**항목 2. #define을 쓰려거든 const, enum, inline을 먼저 떠올리자.**

(가급적 선행 처리자보다 컴파일러를 더 가까이하자)

#define은 자체의 일부가 아닌 것으로 취급될 수 있음.

|  |
| --- |
| #define ASPECT\_RATIO 1.653 |

우리에겐 ASPECT\_RATIO가 보이지만 컴파일러로 넘어 가기 전에 선행처리자가 ASPECT\_RATIO는 밀어버리고 상수로 바꾸어버리기 때문에 에러라도 발생하게 되면 컴파일러에서는 ASPECT\_RATIO라는 것은 모르고 1.653 밖에 모르기 때문에 1.653때문에 에러가 났다고 에러 메세지를 띄우게 된다.

때문에 만약 자신이 짠 코드가 아니라면 디버깅에 많은 시간이 소요 될 수 있는 문제를 야기할 수 있습니다.

이 문제의 해결방법은 **매크로 대신 상수를 쓰는 것** 입니다.

|  |
| --- |
| const double AspectRatio = 1.653; |

AspectRatio는 언어차원에서 지원하는 상수 타입의 데이터 이기 때문에 당연히 컴파일러 눈에도 보이며 기호 테이블에도 당연히 들어갑니다.

위에 경우처럼 상수가 부동소수점 실수 타입일 경우에는 컴파일을 거친 최종 코드의 크기가 #define을 썼을 때보다 작게 나올 수 있습니다. 매크로를 쓰면 코드에 ASPECT\_RATIO가 등장하기만 하면 선행 처리자에 의해 1.653으로 모두 바뀌면서 결국 목적 코드 안에 1.653의 사본이 등장 횟수만큼 들어가게 되지만, **상수타입의 AspectRatio는 아무리 여러 번 쓰이더라도 사본은 딱 한 개만 생기기 때문**입니다.

#define으로 상수를 교체하는 것에는 **두 가지 주의 사항이 존재**

1. **상수 포인터(constant pointer)를 정의하는 경우**입니다. 상수 정의는 대개 헤더 파일에 넣는 것이 상례이므로 (다른 소스파일이 이것을 include해서 쓰게 되지요) 포인터는 꼭 const로 선언해 주어야 하고, 아울러 포인터가 가리키는 대상까지 const로 선언하는 것이 보통입니다. 어떤 헤더 파일 안에 char\* 기반의 문자열 상수를 정의한다면 다음과 같이 const를 두번 써야 한다는 말입니다.

|  |
| --- |
| const char\* const authorName = "Scott Meyers" |

\*기호테이블(심볼테이블)

심볼 테이블은 내부에서 사용된 심볼들의 정보가 들어있는 테이블

- 심볼이란 메모리 상의 주소를 가지는 모든 것들을 의미하는데 전역변수나 함수, 구조체 정적변수들의 이름, 크기와

종류, 전역인가 지역인가를 의미하는 스코프 범위 등의 정보가 들어있다.

AspectRatio는 double이라는 변수지만 컴퓨터는 숫자로만 기억한다. 그래서 **숫자와 변수 명을 매칭**시키기 위해서

**심볼테이블**이라는 것을 생성한다. 심볼테이블은 주소기반으로 **변수가 선언되면 테이블을 생성하여 변수의 타입과 이름, 주소를 생성한다**. 그리고 변수가 초기화되면 심볼 테이블에서 변수를 찾아 해당 주소에 값을 넣는다.

**2. 클래스 멤버로 상수를 정의하는 경우**, 즉 클래스 상수를 정의하는 경우입니다.

어떤 상수의 유효범위를 클래스로 한정하고 할 때는 그 상수를 멤버로 만들어야 하는데, 그 상수의 사본 개수가 한 개를 넘지 못하게 하고 싶다면 **정적(static)멤버로** 만들어야 합니다.

|  |
| --- |
| class GamePlayer{  private:  static const int NumTurns = 5; // 상수선언  int scores[NumTurns]; // 상수를 사용하는 부분  }; |

위에서 보신 NumTurns는 '선언(declaration)'된 것입니다. '정의'가 아니니 주의하세요.

C++에서는 여러분이 사용하고자 하는 것에 대해 '정의'가 마련되어 있어야 하는 게 보통이지만, 정적 멤버로 만들어지는 정수 류(각종 정수 타입, char, bool 등) 타입의 클래스 내부 상수는 예외입니다. 이들에 대해 주소를 취하지 않는 한, 정의 없이 선언만 해도 아무 문제가 없게 되어 있습니다. 단, 클래스 상수의 주소를 구한다든지, 여러분이 주소를 구하지 않는데도 여러분이 쓰는 컴파일러가 잘못 만들어진 관계로 정의를 달라고 떼쓰는 경우에는 별도의 정의를 제공해야 합니다. 아래가 그 예입니다.

|  |
| --- |
| const int GamePlayer::NumTurns;   // NumTurns의 정의 |

이때 클래스 상수의 정의는 구현 파일에 둡니다. 헤더 파일에는 두지 않습니다. 정의에는 상수의 초기값이 있으면 안되는데, 왜냐하면 클래스 상수의 초기값은 해당 상수가 선언된 시점에서 바로 주어지기 때문입니다.

(즉, NumTurns는 선언될 당시에 바로 초기화 된다는 것입니다.)

그런데 주의할 것이 하나 있습니다. 혹시 클래스 상수를 #define으로 만드실 생각을 해보는 건 아니죠?

방법 자체가 말이 안 됩니다. 대저 #define은 유효범위란 게 뭔지도 모르는 피조물이니까요. 매크로는 일단 정의되면 컴파일이 끝날 때까지(중간에 #undef되지만 않으면) 유효하다는 점을 기억해 두시기 바랍니다.

정리하면 #define은 클래스 상수를 정의하는데 쓸 수도 없을 뿐 아니라

어떤 형태의 캡슐화 혜택도 받을 수 없습니다.

말하자면 **'private'성격의 #define 같은 것은 없다는 이야기** 입니다.

물론, 이와 대조적으로 상수 데이터 멤버는 캡슐화가 되죠 NumTurns가 그거잖아요.

**\*나열자 둔갑술(enum hack)**

enum 타입의 값은 int가 놓일 곳에도 쓸 수 있다는 C++의 진실을 적극 활용하는 것입니다.

이 나열자 둔갑술을 알아 두는 것이 여러가지 이유로 피가 되고 살이 됩니다.

**나열자 둔갑술은 동작방식이 const보다는 #define에 더 가깝습니다**.

예를 들어, const의 주소를 잡아내는 것은 합당하지만, enum의 주소를 취하는 일이 불법이며 #define의 주소를 얻는 것 역시 맞지 않습니다.

혹시 여러분이 선언한 정수 상수를 가지고 다른 사람이 주소를 얻는다든지 참조자를 쓴다든지 하는 것이 싫다면 enum이 아주 좋은 자물쇠가 될 수 있습니다. 또한 제대로 만들어진 컴파일러는 정수 타입의 const객체에 대해 저장공간을 준비하지 않겠지만(그 객체의 포인터나 참조라를 만들지 않는한) 조금 편찮게 만들어진 컴파일러는 반대로 동작 할 수도 있어서, 소심한 프로그래머 측면에선 양쪽모두에 대해 안전하게 const 객체에 대한 메모리를 만들지 않는 방법을 쓰고 싶을 것 입니다.

\***enum은 #define처럼 어떤 형태의 쓸데없는 메모리 할당도 절대 저지르지 않습니다.**

나열자 둔갑술을 알아야 하는 두 번째 이유는 지극히 실용적인 이유인데, 상당히 많은 코드에서 이 기법이 쓰이고 있으므로 혹시 이런 것을 발견하면 쉽게 알아보도록 눈을 단련시켜두라는 것!

\* 템플릿 메타프로그래밍의 핵심 기법이기도 한다.

**#define에 다른 오용사례는 매크로함수입니다.**

함수처럼 보이지만 **함수호출 오버헤드를 일으키지 않는 매크로를 구현하는 것**이지요.

//a와 b 중에 큰 것을 f에 넘겨 호출합니다.

|  |
| --- |
| #define CALL\_WITH\_MAX(a,b) f((a) > (b) ? (a) : (b)) |

이런 식의 **매크로는 단점이 한두 개가 아닙니다**.

\* 이런 매크로를 작성할 때는 매크로 본문에 들어있는 인자마다 반드시 괄호를 씌워주는 센스를 잊지 말아야 합니다.

(이게 안되있으면, 표현식을 매크로에 넘길 때 골치 아픈 일이 발생할 수 있으니까요.)

int a = 5, b = 0;

CALL\_WITH\_MAX(++a, b); // a가 두 번 증가합니다.

CALL\_WITH\_MAX(++a, b+10); // a가 한 번 증가합니다.

이런 골치 아픈 일을 해결할 수 있는 방법은 **인라인 함수에 대한 템플릿을 준비하는 것**입니다.

|  |
| --- |
| template<typename T> // T가 정확이 무엇인지 모르기 때문에, 매개변수로 상수 객체에 대한 참조자를 씁니다.  inline void callWithMax(const T& a, const T& b)  {  f(a > b ? a : b);  } |

이 함수는 템플릿이기 때문에 **동일 계열 함수군(family of function)을 만들어 냅니다**. 동일한 타입의 객체 두 개를 인자로 받고

둘 중 큰 것을 f에 넘겨서 호출하는 구조입니다. 보시면 알겠지만 함수 본문에 괄호로 분 칠을 해 댈 필요가 없고

인자를 여러 번 평가할지도 모른다는 걱정도 없어집니다. 그뿐 아니라 **callWithMax는 진짜 함수이기 때문에 유효범위 및 접근규칙을 그대로 따라갑니다**.

**\* const, enum, inline의 친절한 손길이 우리 가까이에 있다는 사실을 늘 유념해 두면 선행 처리자(특히 #define)를 꼭 써야 하는 경우가** **많이 줄어들게 됩니다. 하지만 #include, #ifdef/#ifndef도 계속 사용되고 있습니다.**

**요약**

\* 단순한 상수를 쓸 때는, #define보다 const 객체 혹은 enum을 우선 생각합시다

\* 함수처럼 쓰이는 매크로를 만들려면, #define 매크로 보다 인라인 함수를 우선 생각해 봅시다.