**Chapter 1 ~ 2 학습 필기**

**1. 출력 스트림과 버퍼 & std::endl**

출력 스트림 : 데이터를 외부로 보내는 Out 객체(입력 까지 해서 Iostream)

출력 버퍼 : 스트림이 데이터를 바로 출력하지 않고 임시로 저장하는 메모리

std: :endl 줄바꿈 문자(‘\n’)를 삽입한 뒤, 연결된 출력 장치의 버퍼를 강제로 비웁니다. 이로 인해 메모리에 잠시 머물러 있던 출력이 즉시 콘솔이나 파일로 쓰여짐

C style : printf(“\n”); fflush(); 의 기능을 수행함

**2. std::string의 가변 구조**

std::string은 내부적으로 동적 메모리를 사용해 가변 길이 데이터를 관리합니다. size()와 capacity()를 통해 현재 길이와 할당된 용량을 추적하며, append(), insert(), erase(), replace(), push\_back(), pop\_back() 등을 통해 내용 수정. 용량이 부족할 때는 새 버퍼를 할당하고 기존 데이터를 복사해 확장.(항상 메모리를 필요 이상으로 할당된 상태, 문자열 사이즈가 충분히 예측 되며 다량의 데이터가 필요한 ex 유저 이름, 에서는 사용을 권장하지 않음)

**3. 메모리 할당: new vs malloc + memset**

new 연산자는 메모리 할당 후 생성자를 호출delete 시 소멸자를 자동 실행합니다. 반면 malloc은 단순히 메모리 블록만 할당하고 생성자·소멸자를 호출하지 않는다.

memset은 단순히 특정 값으로 초기화하며 생성자와 같이 객체의 초기화 개념이 아님. C++ 객체는 반드시 new/delete를 써서 생성자와 소멸자로 관리필요.

**4. 초기화 방식: =, (), {}**

– 복사 초기화 (int x = 0;) – 직접 초기화 (int x(0);) – 리스트 초기화 (int x{0};)

**5. 선언(declaration) vs 정의(definition)**

선언은 extern int count;처럼 심볼의 존재만 알리는 것이고, 메모리 할당을 수반하지 않습니다. 정의는 int count = 0;처럼 실제 메모리를 할당하고 초기화합니다. 전역 심볼은 정의 한 번, 선언 여러 번 허용되며, 중복 정의 시 링크 오류가 발생합니다.

**5 타입주의 사항**

**5.1 음수가 될 수 없는 값은 unsigned , size\_t 등 부호 없는 타입 사용**

인덱싱 과정 등 논리적 오류 방지

**5.2 산술 표현식에서 char , bool 사용 금지**

char , bool 의 자료형의 경우 저장용으로만 사용 계산과정에 포함 x  
char 의 경우 머신 환경에 따라 signed or unsigned 로 해석될 수 있어 문제 발생가능

bool 타입의 경우 논리 타입이므로 연산이 모호해지며 가독성을 해칠 수 있음

**5.3 unsigned 타입의 자료형의 범위이상의 값/ 하드웨어적 MOD 연산 혼동 주의**

책의 본문의 경우 “If we assign an out-of-range value to an object of unsigned type, the result is the remainder of the value modulo the number of values the target type can hold.” 다음과 같이 범위를 넘어서면 값의 모듈 값이 결과로 들어간다 기재되어 있음 해당 결과는 초과된 상위 비트가 잘려 하위 비트만 표기한 결과로 하드웨어적인 모듈러 연산 과정은 동작하지 않음

**5.4부동소수점 사용시 double 타입 사용 권장**

IEEE 754 float 기수 지수부가 8bit 10진수 소수점 이하 7자리의 정밀도를 가짐 이러한 경우 cos 과 같은 삼각함수 등 0으로 수렴되는 극한 값을 다루는 경우 왜곡이 발생할 수 있으며 책에서 또한 “cost of double-precision calculations versus single-precision is negligible.(직역하면 double-precision 연산 대비 single-precision 연산 비용은 미미하다.) ”

**6. 참조(reference)와 포인터(pointer)**

참조(int&)는 선언 시점에 한 번만 대상이 정해지며, 이후 변경 불가합니다. 참조 변수는 실제 메모리를 새로 할당하지 않고, 대상과 동일한 메모리 공간을 가리킵니다. – int\*&는 포인터 자체를 참조하는 C++ 전용

C 와의 호환성이 필요하다면 호환되어야 하는 함수 라면 매개 변수에 이중 포인터(int\*\*)사용

**6.1. 해제된 메모리 참조 위험**

int\* p = (int\*)malloc(sizeof(int));

\*p = 1024;

int& r = \*p;

free(p);

r을 사용하면 시 메모리 접근 오류!

free 후에도 참조는 자동 무효화되지 않아, 해제된 메모리를 계속 가리켜 오류 발생

**6.2 static 지역 변수와 안전한 참조**

int& getRef() {

static int B = 10;

return B;

}

static으로 선언된 지역 변수는 함수 호출이 끝나도 소멸되지 않고 프로그램 종료 시까지 유지됩니다. 따라서 이 변수를 반환해 참조해도 안전합니다.

**7.클래스간 ’=’ 연산자 재정의 필요성 (얕은 복사 / 깊은 복사)**

C++ 에서 기본적인 = 을 통한 값 복사의 경우 객체 내부의 데이터를 그대로 복사

즉 포인터의 경우 주소가 그대로 복사되어 동일한 주소를 가진 포인터 발생!(얕은 복사)

다음의 경우 값을 넘겨준 객체의 소멸자에 메모리 해제를 명시하였을 것으로(생성자에서 할당을 했는데 소멸자에서 해당공간을 해지 하지 않은 경우는 그거 대로 문제..) 원본 객체 소멸과 함께 메모리도 해지된다.

값을 넘겨받은 객체에서 주소에 접근시 메모리 접근 오류 발생!

Ex)

class OBJECT{

public:

char\* buffer;

OBJECT(size\_t size) {

buffer = new char[size];

}

~OBJECT(){

delete[] buffer;

}

};

OBJECT a(100);

OBJECT b = a;

a.~OBJECT(); // b에서 버퍼 호출시 메모리 오류 발생!

포인터를 가진 객체의 경우 깊은 복사 필요!

OBJECT& operator=(const OBJECT & other) {

if (this != &other) {

delete[] buffer;

buffer = new char[strlen(other.buffer) + 1];

strcpy(buffer, other.buffer);

}

return \*this;

}

다음과 같이 =을 재정의하여 포인터주소가 아닌 새로운 메모리를 할당하여 메모리의 값을 복사 해주어야함.

**8. 상수 포인터 vs 포인터 상수와 typedef 우선순위**

const T\* p: 가리키는 값이 const (값 변경 불가), 포인터는 변경 가능

T\* const p: 포인터 자체가 const (주소 변경 불가), 값은 변경 가능

const T\* const p: 주소와 값 모두 const

typedef char\* pstring;

const pstring cstr; 의 경우 char\* const cstr로 해석되어 “상수 포인터”가 됩니다. 가리키는 값은 mutable, 포인터만 const입니다.

**8.1 const vs constexpr**

const 변수는 런타임에 초기화

constexpr 변수는 컴파일 타임초기화 함수의 반환 값을 저장할 수 있음,

암시적으로 const

**9. typedef vs using**

typedef Sales\_item SI;

using SI = Sales\_item;

두 문법은 타입 별칭(type alias)을 만드는 동일 기능 but,

using은 템플릿 별칭(template alias) 지원.

**10. auto vs decltype**

– auto x = expr;는 초기화 식 expr의 타입을 추론하며, 참조와 const 한정자는 보통 무시됩니다. – decltype(expr)는 표현식 expr의 타입을 있는 그대로 가져와, 참조와 const 특성을 그대로 유지합니다. – decltype(auto)는 auto와 decltype을 조합해 함수 반환 타입 등을 정밀하게 추론할 때 사용.

**11. 전처리기 매크로 스코프**

#define SIZE 100

매크로는 C++의 네임스페이스나 블록 스코프를 따르지 않고 전역적으로 치환됩니다. 이로 인해 이름 충돌 위험이 높아, 가능한 constexpr나 const 변수로 대체하는 것이 안전합니다.