01 Basic Dynamic Array Using Functions

jjcao

Modified from the assignment of Professor Ligang Liu

Why Dynamic Array

- 使用固定大小的数组，如
  - double a[100];

- 有使用不方便的地方：数组大小固定，不灵活；如果程序使用的情况数组元素的个数多于100，则要改数组大小，然后重新编译程序；如果程序使用的情况数组元素的个数很少，则有大量的空间被浪费。

- 为了解决这个矛盾，可以使用动态分配内存空间的方法

Basic Dynamic Array

- 采用一个指针p记录数组的头地址；采用一个整数n记录数据元素的个数

- 程序运行时，从内存中申请n个空间存储数组p。n一旦变化，p即申请新的空间用来存储数组（当然需释放掉以前的内存空间，以免造成内存泄漏）。这样，对数组的操作是通过一些函数来操作：这些函数是操作这个数组的“接口”。

Interface – Data Structure

```
int g_arraysize(0); // size of the array
double *p_arraydata(0); // data of the array
```

Variable Names

Variable names are all **lowercase**, with **underscores** between words.

- **Common Variable names**

  For example:

  string table_name; // OK - uses underscore.

  string tablename; // OK - all lowercase.

  string tableName; // Bad - mixed case.

- **Global Variables**

  no special requirements for global variables, which should be **rare** in any case, but if you use one, consider prefixing it with **g_** or some other marker to easily distinguish it from local variables.

Interface – Data Structure

```
int g_arraysize(0); // size of the array
double *p_arraydata(0); // data of the array
```

Guidelines for Initializing Variables [McConnell2] ch10.3

- ***Initialize each variable as it's declared***
  is an inexpensive form of defensive programming.
- ***Ideally, declare and define each variable close to where it's used***
  [McConnell2] ch10.4 keep Variables "Live" for as Short a Time as Possible

- ***Initialize the variable with a "proper" value!!***

Interface – Functions (Routines)

```
int SetArraySize( int size);
int FreeArray();
int SetValue( int k, double value);
void PrintArray();
```

Function Names

Regular functions have **mixed case**;

- **Regular Functions**
  - Functions should start with a **capital** letter and have a capital letter **for each new word**. No underscores.

  - If your function crashes upon an error, you should append **OrDie** to the function name. This only applies to functions which could be used by production code and to errors that are reasonably likely to occur during normal operation.

  - AddTableEntry()
  - DeleteUrl()
  - OpenFileOrDie()

High-Quality Routines, [McConnell2] ch7

- **Valid Reasons to Create a Routine**
  - *Reduce complexity*
  - *Make a section of code readable*
  - *Avoid duplicate code*
- **Good Routine Names**
  - ***To name a procedure, use a strong verb followed by an object***
    - *PrintDocument()*
    - *CheckOrderInfo()*
  - ***Describe everything the routine does***
    - *ComputeReportTotals()* is not an adequate name for the following routine:
    - *ComputeReportTotalsAndOpenOutputFile()*
  - ***Make names of routines as long as necessary***
- **How Long Can a Routine Be?**
  - 1 || 2 pages, up to 200lines.
  - IBM once limited routines to 50 lines (McCabe 1976).
  - Errors & limit of understandability

Invocations & Testing by Developers

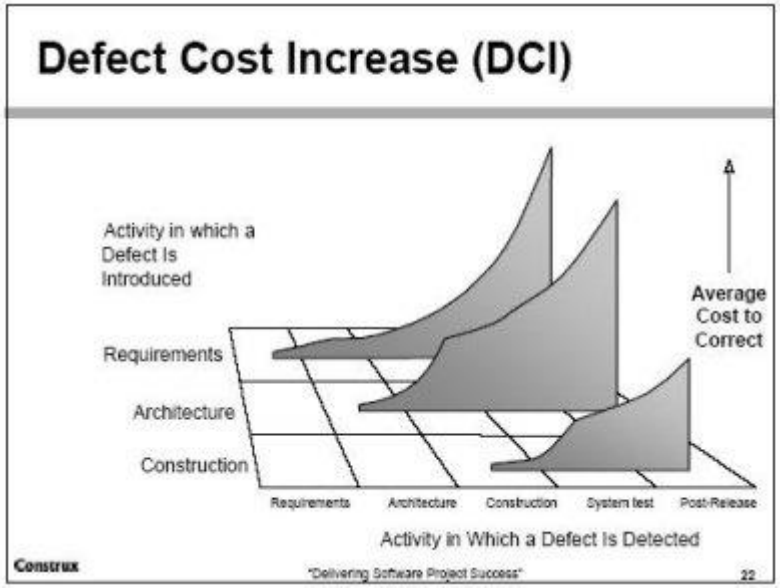
```
#include "BasicDynamicArray_function.h"
#include <iostream>
#include <assert.h>

int main()
{
   SetArraySize(2);
   assert( 2 == g_arraysize); // test-driven development (TDD)
   assert( 0 == p_arraydata[1]);

   SetValue( 0, 1.0);
   SetValue( 1, 2.0);
   assert( 2.0 == p_arraydata[1]);

   PrintArray();

   FreeArray();
   assert ( 0 == g_arraysize);
   assert ( 0 == p_arraydata);

   return 0;
}
```

- **Developer Testing, [McConnell2] ch22**
  - **Test First or Test Last?**
    - **Next page**

- **TDD: see any book about this**
- **Plan Driven**
- **User-case Driven**
  - *The Rational Unified Process: An Introduction*

Test First or Test Last?

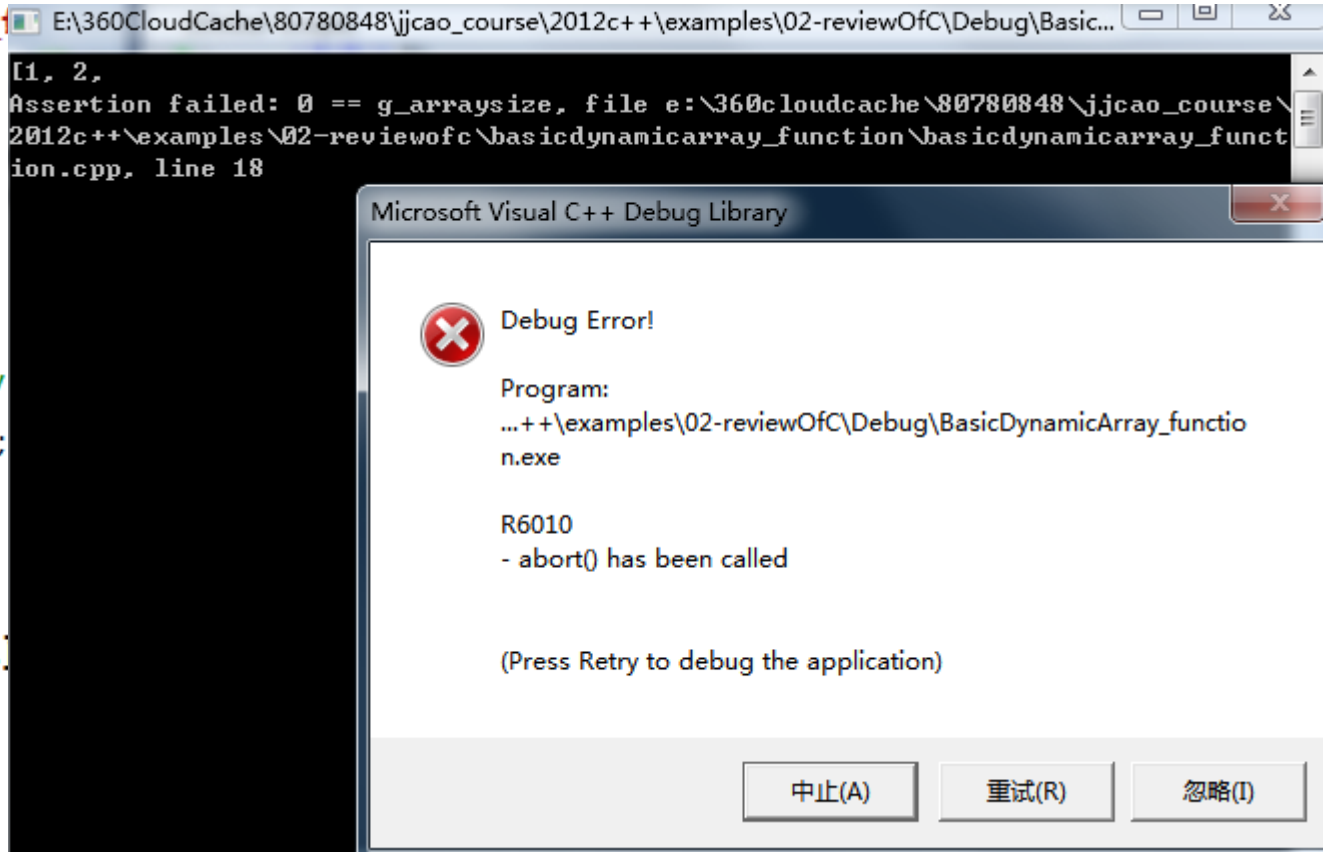
"longer a defect remains undetected, the more expensive it becomes to correct"

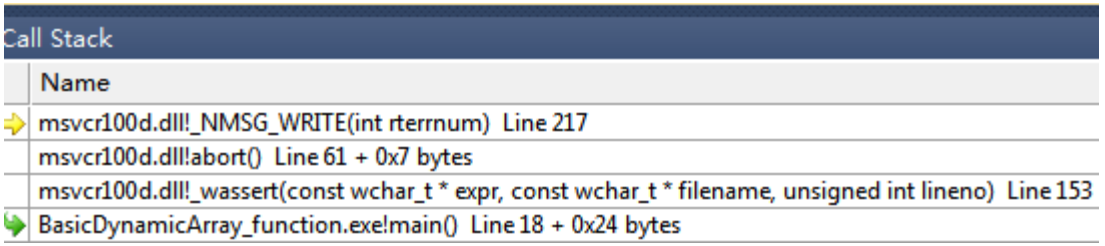


Phases	relative costs (J. Martin, C. McClure (1983) [3]
requirements analysis	3%
specification	3%
design	5%
coding	7%
testing	15%
operations + maintenance	67%

Invocations & Testing by Developers

```
#include "BasicDynamicArray_f
#include <iostream>
#include <assert.h>

int main()
{
    SetArraySize(2);
    assert( 2 == g_arraysize); //
    assert( 0 == p_arraydata[1]);

    SetValue( 0, 1.0);
    SetValue( 1, 2.0);
    assert( 2.0 == p_arraydata[1

    PrintArray();

    FreeArray();
    assert ( 0 == g_arraysize);
    assert ( 0 == p_arraydata);

    return 0;
}
```





Interface – Functions (Routines)

```
int SetValue( int k, double value)
{
   if (NULL == p_arraydata) // Defensive Programming
      return 0;             // 检查输入参数的合法性 [McConnell2] ch8.1

   if ( k < 0 || k >= g_arraysize) // Defensive Programming
      return 0;

   p_arraydata[k] = value;
   return k;
}
```

```
int SetArraySize( int size)
{
   g_arraysize = size;
   p_arraydata = new double [size];
   if (NULL == p_arraydata) // Defensive Programming
   {
      std::cout << " no enough memory" << std::endl;
      return 0;
   }
   return 1;
}
```

Is this enough?

- meticulous
- logic reasoning ability

Coding environment

1. Familiar with Visual C++ 2010 or VC6
2. Master Win32 Console Project

- appendix1_ShortcutKey.pptx
- appendix1_VC.doc
- 00-Win32ConsoleApplication.pptx

A template project with requirements

实现hw01_BasicDynamicArray_Function要求的动态数组

a. **完成满足上述接口的动态数组(Dynamic array)的程序，递交工程文件(*.vcxproj, *.vcxproj.user, (旧的版本*.dsp)) 和源文件(*.h, *.cpp)；**
b. **工程目录中的debug目录删除掉，其他文件压缩打包发给我，参照我们提供的code template。**

评分标准	分数
程序成功运行（通过所有assert）	60
正确使用new/delete，不出现内存泄漏	10
需要遵循基本的编程规范和风格；	20
按时完整的提交项目文件，不包含无用文件	10

抄袭则 平均分配

成绩 = ?/n

A template project: http://yunpan.cn/lk/48jiobzxvc

通过完成该作业需达到的目标：

1. 学习变量、函数的命名规范
2. 了解防御式编程和测试驱动的开发
3. 正确的使用new和delete
4. 学会利用VC2010的debug工具来调试程序；
5. 学会构造各种情况和极端情况来测试程序的鲁棒性。

Coding Standard & Style

- 编写程序一定要养成良好的代码习惯。“程序不是写给自己看的，是写给别人看的”
- 学习规范化编程，养成良好的习惯
  - appendix1_ShortcutKey
  - appendix2_CodingStandard&Style
- 注意：不同的团体、公司都有各自的编程规范，没有统一的编程规范。在编程训练的初期，只要选择自己喜欢的一套比较统一的编程规范来写即可，不必追求与上述文档完全一致的编程规范。以后随着编程水平的提高以及查看其他规范的代码后，可根据自己的喜好和合作团队的要求来不断调整自己的编程规范。

Essentials for high quality coding

- The Power of Variable Names, [McConnell2] ch11
- High-Quality Routines, [McConnell2] ch7
- Self-Documenting Code, [McConnell2] ch32

- Defensive Programming, [McConnell2] ch8
- Developer Testing, [McConnell2] ch22

学习编程的有效方法及TIPS 1

- 编程是很有用的，不论你以后是否从事计算机相关的工作，请相信：较强的编程能力都能对你未来有所帮助！因为它所提供的不仅是个思维的训练，而且还是解决问题的方法的训练。另外，编程能力能帮助你实现你的很多想法。如果你不能实现你的想法，就等于没有想法！
- 如果你对编程没有兴趣，甚至对编程有抵制情绪，请尽早告知（否则是在浪费你的时间，也是在浪费我的时间），我建议你别学C++编程了，做你自己喜欢做的事。生活中没有C++，也同样美好。
- 如果你觉得编程还挺有意思，首先需要有这个信念：学习编程比学习数学容易得多得多！能学好数学的同学一定能学好编程。
- 学习编程有一定的方法，需掌握好方法才能快速提高编程的能力！“方法不对，努力白费”！
- 不要像学数学那样来学习编程，即不要花很多时间（比如，花很多周，甚至1－2个月来研读某教材）来研究C++的各种语法特性；只要初步了解一定的基本语法，就可以开始动手写程序；通过边做边查找来体会各种知识。没有必要去研究(i+++++i)这种钻牛角尖的语句，因为没人会写出这种代码的！

学习编程的有效方法及TIPS 2

- 学习编程一定要有足够的时间（最好能在1 - 2个月之内集中）。如果你时间不够，先暂时不要学。
- Mission Driven: 通过各种任务及目标的设定来学习编程是最有效的方法，如果，要相信我布置给你的一系列练习会在短时间给你很大的提高，甚至精通面向对象(C++)编程思想！（该方法的有效性已得到充分的验证）
- 正如骑自行车、游泳、说英语一样，编程是一种技能学习，而不能把它当作知识来学。只有通过动手来学习是正确的方法，光看书不动手是学不会的！
- 对于某个困难点，可通过自己的努力去解决；若一个问题长久时间还得不到解决，可能是陷入到一个思维定势，或者方法不正确，这时可寻求比你更有经验的人 or Google来帮助解决。在找人帮助解决的时候，最好能坐在他/她边上看他是如何解决的（包括设置断点、调试过程等），这样学到的东西最多。注意：不要太依赖与他人的帮助，不经过思考和失败而寻求他人的帮助对你是无效的！

学习编程的有效方法及TIPS 3

- 学习方法：首先找本**较好的C++教材**（C++ Primer），快速浏览且大致了解一下C++语言的语法，在脑子内有个大致印象；然后将教材当作**参考书/字典**，按照训练作业的要求一步一步去完成任务；若在参考书中找不到的话，可通过google或百度去搜索；若还有困难，可寻求周围更有经验的人的帮助。
- 希望在不久的将来，看到你已成为一名编程好手！如果你没有成为编程好手，那唯一可能的原因就是你并没有上心来学（没有兴趣、没有时间、或者没有投入）。当然，要成为高手，路还很长，这是一个不断探索和努力的过程，以后会跟你具体讲的！
- **快捷键**的使用能很大程度上提高你的**效率**，可在平时有意识积累（不必在一次作业中全部掌握，需日积月累），以后会带给你巨大的价值。另外，建议你迫使自己键盘盲打，学会盲打同样会带给你巨大的价值。