Aufgabe 1

Zeile	Konstruktoren, Zuweisungen, Destruktoren
37	a int-Konstruktor, b int-Konstruktor
38	c Copy-Konstruktor, d Konstruktor, e Konstruktor
39	d int-Zuweisung
40	A Copy-Konstruktor, A Addition, T int-Konstruktor, U Copy-Konstruktor, T Destruktor, V Copy-Konstruktor, A Destruktor, d Zuweisung, U Destruktor, V Destruktor
41	d Addition, T int-Konstruktor, U Copy-Konstruktor, T Destruktor, U Addition, V int-Konstruktor W Copy-Konstruktor, V Destruktor, e Zuweisung, W Destruktor, U Destruktor
42	a Destruktor, b Destruktor, c Destruktor, d Destruktor, e Destruktor

Listing 1: aufgabe2.cc

```
1  #include </home/josua/uni/ipi/ipiclib/fcpp.hh>
2
3  // Ein Listenelement
4  struct IntListElem {
5    IntListElem* next;   // Zeiger auf naechstes Element
6    int value;           // Daten zu diesem Element
7  } ;
8
9  // Eine Liste
10 struct IntStruct {
11   int count;           // Anzahl Elemente in der Liste
12   IntListElem* first; // Zeiger auf erstes Element der Liste
13 } ;
14
15
16 // Initialisiere eine Listenstruktur
17 void empty_list (IntStruct* l)
18 {
19   l->first = 0;    // 0 ist keine gueltige Adresse: Liste ist leer
20   l->count = 0;
21 }
22
23 // Fuege ein Element nach einem gegebenem ein
24 void insert_in_list(IntStruct* list,IntListElem* where,IntListElem* ins)
25 {
26   if (where==0) // fuege am Anfang ein
27     {
28       ins->next = list->first;
29       list->first = ins;
30       ++list->count;
31     }
```

```
32    else          // fuege nach where ein
33      {
34        ins->next = where->next;
35        where->next = ins;
36        ++list->count;
37      }
38  }
39
40  // Entferne ein Element nach einem gegebenem
41  // Liefere das entfernte Element zurueck
42  IntListElem* remove_from_list (IntStruct* list, IntListElem* where)
43  {
44    IntListElem* p;  // das entfernte Element
45
46    // where==0 dann entferne erstes Element
47    if (where==0)
48      {
49        p = list->first;
50        if (p!=0)
51      {
52        list->first = p->next;
53        list->count = list->count - 1;
54      }
55        return p;
56      }
57
58    // entferne Element nach where
59    p = where->next;
60    if (p!=0)
61      {
62        where->next = p->next;
63        list->count = list->count - 1;
64      }
65    return p;
66  }
67
68  class IntList
69  {
70  public:
71  // Konstruktor, erzeugt eine leere Liste
72  IntList();
73  //Copy-Konstruktor
74  IntList( IntList& oldlist);
75  //Zuweisung
76  IntList& operator=( IntList& oldlist);
77  // Destruktor, loescht gesamten Listeninhalt
```

```
78  ~IntList();
79  // Gibt Anzahl der Elemente zurueck
80  int getCount();
81  // Gibt zurueck, ob die Liste leer ist
82  bool isEmpty();
83  // Gibt die Liste aus
84  void print();
85  // Fuegt die Zahl 'element' an der (beliebigen) Position 'position' ein
86  void insert(int element, int position);
87  //insert ganz hinten
88  void insert(int element);
89  // Loescht das Element an der Position 'position'
90  void remove(int position);
91  // Gibt den Wert des Elements an der Position 'position' zurueck
92  int getElement(int position);
93  private:
94  // ... (hier folgen private Member der Klasse)
95  IntStruct liste;
96  IntListElem* elementpointer(int position);
97  };
98
99
100 IntList::IntList() {empty_list(&liste);}
101 //Copy-Konstruktor
102 IntList::IntList( IntList& oldlist )
103 {
104     empty_list(&liste);
105     for (int i = 0; i < oldlist.getCount(); ++i)
106     {
107         this->insert(oldlist.getElement(i));
108     }
109     //std::cout << "hello" << std::endl;
110 }
111 //Zuweisungsoperator
112 IntList& IntList::operator=( IntList& oldlist)
113 {
114     if (this != &oldlist)
115     {
116         empty_list(&liste);
117         for (int i = 0; i < oldlist.getCount(); ++i)
118         {
119             this->insert(oldlist.getElement(i));
120         }
121     }
122 }
123 //Destruktor
```

```
124 IntList::~IntList() {empty_list(&liste);}
125 int IntList::getCount() {return liste.count;}
126 bool IntList::isEmpty()
127 {
128     if (liste.count == 0)
129     {
130         return true;
131     }
132     else
133     {
134         return false;
135     }
136 }
137 void IntList::print()
138 {
139     std::cout << "[";
140     for (IntListElem* p = liste.first; p != 0; p = p->next)
141     {
142         std::cout << p->value <<  " ";
143     }
144     std::cout << "]"<<std::endl;
145 }
146 void IntList::insert(int element, int position)
147 {
148     IntListElem* where = elementpointer(position);
149     IntListElem* ins = new IntListElem;
150     ins->value = element;
151     insert_in_list(&liste, where, ins);
152 }
153 void IntList::insert(int element)
154 {
155     IntListElem* where = elementpointer(liste.count);
156     IntListElem* ins = new IntListElem;
157     ins->value = element;
158     insert_in_list(&liste, where, ins);
159 }
160 void IntList::remove(int position)
161 {
162     IntListElem* where = elementpointer(position + 1);
163     remove_from_list(&liste, where);
164 }
165
166 int IntList::getElement(int position)
167 {
168     IntListElem* pointer = elementpointer(position);
169     return pointer->value;
```

```
170 }
171 IntListElem* IntList::elementpointer(int position)
172 {
173     IntListElem* where = liste.first;
174     int counter = 0;
175     while (counter < position)
176     {
177         where = where->next;
178         counter = counter + 1;
179         //std::cout << where->value <<  " ";
180     }
181     return where;
182 }
183
184
185 int main()
186 {
187 IntList list;
188 list.insert(30);
189 list.insert(20);
190 list.insert(10);
191 list.print();
192 list.remove(2);
193 list.print();
194 list.insert(30,2);
195 list.print();
196 list.insert(40,3);
197 list.print();
198 IntList copy(list);
199 copy.print();
200 copy.remove(0);
201 copy.print();
202 list.print();
203 copy.print();
204 return 0;
205 }
```