**目录**

[1.assert用法 2](#_Toc484454560)

[2.<cctype>头文件 4](#_Toc484454561)

[3.C++中的C\_str()函数用法 4](#_Toc484454562)

[4.对C++的atoi(),substr(),c\_str()函数的理解 5](#_Toc484454563)

[5. C++中substr函数的用法 6](#_Toc484454564)

[6.基本的函数概念 6](#_Toc484454565)

## 1.assert用法

assert宏的原型定义在<assert.h>中，其作用是如果它的条件返回错误，则终止程序执行，原型定义：  
#include <assert.h>  
void assert( int expression );

assert的作用是现计算表达式 expression ，如果其值为假（即为0），那么它先向stderr打印一条出错信息，  
然后通过调用 abort 来终止程序运行。

请看下面的程序清单badptr.c：  
#include <stdio.h>  
#include <assert.h>  
#include <stdlib.h>

int main( void )  
{  
       FILE \*fp;  
      
       fp = fopen( "test.txt", "w" );//以可写的方式打开一个文件，如果不存在就创建一个同名文件  
       assert( fp );                           //所以这里不会出错  
       fclose( fp );  
      
       fp = fopen( "noexitfile.txt", "r" );//以只读的方式打开一个文件，如果不存在就打开文件失败  
       assert( fp );                           //所以这里出错  
       fclose( fp );                           //程序永远都执行不到这里来

       return 0;  
}

使用assert的缺点是，频繁的调用会极大的影响程序的性能，增加额外的开销。  
在调试结束后，可以通过在包含#include <assert.h>的语句之前插入 #define NDEBUG 来禁用assert调用，示例代码如下：

#include <stdio.h>  
#define NDEBUG  
#include <assert.h>

用法总结与注意事项：  
1)在函数开始处检验传入参数的合法性  
如:

int resetBufferSize(int nNewSize)  
{  
//功能:改变缓冲区大小,  
//参数:nNewSize 缓冲区新长度  
//返回值:缓冲区当前长度   
//说明:保持原信息内容不变     nNewSize<=0表示清除缓冲区  
assert(nNewSize >= 0);  
assert(nNewSize <= MAX\_BUFFER\_SIZE);

...  
}

2)每个assert只检验一个条件,因为同时检验多个条件时,如果断言失败,无法直观的判断是哪个条件失败

不好: assert(nOffset>=0 && nOffset+nSize<=m\_nInfomationSize);

好: assert(nOffset >= 0);  
assert(nOffset+nSize <= m\_nInfomationSize);

3)不能使用改变环境的语句,因为assert只在DEBUG个生效,如果这么做,会使用程序在真正运行时遇到问题  
错误: assert(i++ < 100)  
这是因为如果出错，比如在执行之前i=100,那么这条语句就不会执行，那么i++这条命令就没有执行。  
正确: assert(i < 100)  
         i++;

## 2.<cctype>头文件

|  |  |
| --- | --- |
| **isalnum(c)** | **如果c是字母或数字，则返回True** |
| **isalpha(c)** | **如果c是字母，则返回true** |
| iscntrl(c) | 如果c是控制字符，则返回true |
| **isdigit(c)** | **如果c是数字，则返回true** |
| isgraph(c) | 如果c不是空格，但可打印，则为true |
| **islower(c)** | **如果c是小写字母，则为true** |
| isprint(c) | 如果c是可打印字符，则为true |
| ispunct(c) | 如果c是标点符号，则为true |
| isspace（c） | 如果c是空白是否，则为true |
| **isupper（c）** | **如果c是大写字母，则为true** |
| **isxdigit(c)** | **如果c是十六进制的数，则为true** |
| **tolower(c)** | **如果c是大写字母，返回其小写字母的形式，否则直接返回c** |
| **toupper(c)** | **如果c是小写字母，返回其大写字母的形式，否则直接返回c** |

附：黑体为常出现的！

空白字符是：空格、制表符、垂直制表符、回车符、换行符、进制符

标点符号：处数字、字母、可打印的空白符（空格）以外的其他字符。

## 3.[C++中的C\_str()函数用法](http://blog.csdn.net/nancy_m/article/details/7583550)

语法:

const char \*c\_str();

c\_str()函数返回一个指向正规C字符串的指针常量, 内容与本string串相同.

这是为了与c语言兼容，在c语言中没有string类型，故必须通过string类对象的成员函数c\_str()把string 对象转换成c中的字符串样式。

注意：一定要使用strcpy()函数 等来操作方法c\_str()返回的指针

比如：最好不要这样:

char\* c;

string s="1234";

c = s.c\_str(); //c最后指向的内容是垃圾，因为s对象被析构，其内容被处理，同时，编译器也将报错——将一个const char \*赋与一个char \*。

应该这样用：

char c[20];

string s="1234";

strcpy(c,s.c\_str());

这样才不会出错，c\_str()返回的是一个临时指针，不能对其进行操作

再举个例子

c\_str() 以 char\* 形式传回 string 内含字符串

如果一个函数要求char\*参数，可以使用c\_str()方法：

string s = "Hello World!";

printf("%s", s.c\_str()); //输出 "Hello World!"

## 4.对C++的atoi(),substr(),c\_str()函数的理解

刚开始接触C++，很多不懂的地方，翻看老大给的代码其中  year = atoi(dateStr.substr(0, 4).c\_str());云里雾里。  
year = atoi(dateStr.substr(0, 4).c\_str());  
里面包含三个函数,分别是atoi(),substr(),c\_str().  
百度：  
atoi()函数原型为: int atoi(char \*str),用途是将字符串转换成一个整数值,str是待转化成整数值的字符串.成功则返回转化后的整数值,失败返回0.  
substr()函数原型为:basic string::substr(string,start,length),也可把string移到外面,为string &a,a.substr(start,length),其中a是待截取的字符串,start表示从截取开始的前一位,length表示截取长度,例如string &a="hello world",则a.substr(6,5)=world.  
c\_str()函数原型为:const char \*c\_str(),如果要将string对象，转化为char\*对象，c\_str()提供了这样一种方法，它返回一个客户程序可读不可改的指向字符数组的指针。

所以year=atoi(dateStr.substr(0,4).c\_str())的作用就是,截取string型的对象dateStr,并转化为char\*对象,然后将此字符串转换成一个整数值,赋值给year(year是int型).

## 5. C++中substr函数的用法

#include<string>  
#include<iostream>  
using namespace std;

main()  
{  
string s("12345asdf");  
string a=s.substr(0,5);       //获得字符串s中 从第0位开始的长度为5的字符串//默认时的长度为从开始位置到尾  
cout<<a<<endl;  
}

输出结果为:

12345

## 6.基本的函数概念

双射是单射+满射同时满足  
映射是单射满射双射的前提，映射是1对1或多对1  
单射只能1对1  
满射是两个\*\*都要取满，就是说两个\*\*中不存在一个元素没有对应关系

单射可以理解为没有多对一，满射可以理解为每个y都有x对应，满射可以理解为单射和满射的合体。