**BN1000. Function printStack**

**Description**

Rewrite the Stack class to add the printStack function as an instance function to display all the elements in the stack, as follows:  
template<typename T>  
class Stack  
{  
public:  
  Stack();  
  bool empty();  
  T peek();  
  void push(T value);  
  T pop();  
  int getSize();  
  void printStack();  
};  
  
The printStack() function output the elements in the stack from top to bottom, each element in one line. Just output the elements, don't erase the elements in the stack.  
  The following codes:  
Stack<int> s;  
for(int i=0;i<3;++i) s.push(i);  
s.printStack();  
s.push(3);  
s.printStack();  
  
  Output:  
2  
1  
0  
3  
2  
1  
0

**BN1001. The stack class**

**Description**

You are to implement the following Stack class template, using the nontype parameter capacity to allocate the capacity of the stack, i.e. maximum elements that can be stored in the stack.

template<typename T, int capacity>  
class Stack  
{  
public:  
    Stack();                 // Constructs an empty stack.  
    bool empty();         // Returns true if the stack is empty.  
    T peek();               // Returns the element at the top of the stack without removing it from the stack.  
    void push(T value); // Stores an element into the top of the stack.  
    T pop();                 // Removes the element at the top of the stack and returns it.  
    int size();               // Returns the number of elements in the stack.  
private:  
    T\* elements;          // Points to an array that stores elements in the stack.  
    int num;                 // The number of the elements in the stack.  
};

Your submitted source code should only include the implementation of Stack class template, without the declaration above.

No main() function should be included.

**BN1002. MaxValue template function**

**Description**

Design a generic function that returns a maximum element from an array. You should test your function with the array of int, double, std::string and char.  
Using the following function header:  
template < typename T >  
T maxValue (const T value1, const T value2)  
当比较两个char大小时，如果char是字母，将其视为大写字母进行比较。  
当两个char大小相等时，返回value1.  
例如，maxValue('B','a')返回B, maxValue('B','b')返回B, maxValue('B','c')返回c.

**BN1003. typename! typename!**

**Description**

Given the following two classes TypeA, and TypeB:   
class TypeA  
{  
public:  
  class SubType  
  {  
  public:  
    string toString() {return "subType in TypeA";};  
  };  
};  
  
class TypeB  
{  
public:  
  class SubType  
  {  
  public:  
    string toString() {return "subType in TypeB";};  
  };  
};  
  
Write a template class MyClass that has a public variable named **subtypeobj** with type **T::SubType**.  
template <class T>  
class MyClass  
{  
public:  
//add your public member here  
}  
  
  The following codes:  
MyClass<TypeA> obj1;  
cout << obj1.subtypeobj.toString() << endl;  
MyClass<TypeB> obj2;  
cout << obj2.subtypeobj.toString() << endl;  
  
  output:  
subType in TypeA  
subType in TypeB

**BN1004. The Max of a sequence**

**Description**

Write a generic function T max(T seq[], int n) that returns the maximum element of a given sequence with n elements (n>=1). You are ensured the type T could be compared with operator '<'.

Your submitted source code should include the implementation of the max function template.

No main() function should be included.

**BN1005. Search a value**

**Description**

Write a generic function int find\_lower\_bound(T seq[], int n, const T& value). The function returns the index of the largest element in the given sequence that is less than the given value. If multiple elements satisfy, return the one with smallest index. Return -1 if no such element exists.

Your submitted source code should include the implementation of the find\_lower\_bound function template.

No main() function should be included.

**BN1006. Swap Values**

**Description**

Write a generic function that swap values in two variables. You should test the function with int, double and string.  
Using the following function header:  
  
template<typename T>  
void swapVar(T &var1, T &var2)