Programmiersprachen
Sommersemester 2019

Aufgabensammlung 1

Dieses Aufgabenblatt soll eine Einführung in die Programmiersprache C++ geben. Lösen Sie die Aufgaben selbstständig und gründlich, um sich mit der Syntax und den Begrifflichkeiten im Umgang mit C++ vertraut zu machen.
Informieren Sie sich ü¼ber wichtige Fachbegriffe (*kursiv*) und benutzen Sie diese auch, wenn Sie über Quellcode reden.
Die Lösungen der Aufgaben sind in den Übungen **am 15.April** vorzustellen.

Nutzen Sie ausschließlich aktuelle Fachliteratur, z.B.

- Stroustrup, B.: Einführung in die Programmierung mit C++ (2010)
- Stroustrup, B.: Programming: Principles and Practice Using C++ (2014)

Die 2. und 3. Kapitel in diesen Büchern enthalten für dieses Aufgabenblatt hilfreiche Informationen.
Folgende Webseite ist eine gute Online-Referenz:

- http://en.cppreference.com/

Zum schnellen Testen kleinerer Codefragmente ist diese Seite geeignet:

- http://cpp.sh

Zur Verwaltung Ihrer Software verwenden Sie das in der Übung vorgestellte Versionierungssystem `git`.

- https://git-scm.com/book/en/v2

Bei Fragen und Anmerkungen schreiben Sie bitte ein Email an adrian.kreskowski@uni-weimar.de .

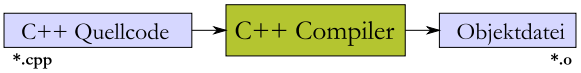
Hello, World!

Das klassische erste Programm in C++ sieht wie folgt aus:

```
#include <iostream> // Ein- / Ausgabe

// C++ Programme beginnen mit der Ausfuehrung der Funktion main
int main()
{
  std::cout << "Hello, World!\n";
  return 0;
}
```

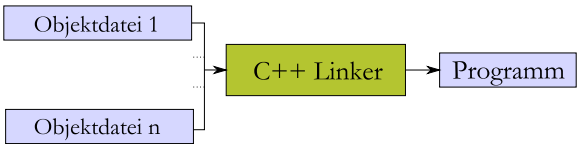
Bevor Sie das Programm ausführen können, müssen Sie es *kompilieren* und *linken*. Für den ersten Schritt benutzen wir einen Compiler, welcher den Quellcode in Objektcode übersetzt .



Schreiben Sie den Quelltext in eine Textdatei mit dem Namen *helloworld.cpp* und kompilieren Sie den Quellcode wie folgt.

```
user@host: g++ -c helloworld.cpp
```

In Ihrem Ordner sollte jetzt die Datei *helloworld.o* liegen.
Um ein ausführbares Programm zu erstellen, müssen Sie den Objektcode linken. In der Regel haben wir es mit mehreren Objektdateien zu tun, die vom Linker zusammengefügt werden.



Beim Linken werden die Objektdateien als Argument übergeben und das ausführbare Programm mit `-o` benannt. In unserem konkreten Falle haben wir nur eine Objektdatei.

```
user@host: g++ helloworld.o -o helloworld
```

Das Programm kann nun ausgeführt werden.

```
user@host: ./helloworld
```

Was bedeutet die Zeile `return 0;` im obigen Programm?
Wie sieht das minimalste C++ Programm aus, dass noch kompiliert? Also was können Sie alles aus dem dem obigen Programm entfernen?

Aufgabe 1.1

Erstellen Sie einen *Fork* des Repository https://github.com/vrsys/programmiersprachen-helloworld. Sie benötigen dazu einen *github*-login. Anschließend *clonen* Sie sich das Repository wie in der Übung vorgestellt, z.B.
`git clone https://github.com/foo/programmiersprachen-helloworld.git`
oder mit TortoiseGit unter Windows: https://vimeo.com/9984118.
Wir verwenden das Programmierwerkzeug *CMake* um den Buildprozess zu managen. Erstellen Sie im Repository einen Unterordner mit dem Namen *build*. Dieser Ordner wird von *CMake* verwendet, um Ihr Projekt zu managen. Rufen Sie folgenden Befehl im Ordner *build* auf: `cmake ..`
Damit haben Sie Ihr Projekt konfiguriert. Nun können Sie die Anwendung bauen lassen: `make`
Wechseln Sie anschließend in den Ordner *source*, der nun im von Ihnen angelegten Ordner *build* erzeugt wurde. Führen Sie die erstellten Programme `helloworld` und `tests` aus.

Aufgabe 1.2

Erstellen Sie nun unter Ihrem eigenen *github*-Account ein neues Repository mit dem Namen *programmiersprachen-aufgabe-1*. Klonen Sie das erstellte Repository. Erstellen Sie darin die gleiche Struktur wie im *helloworld*-Repository. Commiten Sie alle Änderungen und pushen Sie diese auf *github*. Verwenden Sie dieses Repository für Ihre Lösungen. **[2 Punkte]**

Aufgabe 1.3

Schreiben Sie ein Programm, welches die kleinste Zahl, die durch die Zahlen 1 bis 20 teilbar ist errechnet und auf der Konsole ausgibt. **[2 Punkte]**

Aufgabe 1.4

Informieren Sie sich über *Typen* in C++. Benennen Sie im Folgenden Beispiel *Typen*, *Variablen* und *Werte*. Erklären Sie in diesem Zusammenhang das Schlüsselwort `const`. Was ist eine Typkonvertierung und wieso kann es dabei Probleme geben? Wo sehen Sie Probleme im nachfolgenden Programmcode?

```
int main() {
  int        a     = 9;
  bool       b     = false;
  char       c     = 'a';
  double     d     = 1.3;
  int const  five  = 5;
  double     e     = a/two;

  five = d;

  return 0;
}
```

[4 Punkte]

Aufgabe 1.5

Erklären Sie die Begriffe *Initialisierung* und *Zuweisung* und geben Sie jeweils ein Beispiel an. Welche Unterschiede gibt es?

- Erklärung: Initialisierung 1 P
- Erklärung: Zuweisung 1 P
- Erklärung: Beispiele Initialisierung 1 P
- Erklärung: Beispiele Zuweisung 1 P
- Unterschied 1 P

[5 Punkte]

Aufgabe 1.6

Erklären Sie die Begriffe *Deklaration* (manchmal auch Forward-Deklaration genannt) und *Definition* und geben Sie jeweils ein Beispiel an. Welche Unterschiede gibt es? Geben Sie Beispiele für Deklaration und Definition im Kontext von Funktionen, Variablen und Klassen.

Hinweis: extern Variable **[5 Punkte]**

Aufgabe 1.7

Was gehört zur *Signatur* einer Funktion? *Variablen* haben einen Gültigkeitsbereich (*scope*). Bestimmen Sie die Gültigkeitsbereiche aller *Variablen* im folgenden Beispiel.

```
#include <iostream>

int var = 3;

double sum(double a, double b)
{
  return a + b;
}

int square(int var)
{
  return var * var;
}

int main()
{
  for (int i = 0; i != 100; ++i) {
    std::cout << "i^2 = " << square(i) << '\n';
    std::cout << "i+i = " << sum(i,i)  << '\n';
  }
  return 0;
}
```

[3 Punkte]

Aufgabe 1.8

Testgetriebene Entwicklung bzw. TDD (*Test-driven development*) ist eine Vorgehensweise zur Gewährleistung der Programmintegrität bei der Softwareentwicklung. Dabei wird zuerst ein Testprogramm implementiert, welches die gewünschten Fähigkeiten über ein festgelegtes Interface der zu implementierenden Funktionen bzw. Klassen nutzt und prüft, ob Rückgabewerte und Zustände den Erwartungswerten entsprechen.
Da die Festlegung der zu erwarteten Zustände manuell erfolgt, wird sich auf spezielle Testfälle beschränkt.

Getestet werden müssen:

- Parameter, die besonderes Verhalten hervorrufen sollten (z.B. Division durch 0 muss abgefangen werden),
- Parameter, die direkt an Sonderfälle angrenzen (z.B. Division durch -1 und 1)
- und einige zufällige Parameter in einem sinnvollen Bereich, die normales Verhalten hervorrufen sollten (z.B. Division durch 53; durch -23, durch 7513812, ...).

Nach der Definition der Test-Cases wird versucht das Test-Programm, welches die festgelegten Interfaces benutzt, zu kompilieren. Unter Nutzung der Fehlermeldungen des Compilers beginnt man nun die eigentliche Implementierung der Funktionalität, sodass schlussendlich Testprogramm kompiliert und einen erfolgreichen Durchlauf aller zu testenden Funktionalität meldet.

Die C++-Bibliothek Catch unterstützt Test-driven development. Unter folgendem Link finden Sie eine Einführung in Catch:
https://github.com/philsquared/Catch/blob/master/docs/tutorial.md.
Ihre Aufgabe ist es nun eine Funktion `gcd` zu implementieren, die den größten gemeinsamen Teiler zweier ganzen Zahlen bestimmt, die nicht beide Null sein dürfen. Erweitern Sie die Datei `tests.cpp` um den Test `describe_gcd`.

```
#define CATCH_CONFIG_RUNNER
#include "catch.hpp"
#include <cmath>

int gcd(int a, int b)
{
  return 1;
}

TEST_CASE("describe_gcd", "[gcd]")
{
  REQUIRE(gcd(2,4) == 2);
  REQUIRE(gcd(9,6) == 3);
  REQUIRE(gcd(3,7) == 1);
}

int main(int argc, char* argv[])
{
  return Catch::Session().run( argc, argv );
```

```
}
```

Implementieren Sie nun die Funktion `gcd` in der Datei `tests.cpp`. **[5 Punkte]**

Aufgabe 1.9

Entwickeln Sie mit TDD die Funktion `checksum` zur Berechnung der Quersumme einer ganzen Zahl. Testen Sie die Funktion unter anderem mit Ihrer Matrikelnummer.

[3 Punkte]

Aufgabe 1.10

Schreiben Sie eine Funktion `sum_multiples`, die alle Zahlen von 1 bis einschließlich 1000 aufaddiert, die durch 3 oder 5 teilbar sind. Testen Sie die Funktion mittels TDD. **[3 Punkte]**

Aufgabe 1.11

Entwickeln Sie mit TDD die Funktion `fract` zur Berechnung des Nachkomma-Anteils einer Gleitkommazahl. Da hier Gleitkommazahlen verwendet werden, müssen Sie in Ihren Tests `Approx` verwenden. Im nachfolgenden Beispiel wird überprüft, ob der Wert d bis auf einen kleinen Nachkommawert von e abweicht:
`REQUIRE(e == Approx(d));` **[3 Punkte]**

Aufgabe 1.12

Entwickeln Sie mit TDD Funktionen zur Berechnung des Volumens und der Oberfläche eines Zylinders.

Hinweis: Verwenden Sie `#include <cmath>` zur Verwendung mathematischer Funktionen und Konstanten. **[3 Punkte]**

Aufgabe 1.13

Entwickeln Sie die Funktionen `factorial` testgetrieben. Die Funktion soll einer natürlichen Zahl das Produkt aller natürlichen Zahlen kleiner und gleich dieser Zahl zuordnen. **[5 Punkte]**

Aufgabe 1.14

Entwickeln Sie die Funktion `is_prime` testgetrieben.

Hinweis: Der Modulo-Operator `%` gibt den Rest einer ganzzahligen Division zurück. **[5 Punkte]**

Aufgabe 1.15

Entwickeln Sie mit TDD die Funktion `double mile_to_kilometer(double)`, welche Meilen in Kilometer umrechnet.
Schreiben Sie anschließend ein interaktives C++ Programm, das einen Meilenwert von der Kommandozeile einliest, in Kilometer umrechnet und anschließend auf der Konsole ausgibt. Legen Sie dazu die Datei `mile_to_kilometer.cpp` im Ordner `source/` des Projekts an. Kommentieren Sie außerdem die letzten beiden Zeilen in der `source/CMakeLists.txt` ein. Hier ist ein kurzes Beispiel zur Ein- und Ausgabe in C++.

```
#include <iostream>
#include <string>

int main()
{
  std::cout << "Bitte geben Sie Ihren Vornamen ein:\n";
  std::string first_name;
  std::cin >> first_name;

  std::cout << "Bitte geben Sie Ihr Alter ein:\n";
  int age = 0;
  std::cin >> age;
  std::cout << "Hallo " << first_name
            << " (Alter: " << age << ")\n";
  return 0;
}
```

[5 Punkte]

Schlüsselwörter 1.16

Schreiben Sie für jeden Begriff in dieser Liste eine Definition auf. Sie können dazu das Buch “Einführung in die Programmierung mit C++” benutzen.

- C++
- Quellcode
- Compiler
- Linker
- Objektcode
- Ausführbare Datei
- `main()`
- `#include`
- Kommentar
- Header
- Programm
- Ausgabe
- `std::cout`
- `std::cin`
- `<<`
- `>>`
- Funktion
- Funktionssignatur
- Deklaration
- Definition
- Typ
- Typkonvertierung
- Variable
- Name
- Wert
- Initialisierung
- Zuweisung
- const
- Gültigkeitsbereich

Diese Aufgabe soll Ihnen als Einstieg dienen ein eigenes Glossar aufzubauen.

[5 Punkte]

[Gesamt: 58 Punkte]