```
bool search(Node* current, int num)
{
	bool result;
	if (current == nullptr) {
		result = false;
	} else if (current->value == num) {
		result = true;
	} else {
		result = search(current->next, num);
	}
	return result;
}
```

```
template <typename T>
T* LinkedList<T>::search(const T& item)
{
	Node<T>* current = first;
	bool found = false;
	while ((current != nullptr) && !found) {
		if (current->value == item) {
			found = true;
		}
		else {
			current = current->next;
		}
	}
	T* ptr = nullptr;
	if (current != nullptr) {
		ptr = &current->value;
	}
	return ptr;
}
```

```
template <typename T>
class HashTable {
public:
	HashTable(int s);
	~HashTable();
	void insert(const T& item);
	void remove(const T& item);
	T* search(const T& item) const;
	void display() const;
private:
	LinkedList<T>* table;
	int size;
};
template <typename T>
HashTable<T>::HashTable(int s)
{
	size = s;
	table = new LinkedList<T>[size];
}
```

```
template <typename T>
HashTable<T>::~HashTable()
{
        delete [] table;
}

template <typename T>
void HashTable<T>::insert(const T& item)
{
        int index;
        index = item.hash(size);       // Assume that 'hash' is a
                                       // member function of T
        table[index].insertFront(item);
}
```

```
template <typename T>
T* HashTable<T>::search(const T& item) const
{
        int index;
        index = item.hash(size);
        T* result = table[index].search(item);
                        // Returns ‘null’, or a pointer
                        // to the Person if found
        return result;
}

template <typename T>
void HashTable<T>::display() const
{
        for (int i = 0; i < size; ++i) {
                table[i].display();
                cout << "---" << endl;
        }
}
```

```
class Person {
public:
    Person();
    int hash(int hashTableSize) const;
    void populate();
    void setAge(int a);
    void setName(const std::string& n);
    std::string getName() const;
    int getAge() const;
    bool operator==(const Person& p) const;
    bool operator!=(const Person& p) const;
    friend std::ostream& operator<<(std::ostream& os, const
    Person& p);
private:
    std::string name;
    int age;
};
```

```
int Person::hash(int hashTableSize) const
{
	int sum = 0;
	for (int i=0; i < name.size(); ++i) {
		sum += name[i];
	}
	return sum % hashTableSize;
}


bool Person::operator==(const Person& p) const
{
	return name == p.getName();
}
bool Person::operator!=(const Person& p) const
{
	return !(*this == p);
}
```

```
HashTable<Person> h(100);
Person p;
for (int i=0; i < 10; ++i) {
        p.populate();
        h.insert(p);
}
string n;
cout << “\n\nEnter Name to Search: “;
getline(cin, n);
p.setName(n);
Person* ptr;
ptr = h.search(p);
if (ptr != nullptr) {
        cout << “Name: “ << ptr->getName() << endl;
        cout << “Age:     “ << ptr->getAge() << endl;
}
else {
        cout << “Not Found” << endl;
}
```