Day-30

Pointers and References
Address-of operator &

Address-of operator &

- 주소 연산자(&)는 변수의 주소를 반환

```
int a = 3;
int* b = &a; // Address-of operator
             // 'b'는 'a'의 주소와 동일
a++;
cout << *b; // print 4;
```

- 참조와 혼동하지 않도록 주의해야 함

Wild and Dangling Pointers

- Wild pointer

```
int main()
{
    int *ptr; // wild pointer : 가리키고 있는 곳이 없음
                        // 포인터는 항상 초기화 할 것
}
```

- Dangling pointer

```
int main()
{
    int* arr = new int[10];
    delete[] arr; // ok, arr는 댕글링 포인터
```

```
    delete[] arr; // double free
    // 프로그램 중단
}
```

void Pointer - Generic Pointer

- 다양한 타입의 포인터 변수를 선언하는 대신 모든 포인터 타입으로 작동할 수 있는 단일 포인터 변수를 선언할 수 있음
- void포인터를 사용하기 위해선 대입한 데이터형으로 형변환 시킨 후에 사용해야 함 → void 포인터가 뭘 가리키는지 모르기 때문
  - void* 비교 가능
  - 모든 포인터 타입은 암시적으로 void*로 변환 할 수있음
  - 컴파일러가 실제로 어떤 종류의 객체를 가리키는지 모르기 때문에 다른 연산은 안전하지 않음

```
cout << (sizeof(void*) == sizeof(int*)); // print true

int arr[] = {2, 3, 4};
void* ptr = arr; // 암시적 변환
cout << *arr;    // print 2
// *ptr;         // 컴파일 에러
// ptr+2;        // 컴파일 에러
```