const

作用

- const 为修饰符，用于修饰对象或者变量，表示其不可被修改
- 定义常量 — const 最为直接的使用方式，例如定义 PI 的值，程序中需要使用的常量等。
- 防止变量被修改 — 在函数声明中指出不可被修改的参数，可防止实参（引用 or 指针指向的对象）被修改，降低出现 BUG 的概率，函数返回值同理

多文件编程中的使用

- const 所修饰的对象默认为当前文件的局部变量
- 因此，若需要在多个文件中使用定义的 const 变量，需要显式地声明 extern
  - 定义 — extern const double PI = 3.14;
  - 使用 — extern const double PI;

修饰引用与指针

- 引用
  - 引用本身就带有"const属性"。引用一经初始化便不可以再绑定到其它对象身上。从这一点上来看，引用本身就是 const 的
  - const int& i = j 表示我们不能通过引用 i 去修改 j 的值，至于 j 能不能被修改，由 j 是否被 const 修饰所决定
    - const int j = 1024;
const int& i = j;
    - 这种情况下，不管 i 是不是 const 引用，都无法修改变量 j 的值
- 指针
  - 指向常量的指针
    - 即无法通过该指针对所指的变量进行修改，但指针可重新指向其它的内存区域
      - const char *p;
      - char const char *p;
    - 因为指向常量的指针可以随时的指向其它内存区域，所以此类指针不需要初始化
  - 常量指针
    - 可对指针指向的变量进行修改，但无法使该指针指向其他内存区域
      - char * const p = &a;
      - p = &d; // wrong
    - 此类指针无法转而指向其它内存区域，所以必须初始化

观察 const 修饰的是指针所指向的对象还是指针本身

函数中的const

- **当我们在函数中不会改变某一个函数参数的值时，请使用 const 修饰**
  - 这不仅仅是使得程序更加健壮，同时也会避免错误
  - void print(string &s) {
  cout<<s<<endl;
}
    - 当我们调用 print("Hello")，也就是使用字符串字面量的方式进行参数传递时，编译器将会报错
    - 这是因为C++中字符串字面量的类型为 const char *
  - void print(const string &s) {
  cout<<s<<endl;
}
    - 此时可使用字符串字面量作为函数参数进行调用
- 同样地，对于函数返回值而言，如果返回的对象不允许被修改，也请使用 const 进行修饰
- 另外值得一提的是常量指针在函数参数中没有任何意义，因为函数调用的第一步就是使用实参初始化形参，可以简单的认为这是一种拷贝，拷贝值，拷贝指针，引用类型则重新为变量生成一个别名（alias）
  - void print(char *const s) {/*.....*/}
此时的 const 没有任何意义，应避免
- 函数重载中的const
  - 顶层const与底层const
    - 顶层const
      - 修饰对象、指针，指对象或指针本身不能被修改
      - 例如 const int a, int * const p, const Complex pc;
    - 底层const
      - 专用于引用和指针，指指针所指向的对象不可被修改，或者被引用的对象不可被修改
      - 例如 const int *p, const vector<int>& nums
  - **函数会忽略顶层const**
    - void f(const int a);
    - void f(int a);

    这两个函数定义是相同的
原因在于对于不管是对象还是指针，函数都会对其进行拷贝初始化
  - **函数不会忽略底层const**
    - void g(const char *s);
    - void g(char *s);

    这两个函数定义是不同的。换而言之，可进行函数重载

const 成员变量

- const 类成员变量所表达的意思与 const 修饰普通变量所表达的意思一致: 初始化后不可被修改
- **const 成员变量只能通过 initialization list 进行初始化**
  - class A {
private:
  const int number;
public:
  A(int nbr) {number = nbr;}
};
    - 编译将产生错误
    - constructor for 'A' must explicitly initialize the const member 'number'
  - class A {
private:
  const int number;
public:
  A(int nbr) : number(nbr) {}
};
    - 编译通过，且 number 被正确的初始化

const 成员函数

- 我们可以将 const 修饰符添加到函数声明的末尾，用于表示当前成员函数不会修改成员变量
  - class A {
private:
  int number;
public:
  int getNbr() const { return number; }
};
  - **如果我们在 getNbr() 函数内尝试对任意的成员变量进行修改的话，编译器将抛出错误**
- 因此，对于 getXXX 类的成员方法，需使用 const 修饰符进行限定，确保函数内部不会对成员变量进行修改
- 另外一点就是 const 成员函数和不带 const 修饰的成员函数属于不同的函数原型，即可以重载
  - int getNbr() const { return number; }
  - int getNbr() { return number; }

  不同的函数原型
- const 的位置
  - const getNbr() {/*...*/} — 表明函数返回值不可被修改
  - getNbr() const {/*...*/} — 专用于类成员函数中，表示函数内部不会对成员变量进行修改