```
class Person {
public:
	bool operator<(const Person& rhs) const;
	bool operator==(const Person& rhs) const;
	…
};
bool operator<(const Person& rhs) const
{
	return name < rhs.name;
}
bool operator==(const Person& rhs) const
{
	return name == rhs.name;
}

std::ostream& operator<<(std::ostream& os, const Person& p)
{
	p.display(os);
	return os;
}
```

```
int main()
{
        BinarySearchTree<Person> tree;
        Person p;
        for (int i=0; i < 10; ++i)
        {
                p.populate();
                tree.insert(p);
        }
        tree.inorder();
        …
```

```
template <typename T>
void BinarySearchTree<T>::insert(const T& item)
{
	insert(root, item);
}
```

```
template <typename T>
void BinarySearchTree<T>::insert(Node<T>*& r, const T& item)
{
        using namespace std;
        if (r == nullptr) {
                r = new Node<T>;
                r->value = item;
                r->left = nullptr;
                r->right = nullptr;
        }
        else if (item < r->value) {
                        insert(r->left, item);
                }
                else if (r->value < item) {
                        insert(r->right, item);
                }
                else {
                         cout << "Duplicate node" << endl;
                }
}
```

```
template <typename T>
T* BinarySearchTree<T>::search(Node<T>* r, const T& item)
const
{
        T* result;
        if (r == nullptr)  {
                 result = nullptr;
        }
        else if (item < r->value)  {
                result = search(r->left, item);
        }
        else if (r->value < item)  {
                result = search(r->right, item);
        }
        else  {
                result = new T(r->value);
        }
        return result;
}
```