C/C++ Programmierung
Übungsblatt 13

Florian Oetke

Entwicklungsumgebung
Dieses Übungsblatt kann sowohl mit einer IDE Ihrer Wahl als auch mit godbolt.org bearbeitet werden.

Für Aufgaben die in C statt C++ bearbeitet werden sollen (Aufgabe 1) müssen Sie ihre Dateien mit .c statt .cpp benennen bzw. eine Konfiguration von godbolt.org verwenden, die für C konfiguriert ist.

1 Quadratische Gleichungen Einlesen und Lösen

Lösen Sie die folgende Aufgabe nur mit C und nicht mit C++. Bei Ihrer Lösung sollten Sie also darauf achten, dass sie auch tatsächlich einen C-Compiler verwenden (.c Datei, CMake-Projekt aus dem Handout oder für C konfigurierter Compiler-Explorer), damit Sie nicht unabsichtlich C++-Features verwenden.

Schreiben Sie ein Programm das quadratische Gleichungen nach dem Schema `a x^2 + b x + c = y` mit `scanf` von der Standardeingabe einließt, die Gleichung zur Sicherheit nochmal mit `printf` ausgibt, die Lösungen (d.h. all Werte für x, die zum entsprechenden Ergebnis führen) berechnet und diese mit `printf` ausgibt. Ihr Programm soll solange Gleichungen einlesen bis eine Eingabe unvollständig oder fehlerhaft war.

Beim Einlesen der Werte können Sie sich zunutze machen das der Format-String für `scanf` auch zusätzliche Zeichen enthalten kann, die zwar in der Eingabe vorkommen müssen aber nicht in Variablen gespeichert werden. So führt `scanf("x %f", &val)` dazu das ein `float`-Wert eingelesen wird aber nur wenn vor diesem ein x (gefolgt von beliebig vielen Leerzeichen) steht. Ihr Programm muss nicht alle möglichen Schreibweisen für quadratische Gleichungen akzeptieren, sondern lediglich solche wie sie unten im Beispiel angegeben wurden.

Sie können Ihre Lösung mit der folgenden Eingabe testen:

```
2x^2 + 3x + -1 = 1
1x^2 + 1x + 1 = 0
1x^2 + 0 x + 1 = 1
```

Bei einer korrekten Lösung sollte Ihr Programm für diese Eingaben folgendes ausgeben:

```
2.000000x^2 + 3.000000x + -1.000000 = 1.000000
  L_x = {0.500000, -2.000000}
1.000000x^2 + 1.000000x + 1.000000 = 0.000000
  L_x = {}
1.000000x^2 + 0.000000x + 1.000000 = 1.000000
  L_x = {0.000000}
```

2 Eigenes Assert Makro

Schreiben Sie eine Makrofunktion `MY_ASSERT`, die als Parameter eine Bedingung und eine Nachricht bekommt. Ihr Makro sollte die übergebene Bedingung auswerten und eine Fehlermeldung – bestehend aus der Bedingung, der Datei, der Zeile sowie der Nachricht – ausgeben und eine `std::logic_error` werfen, wenn die Bedingung nicht erfüllt ist.

Im vorgegebenen Code sehen Sie außerdem einen etwas problematischeren Fall, bei dem das Marko in einem `if` ohne Klammern verwendet wird. Optional können Sie überlegen wie Sie ihr Makro erweitern müssen, um auch mit diesem Fall korrekt umgehen zu können.

Als Basis für die Implementierung sollten Sie den im Handout bereitgestellten Programmcode verwenden. Bei einer korrekten Lösung sollte Ihr Programm die folgende Ausgabe erzeugen:

```
Assertion "var==3 || var<0" failed at ./example.cpp line 24: Sollte ausloesen!
Funktioniert
```

3 Int-Vector mit malloc und free

In dieser Aufgabe sollen Sie die C-Funktionen zur manuellen Speicherverwaltung verwenden um selbst einen einfachen Container zu implementieren.

Implementieren Sie eine Klasse `Int_vector` die ein Array von int-Objekten auf dem Heap verwaltet[1] und folgende Methoden bereitstellt:

- `push_back`: Bekommt einen `int` als Parameter und fügt in an das Ende des Vectors hinzu.
- `pop_back`: Entfernt das Elemente am Ende des Vectors. Wenn es keine Elemente gibt, soll statt dessen eine `std::out_of_range` Exception geworfen werden.
- `operator[]`: Bekommt einen Index als Parameter und gibt eine Referenz auf das entsprechende Element zurück. Wenn es kein solches Element gibt, soll eine `std::out_of_range` Exception geworfen werden. Der Operator soll dabei keine Kopie sondern eine Referenz zurückgeben und unabhängig davon anwendbar sein ob der `this`-Pointer `const` ist oder nicht. Das heißt Sie müssen zwei Varianten des Operators implementieren. Eine die als `const` markiert ist und entsprechend eine `const` -Referenz zurückgibt sowie eine Variante bei der dies nicht der Fall ist.
- `size`: Gibt die Anzahl Elemente zurück.

Die Größe des verwalteten Speichers auf dem Heap soll anfangs Null sein und exponentiell wachsen, wenn Elemente mit `push_back` hinzugefügt werden. Um zu validieren ob sich ihre Implementierung korrekt verhält, sollten Sie immer wenn Sie neuen Speicher anfordern `"<malloc> "` mit `std::cout` ausgeben, sodass Sie eine Ausgabe wie unten gezeigt erhalten.

Denken Sie auch daran den reservierten Speicher im Destruktor wieder freizugeben. Die anderen Special-Member-Functions brauchen Sie nicht zu implementieren, sollten Sie dann allerdings mit `= delete` unterdrücken.

Als Basis für die Implementierung sollten Sie den im Handout bereitgestellten Programmcode verwenden. Die Ausgabe für eine korrekte Lösung sollte damit so aussehen:

```
<malloc> 0 1 <malloc> 2 3 4 5 <malloc> 6 7 8 9 10 11 12 13 <malloc> 14

42, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
```

[1] Beachten Sie, dass wir mit der hier verwendeten Implementierung lediglich triviale built-in Typen, wie `int`, speichern können. Für komplexere Typen mit Konstruktoren/Destruktoren müssten wir zusätzlich Placement-new verwenden und außerdem einige Regeln zur Gültigkeit von Pointern und Lebenszeit von Objekten berücksichtigen, die wir in der Vorlesung nicht besprochen haben.

4 Beispiele für Klausuraufgaben

Diese Aufgabe besteht aus einigen kleineren Teilaufgaben, die an Klausuraufgaben angelehnt sind. Entsprechend sollten Sie versuchen diese Aufgaben ohne technische Hilfsmittel (Compiler, IDE, ...) zu lösen.

4.1 Dangling Pointer

Beschreiben Sie kurz in eigenen Worten um was es sich bei einem Dangling Pointer handelt, wie es zu einem solchen Pointer kommen kann und was passierend könnten wenn wir ihn dereferenzieren.

4.2 Storage Durations

Nennen Sie die vier unterschiedlichen Storage Durations, die wir in der Vorlesung kennengelernt haben. Beschreiben Sie eine von diesen genauer.

4.3 MIN Makro

Definieren Sie eine Makrofunktion `MIN`, die zwei Parameter erhält und zum kleineren der beiden resultierenden Werte ausgewertet wird, sodass der unten angegebene Code sich wie vorgegeben verhält.

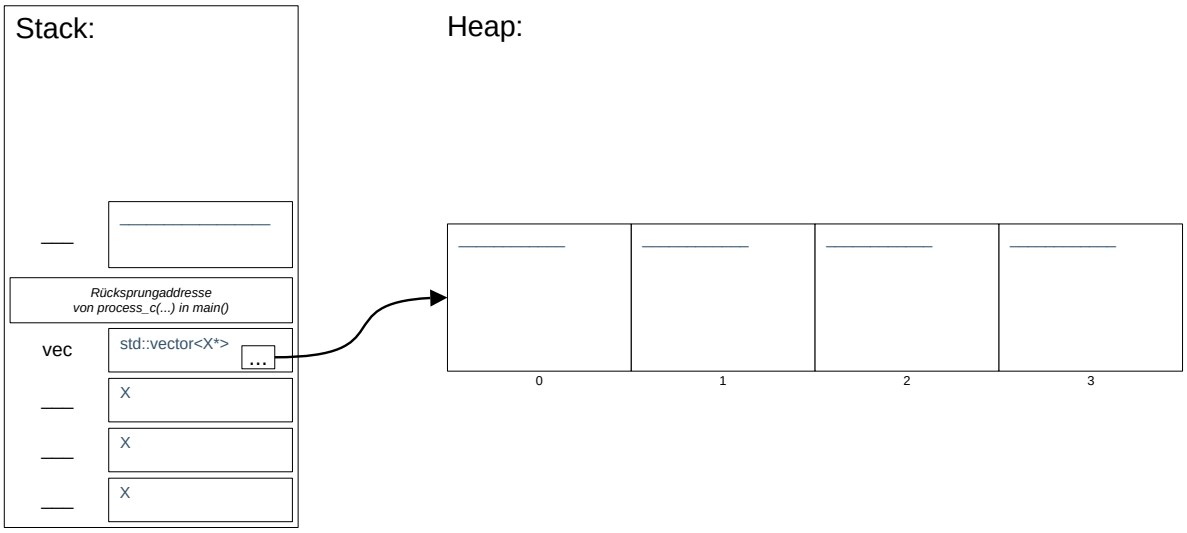
```
1  std::cout << MIN(2,   3) << '\n'; // Ergebnis: 2
2  std::cout << MIN(2|4, 3) << '\n'; // Ergebnis: 3
```

4.4 Quellcode lesen

a) Lesen Sie den folgenden Code und geben Sie in den markierten Zeilen jeweils die Ausgaben an, die von dieser Zeile ausgelöst werden oder – wenn die Zeile keine Ausgabe erzeugt. Achten Sie innerhalb der selben Zeile auch auf die Reihenfolge der Ausgaben!

```
1  #include <iostream>
2  #include <vector>
3
4  class X {
5    public:
6      X()          {std::cout << "1 ";}
7      X(int)       {std::cout << "2 ";}
8      X(const X&) {std::cout << "3 ";}
9      X(X&&)       {std::cout << "4 ";}
10     ~X()         {std::cout << "5 ";}
11
12     X& operator=(const X&) {
13         std::cout << "6 ";
14         return *this;
15     }
16     X& operator=(X&&) {
17         std::cout << "7 ";
18         return *this;
19     }
20 };
21
22 void process_a(X) {}
23 void process_b(X&) {}
24 void process_c(const std::vector<X*>& vec_ref) {} // <= Speicherabbild
25
26 int main() {
27     X x1;                              // Ausgabe:
28     X x2 = X{};                        // Ausgabe:
29     X x3 = 5;                          // Ausgabe:
30     x3 = x2;                           // Ausgabe:
31
32     auto vec = std::vector<X*>{&x3}; // Ausgabe:
33     vec.push_back(&x2);                // Ausgabe:
34     vec.push_back(&x3);                // Ausgabe:
35
36     process_a(x1);                     // Ausgabe:
37     process_b(x1);                     // Ausgabe:
38     process_c(vec);                    // Ausgabe:
39     process_a(std::move(x1));          // Ausgabe:
40
41 }                                      // Ausgabe:
```

b) Ergänzen Sie das folgende Speicherabbild (Namen, Typen und ggf. Pfeile für Pointer oder nullptr), sodass es den Zustand des Programms in Zeile 24 – d.h. direkt nach dem betreten der Funktion aus Z.38 – darstellt.



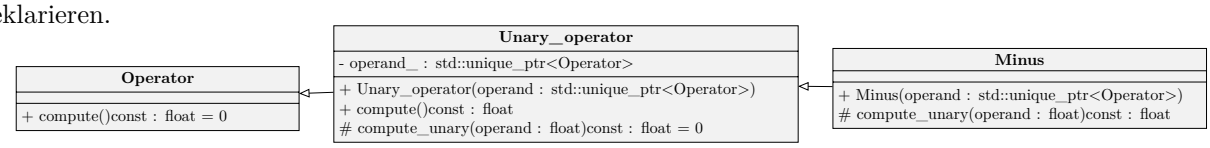
4.5 Taschenrechner

Schreiben Sie eine main-Funktion, die in einer Schleife einfache Eingaben nach dem Schema `Zahl Operator Zahl` von der Standardeingabe einließt und das Ergebnis der Berechnung auf der Standardausgabe ausgibt. Für die Zahlen sollen beliebige Fließkommazahlen und für den Operator ein einzelnes `char` (`'+'`, `'-'`, `'*'` oder `'/'`) erlaubt sein. Eine Eingabe `1.5 * 2` sollte also z.B. `3` ausgeben.

Ihr Programm soll solange weiterlaufen bis eine ungültige Eingabe (keine gültigen Zahlen oder unbekannter Operator) getätigt wurde.

4.6 Klassenhierarchie

Implementieren Sie die Klassenhierarchie, die durch das folgende UML-Diagramm beschrieben wird. Sie müssen keine Definition für die Methoden angeben, d.h. es genügt wenn Sie die Methoden in der Klasse deklarieren.



4.7 Vektoren konkatenieren

Schreiben Sie ein Funktions-Template das für einen beliebigen Typen `T` zwei `std::vector` mit Elementen von diesem Typen als Parameter bekommt und einen neuen `std::vector` zurückgibt, der zuerst alle Elemente des ersten und dann alle Elemente des zweiten Vectors enthält.

4.8 Operatoren

Definieren Sie ein `struct Vec2` mit zwei Member-Variablen vom Typ `float`. Überladen Sie für diesen Typen den Additions- und den <<-Operator, sodass der unten angegebene Code sich wie vorgegeben verhält.

```
1  Vec2 v = Vec2{1,2} + Vec2{3,4};
2  std::cout << v << '\n';          // Ausgabe: 4/6
```

5 Anhang

Zur Lösungen der Klausur-Aufgaben können Sie u.a. die folgenden Klassen und Funktionen aus der Standardbibliothek verwenden[2]. Zur Vereinfachung sind die Funktionen als Pseudocode angegeben und es wurden folgenden Aspekte ausgelassen:

- Special-Member-Functions
- `std` Namespace
- Zusätzliche optionale Template-Parameter
- Weitere Funktionen, die in der Vorlesung nicht behandelt wurden oder die für die Aufgaben nicht relevant sind
- `const` Überladung von Methoden (die entsprechenden Methoden sind mit (C) markiert)
- Aspekte der Typen und Methoden, die als gegeben vorausgesetzt werden und ggf. in anderen Aufgaben abgefragt werden

5.1 `vector<T>`

5.1.1 Konstruktoren

- `vector() noexcept;`
- `vector(size_t count, const T& value);`
- `explicit vector(size_t count);`
- `vector(Iterator src_begin, Iterator src_end);`

5.1.2 Member Funktionen

- `T& at(size_t pos);` (C)
- `T& operator[](size_t pos);` (C)
- `T& front();` (C)
- `T& back();` (C)
- `T* data() noexcept;` (C)
- `iterator begin() noexcept;` (C)
- `iterator rbegin() noexcept;` (C)
- `iterator cbegin() const noexcept;`
- `iterator crbegin() const noexcept;`
- `iterator end() noexcept;` (C)
- `iterator rend() noexcept;` (C)
- `iterator cend() const noexcept;`
- `iterator crend() const noexcept;`
- `bool empty() const noexcept;`
- `size_t size() const noexcept;`
- `void reserve(size_t new_cap);`
- `size_t capacity() const noexcept;`
- `void shrink_to_fit();`
- `void clear() noexcept;`
- `iterator insert(const_iterator pos, const T& value);`
- `iterator insert(const_iterator pos, T&& value);`
- `iterator insert(const_iterator pos, size_t count, const T& value);`
- `iterator insert(const_iterator pos, Iterator in_begin, Iterator in_end);`
- `iterator emplace(const_iterator pos, A&&... args);`
- `iterator erase(const_iterator pos);`
- `iterator erase(const_iterator from, const_iterator to);`
- `void push_back(const T& value);`
- `void push_back(T&& value);`
- `T& emplace_back(A&&... args);`
- `void pop_back();`
- `void resize(size_t count);`
- `void resize(size_t count, const T& value);`

5.1.3 Freie Funktionen

- `void erase(vector<T>& c, const U& value);`
- `void erase_if(vector<T>& c, UnaryPredicate p);`

5.2 Allgemeine freie Funktionen

- `T exchange(T& obj, U&& new_value );`
- `remove_reference_t<T>&& move(T&&) noexcept;`
- `void swap(T&, T&) noexcept;`

[2] Eine Übersicht in dieser Form, mit allen potentiell relevanten Typen und Funktionen, wird es auch auf der letzten Seite der Klausur geben.

5.3 istream

5.3.1 Member Funktionen

- `int get();`
- `int peek();`
- `istream& unget();`
- `istream& ignore(streamsize count=1, int delim=EOF);`
- `istream& read(char* s, streamsize count);`
- `streamsize gcount()const;`
- `streampos tellg();`
- `istream& seekg(streamoff off, ios::seekdir dir=ios::seekdir::beg);`
- `bool good()const;`
- `bool eof()const;`
- `explicit operator bool()const;`
- `void clear(ios::iostate state = ios::goodbit);`

5.3.2 Freie Funktionen

- `istream& getline(istream& input, string& str, char delim);`

5.4 ostream

5.4.1 Member Funktionen

- `ostream& put(char c);`
- `istream& write(const char* s, streamsize count);`
- `streampos tellp();`
- `istream& seekp(streamoff off, ios::seekdir dir=ios::seekdir::beg);`
- `bool good()const;`
- `bool eof()const;`
- `explicit operator bool()const;`
- `void clear(ios::iostate state = ios::goodbit);`