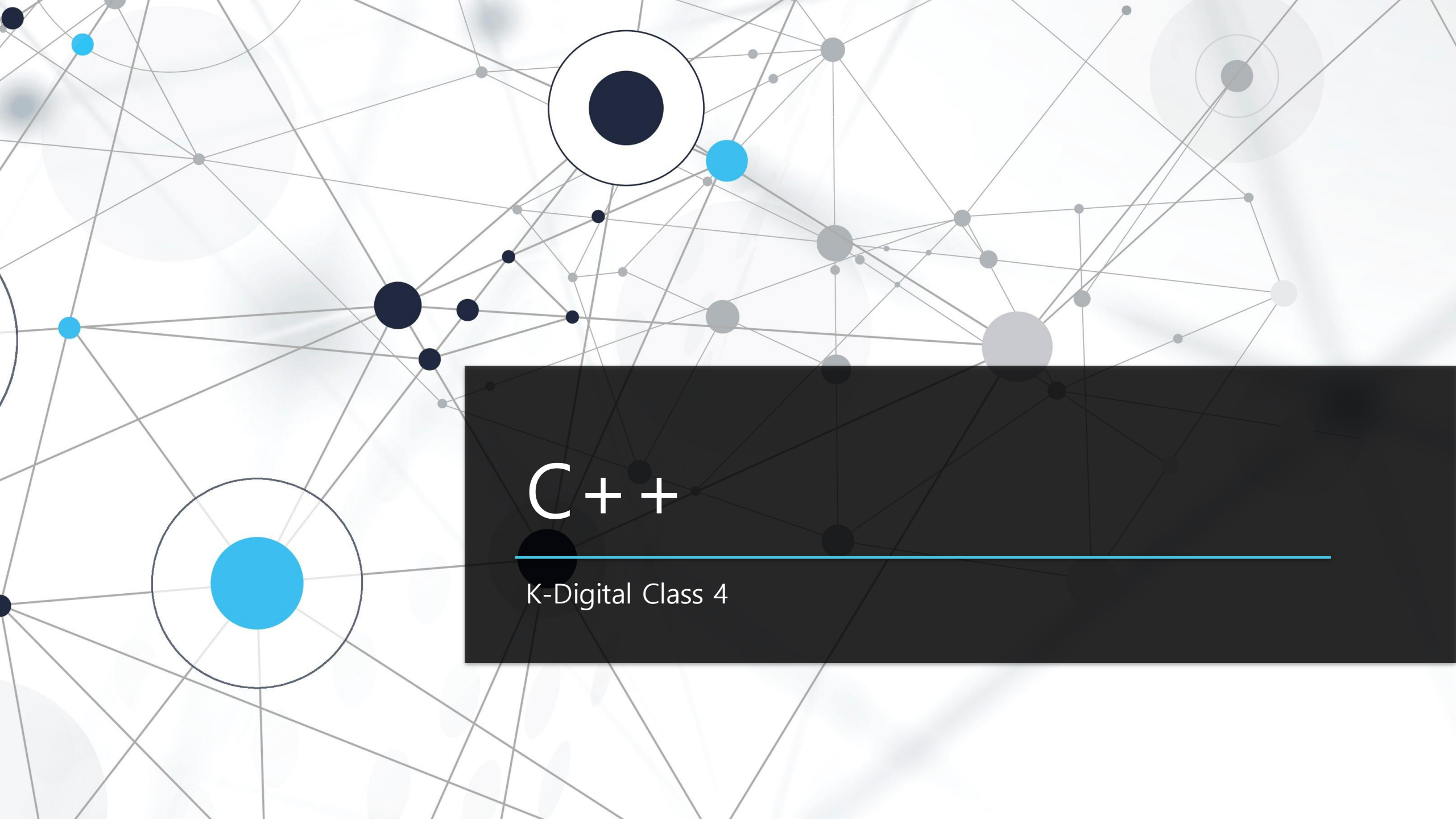


C++

K-Digital Class 4

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.

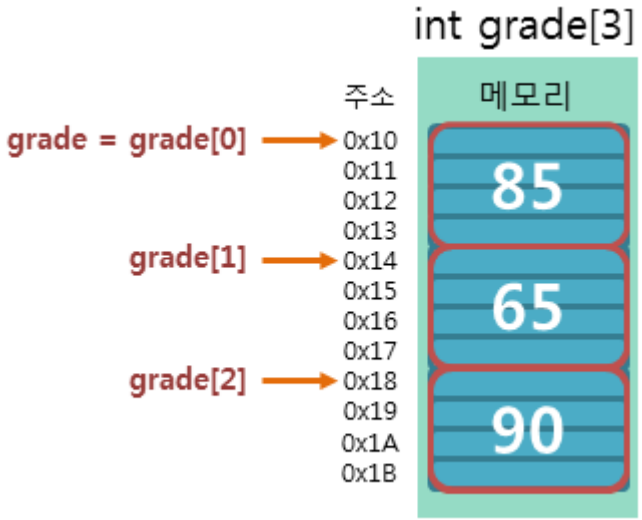
```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  cout << i << "\n";
}
```

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

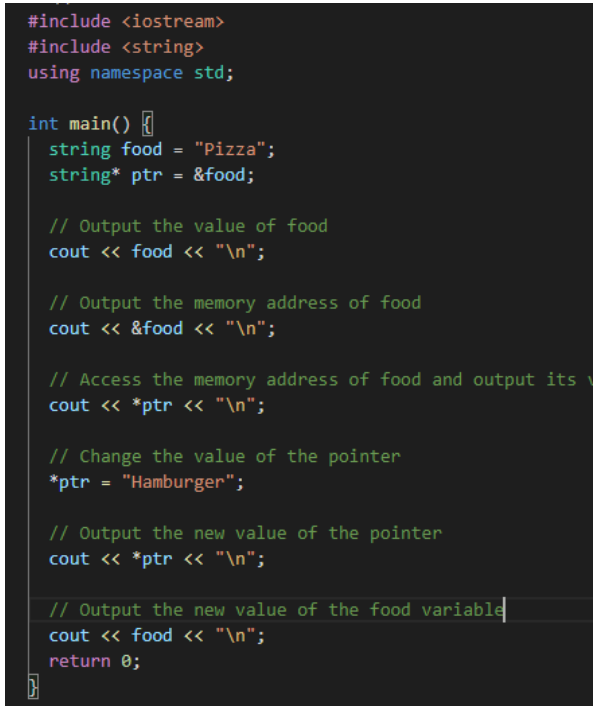
Declaring pointers

- 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.



```cpp
#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

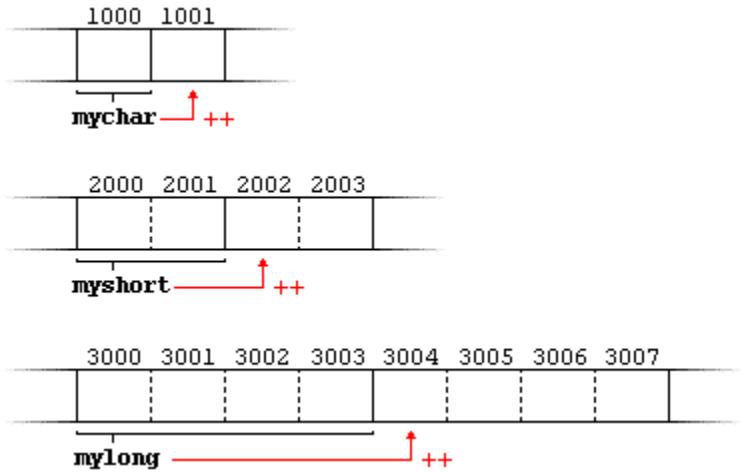
C++ Pointer arithmetics(pointer 연산)

- 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;

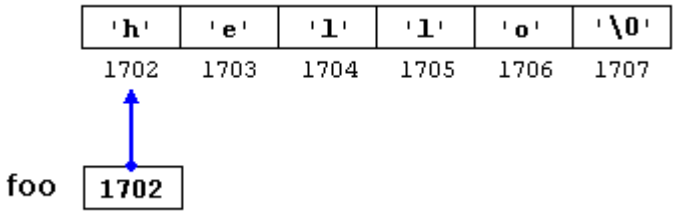


- ++mychar;
- ++myshort;
- ++mylong;

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)