```
using namespace std;

class A {
private:
	int number;

public:
	A() {
		cout << "A()" << endl;
		number = 0;
	}
	A(int n) : A(n) {
		cout << "A(int n)" << endl;
		number = n;
	}
	~A() { cout << "~A()" << endl; }
	virtual void foo(int x) {
		cout << "I am foo in A " << (x + number)<< endl;
	}
	void bar() {
		cout << "I am bar in A " << (number) << endl;
	}
};
class C :	public A {

public:
	C() { cout << "C()" << endl; }
	C(int n) { cout << "C(int n)" << endl; }
	~C() {		cout << "~C()" << endl; }
	void foo(int x) {
		cout << "I am foo in C" << endl;
	}
	void bar() {
		cout << "I am bar in C" << endl;
	}
};

class B : public A {

public:
	B() { cout << "B()" << endl; }
	B(int n) {
		cout << "B(int n)" << endl;
	}
	~B() { cout << "~B()" << endl; }
	void foo(int x) {
		cout << "I am foo in B" << endl;
	}
	void bar() {
		cout << "I am bar in B" << endl;
	}

};
```

```
int main() {
	A* v1 = new B(6);
	cout<<endl;

	B v2;
	cout<<endl;

	A* v3 = new C(4);
	cout<<endl;

	C v4;
	cout<<endl;

	v1->foo(1);
	v1->bar();
	cout<<endl;

	v2.foo(1);
	v2.bar();
	cout<<endl;

	v3->foo(1);
	v3->bar();
	cout<<endl;

	v4.foo(1);
	v4.bar();
	cout<<endl;

	cout << "deleting v1" << endl;
	delete v1;
	cout << "deleting v3" << endl;
	delete v3;
	cout << "done deleting v3" << endl;

}
```

WHAT IS THE OUTPUT?

```
template <class K, class V> class TreeNode
{
public:
	TreeNode(K k, V v): key(k), value(v), left(0), right(0) {}

	K		key;
	V		value;
	TreeNode<K,V> *left;
	TreeNode<K,V> *right;
};

class tree_empty_exception {
};

template <class K, class V> class BinarySearchTree {
public:

	//
	// Purpose:
	//
	// Return the maximum value in the tree.
	//
	// Example:
	//  if t is {3:56, 5:13, 28:19} getMaxVal returns 56
	// Throws:
	//  tree_empty_exception if called on an empty tree
	//
	V getMaxVal() {

		// provide implementation
		// feel free to add private helper methods









	}

	//
	// Purpose:
	//
	// Return the key of the maximum value in the tree.
	//
	// Example:
	//  if t is {3:56, 5:13, 28:19} getKeyOfMaxVal returns 3
	// Throws:
	//  tree_empty_exception if called on an empty tree
	//
	V getKeyOfMaxVal() {

		// provide implementation
		// feel free to add private helper methods









	}

private:

	TreeNode<K,V> *root;
	unsigned int  count;









};
```