

# C++\_배열, 함수, 포인터

*Visual Studio 2022*

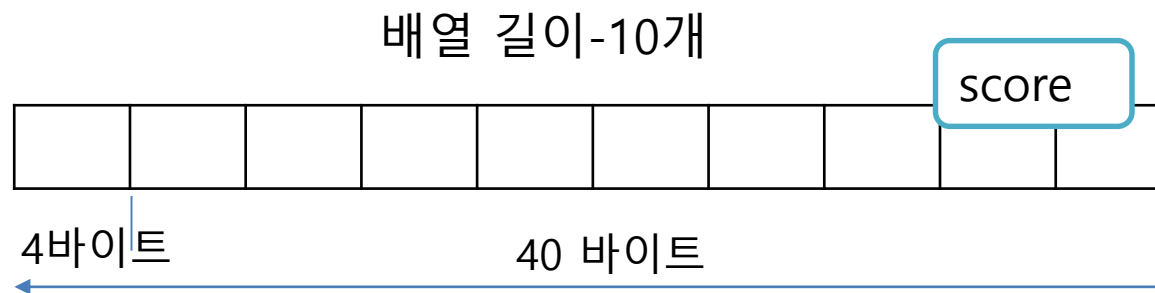
# 배열(Array)

- 배열이란?

여러 개의 연속적인 값을 저장하고자 할 때 사용하는 자료형이다.  
배열 변수는 [ ]안에 설정한 값만큼 메모리를 할당하여 저장한다.

- 배열 변수의 선언과 사용

**int score[10];**



# 배열(Array)

- 문자형 배열 관리 비교

구분	C언어	C++
초기화(생성)	<b>char</b> arr[3][10] = {'a', 'b', 'c'}	<b>string</b> arr[ ] = {'a', 'b', 'c'}
배열의 크기	sizeof(arr) / sizeof(arr[0])	size(arr) 함수

- 범위기반 for문

```
for(자료형 변수이름 : 배열이름){  
    출력 : 변수  
}
```

```
for(string a : arr){  
    printf("%s", a);  
}
```

# 배열(Array)

- 문자형 배열 관리

```
//문자열 배열 관리
//c언어
char season[4][10] = {"봄", "여름", "가을", "겨울"};
//printf("%s ", list[0]);
int len = sizeof(season) / sizeof(season[0]);
for (int i = 0; i < len; i++) {
    printf("%s ", season[i]);
}

//c++
string carts[] = { "라면", "쌀", "생수", "화장지" };

//배열의 크기
cout << "\n배열의 크기(길이): " << size(carts) << endl;

//2번 요소 조회
cout << carts[2] << endl;
```

# 배열(Array)

- 문자형 배열 관리

```
//요소 수정
carts[1] = "빵";

//전체 출력
for (int i = 0; i < size(carts); i++) {
    cout << carts[i] << " ";
}
cout << endl;

//향상된 for문 - for(자료형 변수 : 배열이름){}
for (string cart : carts) {
    cout << cart << " ";
}
```

```
봄 여름 가을 겨울
배열의 크기(길이): 4
생수
라면 빵 생수 화장지
라면 빵 생수 화장지
```

# 배열(Array)

- 정수형 배열 관리

```
/*
| 정수형 배열 선언, 저장 및 출력
*/
int arr[3];

//저장
arr[0] = 1;
arr[1] = 2;
arr[2] = 3;

/*
| 정수형 배열 선언 및 초기화
*/
//int arr[3] = { 1, 2 };
int arr[3] = { 1, 2, 3 };

//배열의 크기 - size()
cout << size(arr) << endl;
```

```
//출력
for (int i = 0; i < size(arr); i++)
{
    cout << arr[i] << " ";
}
cout << "\n===== \n";

//요소 수정 - 1번 인덱스 수정
arr[1] = 5;

for (int i = 0; i < size(arr); i++)
    cout << arr[i] << " ";
```

# 배열(Array)

## ■ 정수형 배열의 연산

```
/*
    성적의 합계와 평균, 최대값, 최소값
*/
int score[4] = { 70, 90, 80, 65 };
int total = 0;    //합계
double avg;       //평균
int max, min;     //최대값, 최소값

//배열의 요소 개수
cout << size(score) << endl;

/*for (int i = 0; i < size(score); i++)
{
    cout << score[i] << " ";
}*/

// 합계 계산
for (int i = 0; i < size(score); i++)
{
    total += score[i];
}
cout << "합계 : " << total << endl;
```

```
// 평균 계산 : 합계 / 개수
avg = total / (double)size(score);
cout << "평균 : " << avg << endl;

//최대값
max = score[0]; //첫째 요소 최대값으로 설정
for (int i = 1; i < size(score); i++)
{
    if (score[i] > max)
    {
        max = score[i];
    }
}
cout << "최대값 : " << max << endl;
```

# 2차원 배열

## ■ 배열의 확장 : 2차원 배열

1. 지도, 게임 등 평면이나 공간을 구현할 때 많이 사용됨.
2. 이차원 배열의 선언과 구조

```
int arr[2][3];
```

3. 선언과 초기화

```
int arr[2][3] = {  
    {1, 2, 3},  
    {4, 5, 6}  
};
```



arr[0][0]	arr[0][1]	arr[0][2]

arr[1][0]   arr[1][1]   arr[1][2]

arr[0][0]	arr[0][1]	arr[0][2]
1	2	3
4	5	6

arr[1][0]   arr[1][1]   arr[1][2]



# 이차원 배열

## ■ 정수형 배열 생성 및 출력

```
/*  
    2차원 배열을 생성하고 1 ~ 6까지 저장하기  
*/  
  
int a[2][3];  
int i, j, k = 0;  
  
//저장  
for (i = 0; i < 2; i++)  
{  
    for (j = 0; j < 3; j++)  
    {  
        a[i][j] = k + 1;  
        k++;  
    }  
}
```

```
//출력  
for (i = 0; i < 2; i++)  
{  
    for (j = 0; j < 3; j++)  
    {  
        cout << a[i][j] << " ";  
    }  
}
```

# 이차원 배열

- 이차원 배열 – 정수형 배열

학생 3명의 2과목 점수

Kim, Lee, Park

이름	수학	영어
Kim	75	80
Lee	85	95
Park	90	100

# 이차원 배열

- 정수형 배열 생성 및 연산

```
// 정수형 2차원 배열 선언 및 초기화
int a[3][2] = {
    {75, 80},
    {85, 95},
    {90, 100}
};

//특정 요소 조회
cout << "a[0][0]=" << a[0][0] << endl;
cout << "a[1][1]=" << a[1][0] << endl;

//전체 조회
for (int x = 0; x < 3; x++)
{
    for (int y = 0; y < 2; y++)
    {
        cout << "a[" << x << "][" << y << "]=" << a[x][y] << ' ';
    }
    cout << '\n';
}
```

# 이차원 배열

- 이차원 배열 – 정수형 배열 생성 및 연산

```
//요소의 수 및 합계
int count = 0;
int total = 0;
for (int x = 0; x < 3; x++)
{
    for (int y = 0; y < 2; y++)
    {
        count++;
        total += a[x][y];
    }
}
cout << "배열의 요소 수: " << count << endl;
cout << "배열의 요소의 총합: " << total << endl;
```

# 함수(function)

## ❖ 함수(Function)란?

- 하나의 기능을 수행하는 일련의 코드이다.(모듈화)
- 함수는 이름이 있고, 반환값과 매개변수가 있다.(함수의 형태)
- 하나의 큰 프로그램을 작은 부분들로 분리하여 코드의 중복을 최소화하고, 코드의 수정이나 유지보수를 쉽게 한다.(함수를 사용하는 이유)
  - 모든 코드를 main(){...} 함수 내에서 만들면 중복 및 수정의 복잡함이 있음

## ❖ 함수의 종류

- 내장 함수 – 수학, 시간, 문자열 함수 등
- 사용자 정의 함수 – 사용자(개발자)가 직접 만들어 사용하는 함수

```
반환자료형 함수이름(매개변수)
{
    구현 코드
}
```

```
int getArea(x, y)
{
    return x * y
}
```

# 함수(function)의 유형

- 반환 자료형이 있는 경우 – **return** 키워드 사용

```
//제곱수 계산 함수
int square(int x)
{
    return x * x;
}

//절대값 계산 함수
int myAbs(int x)
{
    if (x < 0)
        return -x;
    else
        return x;
}
```

# 함수(function)의 유형

- 반환 자료형이 있는 경우 – **return** 키워드 사용

```
//두 수의 합 계산 함수
int add(int x, int y)
{
    return x + y;
}

int main()
{
    //square() 호출
    int value1 = square(4);
    cout << "제곱수: " << value1 << endl;

    //myAbs() 호출
    int value2 = myAbs(-5);
    cout << "절대값: " << value2 << endl;

    //add() 호출
    int value3 = add(10, 20);
    cout << "두 수의 합: " << value3 << endl;

    return 0;
}
```

# 함수(function) 예제

- 배열에서 최대값, 최소값 구하기

```
//최대값 계산 함수
int findMax(int a[], int size)
{
    int max = a[0]; //최대값 설정
    for (int i = 1; i < size; i++)
    {
        if (a[i] > max)    //요소값이 최대값보다 크면
            max = a[i];    //최대값을 요소값으로 지정
    }

    return max;
}
```



# 함수(function) 예제

- 배열에서 최대값, 최소값 구하기

```
//최소값 계산 함수 정의
//코드 작성

int main()
{
    //정수형 배열 생성
    int arr[] = { 21, 35, 71, 2, 97, 66 };

    //최대값 출력
    int maxVal = findMax(arr, size(arr));
    cout << "최대값: " << maxVal << endl;

    //최소값 출력 코드 작성

    return 0;
}
```

# 변수의 메모리 영역

- **코드 영역** : 프로그램의 **실행 코드** 또는 **함수**들이 저장되는 영역
- **스택 영역** : **매개 변수 및 중괄호(블록)** 내부에 **정의된 변수**들이 저장되는 영역
- **데이터 영역** : **전역 변수**와 **정적 변수**들이 저장되는 영역
- **힙 영역** : **동적으로 메모리 할당하는 변수**들이 저장되는 영역



코드 영역  
(실행 코드, 함수)



스택 영역  
(지역 변수, 매개 변수)



데이터 영역  
(전역 변수, 정적 변수)



힙 영역  
(동적 메모리 할당)

# 변수의 적용 범위 - 지역변수

- 전역 변수와 지역 변수의 차이

```
int x = 1; //전역 변수

int add10(){
    //int x = 1; // 지역변수
    x = x + 10;
    return x;
}
```

```
int main()
{
    //add10() 호출
    int value = add10();

    cout << "value = " << value << endl;
    cout << "x = " << x << endl;

    return 0;
}
```

# 변수의 적용 범위 – 정적 변수

- 정적 변수(static variable)
  - 선언된 함수가 종료하더라도 그 값을 계속 유지하는 변수
  - **static** 키워드를 붙임

정적 변수의 메모리 생성 시점 - 중괄호 내에서 초기화될때  
정적 변수의 메모리 소멸 시점: - 프로그램이 종료되었을 때

```
//지역 변수와 정적 변수의 차이
void click()
{
    int x = 10; //지역 변수
    static int y = 10; //정적 변수

    x++;
    y++;

    cout << "x=" << x << ", y=" << y << endl;
}
```

```
int main()
{
    //click() 여러 번 호출
    click();
    click();
    click();
    click();

    return 0;
}
```

```
x=11, y=11
x=11, y=12
x=11, y=13
x=11, y=14
```

# 참조에 의한 호출

- 참조자란?

참조형을 레퍼런스라고도 하는데, 기존의 메모리공간에 별명(alias)을 붙이는 방법을 말한다. (포인터와 유사함)

하나의 변수에 여러 개의 이름을 붙이는 것을 말한다.

**자료형& 참조변수명** (&는 참조 연산자)으로 사용한다.

- 참조자의 활용

① 함수의 매개변수로 사용하기 위해( ★중요 ★)

② 함수의 반환형으로 사용하기 위해

```
int n = 1;
int& x = n; //변수 n의 복사본(별칭)

cout << "x = " << x << endl;

x = 3;
cout << "x = " << x << endl;
```

# 참조에 의한 호출(call-by-reference)

```
void swapVal(int a, int b);
void swapRef(int& a, int& b);
void swapRef2(int* a, int* b);
int main()
{
    //참조(&) - 미리 정의된 변수의 실제 이름 대신 사용하는 이름(별칭-alias)
    int x = 10, y = 20;

    cout << "값에 의한 호출\n";
    swapVal(x, y);
    cout << "x = " << x << ", y = " << y << endl;

    cout << "참조에 의한 호출\n";
    swapRef(x, y);
    cout << "x = " << x << ", y = " << y << endl;

    cout << "포인터에 의한 호출\n";
    swapRef2(&x, &y);
    cout << "x = " << x << ", y = " << y << endl;
    return 0;
}
```

# 참조에 의한 호출(call-by-reference)

```
void swapVal(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

void swapRef(int& a, int& b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

```
void swapRef2(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

값에 의한 호출  
x = 10, y = 20  
참조에 의한 호출  
x = 20, y = 10

# 인라인 함수(inline Function)

- 인라인 함수란?

**inline** 함수란 함수 호출 오버헤드로 인한 프로그램의 실행 속도 저하를 막기 위한 기능으로 인라인 함수의 코드를 그대로 삽입하여 **함수 호출**이 일어나지 않게 한다. (오버헤드란 어떤 명령어를 처리하는데 소비되는 간접적, 추가적인 컴퓨터 자원을 의미한다.)

- 사용 예시

```
inline add(x, y) {return x + y}
```



# 인라인 함수(inline Function)

```
//인라인 함수 정의
inline int square(int x) { return x * x; }
inline int odd(int x) { return x % 2; }
int main()
{
    //제곱수 계산하기
    int val = square(6);
    cout << "제곱수: " << val << endl;

    //홀수의 합 계산
    int sum = 0;
    for (int i = 0; i <= 10; i++) {
        if (odd(i)) { //if(1){}, 1=true
            sum += i; //1+3+5+7+9
        }
    }
    cout << "합계: " << sum << endl;

    return 0;
}
```

# 표준 라이브러리 함수(function)

## ❖ 내장 함수 – 표준 라이브러리 함수

### Standard library header <ctime>

This header was originally in the C standard library as `<time.h>` .

This header is part of the C-style date and time library.

#### Macro constants

<code>CLOCKS_PER_SEC</code>	number of processor clock ticks per second (macro constant)
<code>NULL</code>	implementation-defined null pointer constant (macro constant)

#### Types

<code>clock_t</code>	process running time (typedef)
<code>size_t</code>	unsigned integer type returned by the <code>sizeof</code> operator (typedef)
<code>time_t</code>	time since epoch type (typedef)
<code>tm</code>	calendar time type (class)

# 수학 함수(function)

- ✓ 수학 관련 함수 – <cmath>를 include 해야 함

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

//수학 관련 내장 함수 사용하기
int main()
{
    //반올림
    cout << "2.54 반올림: " << round(2.54) << endl;
    cout << "2.45 반올림: " << round(2.45) << endl;

    //내림
    cout << "3.3 내림: " << floor(3.3) << endl;

    //절대값
    cout << "8 절대값: " << abs(8) << endl;
    cout << "-8 절대값: " << abs(-8) << endl;

    //거듭제곱
    cout << "2의 4제곱: " << pow(2, 4) << endl;

    //제곱근
    cout << "16의 제곱근: " << sqrt(16) << endl;

    return 0;
}
```

# 시간 함수(function)

- ✓ 시간 관련 함수 – <ctime>을 include 함

```
#include <iostream>
#include <ctime>
#include <thread> // 스레드 sleep을 위한 라이브러리
using namespace std;

int main()
{
    // 현재 시간을 초 단위로 가져오기
    time_t now = time(nullptr);

    // 초, 일, 년으로 측정
    cout << "1970년 1월 1일(0시 0분 0초) 이후: " << now << "초" << endl;
    cout << "1970년 1월 1일(0시 0분 0초) 이후: " <<
        now / (24 * 60 * 60) << "일" << endl;
    cout << "1970년 1월 1일(0시 0분 0초) 이후: " <<
        now / (365 * 24 * 60 * 60) << "년" << endl;
```

# 시간 함수(function)

- ✓ 시간 관련 함수 – <ctime>을 include 함

```
// 수행 시간 측정
time_t start, end;

time(&start); // 시작 시간

// 0.5초 간격으로 1~10 출력
for (int i = 1; i <= 10; i++)
{
    cout << i << endl;
    this_thread::sleep_for(chrono::milliseconds(500));
}

time(&end); // 종료 시간
cout << "수행시간: " << (end - start) << "초" << endl;

return 0;
}
```

# 시간 함수(function)

- ✓ 수행 시간 측정하기 - 소수로 출력

```
time_t start, end;
double elapsedTime;

//time(&start); // 시작 시간
start = clock();

// 0.5초 간격으로 1~10 출력
for (int i = 1; i <= 10; i++)
{
    cout << i << endl;
    this_thread::sleep_for(chrono::milliseconds(500));
}

//time(&end); // 종료 시간
end = clock();
elapsedTime = (double)(end - start) / CLOCKS_PER_SEC;
cout << "수행시간: " << elapsedTime << "초" << endl;
```

# rand() 함수

✓ 동전, 주사위 추출하기

```
#include <iostream>
#include <cstdlib> // srand(), rand()
#include <ctime>    // time()
using namespace std;

int main()
{
    // srand(10); // seed 값 설정(고정)
    srand(time(NULL)); // seed 값 설정(변경)

    int rndVal = rand();
    cout << rndVal << endl;
    cout << "=====" << endl;

    // 동전(2가지 경우)
    int coin = rand() % 2;
    cout << coin << endl;
```

```
// 0-앞면, 1-뒷면
if (coin % 2 == 0)
{
    cout << "앞면" << endl;
}
else
{
    cout << "뒷면" << endl;
}

//주사위 눈
/*int dice = rand() % 6 + 1;
cout << dice << endl;*/

//주사위 10번 던지기
for (int i = 1; i <= 10; i++)
{
    int dice = rand() % 6 + 1;
    cout << dice << endl;
}
cout << "=====\n";
```

# rand() 함수

## ✓ 문자열 추출하기

```
//문자 추출
string seasons[] = {"봄", "여름", "가을", "겨울"};
//cout << seasons[1] << endl;
cout << size(seasons) << endl;

int idx = rand() % size(seasons); //배열 인덱스
cout << seasons[idx] << endl;

return 0;
}
```



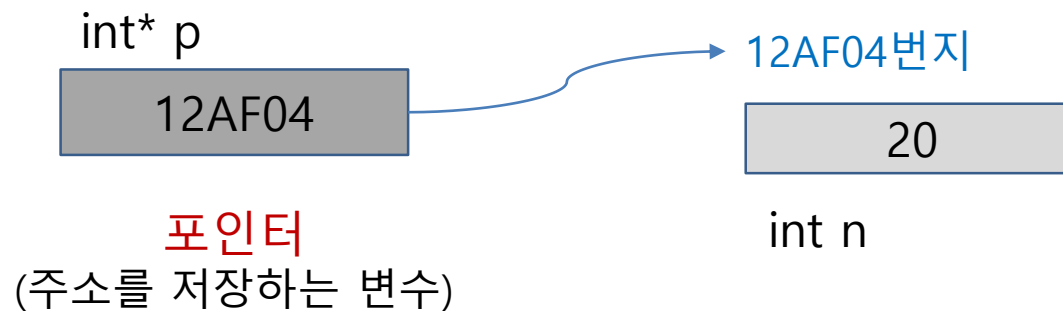
# 포인터(Pointer)

## ➤ 포인터란?

모든 메모리는 주소(address)를 갖는다. 이러한 **메모리 주소를 저장**하기 위해 사용되는 변수를 포인터 변수라 한다.

포인터 변수를 선언할 때에는 데이터 유형과 함께 '\*' 기호를 써서 나타낸다.

(예) 택배 주소만 있으면 집을 찾을 수 있다.



# 포인터(Pointer)

## ➤ 포인터 변수의 선언과 초기화

```
//정수형 변수 선언
int n = 10;

cout << n << endl;
cout << &n << endl;
cout << sizeof(n) << "byte" << endl;

//정수형 포인터 선언
int* pn;
pn = &n;

cout << pn << endl;
cout << &pn << endl;
cout << *pn << endl; //역참조
cout << sizeof(pn) << "byte" << endl;

//역참조 연산
*pn = *pn + 10;
cout << *pn << endl;
```

# 배열과 포인터(Pointer)

## ➤ 정수형 배열과 포인터

```
//정수형 배열 선언
int a[4] = { 10, 20, 30, 40 };

cout << a[0] << endl;
cout << &a[0] << endl;
cout << a << endl;    //배열 이름이 시작 주소이다.

//정수형 포인터 배열
int* pa;
pa = a;    //pa = &a[0]

cout << pa << endl;
cout << *pa << endl;    /*(pa + 0)
cout << *(pa + 1) << endl;

//전체 출력
for (int i = 0; i < size(a); i++) {
    cout << *(pa + i) << " ";
}
```

# 값 & 참조에 의한 호출

- Call-by-reference(참조에 의한 호출)

```
void callByVal(int x)
{
    x++; //1 증가
}

//포인터를 매개변수로 사용
void callByRef(int* pn)
{
    *pn = *pn + 1; //역참조로 1증가
}
```

# 값 & 참조에 의한 호출

## ➤ Call-by-reference(참조에 의한 호출)

```
int main()
{
    int n = 10;

    cout << "=== 값에 의한 호출 ===\n";
    callByVal(n);
    cout << "n = " << n << endl;

    cout << "=== 주소에 의한 호출 ===\n";
    callByRef(&n);
    cout << "n = " << n << endl;

    return 0;
}
```

# 동적 메모리 할당

- 포인터와 동적 메모리 할당

- 정적 메모리 할당 : `int arr[10]`

- 동적 메모리 할당 :  
`int* p = new int,;`  
`int* pa = new int[10]`

주소록 프로그램에 회원  
몇 명을 등록해야할지 미  
정일 때...

- 동적 메모리 할당

- 프로그램 실행 중에 필요한 메모리의 크기를 결정
  - 시스템은 힙(heap)이라는 공간을 관리하고 있는데, 프로그램에서 요청하는 공간을 할당하여 시작 주소를 알려준다.
  - 할당된 시작 주소는 반드시 어딘가에 저장되어야 하고 이때 포인터가 사용됨
  - 할당시 **new** , 해제시 **delete** 사용

# 동적 메모리 할당과 해제

- 정수형 포인터 동적 할당

```
int* p;  
p = new int; //동적 포인터 생성  
if (p == NULL) {  
    cout << "메모리를 할당할 수 없습니다\n";  
    return 0;  
}  
  
*p = 5;  
cout << "*p=" << *p << endl;  
  
delete p; //메모리 반납
```

# 동적 메모리 할당과 해제

- 정수형 배열 동적 할당

```
int* pa;  
pa = new int[10]; //동적 배열 생성  
if (pa == NULL) {  
    cout << "메모리를 할당할 수 없습니다\n";  
    return 0;  
}  
  
for (int i = 0; i < 10; i++) {  
    *(pa + i) = i;  
}  
  
for (int i = 0; i < 10; i++) {  
    cout << "(*pa + " << i << ")----->" << *(pa + i) << endl;  
}  
  
delete[] pa; //메모리 반납
```

```
(*pa + 0)----->0  
(*pa + 1)----->1  
(*pa + 2)----->2  
(*pa + 3)----->3  
(*pa + 4)----->4  
(*pa + 5)----->5  
(*pa + 6)----->6  
(*pa + 7)----->7  
(*pa + 8)----->8  
(*pa + 9)----->9
```

`delete[ ]` 포인터 // 배열로 할당된 메모리 해제



# 동적 메모리 할당과 해제

- 동적 포인터 배열의 연산

```
//동적 포인터 배열 연산
int n;
int sum = 0;
double avg;

cout << "*** 점수의 평균 계산 프로그램 ***\n";
cout << "입력할 정수의 개수: ";
cin >> n; //배열의 크기
int* pn = new int[n];

//점수 입력
for (int i = 0; i < n; i++) {
    cout << i + 1 << "번째 점수 : ";
    cin >> pn[i];
}
```

# 동적 메모리 할당과 해제

- 동적 포인터 배열의 연산

```
//합계 계산
for (int i = 0; i < n; i++) {
    // cout << pn[i] << endl;
    sum += pn[i];
}
//평균 계산
avg = (double)sum / n;

cout << fixed;          //소수점 고정
cout.precision(2);      //소수 2째자리
cout << "평균 : " << avg << endl;

delete[] pn; //메모리 반납

return 0;
```

# 실습 문제 - 함수

main() 함수를 분석하여 거듭 제곱을 계산하는 함수를 정의하세요.

[파일이름: myPow.cpp]

```
int main()
{
    int val = myPow(2, 4);

    cout << "거듭제곱 결과값: " << val << endl;

    return 0;
}
```