| | |
|---|---|
| Name | Matr.-Nr. |

# Klausur Programmieren III – TI3

WS 13

Prof. Dr. K. Baer

Hilfsmittel: 1 Blatt DIN A4, beidseitig beschr.

Bearbeitungszeit: 90 Min.

1. Bitte tragen Sie zuerst Name und Matrikelnummer ein!
2. Kontrollieren Sie die Vollständigkeit der Aufgabenblätter.
3. Die Klausur besteht aus 6 Aufgaben. Verschaffen sie sich einen kurzen Überblick über die Aufgaben und beginnen Sie am besten mit der Aufgabe, die Ihnen am ehesten ein Erfolgserlebnis bringt.
4. Lesen Sie die Aufgabenstellung aufmerksam durch, bevor Sie eine Aufgabe lösen!
5. Verwenden Sie zur Beantwortung der Fragen den vorgesehenen Platz auf den Aufgabenblättern.
6. Schreiben Sie leserlich. Nicht lesbare Teile werden mit 0 Punkten bewertet!

Viel Erfolg!

| Aufgabe | 1 | 2 | 3 | 4 | 5 | 6 | Summe |
|---|---|---|---|---|---|---|---|
| Punkte | 20 | 16 | 18 | 14 | 10 | 12 | 90 |
| erreicht | | | | | | | |

## Aufgabe 1 (20 Punkte = 16+2+2)

Folgende Klassendefinition sei gegeben:

```
class Top {
public:
  Top()      { cout << " cTop"; }
  Top(Top& a) { cout << " copyTop"; }
  ~Top()     { cout << " ~Top"; }
  void doSomething(){ cout << " doTop";}
};

class A : public Top {
public:
  A()      { cout << " cA"; }
  A(A& a)  { cout << " copyA"; }
  ~A()     { cout << " ~A"; }
  void doSomething(){ cout << " doA";}
};

class B : public Top {
private:
     Top* ptrTop;
     A objA;
public:
  B()             { cout << " cB"; }
  B(B& a)         { cout << " copyB"; }
  ~B()            { cout << " ~B"; }

  Top* getptrTop() {return ptrTop; }
  void setptrTop(Top* t){ ptrTop = t; }

  A getobjA() {return objA; }
  void setA(A a){objA = a; }
};
```

a) Geben Sie für untenstehende Funktion `test()` hinter jeder Zeile an, welche Ausgaben bei der Auswertung auf der Konsole erscheinen.

| Nr | Code | Ausgabe |
|---|---|---|
| 1 | void test(){ | |
| 2 | Top t[3]; | cTop cTop cTop |
| 3 | A a1; | cTop cA |
| 4 | B* b1 = new B(); | cTop cTop cA cB |
| 5 | t[1] = a1; | |
| 6 | t[2] = *b1; | |
| 7 | A a2 = a1; | cTop copyA |
| 8 | b1->setptrTop(&a2); | |
| 9 | t[1].doSomething(); | doTop |
| 10 | t[2].doSomething(); | doTop |
| 11 | a2.doSomething(); | doA |
| 12 | b1->doSomething(); | doTop |
| 13 | b1->getptrTop()->doSomething(); | doTop |
| 14 | b1->getobjA().doSomething(); | cTop copyA doA ~A ~Top |
| 15 | delete b1; | ~B ~A ~Top ~Top |
| 16 | } | ~A ~Top ~A ~Top ~Top ~Top ~Top |

b) Was ändert sich an der Ausgabe, wenn die Methode `doSomething()` der Klasse `Top` als `virtual` deklariert wird? Begründen Sie Ihre Antwort!

In Zeile 13 wird statt doTop  jetzt doA ausgegeben

- späte Bindung,
- Der im ptrTop-Attribut gespeichert Pointer ist Basisklassenzeiger auf Exemplare der Klasse Top oder Ableitungen davon. Im konkreten Fall zeigt der Basisklassenzeiger auf ein A-Objekt, d.h durch die virtual Deklaration wird die überschrieben Methode von A verwendet.

c) Was bewirkt die virtual-Deklaration einer Methode und wie wird dies vom Compiler unterstützt?

Dadurch wird späte Bindung erzwungen, d.h. es wird zur Laufzeit geprüft, welcher Objekttyp wirklich vorliegt und es werden die Methoden dieses Objekttyps aufgerufen.

- pro Klasse  wird eine "Virtuelle Methodentabelle" erzeugt  (engl. virtual method table, VMT)
- Die VMT enthält zu jeder virtuellen Methode die Anfangsadresse des Maschinencodes für die Methodenimplementierung
- Jedes Objekt einer Klasse, die mindestens eine virtuelle Methode besitzt, enthält einen Zeiger auf die VMT der Klasse
- Bei einem Aufruf einer virtuellen Methode wird dann in der VMT des Objekts nachgeschaut, wo der Code liegt, der ausgeführt werden soll.

## Aufgabe 2 (16 Punkte=4+4+6+2)

a) Was versteht man unter Elementinitialisierer-Ausdrücken und wozu dienen sie? Geben Sie ein Beispiel.

Initialisierung zusammengesetzter Objekte

Mit sog. Elementinitialisierer-Angaben können von einem Konstruktor einer Klasse Werte an die Konstruktoren von Teilobjekten übergeben werden.

```
C::C(intx, doubley, doublez) : objA(x), objB(y,z)
{
...
}
```

Elementinitialisierer können auch zur initialisierung von Elementen primitven Typs genutzt werden.

b) Welche Probleme können bei expliziten Typumwandlungen (Casts) im Zusammenhang mit abgeleiteten Klassen auftreten?

Downcasts können problematisch sein!

Falls Basisklassenzeiger durch Cast in Zeiger auf abgeleitete Klasse konvertiert wird, kommt es darauf an, ob der Laufzeittyp des Objektes dem Cast entspricht oder nicht. Bei static_cast erhält man einen gültigen Zeiger, unabhängig vom tatsächlichen Laufzeittyp. Falls die Typen nicht übereinstimmen, ist das Verhalten undefiniert, ggf. Absturz.

Dynamic_cast liefert null, falls Typen sich nicht entsprechen, ansonsten einen gültigen Zeiger → vorzuziehen, da dadurch fehlerbehandlung möglich wird!

Bei Upcasts tritt slicing auf.

c) In welchen Zusammenhängen kann “static” verwendet werden und was bedeutet es jeweils?

Für Variablen, Funktionen, Klassenattribute und -funktionen.

Globale Variablen: Lebensdauer über ganzen Programmlauf

Var einer Fkt: dito, status wird zwischen Aufrufen beibehalten

Klassenattribut: existiert 1 x / Klasse, init. ausserhalb Klasse

Memberfkt: kann ohne Objekt aufgerufen werden.

Kann nur auf statische Eleemnte der Klasse zugreifen.

d) Wie ist der Zusammenhang zwischen Iterator und dem zugehörigen Reverse-Iterator? Weshalb?

&*(reverse_iterator(i)) == &*(i-1)

Problem

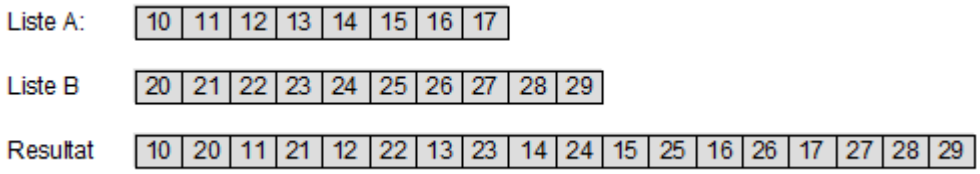
es gibt i.a. keine gültige Adresse vor dem ersten Element.

reverse_iterator liefert beim Dereferenzieren das vorhergehende Element.

## Aufgabe 3 (18 Punkte=5+3+10)

Aus zwei einfach verketteten Listen A und B soll eine neue Liste konstruiert werden, wobei abwechselnd jeweils aus A und dann aus B ein Element an die neue Liste angehängt wird, bis alle Elemente aus A und B in der resultierenden Liste enthalten sind. Falls das Ende einer Liste erreicht wird, werden die restlichen Elemente der anderen Liste angehängt.

Beispiel:




a) Entwickeln Sie eine geeignetes Klassen-Template für diesen Listentyp. Implementieren Sie lediglich die für die beschriebene merge-Operation und den Auf-/Abbau der Liste benötigten Methoden (Konstruktor, Destruktor, Get-/Set-Methoden) sowie ihre Attribute.

```
template<class T>
class SimpleList {
protected:
        T value;
        SimpleList* next;
public:
        SimpleList(T val = 0, SimpleList<T> *n = 0) :
                        value(val), next(n) {
        }
        virtual ~SimpleList() {         //rekusives löschen
                if (next != NULL) {
                        delete next;
                }
        }

        SimpleList<T>* getnext() const {
                return next;
```

```
    }

    void setnext(SimpleList<T> *n) { // kein Kopieren!
        next = n;
    }

    //...

}
```

b) Überladen Sie den << Operator für Ihre Listen-Klasse. Die Ausgabe soll die Werte der Liste getrennt durch Minuszeichen ausgeben, z.B:
10 – 43 – 21 – 23 – 54

```
friend ostream& operator<<(ostream& o, const SimpleList<T> &sl) {
    o << sl.value;
    if (sl.next)
        o << "-" << *sl.next; // rekursiv
    return o;
}
```

c) Schreiben Sie eine globale Template-Methode `merge`, die zwei einfach verkettete Listen `listA` und `listB` Ihrer Klasse zu einer neuen Liste `result` entsprechend obiger Vorgabe zusammenfügt. Die Elemente von `listA` und `listB` sollen nicht kopiert, sondern nach `result` verschoben werden. Die übergebenen Zeiger auf die Listen `listA`, `listB` sollen danach auf `NULL` zeigen. Ihre Methode `merge` könnte z.B. wie folgt in einer `main( )` aufgerufen werden:

```
int main() {
	SimpleList<int> * listA = new SimpleList<int>(10);
	// füllen der Liste

	SimpleList<int> * listB = new SimpleList<int>(20);
	// füllen der Liste

	SimpleList<int> * mergedList = merge(&listA, &listB);
	cout << "mergedList nach Merge: " << (*mergedList);

	if (!listA)
		cout << "\n" << "listA ist leer" << endl;
	if (!listB)
		cout << "\n" << "listB ist leer" << endl;
	//…
	delete mergedList;
} // main( )
```

```
template<class T>
SimpleList<T>* merge(SimpleList<T> **listA, SimpleList<T>
**listB) {
	SimpleList<T> *result = NULL; //  Ergebnisliste

	// Sonderbehandlung für ein oder zwei leere Listen
	if (*listA == NULL)
		result = *listB;
	else if (*listB == NULL)
		result = *listA;
	else { // listA und listB Listen nicht leer

		// Pointer für aktuelle Positionen in allen Listen
		result = (*listA); // Zielliste erhält das erste
listA Element
		SimpleList<T> *posA = (*listA)->getnext();
		result->setnext(*listB); // gefolgt vom ersten listB-
Element
		SimpleList<T> *posB = (*listB)->getnext();
```

```
            SimpleList<T> *posResult = (*listB);

            // abwechselnd Elemente übertragen
            while (posA && posB) {
                posResult->setnext(posA);
                posResult = posA;
                posA = posA->getnext();
                posResult->setnext(posB);
                posResult = posB;
                posB = posB->getnext();
            } // bis zum Ende der kleineren Liste

            // den Rest der verbliebenen Elemente anhängen
            if (posA)
                posResult->setnext(posA);
            else
                posResult->setnext(posB);
        } // else: *listA und *listB != NULL

        // Ausgangslisten zurücksetzen
        (*listA) = (*listB) = NULL;
        return result;
} // merge( )
```

## Aufgabe 4 (14 Punkte = 8+3+3 )

In einem Programm zur Artikelverwaltung müssen die gespeicherten Artikel häufig nach ganz unterschiedlichen Kriterien sortiert werden. Bspw. müssen Artikel

- nach ihrer Bezeichnung alphabetisch aufsteigend oder absteigend
- nach ihrem Preis aufsteigend oder absteigend.

sortiert werden können.

Die STL bietet Ihnen hierzu einiges an Unterstützung. Sie können die Liste der Artikel in einem Container verwalten und mittels des `sort()`-Algorithmus den Container nach den gewünschten Kriterien sortieren.

Den Algorithmus `sort()` gibt es in 2 Varianten:

```
template<class RA_Iter>
void sort( RA_Iter first,  RA_Iter last);

template<class RA_Iter, class Compare>
void sort( RA_Iter first,  RA_Iter last, Compare  comp);
```

`sort()` sortiert die Elemente des Bereichs [first, last) in Bezug auf `operator<`, wenn kein Vergleichskriterium angegeben ist, und ansonsten in Bezug auf `comp`.

a) Implementieren Sie eine Klasse `Artikel` mit den **privaten** Attributen
   `string bezeichnung`
   `double preis`
   (andere Attribute sollen vernachlässigt werden), sowie allen benötigten Methoden, Operatoren, Funktoren, um die oben genannten Sortierungen mit Hilfe des `sort()`-Algorithmus produzieren zu können.

   Die Standardsortierung – also die 1. Variante ohne zusätzliches Funktionsionobjekt – soll eine alphabetisch aufsteigende Sortierung nach Bezeichnung sein.

```
class Artikel {
public:
        Artikel(string t, double p) : bezeichnung(t), preis(p) {}
        bool operator<(const Artikel& other) const {
                return bezeichnung < other.bezeichnung;
        }
        bool operator>(const Artikel& other) const {
                return bezeichnung > other.bezeichnung;
        }
```

```
        double getPreis() const { return preis; }

        friend ostream& operator<< (ostream& os, const Artikel& b);
private:
        double preis;
        string bezeichnung;
};                                                          // 5 P


class PreisAufsteigend {
public:
        bool operator()(const Artikel& b1, const Artikel& b2) const {
        return b1.getPreis() < b2.getPreis();
        }
};                                                          // 2 P

class PreisAbsteigend {
public:
        bool operator()(const Artikel& b1, const Artikel& b2) const {
        return b1.getPreis() > b2.getPreis();
        }
};                                                          // 2 P

ostream& operator<< (ostream& os, const Artikel& b){
        os << "Bezeichnung: \"" << b.bezeichnung << "\" Preis: " << b.preis;
        return os;
}                                                           // 2 P


                                                    // Teil a: 8 P
                                                    // Teil c: 3 P


#endif /* ARTIKEL_H_ */
```

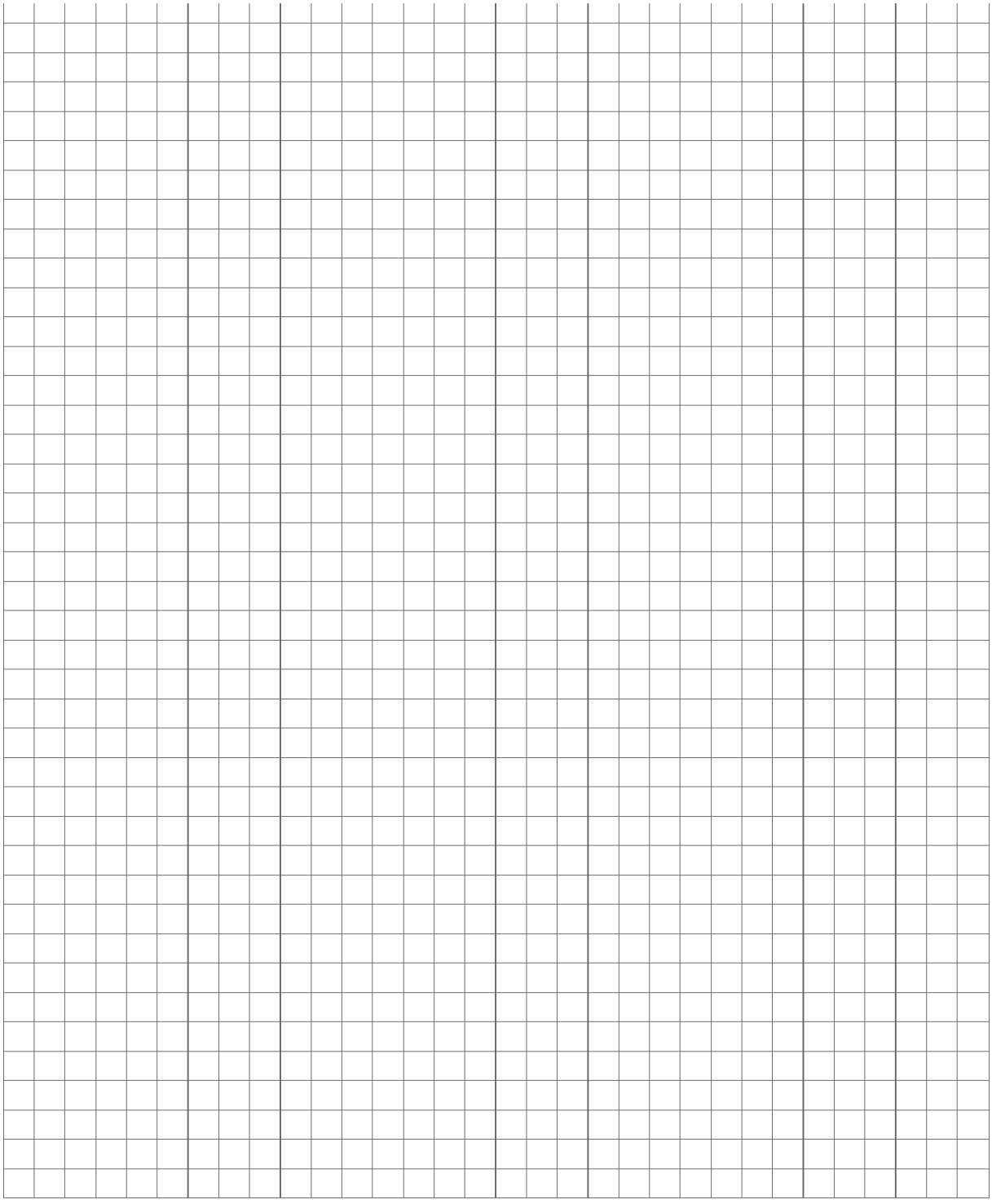
b) Ergänzen sie folgendes kleines Testprogramm um Ihre Funktionsaufrufe gemäß den Angaben in den Kommentaren. Um Schreibarbeit zu sparen, geben Sie nur die Veränderungen zur vorherigen Angabe an.

```
void test(){
      vector<Artikel> b;        // Liste von Artikeln

      // Einfuegen von Buechern
      b.push_back(Artikel("C-Artikel", 40));
      b.push_back(Artikel("A-Artikel", 50));
      b.push_back(Artikel("B-Artikel", 30));


      // Ausgeben der unsortierten Liste


copy(b.begin(), b.end(), ostream_iterator<Artikel>(cout
"\n"));   // 1 P



      // alphabetisch aufsteigend sortieren & ausgeben:

sort(b.begin(), b.end())


      // alphabetisch absteigend sortieren & ausgeben:

sort(b.begin(), b.end(), greater<Artikel>() );      // 1 P


      // nach Preis aufsteigend sortieren und ausgeben:

sort(b.begin(), b.end(), PreisAufsteigend());// 0,5 P


      // nach Preis absteigend sortieren und ausgeben:

sort(b.begin(), b.end(), PreisAbsteigend());// 0,5 P

}
```

c) Was muss ergänzt werden, um die Ausgabe mittels copy und eines Stream-Iterators machen zu können? Ergänzen Sie Ihre Implementierung um die notwendigen Dinge, falls nicht ohnehin schon geschehen.

Die Ausgabe soll in folgender Form erfolgen:

```
Bezeichnung: "C-Artikel" Preis: 40
Bezeichnung: "A-Artikel" Preis: 50
Bezeichnung: "B-Artikel" Preis: 30
```



## Aufgabe 5 (10 Punkte = 3 + 1 + 6)

Folgendes Programm sei gegeben:

```
int main() {
      typedef multimap<string, string> VokMulMap;
      typedef VokMulMap::value_type V;
      const V f[] = {  V("two", "Zwei"),V("three", "Drei"),
                       V("four", "Vier"), V("four", "Doppelvierer")};

      VokMulMap vokabeln(f, f + 4);

      const VokMulMap::iterator ins = vokabeln.insert(V("one",
                                                        "Eins"));
      vokabeln.insert(ins, V("one", "man"));

      VokMulMap::const_iterator iter = vokabeln.begin();
      for (; iter != vokabeln.end(); ++iter)
            cout << "(" << iter->first <<"," << iter->second << ")\n";

      const VokMulMap::const_iterator
            von = vokabeln.lower_bound("one"),
            bis = vokabeln.upper_bound("one");
      for (iter = von; iter != bis; ++iter)
            cout << iter->second << ' ';
      cout << endl;
      const pair<VokMulMap::iterator, VokMulMap::iterator>
                                    p = vokabeln.equal_range("four");
      for (iter = p.first; iter != p.second; ++iter)
            cout << iter->second << ' ';

      VokMulMap::size_type n = vokabeln.count("one");
      cout << '\n' << n << endl;
      n = vokabeln.erase("one");
      cout << n << endl;
}
```

a) Was gibt das Programm aus?

(four, Vier)
(four, Doppelvierer)
(one, man)
(one, Eins)
(three, Drei)
(two, Zwei)
man Eins
Vier Doppelvierer
2
2

b) Welchen Datentyp nutzt eine Map/MultiMap?

pair <const key, value>

c) Wie kann man z.B. nach dem Wert „vier“ suchen, wenn man den zugehörigen Schlüssel nicht kennt?

Denken Sie an den `find_if`-Algorithmus, der folgendermaßen deklariert ist:

```
template<class T, class Predicate>
InputIterator find_if(InputIterator first,
                      InputIterator last,
                      Predicate pred);
```

Was muss man tun, um mit Hilfe dieses Algorithmus nach einem bestimmten Wert zu suchen?

Hinweis:

Die Template-Klasse `pair` stellt Typdefinitionen bereit, mit denen der Typ des ersten und zweiten Elements ermittelt werden kann:

typedef T1 first_type;

typedef T2 second_type;

```
template<class Pair>
class WertGleich : public unary_function<Pair, bool> {
public:
        WertGleich(const typename Pair::second_type& x){
                this->x = x;
   }
        bool operator()(const Pair& v) const {
                return v.second == x;
        }
private:
        const typename Pair::second_type x;
};
```

[6 P]

## Aufgabe 6 (12 Punkte)

Entwickeln Sie ein Programm, das die Wörter in einem übergebenen Text zählt und ausgibt.

Hierfür sind Stream-Ieratoren sehr nützlich. Der `istream_iterator` hat verschiedene Konstruktoren:

`istream_iterator();`

Der Default-Konstruktor erzeugt einen end-of-stream Iterator.

`istream_iterator (istream_type& s);`

erzeugt einen Iterator, der mit einem Eingabe-Strom s verknüpft ist, von dem Sie den Text lesen können.

```
int main() {


    map<string, unsigned> wordmap;
    istream_iterator<string> ins(cin), endofstream;
    while (ins != endofstream) {
        string word = *ins;
        wordmap[word] += 1;
        ++ins;
    }
    for (map<string, unsigned>::iterator it = wordmap.begin();
            it != wordmap.end(); ++it) {
        cout << it->first << "\t" << it->second << endl;
    }
    return 0;
}
```