)

Declaring (Creating) Variables

- 변수(Variables)를 선언하려면 유형(Type)을 지정하거나 값을 할당한다.
- 여기서 type은 C++ type(예: int) 중 하나이고 variableName은 변수의 이름(예: x 또는 myName)이다. 등호(=)는 변수에 값을 할당하는 데 사용되는 대입 연산자이다.

- Syntax
- type variableName;
- type variableName = value;
- Ex)
  - int myNum = 15;

C++ 변수(VARIABLE)

Declare Multiple Variables

- 동일한 type의 변수를 둘 이상 선언하려면 쉼표(,)로 구분하여 선언한다.

- int x = 5, y = 6, z = 50;
- cout << x + y + z;

C++ 변수(VARIABLE)

Variable Scope in C++

- Scope는 프로그램의 한 영역이며, 대체로 변수를 선언할 수 있는 세 곳이 있다.

- 지역 변수(Local Variable)라고 하는 함수나 블록 내부에 정의.
- 형식 매개변수라고 하는 함수 매개 변수의 정의.
- 전역 변수(Global Variable)라고 하는 모든 함수 외부에서 정의.

C++ 변수(VARIABLE)

Local Variables

- 함수(function)나 블록(block) 내에서 선언된 변수는 지역 변수(local variable)이다. 해당 함수 또는 코드 블록(code block:{}) 내부에 있는 Statements(명령문)에서만 사용할 수 있다.
- 지역 변수는 자신의 외부에 있는 함수에서 사용할 수 없다.

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

int main () {
   // Local variable declaration:
   int a, b;
   int c;

   // actual initialization
   a = 10;
   b = 20;
   c = a + b;

   cout << c;

   return 0;
}
```

C++ 변수(VARIABLE)

Global Variables

- 전역 변수(global variables)는 일반적으로 프로그램의 맨 위에 있는 모든 함수 외부에서 정의된다. 전역 변수는 프로그램 수명(life-time) 내내 그 값을 유지(Keep)한다.
- 전역 변수는 모든 함수에서 access(접근)할 수 있다. 즉, 전역 변수는 선언 후 전체 프로그램에서 사용할 수 있다.

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

// Global variable declaration:
int g;

int main () {
   // Local variable declaration:
   int a, b;

   // actual initialization
   a = 10;
   b = 20;
   g = a + b;

   cout << g;

   return 0;
}
```

C++ 변수(VARIABLE)

Global Variables

- 프로그램은 지역(local) 및 전역(global) 변수에 대해 동일한 이름을 가질 수 있지만 함수 내부의 지역 변수 값이 우선한다.

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

// Global variable declaration:
int g = 20;

int main () {
   // Local variable declaration:
   int g = 10;

   cout << g;

   return 0;
}
```

C++ 변수(VARIABLE)

C++ Constants(상수)

- 다른 사람(또는 자신)이 기존 변수 값을 재정의하지 않도록 하려면 const 키워드를 사용한다.
- 이는 변수를 변경할 수 없고 읽기 전용을 의미하는 “Constant " 로 선언한다.

Example

```
const int myNum = 15;  // myNum will always be 15
myNum = 10;  // error: assignment of read-only variable 'myNum'
```

C++ 변수(VARIABLE)

C++ Constants(상수)

- 변경될 가능성이 없는 값이 있는 경우 항상 변수를 상수로 선언 하는 것이 합리적이다.

Example

```
const int minutesPerHour = 60;
const float PI = 3.14;
```

C++ DATA TYPE

- 변수(Variables)에서 설명한 대로 C++의 변수는 지정된 데이터 유형이어야 한다.

Example

```
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99;     // Floating point number
double myDoubleNum = 9.98;   // Floating point number
char myLetter = 'D';         // Character
bool myBoolean = true;       // Boolean
string myText = "Hello";     // String
```

C++ DATA TYPE

Basic Data Types(기본 데이터 유형)

- 데이터 유형은 변수가 저장할 정보의 크기와 유형을 지정한다.

Data Type	Size	Description
int	4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 7 decimal digits
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits
boolean	1 byte	Stores true or false values
char	1 byte	Stores a single character/letter/number, or ASCII values

C++ DATA TYPE

C++ Numeric Data Types

- 35 또는 1000과 같이 소수 없이 정수를 저장해야 하는 경우 int를 사용하고 9.99 또는 3.14515와 같이 부동 소수점 숫자(소수점 포함)가 필요한 경우 float 또는 double을 사용한다.

int

```
int myNum = 1000;
cout << myNum;
```

float

```
float myNum = 5.75;
cout << myNum;
```

double

```
double myNum = 19.99;
cout << myNum;
```

C++ DATA TYPE

C++ Numeric Data Types

- float vs. double
  - 부동 소수점 값의 정밀도는 소수점 이하 값이 가질 수 있는 자릿수를 나타낸다.
  - Float의 정밀도는 소수점 이하 6자리 또는 7자리에 불과하지만 double 변수는 정밀도가 약 15자리이다.
  - 따라서 대부분의 계산에는 double을 사용하는 것이 더 안전하다.

Scientific Numbers

- 부동 소수점 숫자는 10의 거듭제곱을 나타내는 "e"가 있는 과학적 숫자일 수도 있다.

Example

```
float f1 = 35e3;
double d1 = 12E4;
cout << f1;
cout << d1;
```

C++ DATA TYPE

C++ Boolean Data Types

- Bool data type은 bool keyword로 선언되며 true 또는 false 값만 사용할 수 있다.
- 값이 반환되면 true = 1이고 false = 0이다.

Example

```
bool isCodingFun = true;
bool isFishTasty = false;
cout << isCodingFun;  // Outputs 1 (true)
cout << isFishTasty;  // Outputs 0 (false)
```

C++ DATA TYPE

C++ Character Data Types

- char 데이터 유형은 단일 문자를 저장하는 데 사용된다.
- 문자는 'A' 또는 'c'와 같이 작은따옴표(‘’)로 묶어야 한다.

Example

```
char myGrade = 'B';
cout << myGrade;
```

C++ DATA TYPE

C++ Character Data Types

- ASCII 값을 사용하여 특정 문자를 표시할 수 있다.

Example

```
char a = 65, b = 66, c = 67;
cout << a;
cout << b;
cout << c;
```

C++ DATA TYPE

C++ String Data Types

- String Type은 일련의 문자(텍스트) 열를 저장하는 데 사용된다.
- 이것은 기본 제공 유형이 아니지만 가장 기본적인 사용 유형처럼 작동 한다.
- 문자열 값은 큰따옴표(“”)로 묶어야 한다.

Example

```
string greeting = "Hello";
cout << greeting;
```

C++ DATA TYPE

C++ Modifier Types

- C++ 은 char, int, double data type 앞에 modifier 를 가질 수 있도록 허용한다.
- Modifier 는 base type의 의미를 수정하는데 사용되며, 그래서 다양한 상황에 대한 요구를 더 정확하게 만족시킨다.
- 그 data type modifier 의 리스트는 아래와 같다
  - signed
  - unsigned
  - long
  - short

C++ DATA TYPE

C++ Modifier Types

- signed, unsigned, long, short 는 integer base type 들에 적용될 수 있다. 부가적으로 signed 와 unsigned 는 char 에 적용될 수 있으며, long 은 double 에 적용될 수 있다.
- signed 와 unsigned 는 long 이나 short 앞에 사용될 수도 있다. 예를 들면 unsigned long int 이다.
- C++ 은 unsigned, short, long 의 integer 를 선언하기 위한 단축 표기를 허용한다. 당신은 int 없이 단순히 unsigned, short, long 만을 사용할 수 있다. int 는 내포되어 있다. 예를 들어 다음의 두 문장은 둘 다 unsigned integer 변수를 선언한다.

```
unsigned x;
unsigned int y;
```

C++ DATA TYPE

C++ Modifier Types

```cpp
#include <iostream>

using namespace std;

/* This program shows the difference between
   * signed and unsigned integers.
*/
int main() {
   short int i;           // a signed short integer
   short unsigned int j;  // an unsigned short integer

   j = 50000;

   i = j;
   cout << i << " " << j; cout << endl;

   return 0;
}
```

```
-15536 50000
계속하려면 아무 키나 누르십시오 . . .
```

위의 결과는 50,000을 short unsigned integer로 나타내는 비트 패턴을 short로 -15,536으로 해석하기 때문이다.

C++ DATA TYPE

Type Qualifier in C++(Type 한정)

Qualifier	Meaning
const	const type 의 개체는 프로그램 실행 동안에 변경될 수 없다.
volatile	volatile 은 컴파일러에게 변수의 값이 프로그램에 의해서 명시적으로 지정되지 않은 방식으로 변경될 수 있음을 통보한다.
restrict	restrict 에 의한 pointer qualifier 는 초기에 단지 그것이 가리키는 개체에 의해서만 접근 가능하다는 것을 의미한다. C99 에서만 restrict 라 불리는 새로운 유형의 qualifiler 가 추가되었다

C++ OPERATORS

- Arithmetic operators(산술 연산자)
- Assignment operators(대입 연산자)
- Comparison operators(비교 연산자)
- Logical operators(논리 연산자)
- Bitwise operators(비트 연산자)
- Misc Operators(기타 연산자)

C++ OPERATORS

Arithmetic Operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

C++ OPERATORS

Assignment Operators

- 대입 연산자는 변수에 값을 할당하는 데 사용된다.

Example

```
int x = 10;
x += 5;
```

C++ OPERATORS

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

C++ OPERATORS

C++ Comparison Operators

- 비교 연산자는 두 값을 비교하는 데 사용된다.
- 비교의 반환 값은 true(1) 또는 false(0)이다.

Example

```
int x = 5;
int y = 3;
cout << (x > y); // returns 1 (true) because 5 is greater than 3
```

C++ OPERATORS

C++ Comparison Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

C++ OPERATORS

Logical Operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

C++ OPERATORS

Bitwise operator

- 비트 연산자는 논리 연산자와 비슷하지만, 비트(bit) 단위로 논리 연산을 할 때 사용하는 연산자이다.
- 또한, 비트 단위로 왼쪽이나 오른쪽으로 전체 비트를 이동하거나, 1의 보수를 만들 때도 사용된다.

operator	asm equivalent	description
&	AND	Bitwise AND
|	OR	Bitwise inclusive OR
^	XOR	Bitwise exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift bits left
>>	SHR	Shift bits right

C++ OPERATORS

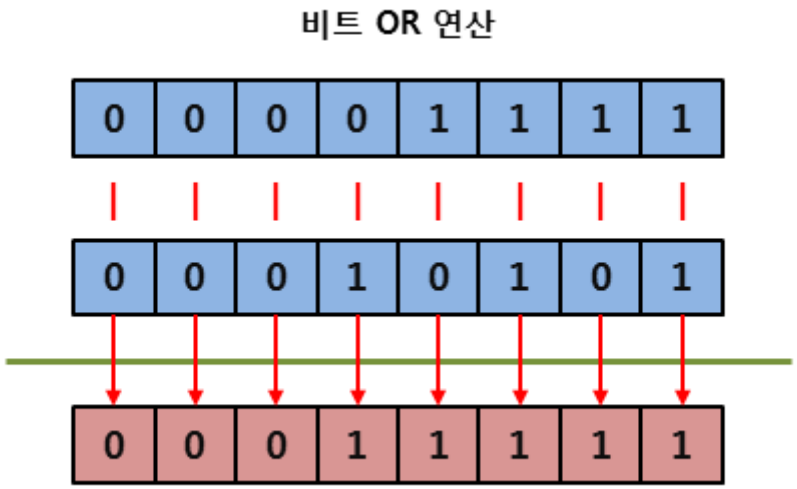
Bitwise operator

- 그림은 비트 AND 연산자(&)의 동작을 나타낸다.
- 이처럼 비트 AND 연산자는 대응되는 두 비트가 모두 1일 때만 1을 반환하며, 다른 경우는 모두 0을 반환한다.

C++ OPERATORS

Bitwise operator

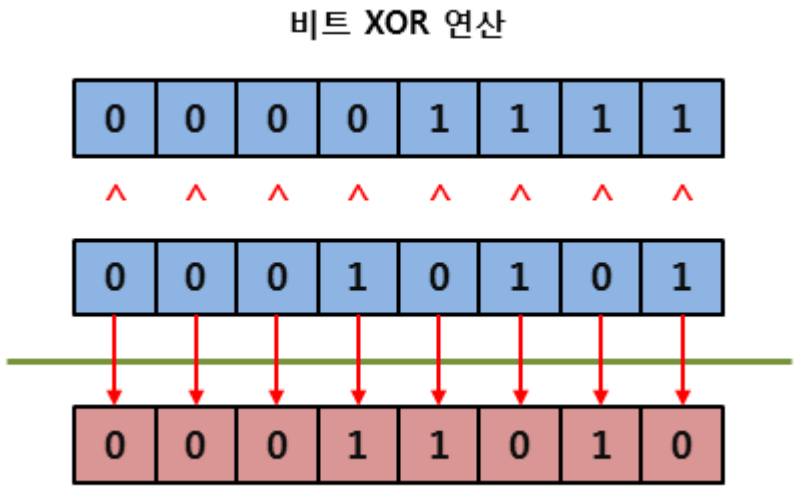
- 그림은 비트 OR 연산자(|)의 동작을 나타낸다.
- 이처럼 비트 OR 연산자는 대응되는 두 비트 중 하나라도 1이면 1을 반환하며, 두 비트가 모두 0일 때만 0을 반환한다.



C++ OPERATORS

Bitwise operator

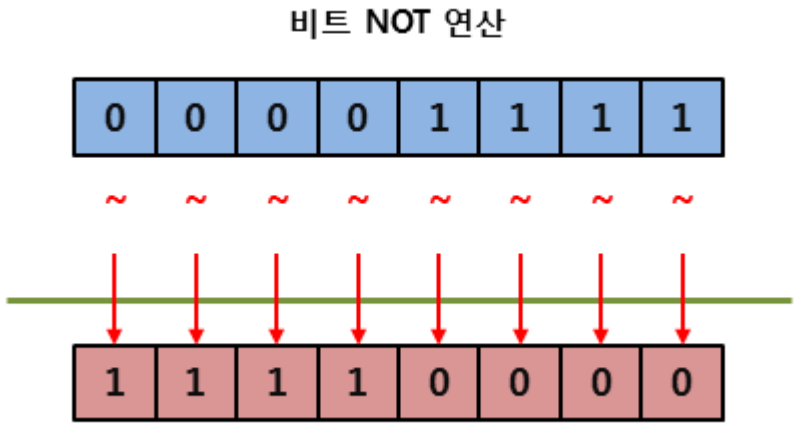
- 그림은 비트 XOR 연산자(^)의 동작을 나타낸다.
- 이처럼 비트 XOR 연산자는 대응되는 두 비트가 서로 다르면 1을 반환하고, 서로 같으면 0을 반환한다.



C++ OPERATORS

Bitwise operator

- 그림은 비트 NOT 연산자(~)의 동작을 나타낸다.
- 이처럼 비트 NOT 연산자는 해당 비트가 1이면 0을 반환하고, 0이면 1을 반환한다.



C++ IOSTREAM

C++ Standard Input/Output(표준 입출력) Class

- 사용자가 프로그램과 대화하기 위해서는 사용자와 프로그램 사이의 입출력을 담당하는 수단이 필요하다.
- C++의 모든 것은 Object(객체)로 표현되므로, 입출력을 담당하는 수단 또한 C언어의 함수와는 달리 모두 Object이다.
- C언어의 printf() 함수나 scanf() 함수처럼 C++에서도 iostream Header file(헤더 파일)에 표준 입출력 클래스를 정의하고 있다.
- C++에서는 cout Object로 출력 작업을, cin 객체로 입력 작업을 수행하고 있다.
- C++에서는 기존의 C언어 스타일처럼 printf() 함수나 scanf() 함수로도 입출력 작업을 수행할 수 있다.

C++ IOSTREAM

C++ Output (Print Text)

- cout 객체는 << 연산자와 함께 값을 출력하거나 텍스트를 출력하는 데 사용된다.
- std::cout << “output string”;
- 출력 연산자(<<)는 오른쪽에 위치한 출력할 데이터를 출력 스트림에 삽입한다.
- 이렇게 출력 스트림에 삽입된 데이터는 스트림을 통해 출력 장치로 전달되어 출력된다.

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

int main() {
  cout << "Hello World!";
  return 0;
}
```

C++ IOSTREAM

New Lines

- 줄 바꿈(새 줄)을 삽입하려면 ₩n 문자를 사용할 수 있다.
- 또한 endl을 사용 하여 줄 바꿈(새 줄)을 삽입한다.

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

int main() {
  cout << "Hello World! \n";
  cout << "I am learning C++";
  return 0;
}
```

C++ IOSTREAM

C++ User Input

- cout이 값을 출력(인쇄)하는 데 사용된다는 면 이제 cin을 사용하여 사용자 입력을 받는다.
- cin은 extraction(추출) operator(연산자)(>>)를 사용하여 키보드에서 데이터를 읽는 미리 정의된 입력 스트림을 나타내는 객체이다.
- std::cin >> 저장 할 변수;

```
int x;
cout << "Type a number: "; // Type a number and press enter
cin >> x; // Get user input from the keyboard
cout << "Your number is: " << x; // Display the input value
```

C++ STRINGS

- Strings are used for storing text.
- To use strings, you must include an additional header file in the source code, the <string> library

```
// Include the string library
#include <string>

// Create a string variable
string greeting = "Hello";
```

C++ STRINGS

String Concatenation

- + 연산자는 문자열 사이에 사용하여 함께 추가하여 새 문자열을 만들 수 있다.
- 이것을 Concatenation (연결)이라고 한다.

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName + lastName;
cout << fullName;
```

C++ STRINGS

String Concatenation

- 출력 시 John과 Doe 사이에 공백을 만들기 위해 firstName 뒤에 공백을 추가했다.
- 따옴표(" " 또는 ' ')로 공백을 추가할 수도 있다.

```
string firstName = "John";
string lastName = "Doe";
string fullName = firstName + " " + lastName;
cout << fullName;
```

C++ STRINGS

String Concatenation - Append

- C++의 string은 실제로 문자열에 대해 특정 작업을 수행할 수 있는 함수를 포함하는 object이다.
- 예를 들어, 문자열을 append() 함수로 연결할 수도 있다.

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName.append(lastName);
cout << fullName;
```

+ 또는 append()를 사용할지 여부는 사용자에게 달려 있다.
이 둘의 주요 차이점은 append() 함수가 훨씬 더 빠르다는 것이다.
그러나 테스트 등의 경우에는 +를 사용하는 것이 더 쉬울 수 있다.

C++ STRINGS

C++ Numbers and Strings

- C++는 덧셈과 문자열 연결 모두에 + 연산자를 사용한다.
- 숫자는 덧셈이 된다.
- 문자열은 문자열이 연결된다.

```
int x = 10;
int y = 20;
int z = x + y;    // z will be 30 (an integer)

string xx = "10";
string yy = "20";
string zz = xx + yy;   // zz will be 1020 (a string)
```

C++ STRINGS

C++ Numbers and Strings

```
string x = "10";
int y = 20;
string z = x + y;
```

C++ STRINGS

C++ String Length

- 문자열의 길이를 얻으려면 length() 함수를 사용한다.
- string의 길이를 얻기 위해 size() 함수를 사용하는 일부 C++ 프로그램을 볼 수 있다.
- 이것은 단지 length()의 alias(별칭)이다. length() 또는 size()를 사용하려는 경우 전적으로 프로그래머에게 달려 있다.

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "The length of the txt string is: " << txt.length();
cout << "The length of the txt string is: " << txt.size();
```

C++ STRINGS

C++ Access Strings

- 대괄호 [] 안의 Index(색인)를 참조하여 문자열의 문자에 access할 수 있다.

```
string myString = "Hello";
cout << myString[0];
// Outputs H
```

C++ STRINGS

Omitting Namespace(Namespace 생략)

- Standard Namespace Library 없이 실행되는 일부 C++ 프로그램을 볼 수 있다.
- using namespace std 행은 생략하고 std 키워드로 대체할 수 있으며, 그 뒤에 string(및 cout) 객체에 대한 :: 연산자가 온다.
- Standard Namespace Library 를 포함할지 여부는 개발자에게 달려 있다.

```
#include <iostream>
#include <string>

int main() {
  std::string greeting = "Hello";
  std::cout << greeting;
  return 0;
}
```

C++ CONTROL FLOW STATEMENTS

- C++ 프로그램이 원하는 결과를 얻기 위해서는 프로그램의 순차적인 흐름을 제어해야만 할 경우가 생긴다.
- 이때 사용하는 명령문을 제어문이라고 하며, 이러한 제어문에는 조건문, 반복문 등이 있다.
- 이러한 제어문에 속하는 명령문(statements)들은 중괄호({})로 둘러싸여 있으며, 이러한 중괄호 영역을 블록(block)이라고 한다.

C++ CONTROL FLOW STATEMENTS

조건문(conditional statements)

- C++는 수학의 일반적인 논리 조건을 지원한다.
  - Less than: a < b
  - Less than or equal to: a <= b
  - Greater than: a > b
  - Greater than or equal to: a >= b
  - Equal to a == b
  - Not Equal to: a != b
- 이러한 조건을 사용하여 다른 결정에 대해 다른 작업을 수행할 수 있다.

- C++에는 다음과 같은 조건문이 있다.
  - 지정된 조건이 true인 경우 실행할 코드 블록을 지정하려면 'if'를 사용한다.
  - 동일한 조건이 false인 경우 'else'를 사용하여 실행할 코드 블록을 지정한다.
  - 첫 번째 조건이 false인 경우 테스트할 새 조건을 지정하려면 'else if'를 사용한다.
  - 실행할 많은 대체 코드 블록을 지정하려면 'switch'를 사용한.

C++ CONTROL FLOW STATEMENTS

The ‘if’ Statement

- if 문을 사용하여 조건이 true인 경우 실행할 C++ 코드 블록을 지정한다.
- if 는 소문자이다. 대문자(If 또는 IF)는 오류를 생성한다.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

C++ CONTROL FLOW STATEMENTS

The ‘else’ Statement

- else 문을 사용하여 조건이 false인 경우 실행할 코드 블록을 지정한다.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

C++ CONTROL FLOW STATEMENTS

The “else if” Statement

- 첫 번째 조건이 false인 경우 else if 문을 사용하여 새 조건을 지정한다.

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

C++ CONTROL FLOW STATEMENTS

C++ Short Hand If Else - Ternary Operator

- 세 개의 피연산자로 구성 된 삼항 연산자로 알려진 short-handed if else도 있다.

Syntax

```
variable = (condition) ? expressionTrue : expressionFalse;
```

C++ CONTROL FLOW STATEMENTS

C++ Switch Statements

- switch 문은 if / else 문과 마찬가지로 주어진 조건 값의 결과에 따라 프로그램이 다른 명령을 수행하도록 하는 조건문이다.
- switch 문을 사용하여 실행할 많은 코드 블록 중 하나를 선택한다.
  - switch expression은 한 번 평가된다.
  - expression의 값은 각 case 값과 비교된다.
  - 일치하는 항목이 있으면 연결된 code block이 실행된다.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

C++ CONTROL FLOW STATEMENTS

C++ Switch Statements - The break Keyword

- C++가 break 키워드에 도달하면 switch 블록에서 나온다.
- 이렇게 하면 블록 내에서 더 많은 코드 및 case 테스트 실행이 중지된다.
- 일치하는 항목이 발견되고 작업이 완료되면 break 된다.더 많은 case 테스트가 필요하지 않다.
- Break문은 switch block의 나머지 모든 코드 실행을 " 무시 " 하기 때문에 많은 실행 시간을 절약할 수 있다.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

C++ CONTROL FLOW STATEMENTS

C++ Switch Statements - The default Keyword

- Default 는 일치하는 case가 없는 경우에 실행할 code를 지정한다.
- default 키워드는 switch의 마지막 문으로 사용해야 하며 break가 필요하지 않다.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

C++ CONTROL FLOW STATEMENTS

C++ While Loop

- 루프는 지정된 condition(조건)에 만족할 때까지 블록을 실행할 수 있다.
- 루프는 시간을 절약하고 오류를 줄이며 코드를 더 읽기 쉽게 만들기 때문에 편리하다.
- While 루프는 지정된 condition이 true인경우에 code block을 반복한다.
- condition에 사용된 변수에 내용이 변경되도록 해야 한다. 그렇지 않으면 무한 loop가 될 것이다.

Syntax

```
while (condition) {
  // code block to be executed
}
```

C++ CONTROL FLOW STATEMENTS

C++ Do/While Loop

- do/while loop는 while loop의 변형이다.
- 이 루프는 condition이 참인지 확인하기 전에 code block을 한 번 실행한 다음 condition이 true인 동안 loop를 반복한다.
- condition에 사용된 변수에 내용이 변경되도록 해야 한다. 그렇지 않으면 무한 loop가 될 것이다.

Syntax

```
do {
  // code block to be executed
}
while (condition);
```

C++ CONTROL FLOW STATEMENTS

C++ For Loop

- for loop는 특정 횟수만큼 실행해야 하는 loop를 효율적으로 작성할 수 있는 반복 제어 구조이다.
  - Init(초기식)은 코드 블록이 실행되기 전에 (한 번) 실행된다.
  - Condition(조건식)는 코드 블록을 실행하기 위한 조건을 정의한다.
    - 조건식의 결과가 true인 동안 반복한다.
  - increment(증감식) 코드 블록이 실행된 후 (매번) 실행된다.

```
for ( init; condition; increment ) {
   statement(s);
}
```

C++ CONTROL FLOW STATEMENTS

C++ Range-based For Loop

- C++11부터는 범위 기반의 for 문이라는 새로운 형태의 반복문이 추가되었다.
- 이러한 종류의 for 루프는 범위의 모든 요소를 반복합니다. 여기서 선언(declaration)은 이 범위(range)의 요소 값을 취할 수 있는 일부 변수를 선언한다.
- 범위(range)는 배열, 컨테이너이 시작 및 종료 기능을 지원하는 기타 유형을 포함하는 요소의 sequences 이다.

- Syntax
  - for (declaration : range) statement;

C++ CONTROL FLOW STATEMENTS

C++ Break

- break 문은 loop 내에서 사용하여 해당 loop에서 빠져 나온다.
- 즉 루프 내에서 조건식의 판단 결과와 상관없이 반복문을 완전히 빠져나가고 싶을 때 사용한다.

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  cout << i << "\n";
}
```

C++ CONTROL FLOW STATEMENTS

C++ Continue

- continue 문은 loop 내에서 사용하여 해당 루프의 나머지 부분을 건너뛰고, 바로 다음 조건식의 판단으로 넘어가게 해준다.
- 보통 loop 내에서 특정 조건에 대한 예외 처리를 하고자 할 때 자주 사용된다.

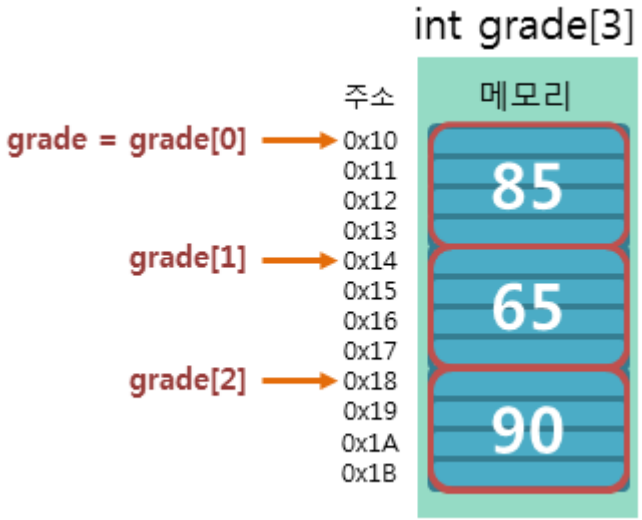
```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  cout << i << "\n";
}
```

C++ ARRAYS AND POINTER

C++ Arrays

- 배열은 각 값에 대해 별도의 변수를 선언하는 대신 단일 변수에 여러 값을 저장하는 데 사용된다.
- 배열을 선언하려면 변수 유형을 정의하고 배열 이름과 대괄호를 지정하고 저장해야 하는 요소 수를 지정한다.
  - type arrayName[length];

```
string cars[4];
```



C++ ARRAYS AND POINTER

C++ Omit Arrays

- 배열의 크기를 지정 하지 않고도 선언할 수 있다.
- 그러나 삽입된 요소만큼만 크기가 커진다.

```
string cars[] = {"Volvo", "BMW", "Ford"}; // size of array is always 3
```

C++ ARRAYS AND POINTER

C++ References

- Reference variable(참조 변수)는 기존 변수에 대한 “참조”이며 & 연산자로 생성된다.

```
string food = "Pizza";  // food variable
string &meal = food;    // reference to food
```

C++ ARRAYS AND POINTER

C++ Memory Address

- 이전 페이지의 예에서는 & 연산자를 사용하여 참조 변수를 생성했다.
- 그러나 변수의 메모리 주소를 얻는 데에도 사용할 수 있다. 메모리에 변수가 저장되는 위치이다.
- C++에서 변수를 생성하면 변수에 메모리 주소가 할당된다. 그리고 변수에 값을 할당하면 이 메모리 주소에 저장된다.
- access하려면 & 연산자를 사용하 변수가 저장된 위치를 나타낸다.

```
string food = "Pizza";

cout << &food; // Outputs 0x6dfed4
```

메모리 주소를 아는 것이 왜 유용한가?
참조와 포인터는 C++에서 중요하다.
왜냐하면 컴퓨터 메모리의 데이터를 조작할 수 있는 능력을 제공하기 때문이다.
이러한 것은 code를 줄이고 성능을 향상시킬 수 있다.
이 두 가지 기능은 C++를 Python 및 Java와 같은 다른 프로그래밍 언어와 차별화하는 요소 중 하나이다.

C++ ARRAYS AND POINTER

C++ Pointer

- 변수가 선언되면 그 값을 저장하는 데 필요한 메모리의 특정 위치(메모리 주소)가 할당된다.
- 일반적으로 C++ 프로그램은 변수가 저장되는 정확한 메모리 주소를 능동적으로 결정하지 않는다.
- 다행히도 그 작업은 프로그램이 실행되는 환경에 맡겨집니다. 일반적으로 런타임 시 특정 메모리 위치를 결정하는 것은 운영 체제이다.
- 그러나 프로그램이 런타임 중에 변수에 대한 특정 위치에 있는 데이터 셀에 access(접근)하기 위해 변수의 주소를 얻어오는 것이 유용할 수 있다.
- C++에서 포인터(pointer)란 메모리의 주소 값을 저장하는 변수이며, 포인터 변수라고도 부른다.
- Char형 변수가 문자를 저장하고, int형 변수가 정수를 저장하는 것처럼 포인터는 주소 값을 저장하는 데 사용된다.

C++ ARRAYS AND POINTER

Address-of (주소) operator - &

- 변수의 주소는 변수 이름 앞에 주소 연산자라고 하는 ampersand 기호 (&)를 붙여서 얻을 수 있다.

Dereference(역 참조) operator-*

- 다른 변수의 주소를 저장하는 변수를 포인터라고 한다.
- 포인터는 주소가 저장되어 있는 변수를 "가리키는" 것이다.

```
string food = "Pizza";  // A string variable
string* ptr = &food;  // A pointer variable that stores the address of food
```

C++ ARRAYS AND POINTER

Declaring pointers

- 포인터 변수 선언의 일반적인 형식은 다음과 같다.
- type * name;
  - int * number;
  - char * character;
  - double * decimals;

C++ ARRAYS AND POINTER

C++ Modify Pointers

- 포인터의 값을 변경할 수도 있다.
- 그러나 이렇게 하면 원래 변수의 값도 변경된다.

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

int main() {
  string food = "Pizza";
  string* ptr = &food;

  // Output the value of food
  cout << food << "\n";

  // Output the memory address of food
  cout << &food << "\n";

  // Access the memory address of food and output its
  cout << *ptr << "\n";

  // Change the value of the pointer
  *ptr = "Hamburger";

  // Output the new value of the pointer
  cout << *ptr << "\n";

  // Output the new value of the food variable
  cout << food << "\n";
  return 0;
}
```

C++ ARRAYS AND POINTER

C++ Pointers and arrays

- 배열의 개념은 포인터의 개념과 관련이 있다. 실제로 배열의 첫 번째 요소는 포인터와 매우 유사하게 작동하며 실제로 배열은 항상 적절한 유형의 포인터로 묵시적으로 변환될 수 있다.
- 포인터와 배열은 동일한 작업 집합을 지원하며 둘 다 동일한 의미를 갖는다. 주요 차이점은 포인터에는 새 주소를 할당할 수 있지만 배열에는 할당할 수 없다는 것이다.

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

int main ()
{
  int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
  return 0;
}
```

C++ ARRAYS AND POINTER

C++ Pointers and arrays

- 배열에 대한 장에서 대괄호([])는 배열 요소의 인덱스를 지정하는 것이다. 사실 이 괄호는 오프셋 연산자로 알려진 역참조 연산자이다. 그들은 뒤따르는 변수를 *처럼 역참조하지만 역참조되는 주소에 대괄호 사이의 숫자도 추가한다.
- 이 두 표현식은 포인터인 경우 뿐 아니라 가 배열인 경우에도 동일하고 유효하다. 배열인 경우 해당 이름을 첫 번째 요소에 대하여 포인터처럼 사용할 수 있음을 기억해야 한다.

```
a[5] = 0;       // a [offset of 5] = 0
*(a+5) = 0;     // pointed to by (a+5) = 0
```

C++ ARRAYS AND POINTER

C++ Pointer initialization

- 포인터를 선언한 후 참조 연산자(*)를 사용하기 전에 포인터는 반드시 초기화되어야 한다.
- 초기화하지 않은 채로 참조 연산자를 사용하게 되면, 어딘지 알 수 없는 메모리 장소에 값을 저장하는 것이 된다.

```cpp
#include <iostream>

using namespace std;

int main () {
   int  var = 20;   // actual variable declaration.
   int  *ip;        // pointer variable

   ip = &var;       // store address of var in pointer variable

   cout << "Value of var variable: ";
   cout << var << endl;

   // print the address stored in ip pointer variable
   cout << "Address stored in ip variable: ";
   cout << ip << endl;

   // access the value at the address available in pointer
   cout << "Value of *ip variable: ";
   cout << *ip << endl;

   return 0;
}
```

C++ ARRAYS AND POINTER

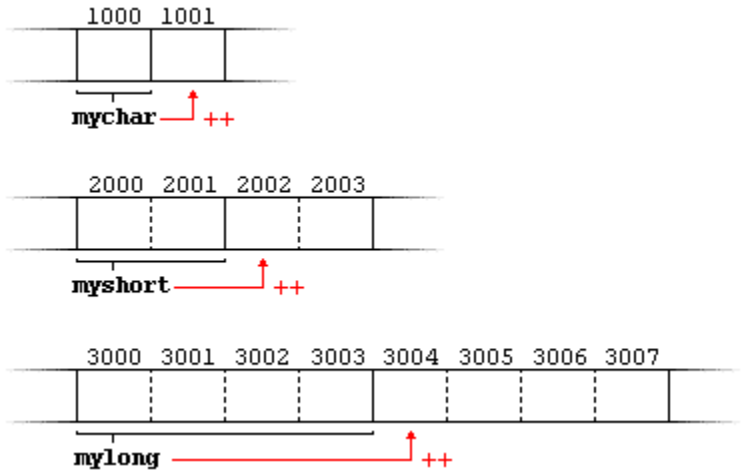
C++ Pointer arithmetics(pointer 연산)

- 포인터에서 산술 연산을 수행하는 것은 일반 정수 유형에서 수행하는 것과 약간 다르다.
- 우선 덧셈과 뺄셈 연산만 허용되며 나머지 연산은 포인터의 세계에서 의미가 없다.
- 그러나 덧셈과 뺄셈은 포인터가 가리키는 데이터 유형의 크기에 따라 포인터에 대해 다른 동작을 한다.
- 기본 데이터 유형은 크기가 서로 다른 것을 배웠다. 예를 들어: char의 크기는 항상 1바이트이고 short는 일반적으로 그보다 크며 int와 long은 훨씬 더 큰다. 이들의 정확한 크기는 시스템에 따라 다르다. 예를 들어, 사용하는 시스템에서 char은 1바이트, short는 2바이트, long은 4를 사용한다고 가정해 보겠다.

C++ ARRAYS AND POINTER

C++ Pointer arithmetics(pointer 연산)

- char *mychar;
- short *myshort;
- long *mylong;

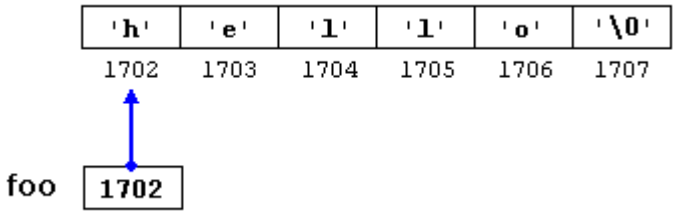


- ++mychar;
- ++myshort;
- ++mylong;

C++ ARRAYS AND POINTER

C++ Pointers and string literals

- 문자열 literal은 null로 끝나는 문자 sequence를 포함하는 배열이다.
- 문자열 literal은 cout에 직접 사용되어 문자열을 초기화하고 문자 배열을 초기화하는 데 사용되었다.
  - const char * foo = "hello";
- "hello"에 대한 literal 표현으로 배열을 선언하고 첫 번째 요소에 대한 포인터가 foo에 할당된다. " hello " 가 주소 1702에서 시작하는 메모리 위치에 저장되어 있다고 상상하면 이전 선언을 다음과 같이 나타낼 수 있다.



C++ ARRAYS AND POINTER

C++ Pointers and string literals

- 여기에서 foo는 포인터이고 주소 값 1702를 포함하고 ' h ' 나 " hello " 가 아니라 실제로 1702가 이 두 가지의 주소이다.
- 포인터 foo는 일련의 문자를 가리킨다. 그리고 포인터와 배열은 표현식에서 본질적으로 같은 방식으로 동작하기 때문에 foo는 null로 끝나는 문자 sequence의 배열과 같은 방식으로 문자에 액세스하는 데 사용할 수 있다.

- *(foo+4)
- foo[4]
- 두 표현식 모두 'o'(배열의 다섯 번째 요소) 값을 갖는다.

C++ ARRAYS AND POINTER

C++ Pointers to pointers(이중 포인터)