**항목 3. 낌새만 보이면 const를 들이대 보자**

낌새만 보이면 const를 들이대 보자!

|  |
| --- |
| char greeting[] = “hello”    char \*p = greeting // 비상수 포인터, 비상수 데이터  const char \*p = greeting // 비상수 포인터, 상수 데이터  char \* const p = greeting // 상수 포인터, 비상수 데이터  const char \* const p = greeting // 상수 포인터, 상수 데이터 |

**const 키워드가 \*표의 왼쪽에 있으면 포인터가 가리키는 대상이 상수인 반면,**

**const가 \*표의 오른쪽에 있는 경우엔 포인터 자체가 상수입니다.**

|  |
| --- |
| void f1(const Widget \*pw); // f1은 상수 Widget 객체에 대한 포인터를 매개변수로 취한다.  void f2(Widget const \*pw); // f2도 그렇고요. |

STL반복자를 const로 선언하는 일은 포인터를 상수로 선언하는 것과 같다(T\* const 포인터).

- 반복자는 **자신이 가리키는 대상의 멤버가 변경 되는 것은 불가능 하지만**

- 반복자가 **가리키는 대상 자체는 변경이 가능하다**.

만약 **변경이 불가능한 객체를 가리키는 반복자**(즉, const T\* 포인터의 STL 대용물)가 필요하다면 **const\_iterator**를 쓰면 된다.

|  |
| --- |
| std::vector<int> vec;  …  const std::vector<int>::iterator iter = vec.begin(); // iter는 T\* const처럼 행동한다.  \*iter = 10; // **OK, iter가 가리키는 대상을 변경**합니다.  ++iter; **// 에러! Iter는 상수입니다.**    std::vector<int>::const\_iterator cIter = vec.begin(); // cIter는 const T\* 처럼 행동한다.  \*cIter = 10; **// 에러! \*cIter가 상수이기 때문에 안 됩니다.**  ++cIter; **// 이건 문제없습니다. cIter를 변경하니까요.** |

**const 키워드가 있고 없고의 차이만 있는 멤버 함수들은 오버로딩이 가능합니다.**

|  |
| --- |
| class TextBlock {  public:  …  const char\* operator[](std::size\_t position) const // **상수 객체에 대한 operator[]**  { return text[position]; }  char\* operator[](std::size\_t position) // **비 상수 객체에 대한 operator[]**  { return text[position]; }  private:    std::string text;  } |

**비트수준 상수성(bitwise constness) == 물리적 상수성(physical constness)**

- 어떤 멤버함수가 그 객체의 어떤 데이터 멤버도 건드리지 않아야(정적 멤버는 제외) 그 **멤버 함수가 ‘const’임을 인정하는 개념.**

|  |
| --- |
| class CTextBlock {  public:  …  char& operator[](std::size\_t position) **const**// 부적절한 (그러나 비트수준  { return pText[position]; }            // 상수성이 있어서 허용되는 operator[]의 선언    private:    char \*pText  } |
| const CTextBlock cctb(“Hello”); **// 상수 객체를 선언.**  char \*pc = &cctb[0]; **// 상수 버전의 operator[]호출하여 cctb의 내부 데이터에 대한 포인터 획득**  \*pc = ‘J’; |
| 어떤 값으로 초기화된 상수 객체를 하나 만들어 놓고 이것에다 상수 멤버함수를 호출했더니 값이 변해 버린 것이다. |

**논리적 상수성(logical constness)**

**-** 비트수준 상수 성이 갖는 문제점을 보완하기 위해 나온 개념.

상수 멤버함수라고 해서 객체의 한 비트도 수정할 수 없는 것이 아니라 일부 몇 비트 정도는 바꿀 수 있 되, 그것을 사용자측에서 알아채지 못하게만 하면 상수 멤버 자격이 있다는 것입니다.

|  |
| --- |
| class CTextBlock {  public:   …  std::size\_t length() const;    private:  char\* pText;  **mutable std::size\_t textlength**; // 바로 직전에 계산한 텍스트 길이  **mutable bool lengthIsValid**; // 이 길이가 유효한가?  } |
| std::size\_t CTextBlock::length() const  {  if (!lengthIsValid) {    textLength = std::strlen(pText); // 에러! 상수 멤버함수안에서는 textLength및 lengthIsValid에 대입할 수 없다.  lengthIsValid = true;  }  Return textLength;  } |
| 컴파일러는 에러를 쏟아낼 게 뻔합니다.  컴파일러의 검열을 통과하려면 비트 수준의 상수성이 지켜져야 합니다.    해답은 const에 맞서는 C++의 변신 비기, mutable을 사용하는 것입니다.  Mutable은 비정적 데이터 멤버를 비트수준 상수성의 족쇄를 풀어 주는 아름다운 오색약수 같은 키워드 입니다. |

**요약**

\* const를 붙여 선언하면 컴파일러가 사용상의 에러를 잡아내는데 도움을 준다.const 는 어떤 유효 범위에 있는 객체에도 붙을 수 있으며, 함수 매개변수 및 반환 타입에도 붙을 수 있으며, 멤버 함수에도 붙을 수 있습니다.

\* 컴파일러 쪽에서 보면 비트 수준 상수성을 지켜야 하지만, 여러분은 개념적인(논리적인) 상수성을 사용해서 프로그래밍 해야 합니다.

\* 상수 멤버 및 비상수 멤버 함수가 기능적으로 서로 똑같게 구현되어 있을 경우에는 코드 중복을 피하는 것이 좋은데, 이 때 비 상수 버전을 호출하도록 만든다.