02 Basic Dynamic Array_Class

jjcao

Modified from the assignment of Professor Ligang Liu

Encapsulation

- 由于p 和 n之间有强烈的逻辑关系，自然可以用struct将它们“包”在一起处理

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

#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;
}
```

Encapsulation

```
typedef struct
{
    int n; // the size of the array
    double *pData;  // the data of the array
}DArray;

int     InitArray( DArray &a );
int     SetArraySize(DArray &a,  int size );
int     FreeArray(DArray &a );
int     SetValue( DArray &a, int k, double value );
int     PrintArray(DArray &a );

void main()
{
    DArray pa;
    InitArray( pa );

    SetArraySize( pa, 3 );
    SetValue( pa, 0, 1.0 );
    SetValue( pa, 1, 2.0 );
    SetValue( pa, 2, 3.0 );

    PrintArray(pa);

    FreeArray(pa);
}
```

// [McConnell2] ch10.4 keep Variables "Live" for as Short a Time as Possible

```
// [McConnell2] ch10.3 Guidelines for Initializing Variables
int InitArray( DArray &a )
{
    a.n = 0;
    a.pData = NULL;

    return 1;
}
int SetArraySize( DArray &a, int size )
{
    a.n = size;
    a.pData = (double*)malloc( size * sizeof(double) );
    if( a.pData == NULL )
    {
        printf("no enough memory!\n");
        return 0;
    }

    return 1;
}
```

```
int FreeArray(DArray &a)
{
    if( a.pData != NULL )
    {
        free( a.pData );
        a.pData = NULL;
    }

    return 1;
}
```

More encapsulation

- 这时发现那些操作数组的**接口**函数，都是操作struct的数据，自然可以想到将这些函数“放”到struct里面。在struct中放函数在C中是不行的，但是在C++中是可以的
-
- 此时发现，数据p,n及处理它们的函数都“包”在一起，放在一个struct中，这就是对它们的一种**封装**。用户只要操作这个struct的函数，就可以操作一个“数组”。

More encapsulation

```
typedef struct
{
   int n; // the size of the array
   double *pData;  // the data of the array

   int      InitArray( );
   int      SetArraySize( int size );
   int      FreeArray( );
   int      SetValue( int k, double value );
   int      PrintArray(  );
}DArray;
```

It is better to use

- Constructor for InitArray()
- Desconstructor for FreeArray()
- operator overloading
  - << for PrintArray()
  - () for SetValue()

```
void main()
{
   DArray pa;

   pa.InitArray( );

   pa.SetArraySize(  3 );
   pa.SetValue(  0, 1.0 );
   pa.SetValue(  1, 2.0 );
   pa.SetValue( 2, 3.0 );

   pa.PrintArray( );

   pa.FreeArray( );
}
```

```
int DArray::PrintArray()
{
   if(n==0)
      return 0;

   if( pData == NULL )
      return 0;

   for( int i=0; i<n; i++)
   {
      printf("%lf \n", pData[i] );
   }

   return 1;
}
```

A Simple Class for Dynamic Array

- 我们将关键词struct改为class，然后将函数的类型改为public，其他不变，这个程序就可以编译运行。这事实上就是一个简单的C++的程序，并且大致完成了一个class！

- 在写C++程序时，一般一个类需要2个文件，一个头文件*.h（定义了这个类的接口），一个实现文件*.cpp（具体这个类的实现）

- 希望上述过程好好体会一下。注意：上述的代码都不太符合编程规范，只是演示了大致的一个过程。你需按照严格的编程规范来完成该练习。

```
#ifndef __JJ_BASICARRAY__
#define __JJ_BASICARRAY__

#include <iostream>

namespace jj{

// interfaces of Dynamic Array class BasicArray
class BasicArray
{
private:
   double* data_; // the pointer to the array memory
   int     size_; // the size of the array
   //int   m_capacity; // the max memory of the array
```

Class Data Members

Data members (also called instance variables or member variables) are **lowercase** with optional **underscores** like regular variable names, but always end with a **trailing underscore**.

Constructor & Deconstructor

```
public:
    BasicArray():data_(0),size_(0){} // default constructor
    // other constructor, set an array with default values.
    // Of course, more constructors can be provided.
    BasicArray(int size, double value = 0);
    // copy constructor (It is a best practice to provide copy constructor
    // for all classes which contians members allocated dynamically)
    BasicArray(const BasicArray& ba);
    // overload "=" operator
    BasicArray& operator = (const BasicArray& array);
    ~BasicArray(){ delete[] data_;} // deconstructor
```

Member Functions

```
int size() const { return size_;}          // get the size of the array
// re-set the size of the array. 注：若size_小于原数组大小，
// 可截断取前size_个元素作为新数组的元素；
// 若size_大于原数组大小，新增的元素的值设置缺省值0即可
int resize(int num, double elem=0.0);

double  at(int ind);            // get an element at an index
double  operator[] (int ind) const;  // overload "[]" operator, get an element, such as double tmp = a[k]
double& operator[] (int ind) ; // overload "[]" operator, set value of specified position, such as a[k]=3.14;

int push_back(double elem);       // add a new element at the end of the array, return the size of the array.
int insert(int ind, double value); // insert a new element at some index, return the size of the array
void erase(int ind);         // delete an element at specified index
```

Member functions should start with a lowercase letter & with underscores between words.

-- Google C++ Style

friend & Operator Overload

```
friend std::ostream& operator <<(std::ostream& os, const BasicArray &ba); // print all elements
private:
    inline bool isValidateIndex(int ind);// judge the validate of an index
};

std::ostream& operator <<(std::ostream& os, const BasicArray &ba);

bool BasicArray::isValidateIndex(int ind)
{
    return ind>-1 && ind<size_;
}
```

A template project with requirements

1. **实现hw02_BasicDynamicArray_class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

hw01_BasicDynamicArray_Template in http://www.jjcao.net/c.html

通过完成该作业需达到的目标：

1. 务必体会类（对象）的抽象和封装特性：数组抽象为通过各种操作来完成的一个“容器”。对象的特性完全是通过其“接口”（public函数）来表达的；

2. 掌握和熟悉类的定义和语法，包括构造函数（缺省构造函数，重载构造函数，拷贝构造函数），析构函数，函数的重载；
3. 熟悉动态内存的操作原理及其分配和释放，学会使用new/delete；
4. 赋值操作符号"="的重载，操作符（如”[]”）的重载；
5. 理解和使用预编译头的机制（在头文件(*.h)中使用“预编译#define #ifdef #endif”等来避免重复编译）；

6. 学会利用VC2010的debug工具来调试程序；
7. 学会构造各种情况和极端情况来测试程序的鲁棒性。

Standard solution