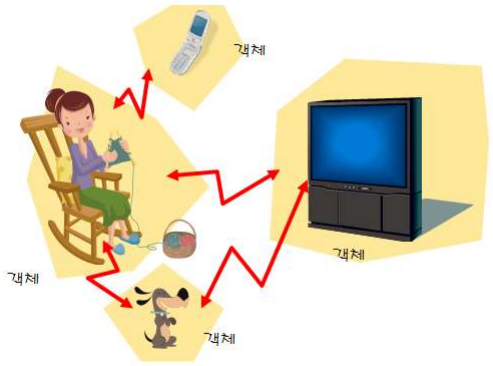
C++ Espresso

제3장 배열과 포인터



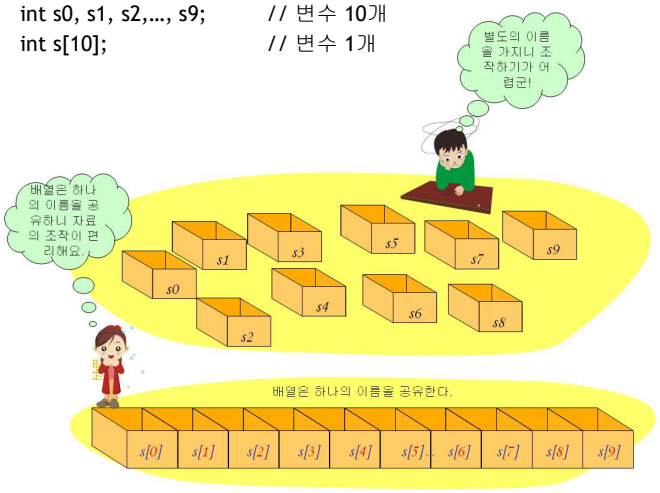
이번 장에서 학습할 내용

- 배열의 개념
- 배열의 선언과 초기화
- 포인터의 개념
- 참조에 의한 호출
- 참조자의 개념
- 문자열의 개념

배열이란?

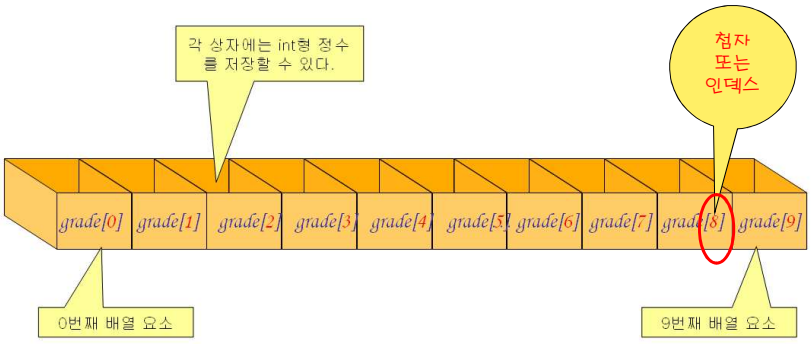
- 배열(array): 같은 종류의 데이터들이 순차적으로 저장되어 있는 자료 구조

```
int s0, s1, s2,…, s9;        // 변수 10개
int s[10];                   // 변수 1개
```

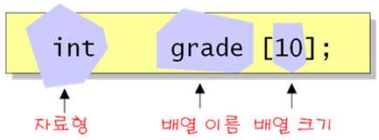


배열 원소와 인덱스

- *인덱스(index):* 배열 원소의 번호, 0 부터 시작
  - int grade[10] : 인덱스 0~9



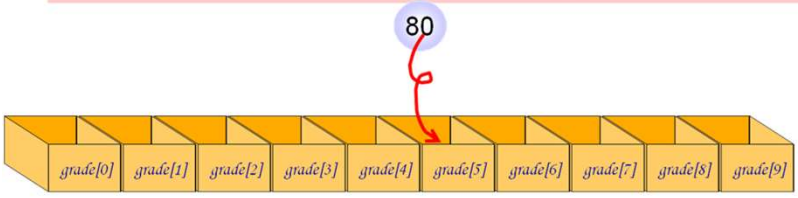
배열의 선언



- 자료형: 배열 원소들이 int형라는 것을 의미
- 배열 이름: 배열을 사용할 때 사용하는 이름이 grade
- 배열 크기: 배열 원소의 개수가 10개
- 인덱스(배열 번호)는 항상 0부터 시작한다.

```
int  score[60];          // 60개의 int형 값을 가지는 배열 grade
float  cost[12];         // 12개의 float형 값을 가지는 배열 cost
char  name[50];          // 50개의 char형 값을 가지는 배열 name
char  src[10], dst[10];  // 2개의 문자형 배열을 동시에 선언
int   index, days[7];    // 일반 변수와 배열을 동시에 선언
```

배열 원소 접근



// 배열 모두 0으로 초기화 가정
int grade[10], index[5];

grade[5] = 80
인덱스

0	0
1	0
2	0
3	0 → 100 → 8
4	
5	0 → 80 →100
6	
7	
8	100
9	

0	0
1	0
2	0
3	0 → 8
4	0

```
grade[5] = 80;
grade[1] =  grade[0];
i=3;
grade[i] = 100;
grade[i+2] = 100;        // 수식이 인덱스가 된다.
index[3] = 8;
grade[index[3]] = 100;  // grade[8] =100 과 동일
```

예제

```
#include <iostream>  // 출력 위한 것
using namespace std;
int main(void)
{
    const int STUDENTS=5;
    int grade[STUDENTS];
    int sum = 0;
    int i, average;
    for(i = 0; i < STUDENTS; i++)          {
            cout << "학생들의 성적을 입력하시오: ";
            cin >> grade[i];
    }
    for(i = 0; i < STUDENTS; i++)
            sum += grade[i];  // sum = sum + grade[i];

    average = sum / STUDENTS; // 150/5=30
    cout << "성적 평균= " << average << endl;
    return 0;
}
```

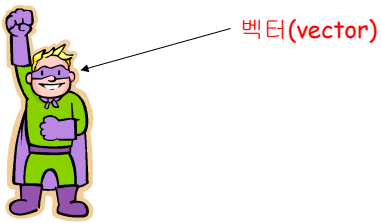
학생들의 성적을 입력하시오: 10
학생들의 성적을 입력하시오: 20
학생들의 성적을 입력하시오: 30
학생들의 성적을 입력하시오: 40
학생들의 성적을 입력하시오: 50
성적 평균 = 30

0	10
1	20
2	30
3	40
4	50

sum = 0+10+20+30+40+50

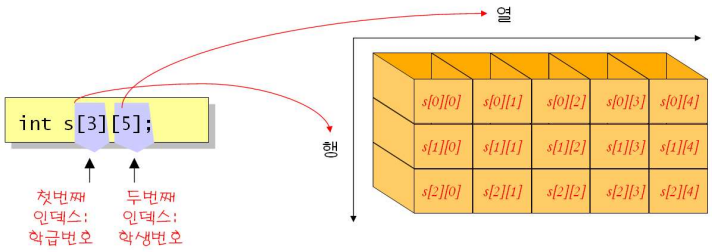
동적 배열

- C++에는 더 좋은 배열이 존재한다.
- STL 라이브러리로 제공(14장~15장)



2차원 배열

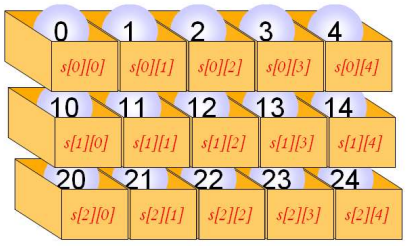
```
int s[10];      // 1차원 배열   s[x]
int s[3][10];   // 2차원 배열   s[y][x]
int s[5][3][10]; // 3차원 배열
```



2차원 배열의 초기화

```
int s[ ][5] = {          // 크기 지정 안하면 초기값 개수 이용 자동 지정
  { 0,  1,  2,  3,  4 }, // 첫 번째 행의 원소들의 초기값
  { 10, 11, 12, 13, 14 }, // 두 번째 행의 원소들의 초기값
  { 20, 21, 22, 23, 24 }, // 세 번째 행의 원소들의 초기값
};
```

int s[3][5] 로 하는 것이 좋음



다차원 배열을 이용한 행렬의 표현

```
#include <iostream>
using namespace std;
const int ROWS=3;
const int COLS=3;

int main()
{
	int A[ROWS][COLS] = { { 2,3,0 }, { 8,9,1 }, { 7,0,5 } };
	int B[ROWS][COLS] = { { 1,0,0 }, { 1,0,0 }, { 1,0,0 } };
	int C[ROWS][COLS];
	int r, c;
	for(r = 0;r < ROWS; r++)		// 아래와 같이 { , } 사용하는것이 좋음
		for(c = 0;c < COLS; c++)
			C[r][c] = A[r][c] + B[r][c];

	for(r = 0;r < ROWS; r++)
	{
		for(c = 0;c < COLS; c++)
			cout << C[r][c] << " ";
		cout << endl;
	}
	return 0;
}
```

```
3 3 0
9 9 1
8 0 5
```

2	3	0
8	9	1
7	0	5

1	0	0
1	0	0
1	0	0

포인터

- 포인터: 변수의 주소를 저장하는 변수

```
int i = 10;		// 정수형 변수 i 선언		→ 값을 저장하는 변수
int *p = &i;		// 변수 i의 주소가 포인터 p로 대입	→ 주소를 저장하는 변수
```

- 주소 연산자 &
  - &i → 변수 i 의 저장공간 주소를 반환
- 변수 i 는 값을 저장하는 변수
- 변수 p 는 주소를 저장하는 변수 → 포인터 변수

	주소	값
i	1000	10
	1001	
p	1002	1000
	1003	

간접 참조 연산자

- 간접 참조 연산자 * : 포인터가 가리키는 값을 가져오는 연산자

```
int i = 10;
int *p = &i;            // 선언하며 초기화 필요

cout << *p;             // 10 이 출력된다. → p 가리키는 곳의 값(내용)

*p = 20;
cout << *p;             // 20 이 출력된다.
```

- 교재의 94쪽 예제(포인터에 직접 주소 저장)는 불필요
- 포인터 이전 내용은 교재 내용 필요
- 포인터 이후 내용은 유인물만 필요

	주소	값
i	1000	10 → 20
	1001	
p	1002	1000
	1003	

포인터 사용시 주의할 점

- **포인터 타입**

```
int x=100;
float y = 2.5;

int *p=&x ;             // p 가리키는 공간에 저장되는 값은 int
float *q = &y;          // q 가리키는 공간에 저장되는 값은 float
```

- **포인터 타입과 포인터가 가리키는 변수의 데이터 타입 동일해야 함.**

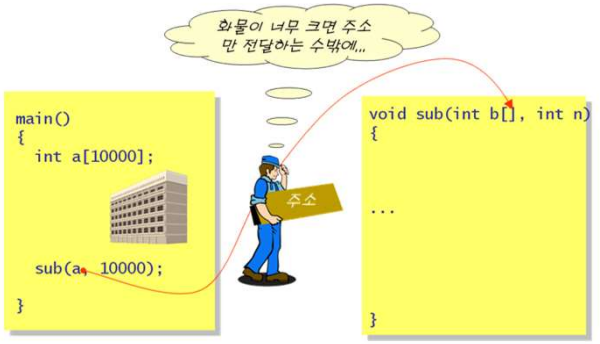
```
int i = 10;
float *pd = &i;         // Error → float 형 포인터에 int 형 저장 공간 주소 저장
```

- **위와 같이 포인터 선언시 초기값 지정하는 것이 안전**

```
int *p;                 →       int *p =NULL;  // ok
*p=10;      // Error 발생 가능
```

배열과 함수

- 배열이름은 배열 시작주소 가리킨다. → 배열은 포인터로 취급 가능
- 배열을 함수의 인수로 전달하는 경우에는 "배열 이름"이 포인터(주소)이므로 포인터(주소)가 전달된다.



a →

주소	값
1000	
1001	3
1002	4
1003	5
...	...

예제

```
#include <iostream>

int get_average(const int score[], int n);
void increment(int score[], int n);

int main(void){
    const int STUDENTS=5;
    int grade[STUDENTS] = { 1, 2, 3, 4, 5 };
    int avg;

    increment(grade, STUDENTS);
    avg = get_average(grade, STUDENTS);
    printf("평균은 %d입니다.\n", avg);

    return 0;
}
void increment(int score[], int n){  // 함수정의
    for(int i = 0; i < n; i++)
        ++score[i];
}
```

```
int get_average(const int score[], int n)
{
    int i;
    int sum = 0;

    for(i = 0; i < n; i++)
        sum += score[i];

    return sum / n;
}
```

```
평균은 4입니다.
계속하려면 아무 키나 누르십시오 . . .
```

배열과 포인터 사용 함수 호출

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

void prn(int ax[], int *bx);
int main() {
      int a[3], *b;
      b = new int[3];     // 동적 배열
      for (int i = 0; i < 3; i++){
            a[i] = i * 10;        b[i] = i * 100;
      }
      prn(a, b);      // 배열 이름은 배열 시작 주소
      delete[] b;    // 동적 배열 삭제
      return 0;
}
void prn(int ax[], int* bx){
      for (int i = 0; i < 3; i++){
            cout << ax[i] << endl;
            cout << bx[i] << endl;
      }
}
```

	주소	값
a	1000	1001
a[0]	1001	0
a[1]	1002	10
a[2]	1003	20
b	1004	1005
	1005	0
	1006	100
	1007	200
ax	1008	1001
bx	1009	1005

배열(포인터 상수) ← → 포인터(포인터 변수)

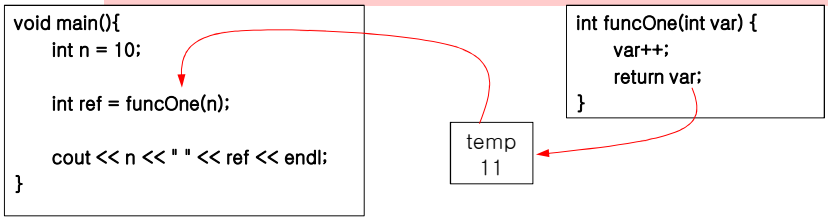
배열과 포인터 사용 함수 호출

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

void prn(int *ax, int *bx);
int main() {
      int a[3], *b;
      b = new int[3];
      for (int i = 0; i < 3; i++){
            a[i] = i * 10;        b[i] = i * 100;
      }
      prn(a, b);      // 배열 이름은 배열 시작 주소
      delete[] b;
      return 0;
}
void prn(int *ax, int* bx){
      for (int i = 0; i < 3; i++){
            cout << ax[i] << endl;
            cout << bx[i] << endl;
      }
}
```

앞쪽과 동일 동작

	주소	값
a	1000	1001
a[0]	1001	0
a[1]	1002	10
a[2]	1003	20
b	1004	1005
	1005	0
	1006	100
	1007	200
ax	1008	1001
bx	1009	1005

함수의 값 반환 과정...



- 매개변수 전달→ int var = n; → var 은 지역변수, n의 값(=10)이 var 에 복사
- 반환시 → return var; → var의 값(=11) 을 반환, 지역변수 var 은 소멸
  → 임시변수(temp) 만들고 var 값(=11) 을 저장
  → 임시변수값(=11) 을 ref 에 저장
  → 임시변수(temp) 소멸

포인터 형변환 불가 ...

- C++에서는 묵시적인 포인터 변환은 허용되지 않는다.
  ➔포인터들의 형변환은 안됨

```
int a=5, b;
float c=4.5, d;
b = c;      // float 를 int 로 형변환
d = a;      // int 를 float 로 형변환

cout << b << " " << d << endl;

int *pi;
float *pf;

pi = &c;            // Error ← int 형 포인터변수에 float 형의 방 주소를 저장
pf = &a;            // Error ← float 형 포인터변수에 int 형의 방 주소를 저장
```

참고) int(4 byte), float(8 byte)

- int 형 포인터 → 저장한 주소값부터 4 byte 에 걸쳐 저장된 값을 정수값으로 사용
- float 형 포인터 → 저장한 주소값부터 8 byte 에 걸쳐 저장된 값을 실수값으로 사용

동적 메모리

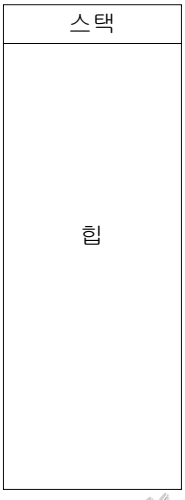
- 동적 메모리 → 배열과 비교(배열은 방 개수를 미리 지정)
  - 실행 도중에 동적으로 메모리를 할당받는 것
  - 사용이 끝나면 시스템에 메모리를 반납
  - 필요한 만큼만 할당을 받고 메모리를 매우 효율적으로 사용
  - new와 delete 키워드 사용, 포인터 변수 사용

그림 3.5 동적 메모리 사용 단계

동적 메모리 …

- 배열
  - 초기에 메모리 할당(방 개수 지정) → int grade[100]
  - 사용이 편리, 초보자가 사용, but
  - 메모리의 스택 부분에 저장공간 생성(작은 공간)

- 동적 메모리
  - 실행 도중에 동적으로 메모리를 할당 (방 개수 실행 중 지정)
    - new int[num]; // 정수 저장용 공간 생성
    - new float[num]; // 실수 저장용 공간 생성
  - new와 delete 키워드 사용, 포인터 변수 사용
  - 사용이 약간 복잡, but,
  - 메모리의 힙 부분에 저장공간 생성(무한한 공간)
  - 대부분 프로그램(C++, C#, Java)은 동적 메모리 할당을 사용
  - 동적 메모리 사용시 C 에서는 심각한 실행오류 자주 발생, but C++ 에서는 규칙만 따르면 오류 거의 없음



동적 메모리 할당 과정

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

int main() {
    int *pi;      // 동적 메모리 시작 주소를 가리키는 포인터

    // ①동적 메모리 할당(100 대신에 변수값 사용 가능)
    pi = new int[100];
    for(int i=0; i < 100;  i++)
        *(pi+i) = i;   // ②동적 메모리 사용 → pi[i] = i;

    for(int i=0; i < 100;  i++)
        cout << *(pi+i) << endl

    delete[] pi;          // ③동적 메모리 반납

    return 0;
}

// 배열과 비슷 ?? → 실제는 프로그램 중간에 방의 개수를 정한다
// 실행중 방 개수를 정하고(100), 100 개 방을 생성
```

	주소	값
pi	1000	1002
	1001	
	1002	0
	1003	1
	1004	2
	1005	3
	...	...

*(1002+0)
*(1002+1)
...

동적 메모리 할당(배열과 비슷)

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

int main() {
        int *pi;              // 동적 메모리 시작 주소를 가리키는 포인터

        pi = new int[100];            // ①동적 메모리 할당

        for(int i=0; i < 100;  i++)
                pi[i] = i;            // ②동적 메모리 사용 → *(pi + i)

        for(int i=0; i < 100;  i++)
                cout << pi[i] << endl

        delete[] pi;                  // ③동적 메모리 반납

        return 0;
}

// 배열과 비슷 ?? → 실제는 프로그램 중간에 방의 개수를 정한다
// 실행 중 결정한 방 갯수에 따라 100 개 방을 생성
```

동적 메모리 할당의 과정

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

int main() {
        int *pi, num;                   // 동적 메모리 시작 주소를 가리키는 포인터

        cin >> num;                     // 입력받은 개수 의 방을 생성
        pi = new int[num];              // ①동적 메모리 할당

        for(int i=0; i < num;  i++)
                *(pi+i) = i;            // ②동적 메모리 사용 → pi[i] = i;

        delete[] pi;                    // ③동적 메모리 반납

        return 0;
}
```

동적 메모리 할당과 반납

```
int *pi = new int;                  // 하나의 int형 공간할당
int *pia = new int[100];            // 크기가 100인 int형 동적배열할당
double *pd = new double;            // 하나의 double형 공간할당
double *pda = new double[100];      // 크기가 100인 double형 동적배열할당
```

```
delete pi;                  // 동적할당 int형공간반납
delete[] pia;               // 동적할당 배열반납
delete pd;                  // 동적할당 double형공간반납
delete[] pda;               // 동적할당 배열반납
```

참고)
- 일차원 배열은 위와 같이 간단하게 할당받을 수 있지만
- 이차원 배열 할당받는 방법은 좀 복잡

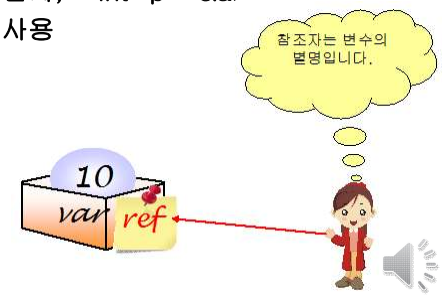
배열과 포인터

```
int a[2], b[2], *p=NULL;
p = a;      // ok
p = b;      // ok
a = b;      // error
```

- 포인터 변수는 변수 → 저장 값(주소)가 변경 가능
- 배열 이름은 주소 저장 → but 포인터 상수 → 저장 값(주소)가 변경 불가능

참조자

- 참조자(reference): 변수에 별명을 붙이는 것, C++ 에서 많이 사용
  int var = 10;
  int &ref = var;
- "참조자 ref 는 변수 var의 별명(alias)이다"
  - "&" 는
  - "=" 오른쪽에 있으면 주소 연산자, int *p = &a;
  - "=" 왼쪽에 있으면 참조자로 사용



예제

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

int main()
{
        int var;
        int &ref = var;                 // 참조자선언

        var = 10;
        cout << "var의값: " << var << endl;
        cout << "ref의값: " << ref << endl;

        ref = 20;               // ref의 값을 변경하면 var의 값도 변경된다.
        cout << "var의값: " << var << endl;
        cout << "ref의값: " << ref << endl;

        return 0;
}
```

	주소	값
var, ref	1000	10 → 20
	1001	
	1002	

```
var의 값: 10
ref의 값: 10
var의 값: 20
ref의 값: 20
```

참조자와 포인터 …

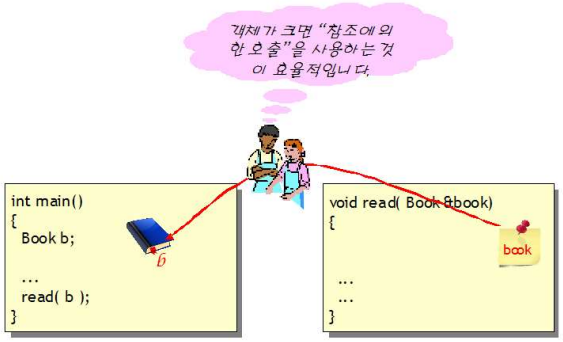
- 참조자는 반드시 선언과 동시에 초기화
  int &ref; // 오류! → int &ref = var; 와 같이 해야함..

- 포인터는 변경될 수 있지만 참조자는 변경이 불가능하다.
  int *p, var1=5, var2=3, &ref = var1;
  p = &var1;
  p = &var2;
  ref = var2; // 문법 오류 아님
  // 실행 오류! → var2 값(3)을 var1 에 저장

- 참조자를 상수로 초기화하면 오류
  int &ref = 10; // 오류!

참조자를 통한 효율성 향상...

- 객체의 크기가 큰 경우, 복사는 시간이 많이 걸린다. 이때는 참조자로 처리하는 것이 유리



- C++ 에서는 함수 매개변수로 객체 전달시 참조자 많이 사용, 함수 반환시에도 사용

참조자를 통한 변경을 방지하려면

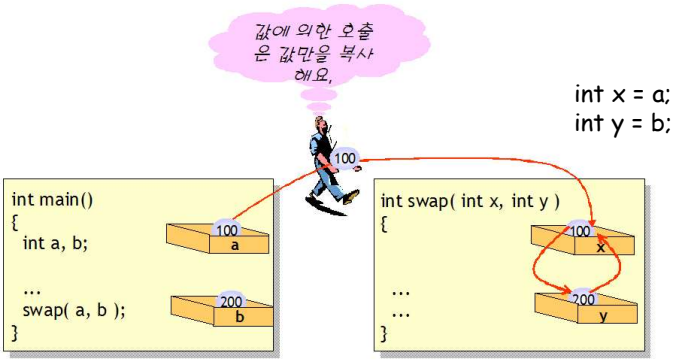
- const를 앞에 붙이면 참조자가 가리키는 내용이 변경 불가능한 상수라는 뜻

```
void print(const int& r)      // → r 값을 함수 안에서 변경 못하게 할 때 사용
{                             // const int &r = 100;

        cout << "현재의 값= " << r  << endl;
        return;
}
int main()
{
        print(100);
        reurn 0;
}
```

함수 호출시 인수전달 방식

- C/C++에서의 인수 전달 방법
  - 값에 의한 호출 (call-by-value)
    - 함수로 인수의 복사본이 전달된다.
  - 참조에 의한 호출 (call-by-reference)
    - 2가지 방식 → 포인터로 전달, 참조자로 전달
    - 함수로 인수의 원본이 전달된다.

인수 전달 방법- 값에 의한 호출



swap() 함수

- 변수 2개의 값을 바꾸는 작업을 함수로 작성

```
#include <iostream>
using namespace std;
void swap(int x, int y);

int main()
{
    int a = 100, b = 200;
    cout << "swap() 호출전: a = " << a << ", b = " << b << endl;
    swap(a, b);
    cout << "swap() 호출후: a = " << a << ", b = " << b << endl;
    return 0;
}
```

```
swap() 호출전: a = 100, b = 200
swap() 호출후: a = 100, b = 200
계속하려면 아무 키나 누르십시오 . . .
```

```
void swap(int x, int y)
{
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
}
```

```
int x = a;
int y = b;
```

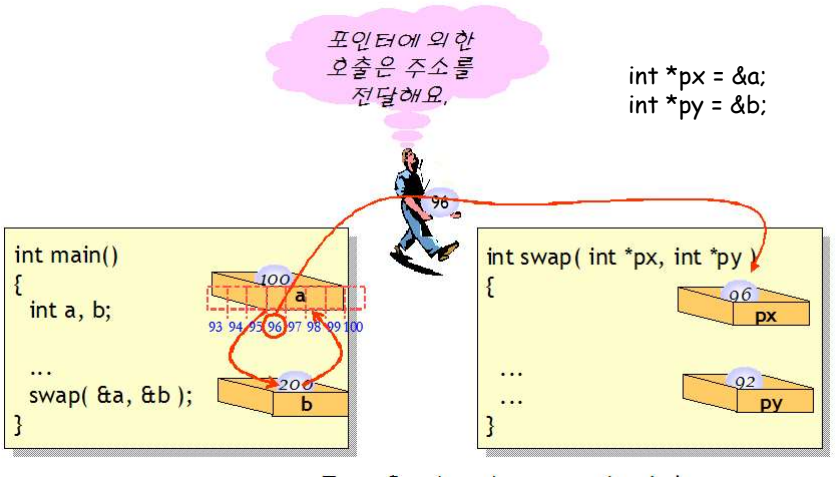
	주소	값
a	1000	100
b	1001	200
	1002	
x → x	1003	100 → 200
y → x	1004	200 → 100
	1005	

참조에 의한 호출

참조에 의한 호출
(call-by-reference)

- 포인터를 사용하는 방법
- 참조자를 이용하는 방법

포인터 의한 호출(포인터 이용)



포인터를 사용하는 참조호출 (주소를 주고 포인터로 받음)

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

void swap(int *px, int *py);
int main()
{
        int a = 100, b = 200;
        cout << "swap() 호출전: a = " << a << ", b = " << b << endl;
        swap(&a, &b);
        cout << "swap() 호출후: a = " << a << ", b = " << b << endl;
        return 0;
}
void swap(int *px, int *py)
{
    int tmp;

    tmp = *px;
    *px = *py;
    *py = tmp;
}
```

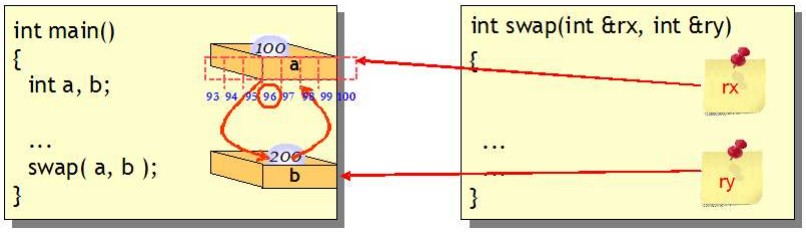
```
// int *px = &a;
// int *py = &b;
```

	주소	값
a	1000	100 → 200
b	1001	200 → 100
	1002	
px → x	1003	1000
py → x	1004	1001
tmp	1005	100

```
swap() 호출전: a = 100, b = 200
swap() 호출후: a = 200, b = 100
```

참조에 의한 호출(참조자)

```
// int &rx = a;
// int &ry = b;
```



참조에 의한 호출(참조자로 받음)

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

void swap(int &rx, int &ry);
int main()
{
        int a = 100, b = 200;
        cout << "swap() 호출전: a = " << a << ", b = " << b << endl;
        swap(a, b);
        cout << "swap() 호출후: a = " << a << ", b = " << b << endl;
        return 0;
}
void swap(int &rx, int &ry)
{
        int tmp;

        tmp = rx;
        rx = ry;
        ry = tmp;
}
```

```
// int &rx = a;
// int &ry = b;
```

	주소	값
a, rx	1000	100 → 200
b, ry	1001	200 → 100
	1002	
tmp	1005	100

```
swap() 호출전: a = 100, b = 200
swap() 호출후: a = 200, b = 100
```

예제

x 는 제거 의미

```
void dec_by_v(int v) {                  // int v = time
    v--;
    return;
}
void dec_by_r(int& r) {                 // int &r = time;
    r--;
    return;
}

void dec_by_p(int* p) {                 // int *p = &time
    --(*p);
    return;
}

int main() {
    int time = 10;
    dec_by_v(time);     cout << time;   10
    dec_by_r(time);     cout << time;   9
    dec_by_p(&time);    cout << time;   8

    return 0;
}
```

	주소	값
time r → x	1000	10 → 9 → 8
	1001	
v → x	1002	10 → 9 → x
	1003	
p → x	1004	1000 → x
	1005	
	1006	

일반적인 매개변수 전달...

값을 받음

```
// int x = n; // 복사
int funcOne(int x) {
    x++;
    return x;
}

void main(){
    int n = 10;
    int k = funcOne(n);

    cout << n << k << endl;

}
// 10, 11
```

포인터로 받음

```
// int *x = &n
void funcOne(int *x) {
    (*x)++;
}

void main(){
    int n = 10;
    funcOne(&n);

    cout << n << endl;

}

// 11
```

참조자로 받음

```
// int &x = n
void funcOne(int& x) {
    x++;
}

void main(){
    int n = 10;
    funcOne(n);

    cout << n << endl;

}

// 11
```

함수의 반환값 들

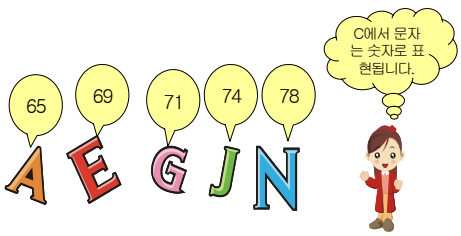
```
int main() {
      int a[2], b[2], a1, a2, *c, *d;

      for (int i = 0; i < 2; i++) {
            a[i] = i * 10;
      }
      c = add_4(a);       prn(c);   // 결과는 ?

      for (int i = 0; i < 2; i++) {
            a[i] = i * 10;
      }
      d = add_5(a);       prn(d);   // 결과는 ?

      return 0;
}
```

```
int* add_4(int ax[]) {
      int bx[2];
      for (int i = 0; i < 2; i++)
            bx[i] = ax[i] + 1;
      return bx;
}

int* add_5(int ax[]) {
      int *bx;
      bx = new int[2];
      for (int i = 0; i < 2; i++)
            bx[i] = ax[i] + 1;
      return bx;
}

void prn(int* ax) {
      for (int i = 0; i < 2; i++)
            cout << ax[i] << endl;
}
```

- add_4 : 함수 안에서 배열로 생성한 방은 함수 빠져 나오면 없어짐. → bx 주소가 반환되지만 그 곳은 삭제되어 쓰레기만 존재함 → 쓰레기 값 출력됨
- add_5 : 함수 안에서 new 로 생성한 방은 함수 빠져 나와도 존재. → bx 주소가 반환되고 그 곳이 값은 존재 → ok

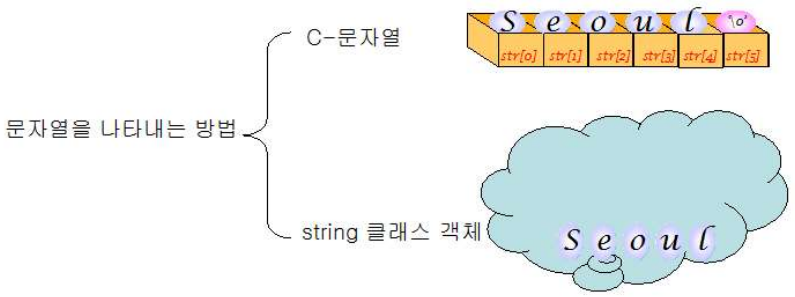
문자표현방법

- 컴퓨터에서는 각각의 문자에 숫자코드를 붙여서 표시한다.
- 아스키코드(ASCII code): 표준적인 8비트 문자코드
  - 0에서 127까지의 숫자를 이용하여 문자표현
- 유니코드(unicode): 표준적인 16비트 문자코드
  - 전세계의 모든 문자를 일관되게 표현하고 다룰 수 있도록 설계

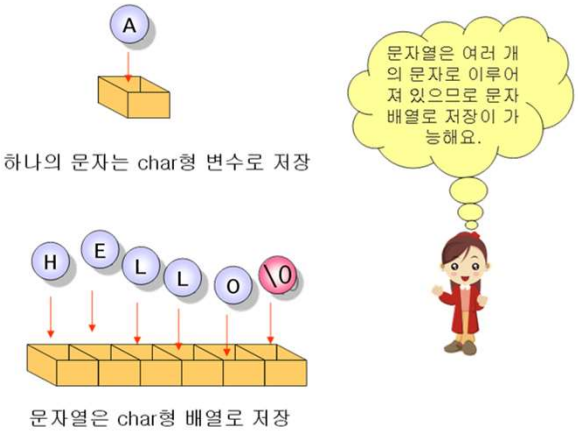


문자열 표현 방법

- *문자열(string):* 문자들이 여러 개 모인것
  - "A"
  - "Hello World!"



C-문자열



문자 배열의 초기화

1. 문자 배열 원소들을 중괄호 안에 넣어주는 방법
   - char str[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };

2. 문자열 상수를 사용하여 초기화하는 방법
   - char str[6] = "Hello";

3. 만약 배열을 크기를 지정하지 않으면 컴파일러가 자동으로 배열의 크기를 초기화값에 맞추어 설정
   - char str[] = "C Bible"; // 배열의 크기는 7이 된다.

문자 배열에 문자를 저장

1. 각각의 문자 배열 원소에 원하는 문자를 개별적으로 대입하는 방법이다.
   - str[0] = 'W';
   - str[1] = 'o';
   - str[2] = 'r';
   - str[3] = 'l';
   - str[4] = 'd';
   - str[5] = '\0';

2. strcpy()를 사용하여 문자열을 문자 배열에 복사

```
char str[6];
//strcpy(str, "World"); // 예전 방법, error
strcpy_s(str, 6, "World");   // ok,  strcpy_s(src, src 크기, des);
```

예제 #1

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

int main()
{
	char str1[7] = "Seoul ";
	char str2[3] = { 'i', 's' };
	char str3[] = " the capital city of Korea.";

	cout << str1 << str2 << str3 << endl;
	return 0;
}
```

```
Seoul is the capital city of Korea.
```

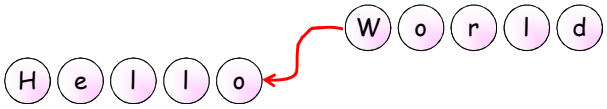
문자열 길이 계산 예제

```
// 문자열의 길이를 구하는 프로그램
#include <iostream>
using namespace std;

int main()
{
	char str[30] = "C++ language is easy";
	int i = 0;

	while(str[i] != 0)   // '\0'
		i++;

	cout << "문자열 "<< str << "의 길이는 " << i << "입니다." << endl;

	return 0;
}
```

```
문자열 C++ language is easy의 길이는 20입니다.
계속하려면 아무 키나 누르십시오 . . .
```

문자열 처리 라이브러리(2010 버젼)

함수	설명
strlen(s)	문자열 s의 길이를 구한다.
strcpy(s1, s2)	s2를 s1에 복사한다.
strcat(s1, s2)	s2를 s1의 끝에 붙여넣는다.
strcmp(s1, s2)	s1과 s2를 비교한다.
strncpy(s1, s2, n)	s2의 최대 n개의 문자를 s1에 복사한다.
strncat(s1, s2, n)	s2의 최대 n개의 문자를 s1의 끝에 붙여넣는다.
strncmp(s1, s2, n)	최대 n개의 문자까지 s1과 s2를 비교한다.
strchr(s, c)	문자열 s안에서 문자 c를 찾는다.
strstr(s1, s2)	문자열 s1에서 문자열 s2를 찾는다.



VS 2015 이상에서 문자열 처리함수

- Studio 2010
  - strcpy(복사받을 변수, 복사할 변수);
  - strcat(복사받을 변수, 복사할 변수);

- Studio 2015 이상 버젼
  - 보안 문제 때문에 다음과 같이 함수 변경
  - strcpy_s(복사받을 변수, 복사받을 변수 크기, 복사할 변수);
  - strcat_s(복사받을 변수, 복사받을 변수 크기, 복사할 변수);

```
char* f1(char *x) {
  char *t = new char[40];// 동적 할당

  strcpy_s(t, 40, x);
  strcat_s(t, 40, " ddd");
  return t;
}
```

예제(교재) → error

```
// strcpy와 strcat
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
      char string[80];

      strcpy( string, "Hello World from " );        // strcpy_s( string, 80, "Hello World from " );
      strcat( string, "strcpy() " );
      strcat( string, "and " );
      strcat( string, "strcat()!" );
      cout << string << endl;
      return 0;
}
```

```
Hello World from strcpy() and strcat()!
```

예제(수정 후)

```
// strcpy와 strcat
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
      char string[80];

      strcpy_s( string, _countof(string), "Hello World from " );
      strcat_s( string, 80, "strcpy() " );
      strcat_s( string, 80, "and " );
      strcat_s( string, _countof(string), "strcat()!" );
      cout << string << endl;
      return 0;
}
```

int arr[10]

- **sizeof(arr) 의 결과 값은 byte 수** (4 바이트인 int 타입이 10개, 40 byte)
- **_countof(arr) 의 결과 값은 공간 갯수(=10)** (10개의 배열)

```
Hello World from strcpy() and strcat()!
```

스택, 힙

- 변수의 메모리 저장 장소
  - 스택 :
    - 지역변수의 저장공간이 생성
    - 함수가 끝나면 저장공간이 자동으로 소멸
  - 힙
    - 동적으로 저장공간이 할당되는 공간
    - 함수를 사용하여 공간 할당
      - malloc()/calloc() ← C 방식
      - new( ) ← C++ 방식
    - 저장공간이 자동으로 없어지지 않음 → 코드로 소멸시켜야 함
      - free() ← C 방식
      - delete( ) ← C++ 방식

Report

- 122 쪽~
- 3번, 5번 , 9번, 11번
- 함수로 배열을 전달하고 그 값을 반환 받는 방법 잘 아는지 test

- 제출파일 이름 : OOP_학번_3.cpp → OOP_학번_3.txt

```
void main(){
    p3( );
    P5();
    P9();
    P11();
}
```

Report

- p3() 에서 평균, 편차를 구하는 다음 두 함수를 호출하고 결과 값을 반환 받아 출력
  - calc_Avg(…), calc_Dev(…)
- p5() 에서 행 합, 열 합을 구하는 다음 두 함수를 호출하고 결과 값을 반환 받아 출력
  - sum_Row(…), sum_Col(…)
- p9() 에서 copy(…) 를 호출하면 copy 함수에서 복사하고 출력
- p11() 에서 get_stat(…) 호출하고 3개 값을 반환받아 출력

Report

- 제출처
  - DOOR → 수업결과 → 수업활동일지 → 해당 과제란
  - 수업활동 일지의 과제란에 있는 한글 파일에 코드 작성하여 제출
- 제출기한 엄수

Q & A