Softwaretechnik (WS 2005/2006) - Übungsblatt 10

Jan Hendrik Dithmar - Matrikelnummer 2031259
Übungsgruppe Mi, 8-10 Uhr
Dirk Heine - Matrikelnummer 2030941
Übungsgruppe Di, 14-16 Uhr

Rückgabe: Übungsgruppe Di, 14-16 Uhr

Aufgabe 1

(a)

```
a1.cpp: In function ‘void increment_set(std::set<int, std::less<int>,
std::allocator<int> >&)’:
a1.cpp:6: error: increment of read-only location
```

(b)

In Zeile 6 wird die Adresse des Iterators erhöht. Da dies in der Form nicht erlaubt ist (man würde die Einzigartigkeit von `set` gefährden), steigt der Compiler dort aus. Eine mögliche Fehlermeldung wäre z.B.: „read-only location in set“

(c)

Es ist möglich, `increment_set` zu implementieren, ohne seine Signatur zu verändern, da man eine temporäre Variable des Typs `std::set<int>` und einen temporären Iterator einführen und mit diesen arbeiten kann (siehe Teil d). Somit ist die Eindeutigkeit der ursprünglichen Menge (`set`) garantiert.

(d)

```
static void increment_set(std::set<int>&s) {
  std::set<int> s_temp;
  for (std::set<int>::iterator i = s.begin(); i != s.end(); i++) {
    int i_temp = *i + 1;
    s_temp.insert(i_temp);
  }
  s = s_temp;
}
```

Aufgabe 2

```
#include <iostream>
#include <stdexcept>

template <typename T,unsigned int size>
class Buffer {
    private:
        T* buffer; // Zeiger auf T (1.)
    public:
        Buffer();
        ~Buffer();
        const T operator[] (unsigned int) throw (std::out_of_range);
};
```

```
template <typename T,unsigned int size>
Buffer<T,size>::Buffer() {
    buffer = new T(size); // neues Array vom Typ T der Länge size (2.)
}

// Arrays wird bei Destruktion wieder freigegeben (3.)
template <typename T,unsigned int size>
Buffer<T,size>::~Buffer() {
    free(buffer);
}

// Falls das Element nicht im Array ist,
// wird eine out_of_range-Exception geworfen (4.)
template <typename T,unsigned int size>
const T Buffer<T,size>::operator[] (unsigned int n) throw (std::out_of_range) {
    if(n < size) {
        buffer = buffer + (n*sizeof(T)); // das n-te Element wird zurückgegeben (5.)
    } else {
        throw std::out_of_range();
    }
    return *buffer;
}

// Für <bool,16> soll es ein spezialisiertes Template geben (6.)
template<> class Buffer<bool,16> {
    private:
        unsigned short buffer;
    public:
        Buffer();
        ~Buffer();
        const bool operator[] (unsigned int) throw (std::out_of_range);
};

Buffer<bool,16>::Buffer() {
    // Test, ob sizeof(short) >= 2 ist.
    assert(sizeof(short)>=2);

    buffer = 0;
}

Buffer<bool,16>::~Buffer() {
    // hier muss nichts getan werden, da in der speziellen Implementierung
    // kein Pointer verwendet wird
}

const bool Buffer<bool,16>::operator[] (unsigned int n) throw (std::out_of_range) {
    if (n < 16) {
```

```
        short temp = buffer;
        short temp2 = buffer;
        for (int i = 0; i < n; i++) {
            temp = temp2 % 2;
            temp2 = temp2 / 2;
        }
        return temp;
    } else {
        throw std::out_of_range(false);
    }
}

// Programm (7.)
int main() {
    Buffer<bool,15> b1;
    Buffer<bool,16> b2;

    std::cout << "Size 1 = " << sizeof(b1) << std::endl;
    std::cout << "Size 2 = " << sizeof(b2) << std::endl;

    return 0;
}
```

Das Programm gibt folgendes aus:

```
Size 1 = 4
Size 2 = 2
```