Day-29

Pointers and References
Pointer Operations

Pointers and Pointer Operations

- Pointer
  - 포인터 T*는 메모리 내 위치를 가리키는 값
- Pointer Dereferencing(포인터 역참조)
  - 포인터 역참조 (*ptr)는 포인터가 참조하는 위치에 저장된 값을 가져오는 것을 의미함
- Subscript Operator(아래 첨자 연산자) []
  - 아래첨자 연산자(ptr[])를 사용하면 지정된 위치의 포인터 요소에 액세스 할 수 있음
- 포인터 타입(ex : void*)은 기본 아키텍처에 따라 32비트/64비트의 부호 없는 정수임
  - 연산자 +, -, ++, -- , 비교 ==, !=, <, <=, >, >= , 아래 첨자 [] , 역참조 *만 지원
  - 포인터를 정수형으로 명시적으로 변환할 수 있음

```
void* x;
size_t y = (size_t) x; // ok, 명시적 변환
// size_t y = x;       // 컴파일 에러(암시적 변환)
```

- 역참조

```
int* ptr1 = new int;
*ptr1 = 4;
int a = *ptr1;
```

- 배열 아래첨자

```
int* ptr2 = new int[10];
ptr2[2] = 3;
int var = ptr2[4];
```

- 일반적인 오류

```
int *ptr1, ptr2; // 포인터, 정수형 조합, 잘못된 사용
int *ptr1, *ptr2; // 가능
```

Pointer Conversion

- 모든 포인터 타입은 암시적으로 `void*`로 변환할 수 있음
- Non-`void` 포인터는 명시적으로 변환해야 함
- 정적 형변환은 안전상의 이유로 포인터 변환에 허용되지 않음
  - `void*`는 예외임

```
int* ptr1 = ...;
void* ptr2 = ptr1;          // int* -> void*, 암시적 변환

void* ptr3 = ...;
int* ptr4 = (int*) ptr3;    // void*->int, 명시적 변환 필요
                            // static_cast 가능
int* ptr5 = ...;
char* ptr6 = (char*) ptr5; // int* -> char* 명시적 변환 필요
                            // static_cast 불가, 위험함
```

1 + 1 = 2 : Pointer Arithmetic

- ptr[i] = *(ptr + i)
- 포인터 산술 규칙

```
address(ptr + i) = address(ptr) + (sizeof(T) * i);
// T는 ptr이 가리키는 요소의 타입
```