

# C++\_함수, 포인터

*Visual Studio 2022*

# 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)

## ❖ 사용자 정의 함수

- 사용자(개발자)가 직접 만들어 사용하는 함수

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

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

# 사용자 정의 함수(function)

## ❖ 함수의 정의와 호출

### 1. 반환 자료형이 없는 경우(void 형)

```
#include <iostream>
#include <string> //string 자료형 사용
using namespace std;

//함수 정의
void sayHello()
{
    cout << "안녕하세요~" << endl;
}
```

# 함수(function)의 유형

## 1. 반환 자료형이 없는 경우(void 형)

```
//매개변수가 있는 함수
void sayHello2(string name)
{
    cout << name << "님, 안녕하세요~ " << endl;
}

int main()
{
    sayHello(); //함수 호출

    sayHello2("신유빈");
    sayHello2("한강");

    return 0;
}
```

# 함수(function)의 유형

## 2. 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

## 2. 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 종괄호(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 – 전역 변수

- 전역 변수(global variable)
  - 전체 소스 코드를 범위로 적용되는 변수
  - 소스 파일 내의 어디서든지 사용 가능한 변수

전역 변수의 메모리 생성 시점 - 프로그램이 시작되었을 때  
전역 변수의 메모리 소멸 시점: - 프로그램이 종료되었을 때

- 지역 변수(local variable)
  - 하나의 코드 블록에서만 정의되어 사용되는 변수
  - 함수 또는 제어문의 중괄호{ } 내부에서 사용

지역 변수의 메모리 생성 시점 - 블록(중괄호) 내에서 초기화할 때  
지역 변수의 메모리 소멸 시점: - 블록(중괄호)을 벗어났을 때

# 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();

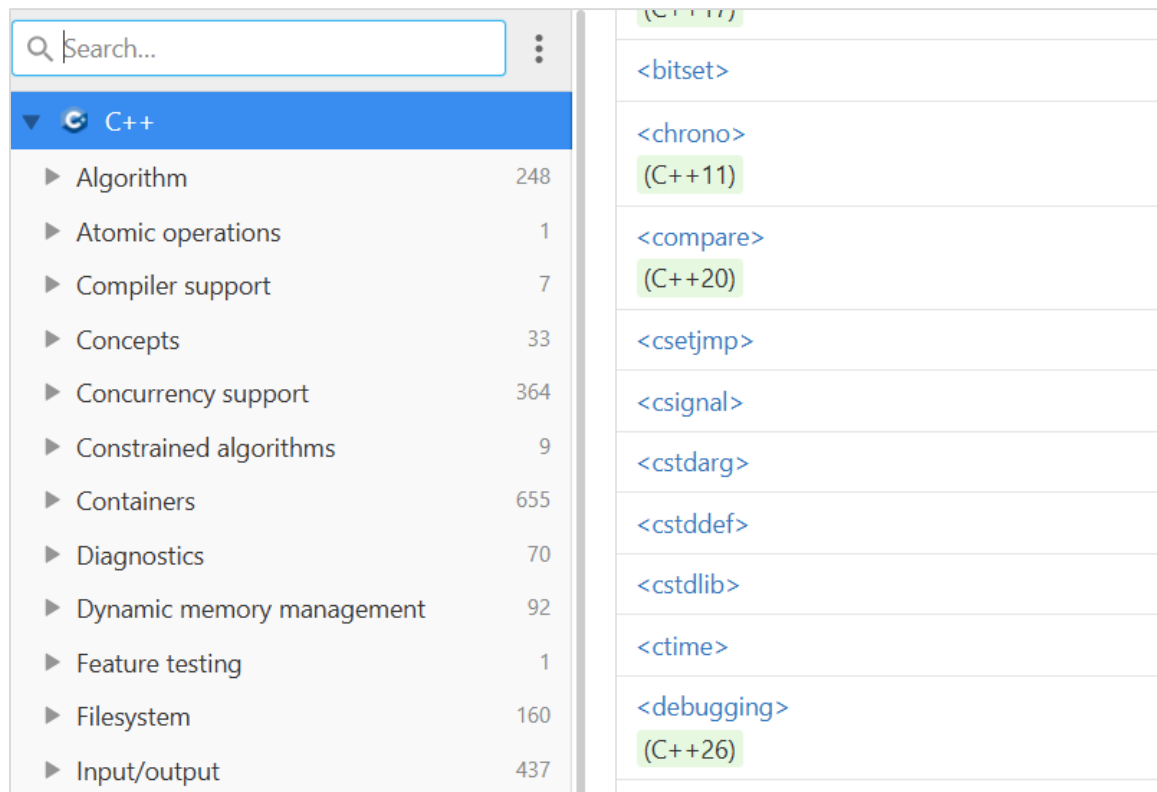
    return 0;
}
```

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

# 표준 라이브러리 함수(function)

## ❖ 내장 함수 – 표준 라이브러리 함수

C언어 Devdocs 검색 : <https://devdocs.io/c++>



# 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;
```

# 날짜 표시

## ✓ 현재 날짜와 시간 표시하기

```
#define _CRT_SECURE_NO_WARNINGS //localtime()
#include <iostream>
#include <ctime>
using namespace std;

int main() {
    // 현재 시간을 가져오기 위한 time_t 변수 선언
    time_t ct;
    struct tm* now; // 현재 날짜와 시간(tm 구조체 포인터 객체)

    // 현재 시간 가져오기
    time(&ct);
    now = localtime(&ct); // localtime 함수로 포매팅

    // 날짜 및 시간 출력
    cout << "현재 년도: " << now->tm_year + 1900 << endl;
    cout << "현재 월: " << now->tm_mon + 1 << endl;
    cout << "현재 일: " << now->tm_mday << endl;
    cout << "현재 날짜: " << now->tm_year + 1900 << ". " <<
        now->tm_mon + 1 << ". " << now->tm_mday << "." << endl;
```

# 날짜 표시

## ✓ 현재 날짜와 시간 표시하기

```
cout << "현재 시: " << now->tm_hour << endl;
cout << "현재 분: " << now->tm_min << endl;
cout << "현재 초: " << now->tm_sec << endl;
cout << "현재 시간: " << now->tm_hour << " : " <<
    now->tm_min << " : " << now->tm_sec << "." << endl;

// 현재 요일
cout << "현재 요일: " << now->tm_wday << endl; // 0-일, 1-월, 2-화...

// 현재 요일을 출력(조건문 사용)
switch (now->tm_wday) {
case 0: cout << "오늘은 일요일입니다." << endl; break;
case 1: cout << "오늘은 월요일입니다." << endl; break;
case 2: cout << "오늘은 화요일입니다." << endl; break;
case 3: cout << "오늘은 수요일입니다." << endl; break;
case 4: cout << "오늘은 목요일입니다." << endl; break;
case 5: cout << "오늘은 금요일입니다." << endl; break;
case 6: cout << "오늘은 토요일입니다." << endl; break;
default: cout << "없는 요일입니다." << endl; break;
}

return 0;
}
```

# rand() 함수

- rand() 함수 – 난수(무작위)를 생성해 주는 함수

$\text{rand()} \% (\text{경우의 수}) + 1$

- rand() 함수를 사용하려면 srand() 함수가 반드시 먼저 사용되어야 한다.
- seed값을 설정하면 한번 만 난수로 되므로, 계속 무작위수가 나오려면 seed값에 시간의 흐름을 넣어준다.

**srand(6) -> srand(time(NULL))**

- srand(), rand()는 <cstdlib>에 정의 되어 있다.
- 동전의 양면, 가위/바위/보, 주사위 눈의 수등 게임이나 통계 확률 등에서 많이 사용된다.

# 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;
}
```



# 영어 타이핑 게임

## ■ 게임 방법

- 무작위로 선택된 단어가 출력된다.
- 사용자가 단어를 따라 입력한다.
- 단어가 일치하면 "통과!", 그렇지 않으면 "오타! 다시 도전" 출력한다.
- 총 10개의 단어를 맞추면 게임이 종료되고, 게임 소요시간을 측정한다.

영어 타자게임, 준비되면 엔터>

문제 1  
galaxy  
galaxy  
통과!

문제 2  
moon  
moon  
통과!

문제 3  
river  
river  
통과!

문제 4  
galaxy  
ga  
오타! 다시 도전!

문제 7  
river  
river  
통과!

문제 8  
galaxy  
galaxy  
통과!

문제 9  
sea  
sea  
통과!

문제 10  
animal  
animal  
통과!  
게임 소요시간 : 23.342초

# 영어 타이핑 게임

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

int main()
{
    string words[] = { "sun", "moon", "earth", "galaxy", "sky", "sea",
                       "mountain", "river", "flower", "tree", "human", "animal"};
    string question; //문제
    string answer;   //사용자 입력
    clock_t start, end;
    double elapsedTime; //게임 소요 시간
    int n = 1; //문제 번호

    srand(time(NULL)); //랜덤 시드 설정
    cout << "영어 타자게임, 준비되면 엔터>";
    getchar();

    start = clock(); //시작 시간
```

# 영어 타이핑 게임

```
while (n <= 10)
{
    cout << "\n문제 " << n << endl;
    int idx = rand() % size(words); //난수(인덱스)
    question = words[idx];
    cout << question << endl; //문제 출제
    cin >> answer;
    if (answer.compare(question) == 0)
    {
        cout << "통과!\n";
        n++;
    }
    else
    {
        cout << "오타! 다시 도전!\n";
    }
}
end = clock(); //종료 시간
elapsedTime = (double)(end - start) / CLOCKS_PER_SEC;
cout << "게임 소요시간: " << elapsedTime << "초" << endl;
return 0;
}
```

# 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;
}
```

# 인라인 함수 & 매크로 함수

- 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}
```

- 매크로 함수란?

**매크로 함수**는 #define에 인수로 함수의 정의를 전달함으로써 함수 처럼 동작한다. 컴파일러 전에 실행되는 전처리기로써 프로그램의 실행 속도를 높여준다.

- 사용 예시

```
#define ADD(x, y) x + y
```

# 인라인 함수 & 매크로 함수

## ■ 예제

```
#include <iostream>
#define M_PI 3.1415          //매크로 상수
#define SQUARE(x) x * x     //매크로 함수
using namespace std;

//inline int square(int x) { return x * x; }
inline int odd(int x) { return (x % 2); } //인라인 함수
/*int odd(int x) { //일반 함수
    return (x % 2);
}*/
int main()
{
    int val = SQUARE(6);
    //int val = square(6);

    cout << "제곱수: " << val << endl;
```



# 인라인 함수 & 매크로 함수

## ■ 예제

```
//1부터 10까지 홀수의 합
int sum = 0;
for (int i = 1; i <= 10; i++)
{
    if (odd(i)) //if(true){}, i % 2 == 1, true=1
    {
        sum += i;
    }
}
cout << "합계: " << sum << endl;

return 0;
}
```

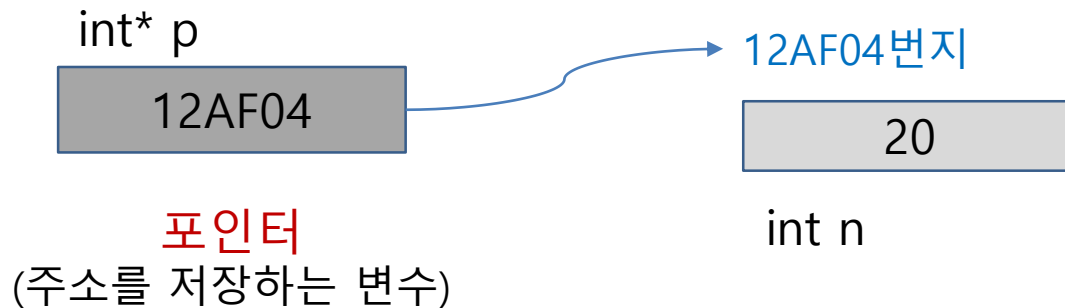
# 포인터(Pointer)

## ➤ 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 선언

### ▪ 선언

**자료형\*** 포인터 이름

```
char* c;    // char형 포인터  
int* n;     // int형 포인터  
double* d;  // double형 포인터
```

- 포인터의 크기 – 모든 자료형에서 8바이트로 동일하다. 포인터에 저장할 수 있는 값은 메모리 번지 뿐이며, 따라서 모든 포인터 변수는 동일한 크기의 메모리가 필요함.

sizeof(포인터)

# 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;

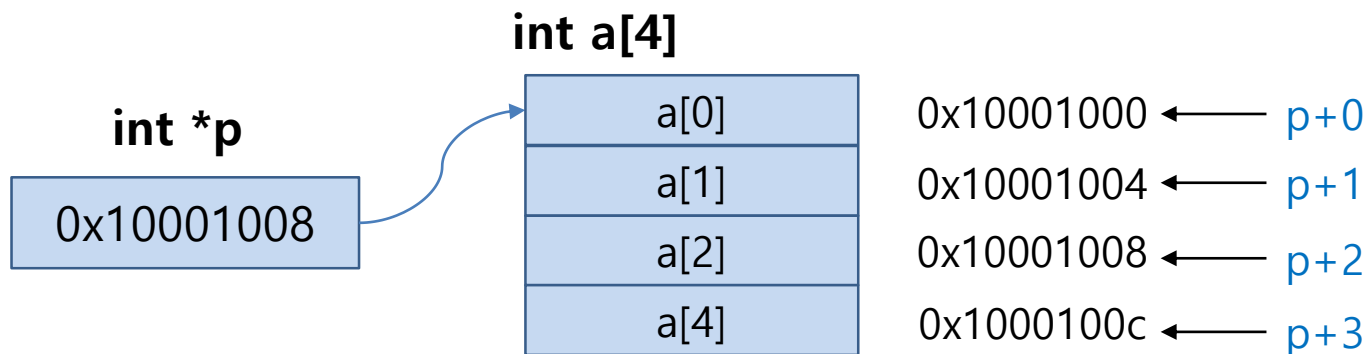
//역참조 연산
*pn = *pn + 10;
cout << *pn << endl;
```

# 포인터 배열

## ➤ 포인터 배열

- 배열은 데이터를 연속적으로 메모리에 저장한다.
- 포인터 역시 메모리에 데이터를 저장하거나 저장된 데이터들을 읽어 올 수 있다.

```
int a[4];           //배열 선언
int* p;             // 포인터 p선언
p = &a[0]           // p에 배열의 첫 번째 항목 주소 복사
```



# 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) << " ";
}
```

# 함수에서 포인터의 전달

- Call-by-value(값에 의한 호출) vs Call-by-reference(참조에 의한 호출)

값을 매개체로 함수호출

CallByVal(int x)



CallByVal(n)

주소를 매개체로 함수 호출

CallByRef(int\* pn)



CallbyRef(&n)

# 값 & 참조에 의한 호출

- Call-by-value(값에 의한 호출) vs 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-value(값에 의한 호출) vs 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;
}
```

# 값 & 참조에 의한 호출

➤ Call-By-Value(값에 의한 호출) vs Call-By-Reference(참조에 의한 호출)

11	$x = x + 1$
10	호출후 n
10	호출전 n

매개변수  $x$ 는 호출된 후 11을 반환하고 소멸됨,  
main()의 지역변수  $n$ 은 그대로 10을 유지함

	&n	$*pn = *pn + 1$
	11	호출후 n
	&n	pn
	10	호출전 n

매개 포인터  $p$ 는 주소에 저장된 값에 접근하고 역참조 계산을으로  $n$ 은 11이 됨.  
호출된 후  $pn$ 의 메모리 공간은 소멸됨.

# 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]
```

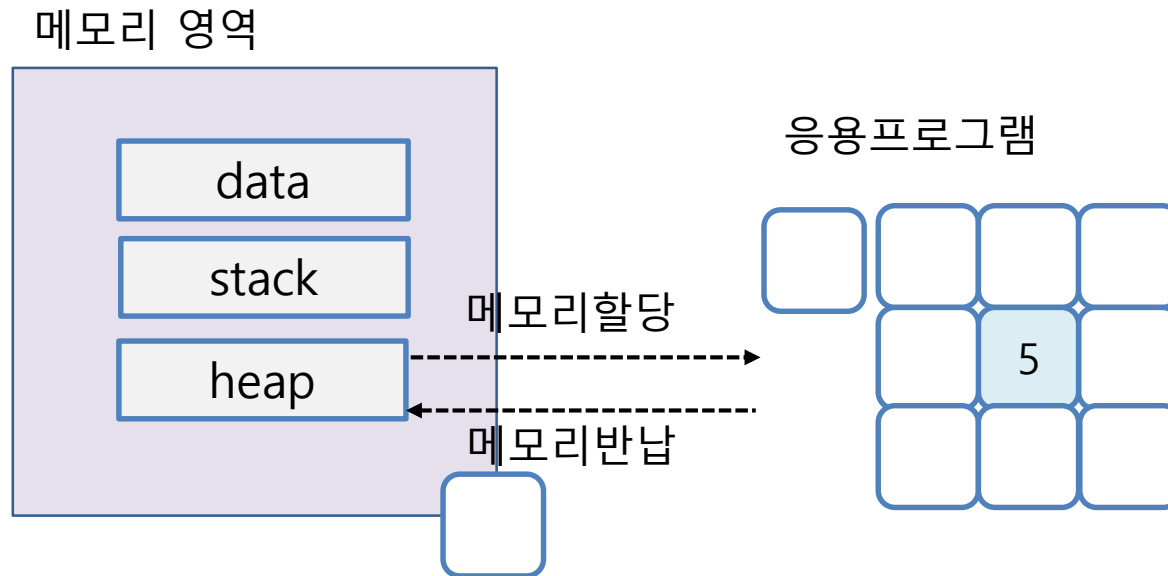
주소록 프로그램에 회원  
몇 명을 등록해야할지 미  
정일 때...

## ■ 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

- 동적 메모리 할당과 해제



# 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 1 - 함수

함수를 정의하여 구구단 7단을 출력하는 프로그램을 작성하세요.

[파일이름: Gugudan.cpp]

👉 실행 결과

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
```