Day-40

Functions
Pass by-Value, Pass by-Pointer, Pass by-Reference

Call-by-value

- 객체가 복사되어 f(T x)메서드의 입력 인수에 할당됨
- 장점
  - 함수 내부의 매개변수 변경은 전달된 인수 원본에 영향을 미치지 않음
- 단점
  - 복사된 인수가 큰 경우 성능 저하
  - ex : 여러 데이터 멤버가 있는 구조
- 사용 하는 경우
  - 내장 데이터 타입, 작은 객체(8 Bytes 이하)
- 사용 하지 않는 경우
  - 고정 크기 배열
  - 큰 객체

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

void CallByValue(int value)
{
    value = 20;
    cout << value << endl;  // value의 값
    cout << &value << endl; // value의 주소
}

int main()
```

```
{
    int num = 10;
    CallByValue(num);
    cout << num << endl;
    cout << &num << endl;

    return 0;
}

// 결과
// 20
// 000000087236F780
// 10
// 000000087236F7A4
```

Call-by-pointer

- 변수의 주소가 복사되어 f(T* x) 메서드의 입력 인수에 할당됨
- 장점
  - 함수가 전달된 인수 값을 변경할 수 있음
  - 인수가 복사되지 않기 때문에 빠름
- 단점
  - 인자가 null 포인터일 경우가 존재할 수 있음
  - 포인터를 역참조하는 것은 값에 직접 액세스하는 것보다 느림
- 사용 하는 경우
  - 기본 배열(읽기 전용인 경우 const T* 사용)
- 사용하지 않는 경우
  - 다른 모든 경우

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

void CallByPointer(int* address)
```

```
{
    *address = 20;
    cout << *address << endl;  // address가 가리키는 주소 값
    cout << &address << endl;  // address의 주소
}

int main()
{
    int num = 10;
    CallByPointer(&num);
    cout << num << endl;
    cout << &num << endl;

    return 0;
}

// 결과
// 20
// 000000C51FF7F570
// 20
// 000000C51FF7F594
```

Call-by-reference

- 변수의 참조가 복사되어 f(T& x) 메서드의 입력 인수에 할당됨
- 장점
  - 함수가 인자의 값을 변경할 수 있음(포인터에 비해 가독성 향상)
  - 인수가 복사되지 않기 때문에 빠름
  - 참조를 초기화해야 함 → 널 포인터 없음
  - 암시적 변환을 피할 수 있음
- 사용 하는 경우
  - 기본 포인터를 제외한 모든 경우
- 사용하지 않는 경우

- Pass by-value를 사용하는 것이 더 뛰어난 성능 이점을 제공하는 경우

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

void CallByReference(int& reference)
{
    reference = 20;
    cout << reference << endl;   // reference가 가리키는 주소
    cout << &reference << endl;  // reference의 주소
}

int main()
{
    int num = 10;
    CallByReference(&num);
    cout << num << endl;
    cout << &num << endl;

    return 0;
}

// 결과
// 20
// 000000477F4FF704
// 20
// 000000477F4FF704
```