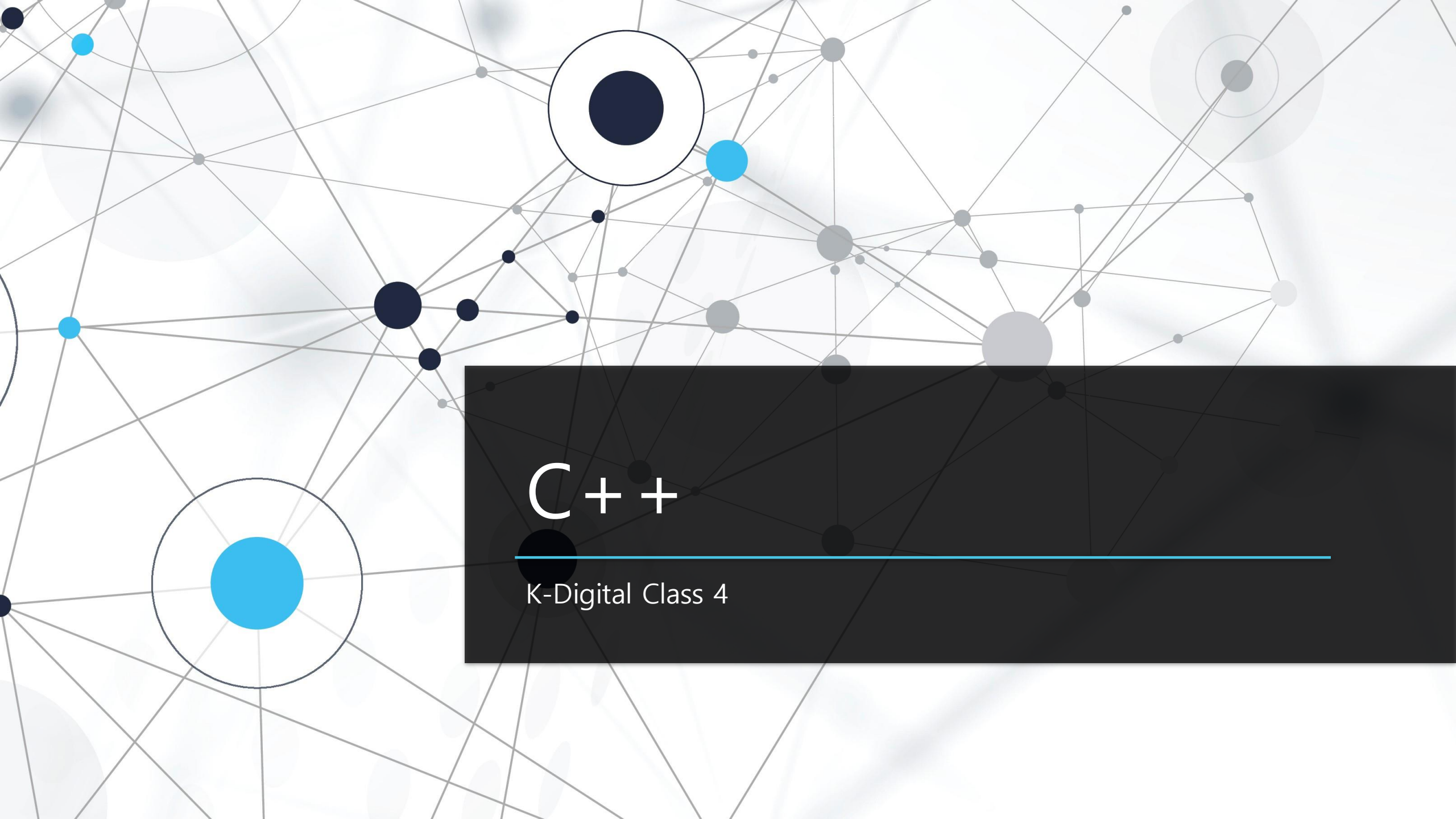


C++

K-Digital Class 4

C++ ARRAYS AND POINTER

C++ new[] and delete[] operator

- new[] operator
  - 문자 배열, 즉 20자의 문자열에 대한 메모리를 할당한다고 가정하면, 예제와 같이 메모리를 동적으로 할당할 수 있다.

```
char* pvalue  = NULL;          // Pointer initialized with null
pvalue  = new char[20];        // Request memory for the variable
```

```
double** pvalue  = NULL;       // Pointer initialized with null
pvalue  = new double [3][4];   // Allocate memory for a 3x4 array
```

C++ ARRAYS AND POINTER

C++ new[] and delete[] operator

- delete[] operator
- 앞장에서 생성 한 배열을 제거하려면 다음과 같이 한다.

```
delete [] pvalue;            // Delete array pointed to by pvalue
```

```
delete [] pvalue;            // Delete array pointed to by pvalue
```

C++ ARRAYS AND POINTER

C++ Two-dimensional Arrays

- 2차원 배열이란 배열의 요소로 1차원 배열을 가지는 배열이다.
- C++에서는 2차원 배열을 나타내는 타입을 따로 제공하지 않는다.
- 대신에 1차원 배열의 배열 요소로 또 다른 1차원 배열을 사용하여 2차원 배열을 나타낼 수 있다.
- type(타입)은 배열 요소로 저장되는 변수의 타입을 설정한다.
- arrName은 배열이 선언된 후에 배열에 접근하기 위해 사용된다.

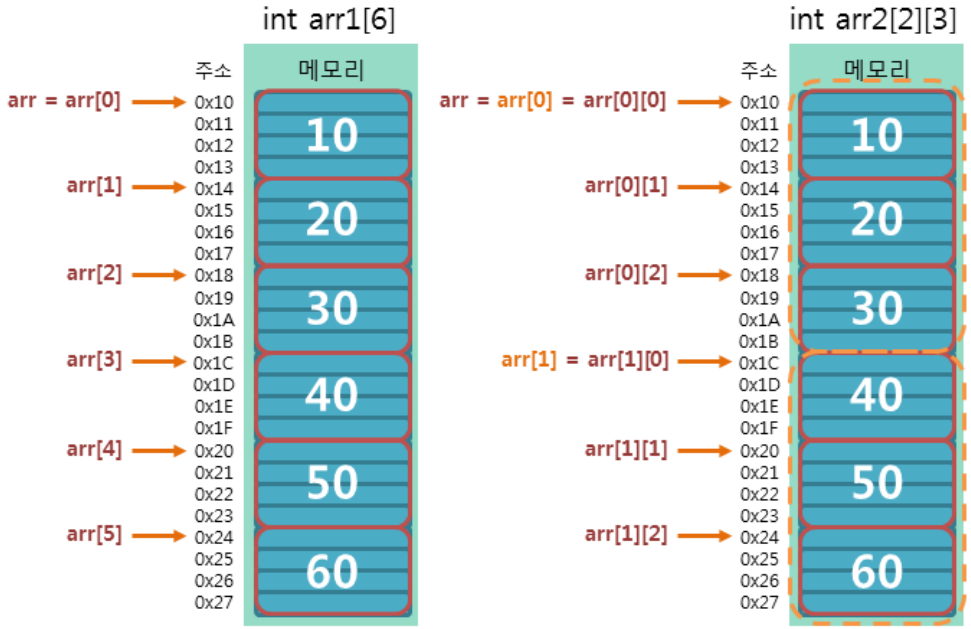
```
type arrayName [ x ][ y ];
```

C++ ARRAYS AND POINTER

C++ Two-dimensional Arrays

- 하지만 컴퓨터의 Memory는 입체적 공간이 아닌 선형 공간이므로 실제로는 다음 그림과 같이 저장된다.

```
int arr1[6] = {10, 20, 30, 40, 50, 60};
int arr2[2][3] = {10, 20, 30, 40, 50, 60};
```



C++ ARRAYS AND POINTER

Initializing Two-Dimensional Arrays

- 다차원 배열은 각 행에 대괄호로 묶인 값을 지정하여 초기화할 수 있다. 다음은 3행4열이 있는 배열의 선언과 함께 초기화 하는 방법이다.

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

C++ ARRAYS AND POINTER

Initializing Two-Dimensional Arrays

- 행을 나타내는 중첩 중괄호는 선택 사항이다.
- 다음 초기화는 이전 예제와 동일하다.

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

C++ ARRAYS AND POINTER

C++ Pointers to pointers & Arrays

- [row][col] 크기의 2차원 배열
  - 원소를 col개 가지고 있는 1차원 배열이 row개 있는 것.
  - 1차원 배열의 주소가 row개 있는 것이나 마찬가지다.
  - 2차원 배열은 포인터의 배열이나 마찬가지다.

```
int main() {
int row = 3, col = 5;
int * row_1 = new int[col] {1, 2, 3, 4, 5}; // int 원소 5개의 1차원 동적 배열 row_1
int * row_2 = new int[col] {6, 7, 8, 9, 10}; // int 원소 5개의 1차원 동적 배열 row_2
int * row_3 = new int[col] {11, 12, 13, 14, 15}; // int 원소 5개의 1차원 동적 배열 row_2

int ** towd_array = new int* [row] {row_1, row_2, row_3};
// int* 포인터 원소 3개(= 1차원 배열이름 3개)의 2차원 동적 배열 2d_array
// 포인터 3개가 들어있는 배열(포인터)의 주소를 리턴받음
// 포인터의 포인터이므로 2d_array의 타입은 int **가 된다.

delete [] row_1;
delete [] row_2;
delete [] row_3;
delete [] towd_array;
```

C++ FUNCTION

- 함수는 호출될 때만 실행되는 코드 블록(code block)이다.
- 매개변수라고 하는 데이터를 함수에 전달할 수 있다.
- 함수는 특정 작업을 수행하는 데 사용되며 코드를 재사용하는 데 중요하다. 코드를 한 번 정의하고 여러 번 사용한다.
- 코드를 별도의 기능으로 나눌 수 있다. 코드를 다른 기능으로 나누는 방법은 사용자에게 달려 있지만 논리적으로 구분은 일반적으로 각 기능이 특정 작업을 수행하도록 한다.

C++ FUNCTION

Defining a Function

- C++는 코드를 실행하는 데 사용되는 main()과 같은 일부 미리 정의된 함수를 제공한다. 그러나 특정 작업을 수행하기 위해 고유한 기능을 만들 수도 있다.
- 함수를 생성(종종 선언이라고도 함)하려면 함수 이름을 지정하고 그 뒤에 괄호()를 붙인다.

```
void myFunction() {
  // code to be executed
}
```

C++ FUNCTION

Call a Function

- C++ 함수를 생성하는 동안 함수가 수행해야 하는 작업에 대한 정의를 제공한다. 함수를 사용하려면 해당 함수를 호출해야 한다. 프로그램이 함수를 호출하면 프로그램 제어가 호출된 함수로 이전된다.
- 호출된 함수는 정의된 작업을 수행하고 return 문이 실행되거나 함수 끝 닫는 중괄호에 도달하면 프로그램 제어를 다시 호출한 메인 프로그램 또는 호출한 함수로 반환한다.

Inside `main`, call `myFunction()`:

```
// Create a function
void myFunction() {
  cout << "I just got executed!";
}

int main() {
  myFunction(); // call the function
  return 0;
}

// Outputs "I just got executed!"
```

C++ FUNCTION

C++ Function Parameters

- 정보는 매개변수로 함수에 전달할 수 있습니다. 매개변수는 함수 내에서 변수 역할을 한다.
- 매개변수는 함수 이름 뒤에 괄호 안에 지정된다. 원하는 만큼 매개변수를 추가할 수 있다. 쉼표(,)로 구분하기만 하면 된다.

Syntax

```
void functionName(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

C++ FUNCTION

C++ Function Default Parameter Value

- 등호(=)를 사용하여 기본 매개변수 값을 사용할 수도 있다.
- 인수 없이 함수를 호출하면 기본값 ("Norway")이 사용된다.

```
void myFunction(string country = "Norway") {
  cout << country << "\n";
}

int main() {
  myFunction("Sweden");
  myFunction("India");
  myFunction();
  myFunction("USA");
  return 0;
}

// Sweden
// India
// Norway
// USA
```

C++ FUNCTION

C++ Function Multiple Parameters

- 함수 내에서 원하는 만큼 매개변수를 추가할 수 있다.

```
void myFunction(string fname, int age) {
  cout << fname << " Refsnes. " << age << " years old. \n";
}

int main() {
  myFunction("Liam", 3);
  myFunction("Jenny", 14);
  myFunction("Anja", 30);
  return 0;
}

// Liam Refsnes. 3 years old.
// Jenny Refsnes. 14 years old.
// Anja Refsnes. 30 years old.
```

C++ FUNCTION

C++ Function Return Values

- 이전 예에서 사용된 void 키워드는 함수가 값을 반환하지 않아야 함을 나타낸다.
- 함수가 값을 반환하도록 하려면 void 대신 데이터 type(유형)(예: int, string 등)을 사용하고 함수 내에서 return 키워드를 사용한다.

```
int myFunction(int x) {
  return 5 + x;
}

int main() {
  cout << myFunction(3);
  return 0;
}

// Outputs 8 (5 + 3)
```

C++ FUNCTION

C++ Function Pass By Reference Argument

- 이전 예제에서는 함수에 매개변수를 전달할 때 일반 변수를 사용했다.
- 함수에 대한 매개변수를 참조로 전달할 수도 있다. 이는 인수 값을 변경해야 할 때 유용할 수 있다.

```
void swapNums(int &x, int &y) {
  int z = x;
  x = y;
  y = z;
}

int main() {
  int firstNum = 10;
  int secondNum = 20;

  cout << "Before swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  // Call the function, which will change the values of firstNum and secondNum
  swapNums(firstNum, secondNum);

  cout << "After swap: " << "\n";
  cout << firstNum << secondNum << "\n";

  return 0;
}
```

C++ FUNCTION

C++ Function Overloading

- 디폴트 인수가 인수의 개수를 달리하여 같은 함수를 호출하는 것이라면, 함수 오버로딩(overloading)은 같은 이름의 함수를 중복하여 정의하는 것을 의미한다.
- C++에서 새롭게 추가된 함수 오버로딩은 여러 함수를 하나의 이름으로 연결해 준다.
- 즉, 함수 오버로딩이란 같은 일을 처리하는 함수를 매개변수의 형식을 조금씩 달리하여, 하나의 이름으로 작성할 수 있게 해주는 것이다.
- 이와 같은 함수 오버로딩은 객체 지향 프로그래밍의 특징 중 바로 다형성(polymorphism)의 구현이다.

C++ FUNCTION

C++ Function signature

- 함수 오버로딩의 핵심은 바로 함수 시그니처(function signature)에 있다.
- 함수 시그니처란 함수의 원형에 명시되는 매개변수 리스트를 가리킨다.
- 만약 두 함수가 매개변수의 개수와 그 타입이 모두 같다면, 이 두 함수의 시그니처는 같다고 할 수 있다.
- 즉, 함수의 오버로딩은 서로 다른 시그니처를 갖는 여러 함수를 같은 이름으로 정의하는 것이라고 할 수 있다.

C++ FUNCTION

C++ Function signature

```
int myFunction(int x)
float myFunction(float x)
double myFunction(double x, double y)
```

```
int add(int x, int y)
{
    return x + y;
}

float add(float x, float y)
{
    return x + y;
}

int main()
{
    cout << "Adding Integer number : " << add(13, 67) << endl;
    cout << "Adding float number : " << add(13.43f, 67.56f) << endl;
    std::cout << "Hello World!\n";

    return 0;
}
```

C++ FUNCTION

C++ Function pointer

- 프로그램에서 정의된 함수는 프로그램이 실행될 때 모두 메인 메모리에 올라간다.
- 이때 함수의 이름은 메모리에 올라간 함수의 시작 주소를 가리키는 포인터 상수(constant pointer)가 된다.
- 이렇게 함수의 시작 주소를 가리키는 포인터 상수를 함수 포인터(function pointer)라고 부른다.
- 함수 포인터의 포인터 타입은 함수의 반환 값과 매개변수에 의해 결정된다.
- 즉 함수의 원형을 알아야만 해당 함수에 맞는 함수 포인터를 만들 수 있다.

포인터 상수(constant pointer)란 포인터 변수가 가리키고 있는 주소 값을 변경할 수 없는 포인터를 의미하며, 상수 포인터(pointer to constant)란 상수를 가리키는 포인터를 의미한다.

C++ FUNCTION

C++ Function pointer

- 함수의 원형(Function Prototype)
- void Func(int, int);

- 함수 포인터
- void (*ptr_func)(int, int);