# C/C++

## const关键字

const关键字常用于数组边界和switch条件分支标号：

const type variable; //常变量等价于 type const variable

const type &reference; //常引用等价于 type const& variable

ClassName const Object; //常对象等价于const ClassName Object

ClassName::func(signature) const; //常成员函数

Type const ArrayName[size]; //常数组等价于 const type ArrayName[size]

const type\* pointer; //常指针定义方法1

type const\* pointer; //常指针定义方法2

C标准中，const变量定义为全局作用域，而C++中则要视其定义位置而定；

使用指针时涉及到两个关键要素：指针本身(地址)和指针所指向的对象(数据类型)；

将一个指针声明为const类型，是限制其指向的对象为指定类型；

将指针本身声明为type \*const pointer，则是将指针(地址)本身声明为常量；

可以将一个非const对象的地址赋给const对象的指针，反之不可；

void Func(const A \*arg); //传指针传递参数

void Func(const A &arg); //传引用传递参数

将函数传入参数声明为const，提高函数运行效率且禁止修改其内容；

在C++中使用const关键字修饰成员函数，则const对象只能访问const成员函数，而非const对象可以访问所有成员函数；const对象的成员不可修改，const成员函数不可修改对象的数据，无论对象是否具有const限定；

常量指针与指针常量：

const int \*p1 = new int(10); //指向常量的指针，不可通过p1修改其指向的内容

int \*const p2 = new int(10); //指向int类型的指针常量，不可指向其他内存

程序载入内存时，会分配常量区存储常量，可以通过直接修改变量数值或通过另外一个非常量指针进行修改：

int a = 10;

const int \*p = &a;

// error

// \*p = 100;

//直接修改变量

a = 100;

//

int \*pi = (int\*) p; //第三方指针

\*pi = 100;

函数参数中的指针常量表示不允许此指针指向其他内容：

void func(int \*const pt){

int \*p = new int(10);

pt = p; //error cannot point to a new pointer

}

函数参数中的常量指针表示此参数不允许被修改：

void func(const int \*pt){

\*pt = 100; //error cannot be modified

}

若将参数中的指针赋给一个新的指针，则会修改其指向的内容；

常量与引用的关系：引用就是变量的别名，常量引用即不允许此引用成为其他变量的别名；

int a = 10;

const int& ra = a; //常量引用，不可通过此引用改变其对应的内容

// int& const ra = a; //error

若不希望函数调用者修改参数本身的值，最可靠的方法是传递引用；

void func(const int& arg){

// arg = 100; //error cannot be modified

}

系统在加载程序时，会将内存分为4个区域：堆、栈、数据段和代码段，使用常量的方式保护数据是通过编译器语法规则限定来实现的，但是仍然可以通过以下方式完成修改：

const int a = 10;

int \*pa = (int\*) &a; //可以通过定义其他指针的方法修改常量数值

\*pa = 100;

常量函数在C++中可防止类的 数据成员被非法访问，将类的成员函数分为两类：常量成员函数和非常量成员函数；

class Test{

public:

void func() const;

private:

int intValue;

};

void Test::func() const{

intValue = 100; // raise error 常量函数尝试改变数据成员intValue的数值

//编译时引发异常

}

class Fred{

public:

void inspect() const;

void mutate();

};

void UserCode(Fred& changeable, const Fred& unChangeable){

changeable.mutate(); //非常量对象调用非常量函数

changeable.inspect(); //非常量对象调用常量函数

unChangeable.mutate(); //常量对象调用非常量函数，错误

unChangeable.inspect(); //常量对象只能调用常量函数

}

常量函数包含一个this的常量指针：

void inspect(const Fred\* this) const;

void mutate(Fred\* this);

对于常量函数，不能通过this指针修改对象对应的内存，但可以通过重新定义指针来修改内存中的内容；

void func(const int \*pt){

int \*ptr = (int\*) pt;

\*ptr = 100;

}

通过常量对象调用非常量函数将产生语法错误；对于常量函数，不能通过this指针修改对象对应的内存块；但是可以通过this指针重新定义一个指向同一内存单元的指针；

void Fred::inspect() const{

Fred \*pFred = (Fred\*) this;

pFred->intValue = 50;

}

对于常量对象，可以构造新指针，指向常量对象所在的内存单元；

C++允许在类的数据成员定义前加上mutable关键字以实现成员在常量函数中可修改：

class Fred{

public:

void inspect() const;

private:

mutable int intValue;

};

常量函数的重载问题：

class Fred{

public:

void func() const;

void func();

}

void Fred::func() const{}

viod Fred::func(){}

void UserCode(Fred& fred, const Fred& cFred){

fred.func(); //call func()

cFred.func(); //call func() const

}

int main(int argc, char\*\* argv){

Fred fred;

UserCode(fred, fred);

return 0;

}

当存在 同名同参数相同返回值的函数重载时，具体调用哪个函数取决于调用对象是常量对象还是非常量对象；

常量返回值：不希望函数调用者修改函数的返回值，将函数返回一个常量。

* Const常量与宏定义的区别：const常量有数据类型(编译器做类型检查)，宏定义没有数据类型(编译器仅作简单的字符替换)；
* Const修饰类的数据成员，const数据成员只在对象的生命周期内是常量，而对整个类而言是可变的，因此，const数据成员的初始化只能在类的构造函数的初始化列表中进行，不能在类中指定；

class A{

const int size = 100; //错误定义

int array[size]; //错误定义，未知的size

}

要在类中定义常量，应当使用enum类型：

class A{

enum{size1=100, size2=200};

int array1[size1];

int array2[size2];

}

枚举常量不占用对象的内存空间，在编译时全部求值，枚举常量隐含使用整型类型，最大数值有限，不能表示浮点型数据；

* const初始化：

T b;

const T a = b; //非指针const常量初始化

T\* p = new T();

const T\* pc = p; //指针常量初始化

const T\* pb = new T(); //同上

T r;

const T& rf = r; //rf只能访问const成员函数

const T\* c = new T();

T\* e = c; //声明指针的目的是要修改其指向的内容，但此处指向常量

T\* const c = new T();

T\* e = c; //声明指针指向的内容可变

* 参数const通常用于参数为指针或引用的情况下使用，若参数为值传递，则函数会自动产生临时变量复制，保护被传递对象的属性；
* 对于非内部数据类型的参数传递，将参数值传递改为const引用传递可以提高效率；而对于内部数据类型的参数传递，避免改为const引用传递，以免降低程序的可理解性；
* 修饰返回值的const关键字，const T func(); const T\* func();对函数返回值进行保护；
* 函数的返回值声明为const，通常用于操作符重载，若使用const修饰函数的返回值类型，则返回的const Object只能访问类中的公有数据成员和const成员函数，且无法对其进行赋值操作；指针传递函数返回值加const修饰符，则函数返回内容(指针)不能被修改，只能赋给同样加const修饰符的同类型指针；

const char\* GetString(void);

char \*str = GetString(); //compile error

const char\* str = GetString(); //compile success

* 函数返回值引用传递通常用于类的赋值函数中，实现链式表达：

class A{

A &operator = (const A& other); //赋值操作符

}

A a, b, c; //class A Object a, b, c

…

a = b = c; //all right

(a = b) = c; //legal but unusual

a.operator = (b)的返回值是const类型的引用，不可再次赋值给c；

返回值的内容不允许被修改；

* 类成员函数const关键字的使用：

class Stack{

public:

void Push(int elem);

int Pop(void);

int GetCount(void) const; //const成员函数

private:

int m\_num;

int m\_data[100];

};

//公有成员函数定义在类外

int Stack::GetCount(void) const{

++m\_num; //编译错误，企图修改数据成员

Pop(); //编译错误，企图调用非const成员函数

return m\_num;

}

* 在C中，const是一个不能被改变的普通变量，需要占用存储空间，编译器不知道编译时的数值，且数组下标必须为常量；而在C++中，将const看做编译时的常量，不为其分配存储空间，只是在编译时将其数值存储在名字列表中；
* 在C语言中，const int size; 语句正确，在C++中不正确；C编译器默认使用外部连接，将其看做声明，可在其他地方分配内存空间；C++编译器默认使用内部链接，必须在声明时初始化，将const对象默认看做文件的局部变量；
* 在C++中，是否为const 分配内存空间取决于是否添加extern关键字或者取const变量地址；

## 1.2函数传值与传引用

## 1.3 字符串转整型问题

## 1.4 操作符重载

# Python

# Algorithm

## 最大堆

## 红黑树

## 算法复杂度

# OS