C/C++ Programmierung
Übungsblatt 5

Florian Oetke

1 Summieren beliebiger Vektoren

In dieser Aufgabe soll ein Funktions-Template `sum` mit einem Template-Parameter `T` implementiert werden, das einen `std::vector` mit Elementen vom Typ `T` als Parameter bekommt und die Summe aller Elemente zurückgibt.

Um die Summe zu bilden, benötigen Sie eine Zählvariable, die mit 0 initialisiert wird. Dies können Sie mit `auto sum = T{};` erreichen[1].

Als Basis für die Implementierung können Sie folgenden Programmcode verwenden:

```
1 #include <iostream>
2 #include <vector>
3
4
5 int main() {
6     std::cout << sum(std::vector<int>{1, 2, 3}) << "\n";
7     std::cout << sum(std::vector<float>{-5.2f, 8.8f}) << "\n";
8     std::cout << sum(std::vector<std::string>{"Hello ", "there\n"});
9 }
```

2 Speicherabbild: min Funktions-Template

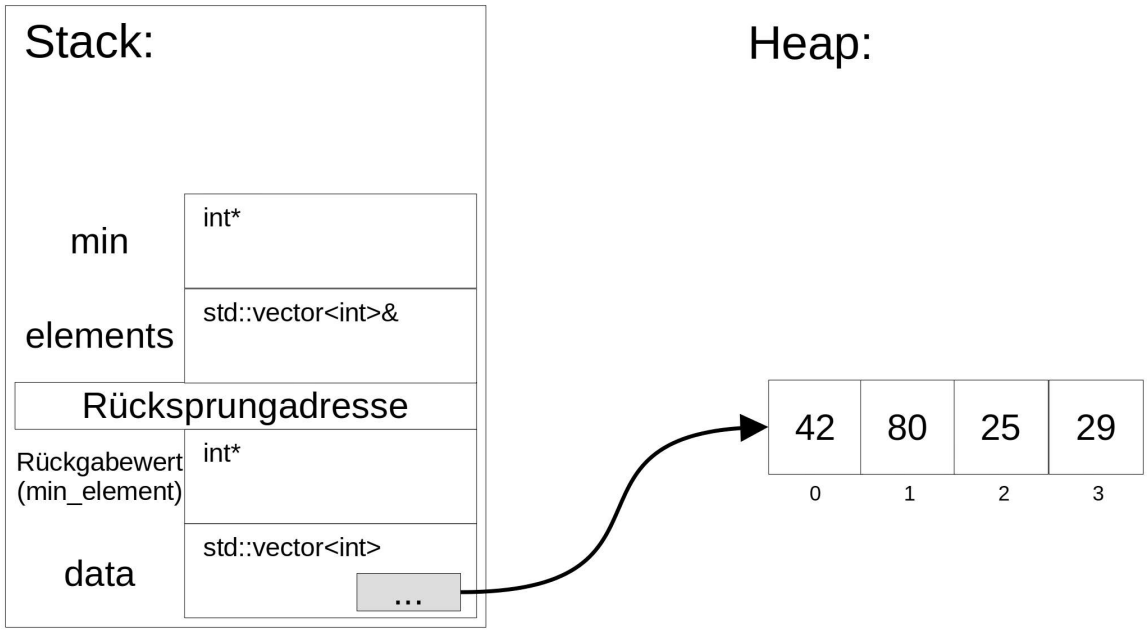
Schauen Sie sich das unten aufgeführte Programm, welches das kleinste Element in einem `std::vector` bestimmt und einen Pointer auf dieses zurückgibt, sowie das unvollständige Speicherabbild[2] zu diesem Programm (nächste Seite) an und bearbeiten Sie anschließend die folgenden Aufgaben.

1. Ergänzen Sie das Speicherabbild um die Pfeile der Zeiger und Referenzen, sodass es den Zustand am Ende der ersten Iteration der `for`-Schleife in der `min`-Funktion darstellt. Das heißt, nachdem das erste Element im Vektor, mit dem Wert 42, verarbeitet wurde.
2. Markieren Sie, auf welches Objekt der `min`-Pointer nach jeder der verbleibenden Iterationen zeigt.
3. Zeichnen Sie ein Speicherabbild vom Zustand des Programms in Zeile 27.
4. Welche Auswirkungen hat es das min einen Pointer statt einen `int` zurückgibt?
5. Ist es notwendig, dass der Parameter `elements` und die Schleifen-Variable `e` Referenzen sind? Was würde ggf. passieren, wenn wir hier stattdessen die Objekte by-value übergeben bzw. kopieren?
6. Könnte `min` auch eine Referenz statt eines Pointers zurückgeben? Warum bzw. warum nicht?
7. Was müssten Sie an der `min` Funktion ändern, wenn sie statt auf einem veränderbaren auf einem `const std::vector<T>&` arbeiten soll?

[1] Hat bei `int` den Wert `0`, bei `float` `0.0`, bei `std::string` `""`, etc.

[2] Der `std::vector` ist hier ähnlich zu Pointern/Referenzen dargestellt, insofern das er lediglich auf eine Folge von Elementen im Speicher verweist. Wie das im Detail funktioniert, werden wir uns in der nächsten Vorlesung anschauen. Aktuell können Sie einfach davon ausgehen, dass Sie beim Zugriff auf ein Element des `std::vector` eine Referenz auf das entsprechende Objekt in dieser Folge erhalten und sich der `std::vector` in Bezug auf `const` ähnlich wie eine Referenz verhält (d.h. bei einem `const std::vector` wären die einzelnen Elemente ebenfalls `const`).

```cpp
#include <iostream>
#include <vector>

template<typename T>
T* min(std::vector<T>& elements) {
    T* min = nullptr;

    for(auto& e : elements) {
        if(!min || *min > e) {
            min = &e;
        }
    }

    return min;
}

int main() {
    auto data = std::vector<int>{
        42, 80, 25, 29
    };

    auto min_element = min(data);
    if(min_element) {
        std::cout << "Min: " << *min_element << "\n";
        *min_element = -1;
    }

    for(auto& e : data) {
        std::cout << e << "\n";
    }
}
```



3 Template Argument Deduction

In dieser Aufgabe geht es darum, dass Sie sich mit der automatischen Bestimmung von Template-Parametern vertraut machen, indem Sie diese händisch nachvollziehen.

Schauen Sie sich das Beispiel unten bzw. im Handout an und überlegen Sie sich bei jedem der Aufrufe in main welcher Typ für den Template-Parameter eingesetzt werden und ob der Aufruf kompilieren würde.

Versuchen Sie die Aufgabe nach Möglichkeit zuerst ohne Zuhilfenahme eines Compilers zu lösen.

```cpp
#include <array>
#include <vector>

template<typename T>
void f1(T) {}

template<typename T>
void f2(T&) {}

template<typename T>
void f3(const T&, const T&) {}

template<typename T>
void f4(const std::vector<T>&) {}


int main() {
    const int x = 42;

    f1(x);  // a) T = ?
    f1(42); // b) T = ?
    f1(&x); // c) T = ?

    f2(x);  // d) T = ?
    f2(42); // e) T = ?

    f3(x, 6);   // f) T = ?
    f3(x, 1.f); // g) T = ?

    f4(std::vector{1, 2, 3}); // h) T = ?
    f4(std::vector{42.f});    // i) T = ?
}
```

4 Ein einfacher Stack

In dieser Aufgabe geht es darum, ein einfaches Klassen-Template `Stack`[3] zu implementieren, das als Template-Parameter den Typ der Elemente bekommt und die folgenden Member-Funktionen bereitstellt:

- `push`: Bekommt ein Objekt vom Typ `T` als Parameter übergeben und fügt es zum Stack hinzu.
- `empty`: Gibt `true` zurück, wenn sich aktuell kein Objekt auf dem Stack befindet und sonst `false`.
- `pop`: Entfernt das zuletzt hinzugefügte Objekt vom Stack und gibt es als Rückgabewert zurück.

Zum Speichern der Elemente können Sie eine Member-Variable vom Typ `std::vector<T>` verwenden, die u.a. die folgenden Member-Funktionen hat:

- `push_back`: Fügt das übergebene Objekt hinten im Vektor hinzu.
- `back`: Gibt eine Referenz auf das letzte Element (hinten) im Vektor zurück.
- `pop_back`: Entfernt das letzte Element (hinten) aus dem Vektor.
- `empty`: Gibt `true` zurück, wenn der Vektor aktuell keine Elemente enthält.

Die beiden Funktionen `back` und `pop_back` von `std::vector` haben undefiniertes Verhalten, wenn der Vektor aktuell keine Elemente enthält. Deswegen sollten Sie in ihrer Implementierung der `pop` Funktion prüfen, ob es noch Elemente gibt und ggf. eine `std::out_of_range` Exception werfen.

Als Basis für die Implementierung können Sie folgenden Programmcode verwenden:

```
1 #include <iostream>
2 #include <stdexcept>
3 #include <vector>
4
5
6 int main() {
7     auto stack = Stack<int>();
8     stack.push(1);
9     stack.push(2);
10    stack.push(42);
11
12    while(!stack.empty()) {
13        std::cout << stack.pop() << "\n";
14    }
15
16    try {
17        stack.pop();
18        std::cout << "Exception wurde nicht geworfen!\n";
19
20    } catch(const std::out_of_range&) {
21        std::cout << "Exception funktioniert!\n";
22    }
23 }
```

[3] Deutsch: Stapelspeicher oder Kellerspeicher