# 常函数

int func() const{};

## 常成员函数不能更新对象的数据成员 ；

## 不能调用该类中没有const修饰的成员函数 。

## 不能返回类的属性的指针和引用，防止被修改，但可以返回类似于const int \*类型的值。因为：int \* const 类型的值不能修改指针，但可以修改指针指向的值

**而const int \*类型的值不能修改指针指向的值，却可以修改指针**

## 同理，const int& func() const{},也是合理的

# 动态分配二维数组

int \*\*p = new int\*[16];

for(int i = 0; i< 16: ++i)

p[i] = new int[5];

for(int i = 0; i< 16; ++i)

delete[] p[i];

delete[] p;

# C++正则化regex

## 1.注解。

1. [\\是转义符](file:///\\是转义符)
2. pat=”[\\w+](\\\\w+)”; line=”{“ 时，match中仍然又一个值，为空字符串。
3. 用regex\_search时：

"\\w+\\s\*((\\(\\w+,\\d+\\)\\s\*){0,})"与 “[\\w+\\s\*(\\(\\w+,\\d+\\)\\s\*){0,}](file:///\\w+\\s*(\\(\\w+,\\d+\\)\\s*)%7b0,%7d)”

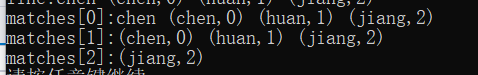
不同，在前一个字符串中，([\\(\\w+,\\d+\\)\\s\*](file:///\\(\\w+,\\d+\\)\\s*))和((\\(\\w+,\\d+\\)\\s\*){0,})可分别作两个子串，并可以独立地参加匹配过程。例如：

string pat = "\\w+\\s\*((\\(\\w+,\\d+\\)\\s\*){0,})";

pattern = pat;

string line = "chen (chen,0) (huan,1) (jiang,2)";

的结果为：



string pat = "\\w+\\s\*(\\(\\w+,\\d+\\)\\s\*){0,}";

pattern = pat;

string line = "chen (chen,0) (huan,1) (jiang,2)";

的结果为：  
 

1. 若pat=”\\w+”;line=”aaaa”;

结果match中只有“aaaa”;

**“\*”,”+”,”?”都是贪婪的，尽可能多的匹配字符串， 若解决这个问题，可以在它们后面加“？”，就可以转化为最小模式或非贪婪模式**

1. if(regex\_search(str\_1, mat, pattern))  
             for(smatch::iterator iter=mat.begin() ; iter!=mat.end() ; ++iter)  
               cout << \*iter <<endl;

只会返回匹配到的第一个字符串

string str\_1 = "chen1234huan12345jiang1234567";  
 21     string::const\_iterator start = str\_1.begin();  
 22     string::const\_iterator end = str\_1.end();  
 23      
 24     smatch mat;  
 25     while(regex\_search(start, end, mat, pattern))  
 26     {    
 27         string msg(mat[0].first, mat[0].second);  
 28         cout << msg<< endl;  
 29         start = mat[0].second;  
 30     }

则会返回所有想要地子串

1. “/”无特殊含义
2. 字符串替换：test\_str = regex\_replace(test\_str,reg1,t)
3. 为了使regex不匹配模式串中用（）标识的子串，需要这样写（?:expression）
4. “\\bCha”匹配“Chank”,但不匹配“hChal”

如果它位于要匹配的字符串的开始，它在**单词**（而不是字符串）的开始处查找匹配项。如果它位于字符串的结尾，它在单词的结尾处查找匹配项

应用：可以查找两个相同的单词

Line=”Is the up up”

pat = “\\b([a-z]+) \\1\\b”

可以找出up up

1. \\n 指定第n各子匹配项

## 2. 基本语法

# string库

1. string.size()可得到字符串的字符数
2. string s(c,3) 把c的前3各字母赋给s

**注意：若iter是字符串的迭代器，则可以通过cout<<&\*iter;的方式输出子字符串**

若c是字符数组，则可以通过cout<<&c[1];的方式输出子

3. s1.compare(s2)若相等，则返回0，s1<s2,返回-1，s1>s2, 返回1.

4. s1.rfind(s2, i), 指的是从s1[i]后面的字符串和s2整个串进行比较。

# vector

## vector.front()和vector.back()返回vector容器中首尾值的引用。

1. vector作为函数参数和返回值时， 最好带&.
2. #<algorithm>

* reverse( vec.begin(), vec.end() ) //逆序排列
* sort( vec.begin(), vec.end() ) //默认从小到大排序

若要从大到小排序，则重写Comp函数

boo Comp( const int& a, const int& b )

{

Return a>b;

}

# priority\_queue

默认队头元素是最大的， 但可以改为从小到大，两种方式

1. struct cmp

{

bool operator ()(int a, int b){

return a>b;

}

}

priority\_queue<int, vector<int>, cmp>;

1. priority中类型为node型时， 可以重载<运算符

bool operator <(const Edge& e){

return this.cost>e.cost;  
}