10-Aug-11

Const correctness

Clarence Bartenhagen

GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

Motivation

- Why use *const*?
  - Avoid accidental mutation
  - Document code
  - No difference in program speed
  - Builds character

Simple uses

- Declare constant variable
  - `const int answer=42;`
  - Better than `#define answer 42`
    - More helpful error messages

Use with pointers

- Rule: ‘const’ applies to type to immediate left
  - If nothing is there, then immediate right
- `const int * ptr_const_int`
- `int const * ptr_const_int2`
  - both point to constant int
  - choose a style and be consistent

- `int * const const_ptr_int`
  - constant pointer to variable integer

- `const int * const const_ptr_const_int`
- `int const * const const_ptr_const_int2`
  - constant pointer to constant integer

Use with references

- `int const & intRef`
  - intRef is a reference to a constant integer
- `int & const intRef`
  - Confusing and redundant

Const member methods

```
class Pretzel {
public:
   void admire() const;
   void consume();
   …
   }
```

- Example:

```
   void snackTime( Pretzel& yours, Pretzel& notyours) {
       yours.admire();  // fine. Doesn’t change changeable
      yours.consume(); // find. Changes a  changeable

      notyours.admire(); // fine. Doesn’t change an unchangable
      notyours.consume(); //WRONG
   }
```

Return-by-reference and const method

```
class Person {
public:
  std::string const& nameGood() const;
  std::string      & nameBad() const;
  ...
};

void awesomeMethod(Person const& p)
{
  p.nameBad() = “Clarence";
}
```

- Non-const reference allows accidental modification
- complier might not catch this

Const overloading

- Similar to normal function overloading
- Rule of thumb: subscript operators usually come in pairs

```
class PretzelList {
   public:
   Pretzel const& operator[](int index)const;
   Pretzel & operator[](int index);
   …
   }
```

Const overloading (cont.)

```
void f(PretzelList & a )  // non-const

  // reference is not modified – OK!
  Pretzel r = a[6];
  a[6].admire();

   // reference is modified – OK!
  Pretzel s;
  a[6] = s;
  a[6].consume();
```

Const overloading (cont.)

```
void f(PretzelList const& a )  // NOTE:const

  // reference is not modified – OK!
  Pretzel r = a[6];
  a[6].admire();

   // reference is modified – ERROR!
  Pretzel s;
  a[6] = s;
  a[6].consume();
```

References

- http://www.parashift.com/c++-faq-lite/const-correctness.html
- http://www.cprogramming.com/tutorial/const_correctness.html
- http://duramecho.com/ComputerInformation/WhyHowCppConst.html
- `Pretzel& yours`

Thank You

Fragen?