C++ / JAVA 프로그래밍

06주차 포인터와 함수의 매개변수 전달

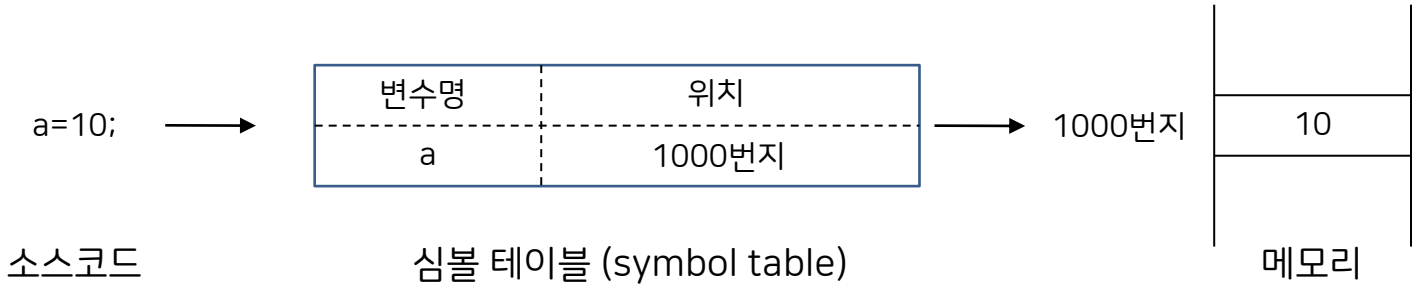
학습목표

- 포인터에 대해 알아보고 포인터 선언 방법과 사용법을 익힌다.
- 포인터 연산에 대해 알아본다.
- 함수의 인자 전달 방식 중에서 값에 의한 전달 방식과 포인터에 의한 전달 방식을 학습한다.

메모리 주소와 변수의 관계

- 메모리 주소(address)
  - 메모리의 위치를 구분하기 위해 붙인 일련번호
  - 0번지부터 시작하여 차례로 증가함

- 변수는 메모리 주소에 대한 별명
  - 컴퓨터는 메모리에 데이터를 올려놓고 사용함
  - 메모리의 데이터를 사용하려면 주소를 알아야 함
  - 메모리 주소를 프로그래머가 일일이 기억하고 사용하는 것은 불가능
  - 컴파일러는 심볼 테이블을 통해 변수명, 데이터 타입, 주소 사이의 관계를 관리함



포인터의 개념

- Pointer
  - 사전적 정의는 '가리키는 것'
  - 특정 메모리 주소를 가리킴
  - C++에서는 포인터를 직접 사용할 수 있도록 연산자 &와 *를 제공

포인터 연산자

```
&변수명          // 변수의 주소를 알려줌
*포인터변수명    // 주소에 저장된 값을 알려줌
```

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

int main() {
    int a=10;
    cout<<" 변수 a에 저장된 값 => "<< a << endl;
    cout<<" 변수 a의 주소 => "<< &a << endl;
    cout<<" 변수 a에 저장된 값 => "<< *&a << endl;
    return 0;
}
```

```
./main
 변수 a에 저장된 값 => 10
 변수 a의 주소 => 0x7ffdf4b72458
 변수 a에 저장된 값 => 10
```

포인터 변수

- 주소만 저장하기 위해 사용하는 변수
  - 일반적인 변수와 구분하기 위해 포인터 변수를 선언 할 때 반드시 * 기호를 붙여야 함

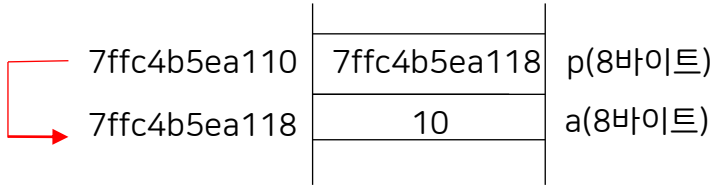
포인터 변수 선언

```
자료형 *포인터변수명
```

- 포인터 변수와 * 연산자
  - * 연산자를 주소 앞에 붙이면,
    그 주소를 찾아가서 저장된 값을 반환함
  - * 연산자를 포인터 변수 앞에 붙이면,
    포인터 변수에 저장된 주소를 찾아가서
    저장된 값을 반환함

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

int main() {
    int a=10;
    int *p;
    p=&a;
    cout<<" a => "<< a << endl;
    cout<<" &a => "<< &a << endl;
    cout<<" p => "<< p << endl;
    cout<<" *p => "<< *p << endl;
    return 0;
}
```

```
./main
a => 10
&a => 0x7ffc4b5ea118
p => 0x7ffc4b5ea118
*p => 10
&p => 0x7ffc4b5ea110
```



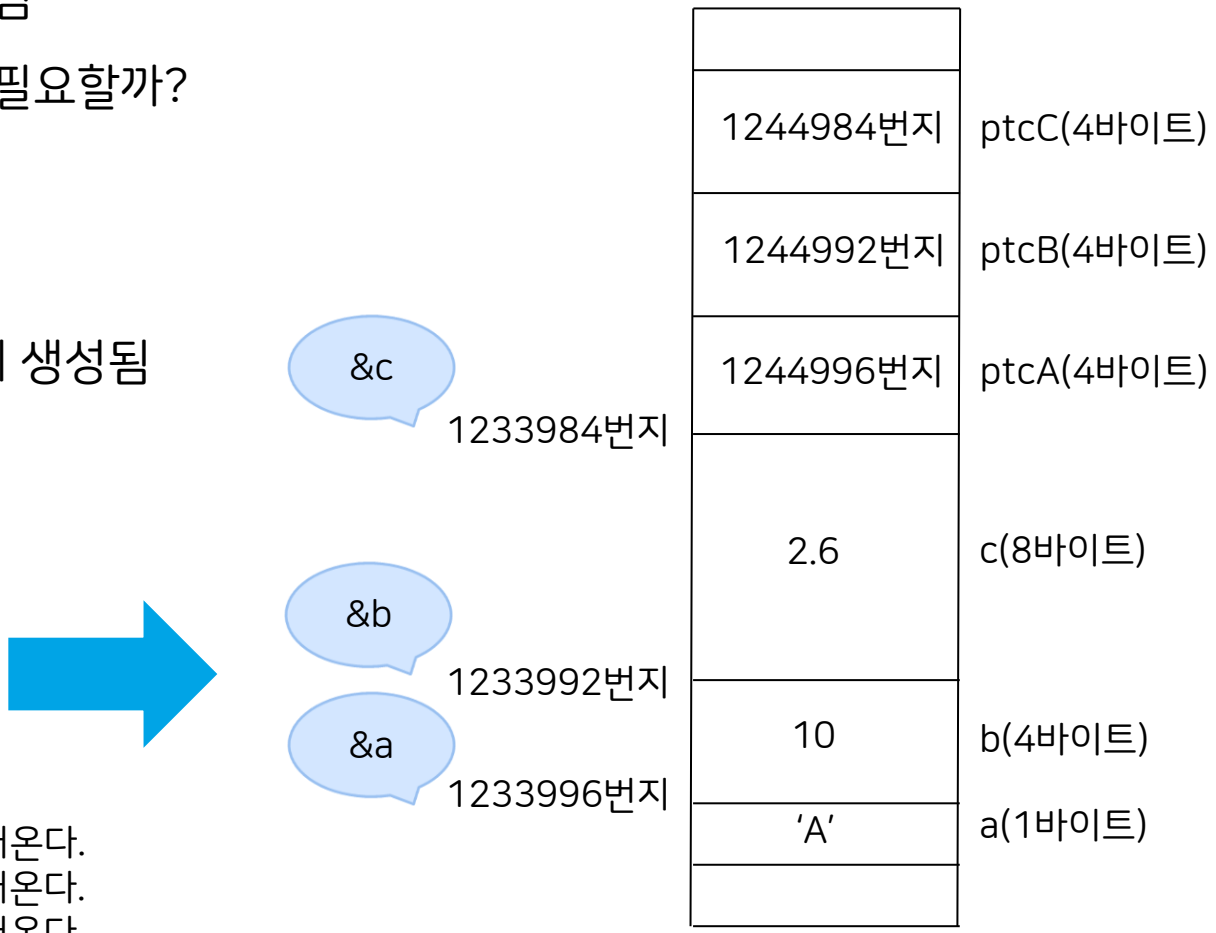
포인터 변수를 메모리에 할당한 구조

포인터 변수의 자료형

- 포인터 변수의 자료형은
  - 주소를 따라갔을 때 저장된 자료의 형태에 따라 결정됨
  - 포인터 변수는 주소를 저장하는 것인데 자료형은 왜 필요할까?

- 참고사항
  - 메모리 할당은 먼저 선언된 변수가 아래쪽(큰 주소)에 생성됨
  - 주소의 크기는 워드의 크기에 따라 결정됨
  - 워드의 크기는 시스템마다 다를 수 있음

```
char a ='A';
int b =10;
double c = 2.6;

char   ptrA = &a; // ptrA에 저장된 주소로부터 1바이트를 읽어온다.
int    ptrB = &b; // ptrB에 저장된 주소로부터 4바이트를 읽어온다.
double ptrC = &c; // ptrC에 저장된 주소로부터 8바이트를 읽어온다.
```



포인터 변수 다루기

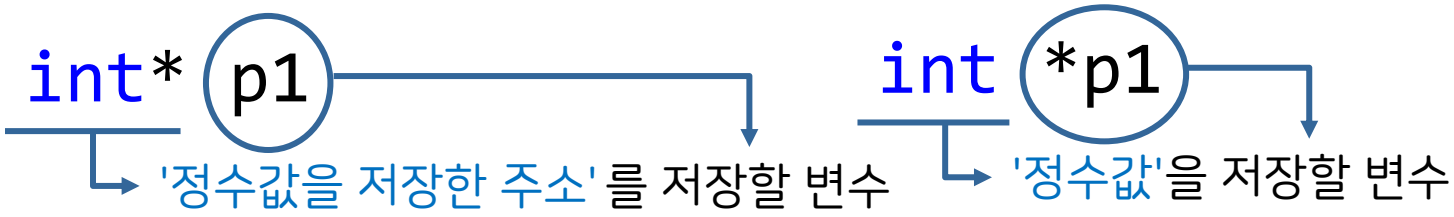
- 포인터 변수에는 주소 값을 대입해야 함
  - 값을 의미하는 일반 변수를 대입할 경우 오류 발생

- 오른쪽 예제의 의미
  - a는 정수형 변수
  - p1과 p2는 '정수값을 저장한 주소'를 저장할 수 있음
  - &a는 정수형 변수의 주소이므로 문제가 되지 않지만, a는 정수값이므로 오류
  - *p1과 *p2는 '정수값'을 저장할 수 있음

```
int a = 10;

int* p1 = &a;    // 정상
int* p2 = a;     // 오류

p1 = &a;         // 정상
p1 = a;          // 오류

*p1 = 20;        // 정상
*p1 = &a;        // 오류
```

포인터 변수에 값 넣기

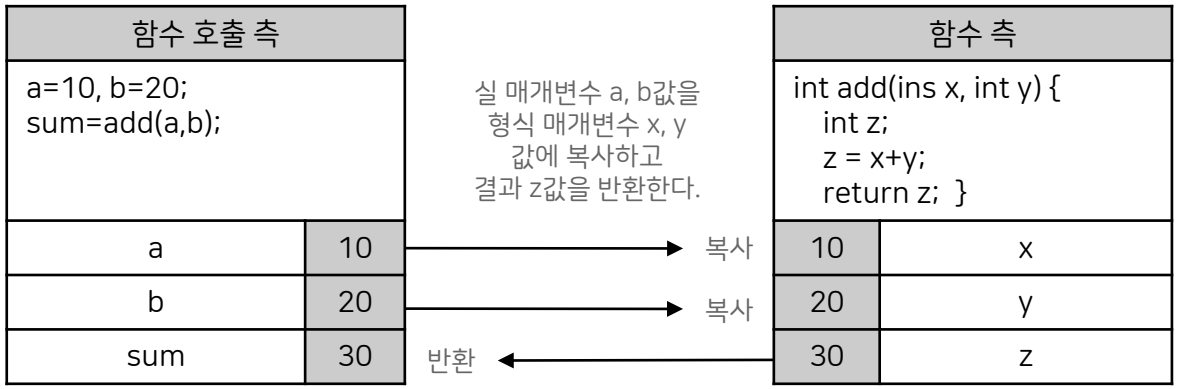
- p1과 *p1에 저장할 값을 구분하는 방법



함수에서 매개변수를 전달하는 방법

- 크게 3가지 방법이 있음
  - 값에 의한 호출 방식(Call by Value)
  - 주소에 의한 호출 방식(Call by Address)
  - 참조에 의한 호출 방식(Call by Reference)

- 값에 의한 호출 방식
  - 지금까지 함수호출 방식과 동일
  - 함수를 수행한 결과를 전달하려면 return문 사용
  - 형식 매개변수가 실 매개변수와 별개의 기억공간을 사용
  - 값을 복사하므로 함수를 정의한 곳에서 형식 매개변수의 값을 변경해도 실 매개변수의 값은 변경되지 않음



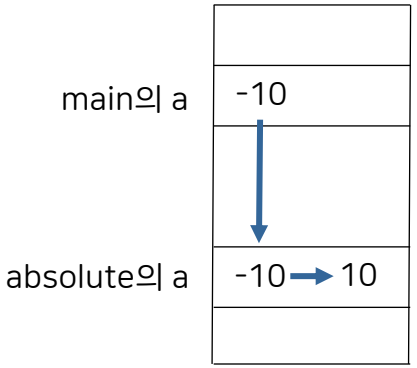
예제. 값 호출 후 변수 값의 변화

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

void absolute(int a);

int main() {
    int a=-10;
    cout<<" main 에서 함수 호출 전 a 값 = "<< a <<endl;
    absolute(a);
    cout<<" main 에서 함수 호출 후 a 값 = "<< a <<endl;
    return 0;
}

void absolute(int a) {
    if(a<0)
        a=-a;
    cout<<" absolute 함수에서의 a 값 = "<< a <<endl;
}
```

```
> ./main
 main 에서 함수 호출 전 a 값 = -10
 absolute 함수에서의 a 값 = 10
 main 에서 함수 호출 후 a 값 = -10
```

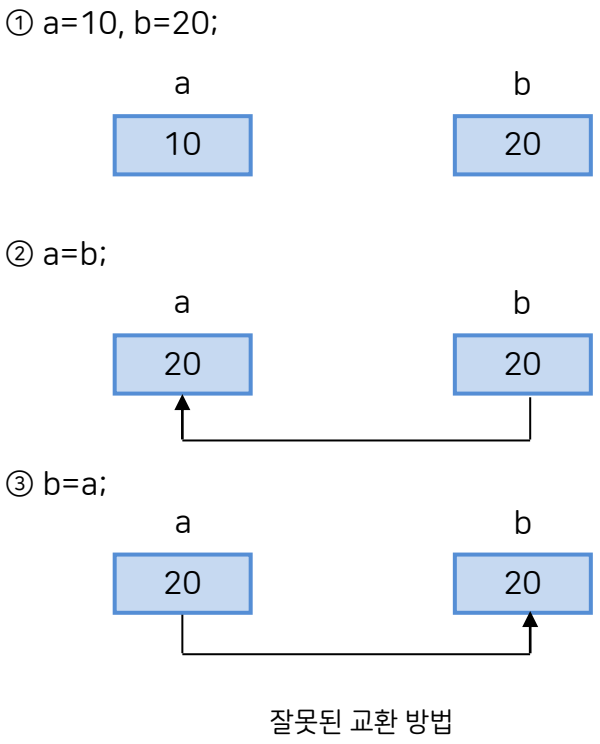
main 함수 측에서 선언된 변수 a와
absolute 함수 측에서 선언한 형식 매개변수 a는
서로 독립된 기억공간을 할당받음

따라서 absolute 함수 내에서 a를 변화시켜도
main 함수의 a에게는 반영되지 않음

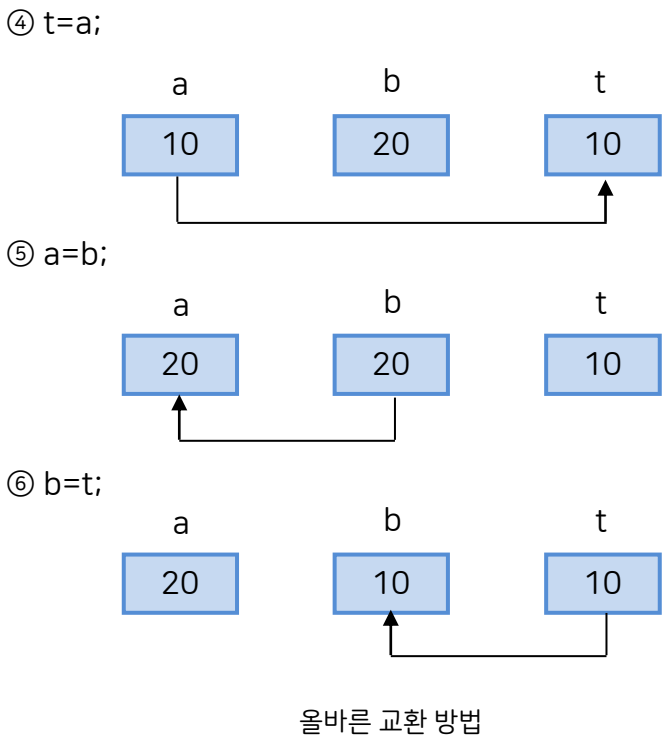


주소에 의한 호출 방식

- 두 변수에 저장된 값을 바꿔보자
  - 변수 a의 값을 임시 기억장소인 t에 저장
  - 변수 b의 값을 임시 변수 a에 저장하더라도 변수 t에 a값을 저장해 두었으므로
  - t의 값을 변수 b에 저장하면 두 변수의 값이 교환된다.



잘못된 교환 방법



올바른 교환 방법

예제. 두 변수에 저장된 값 바꾸기 – 함수 없이

```
// 함수 호출 없이 두 변수에 저장된 값 바꾸기
#include <iostream>
using namespace std;

int main() {
    int a=10, b=20;
    cout<<" a => "<< a <<" b => "<< b <<endl;
    int t;
    t=a;
    a=b;
    b=t;
    cout<<" a => "<< a <<" b => "<< b <<endl;
    return 0;
}
```

```
> ./main
 a => 10 b => 20
 a => 20 b => 10
```

예제. 두 변수에 저장된 값 바꾸기 – 값 호출

```
// 값 호출을 이용해 두 변수에 저장된 값 바꾸기 (실패)
#include <iostream>
using namespace std;
void swap(int a, int b);

int main() {
    int a=10, b=20;
    cout<<" a => "<< a <<" b => "<< b <<endl;
    swap(a, b);
    cout<<" a => "<< a <<" b => "<< b <<endl;
    return 0;
}

void swap(int a, int b) {
    int t;
    t=a;
    a=b;
    b=t;
}
```

```
./main
 a => 10 b => 20
 a => 10 b => 20
```

예제. 두 변수에 저장된 값 바꾸기 – 주소 호출

```
// 주소 호출을 이용해 두 변수에 저장된 값 바꾸기
#include <iostream>
using namespace std;
void swap(int *pa, int *pb);

int main() {
    int a=10, b=20;
    cout<<" a => "<< a <<" b => "<< b <<endl;
    swap(&a, &b);
    cout<<" a => "<< a <<" b => "<< b <<endl;
    return 0;
}

void swap(int *pa, int *pb) {
    int t;
    t=*pa;
    *pa=*pb;
    *pb=t;
}
```

```
> ./main
 a => 10 b => 20
 a => 20 b => 10
```

참조에 의한 호출 방식

- 참조 변수란
  - 별칭(일종의 다른 이름)을 의미함
  - 변수에 별도의 기억공간을 할당하지 않고, 이미 선언되어 있는 변수의 별칭으로 사용함
  - 동일한 메모리 공간을 여러가지 변수명으로 접근할 수 있게 함

- 참조 변수 선언 방법

참조 변수 선언

```
자료형 &새로운_변수명 = 이미_사용_중인_변수명
```

  - 별칭으로 사용할 변수명 앞에 & 기호를 덧붙이며, 이때 사용한 & 기호를 참조 연산자라고 함

- 주의할 점
  - 변수를 선언할 때 반드시 초기값을 주어야 함 (이미 존재하는 변수에 대해 이름만 하나 더 추가하는 것이기 때문)

예제. 참조 변수 선언 및 사용 방법

```
// 참조 변수 선언 및 사용
#include <iostream>
using namespace std;

int main() {
    int a=10;
    int &b = a;
    cout<<" a = "<<a<<" b = "<<b<<endl;
    b+=300;
    cout<<" b = "<<b<<endl;
    cout<<" a = "<<a<<endl;
    return 0;
}
```

```
> ./main
 a = 10 b = 10
 b = 310
 a = 310
```

예제. 두 변수에 저장된 값 바꾸기 – 참조 호출

```
// 참조 호출을 이용해 두 변수에 저장된 값 바꾸기
#include <iostream>
using namespace std;
void swap(int &x, int &y);

int main() {
    int a=10, b=20;
    cout<<" a => "<< a <<" b => "<< b <<endl;
    swap(a, b);
    cout<<" a => "<< a <<" b => "<< b <<endl;
    return 0;
}

void swap(int &x, int &y) {
    int t;
    t=x;
    x=y;
    y=t;
}
```

```
> ./main
 a => 10 b => 20
 a => 20 b => 10
```


HANYANG
UNIVERSITY