Einführung in C/C++

Prof. Dr. Alexander Voß
SS 2014 28.03.2014

Übungsaufgaben

♙ Basisaufgabe ♖ Ähnliche Aufgabe ♕ Anspruchsvolle Aufgabe

Aufgabe 001 ♙

[`hello_world` compilieren und starten]

Führen Sie folgende Schritte aus:

- *VirtualBox* starten.
- Appliance *Lubuntu SS2014* in VirtualBox importieren.
- Virtuelle Maschine (VM) starten.
- Dateimanager *PCManFM* in VM starten.
- In den Ordner `cpp_ss2014/lectures/01_hello_world_C` navigieren.
- Doppelt auf `01_hello_world_C.c` klicken – der Editor *Geany* öffnet sich.
- Im Editor das Programm `01_hello_world_C.cout` durch Ausführen der Befehle *Make* und *Ausführen* (Buttonleiste oder Menü) erzeugen und starten.

Fragen Sie, falls Sie nur *VMware* benutzen können!

Aufgabe 002 ♙

[Kommandozeilen-Befehle]

Starten Sie *LXTerminal* und probieren Sie die folgenden Kommandozeilen-Befehle aus bzw. lesen deren Bedeutung *(z.B.* `man ls`*):*

- *mkdir, rmdir, cd,*
- *ls, cp, mv, rm,*
- *cat, more, chmod, jobs, kill, bg,*
- *zip, unzip*

Beachten Sie:

- Wenn Sie einen Befehl mit einem `&` abschließen, wird dieser im Hintergrund ausgeführt, ohne auf das Ende zu warten (z.B. `geany x.c &`).
- Wenn Sie eine Datei mit `rm` löschen, ist sie endgültig gelöscht.

Fragen Sie, falls Sie keine Erfahrung mit Kommandozeilen haben!

Aufgabe 003 ♙

[Shared Folder]

Richten Sie einen *Shared Folder* in Ihrer VM ein. Heisst dieser `host`, so wird er automatisch eingebunden. Führen Sie diese Schritte durch:

- VM ordnungsgemäß runterfahren
- In den Einstellungen von *VirtualBox* einen Eintrag im Reiter *Shared Folders* unter *Machine Folders* einrichten. Verweisen Sie auf einen Ordner auf Ihrem Rechner und nennen diesen Eintrag `host`.
- VM neu starten.
- Im Verzeichnis `host` sollte der eingebundene Ordner sichtbar sein.

Fragen Sie, falls *mounten* nicht automatisch funktioniert.

Übungsaufgaben

Aufgabe 004 ♙ [printf]

Führen Sie folgende Schritte aus und verstehen Sie den Code:

- Im Dateimanager *PCManFM* ins Verzeichnis `cpp_ss2014/exercises/A004_C` wechseln und die c-Datei dort im Editor *Geany* öffnen.
- Das Programm erstellen und starten.
- Einen beliebigen Text verändern und mit `printf` ausgeben.
- Eine Integer-Variable mit `printf` ausgeben.
- Die 6'er-Reihe von 0*6 bis 9*6 mit `printf` ausgeben.

Aufgabe 005 ♙ [Snippets]

Verstehen Sie alle Programme in den Ordnern:

- `01_hello_world_C`
- `02_first_steps_C`
- `03_control_flow`

Fragen Sie bei Unklarheiten!

Aufgabe 006 ♙ [Datentypen, Schleifen, Testen]

Berechnung der Potenz:

- Berechnen Sie b^n (b hoch n) für feste natürliche Zahlen b und n (alternativ: Lesen Sie b und n von der Konsole ein).
- Schreiben Sie eine Funktion `Pot`, die b und n übergeben bekommt und das Ergebnis zurückgibt und ausgibt.
- Schreiben Sie repräsentative Testfälle für die Funktion in der Art:
  `assert( 8==Pot(2,3) );`

Aufgabe 007 ♖ [Datentypen, Schleifen, printf]

Berechnung von Primzahlen:

- Berechnen Sie alle Primzahlen bis zu einer festen natürlichen Zahl n und geben Sie diese aus (alternativ: Lesen Sie n von der Konsole ein).
- Geben Sie sie rechtsbündig mit der Breite 10 aus.

Aufgabe 008 ♖ [Datentypen, Schleifen, Funktionsaufrufe, Testen]

Berechnung der Fibonacci-Folge:

- Berechnen Sie die ersten Fibonacci-Zahlen bis zu einem festen Index n (alternativ: Lesen Sie n von der Konsole ein):
  $f_0 = 0, f_1 = 1, f_n = f_{n-2}+f_{n-1}$ für $n>1$
  nicht-rekursiv in einer Schleife und geben Sie sie rechtsbündig aus.
- Formulieren Sie die Schleife als *for, do-while* und *while*-Schleife.
- Berechnen Sie die Folge rekursiv. Schreiben Sie repräsentative Testfälle.

Übungsaufgaben

Aufgabe 009 ♙ [Zeiger]

Zeiger definieren und benutzen:

- Legen Sie eine `int` und eine `float` Variable an und initialisieren Sie sie mit beliebigen Werten.
- Legen Sie zwei Zeiger an, die jeweils auf diese Variablen zeigen.
- Verändern Sie die Werte der Variablen mit Hilfe der Zeiger und
- geben Sie jeweils die Variable und den Wert, auf den der zugehörige Zeiger zeigt, aus.

Aufgabe 010 ♙ [Zeiger, Arrays, scanf]

Werte von der Tastatur einlesen:

- Lesen Sie eine ganze Zahl von der Kommandozeile mittels `scanf` ein und geben Sie sie aus.
- Lesen Sie einen max. 15 Zeichen langen String mittel `scanf` ein und geben ihn aus.

Aufgabe 011 ♙ [Felder, Zeiger]

Mit Feldern und Zeigern arbeiten:

- Legen Sie einen C-String mit einem beliebigen Text an:
  `char* str = "hallo";`
- Legen Sie einen Zeiger auf `char` an und laufen Sie mit diesem durch das Feld `str` (einschliesslich '`\0`') um den jeweiligen Charakter, auf den der Zeiger zeigt, auszugeben.
- Geben Sie den jeweils aktuellen Charakter einmal als Charakter und einmal als ASCII Wert aus.
- Geben Sie zusätzlich auch den Wert des Zeigers aus (die Adresse).

Aufgabe 012 ♖ [Felder, Zeiger]

- Legen Sie ein Feld von `int`s der Größe 10 an.
- Initialisieren Sie dieses Feld mit der 3-er Reihe (1*3, 2*3, ...).
- Legen Sie einen Zeiger auf `int` an und laufen Sie mit diesem durch das Feld um die jeweilige Zahl, auf die der Zeiger zeigt, auszugeben.
- Geben Sie zusätzlich auch den Wert des Zeigers aus.

Aufgabe 013 ♙ [Felder, Zeiger, malloc, free]

- Legen Sie das Feld aus Aufgabe 012 dynamisch an (`malloc`) und
- geben Sie es am Ende auch wieder frei (`free`).

Aufgabe 014 ♖ [Zeiger]

- Versuchen Sie, durch Verwendung von Zeigern einzelne Bytes von `int`s zu manipulieren.

Übungsaufgaben

Aufgabe 015 ♕ [Zeiger]

- Versuchen Sie, über Zeiger-Manipulationen lokale Variablen gezielt auf dem Stack zu verändern, ohne diese direkt zu adressieren.

Aufgabe 016 ♕ [Zeiger]

- Versuchen Sie, durch "bösartige" Manipulation des Stacks das Programm zum Absturz zu bringen.

Aufgabe 017 ♙ [Funktionen, Zeiger, Testen, assert]

Aufruf einer Funktion mit Zeigern:

- Schreiben Sie eine Funktion `Swap`, die zwei übergebene double-Zahlen tauscht.
- Schreiben Sie repräsentative Testfälle analog zu Aufgabe 006.

Aufgabe 018 ♙ [Strings, scanf]

- Lesen Sie ein Passwort von der Kommandozeile ein (einen Text) und bitten Sie den Benutzer nach der ersten Eingabe, diesen Text noch mal einzugeben. Stimmen beide Eingaben nicht überein, wird das Spiel wiederholt.

Aufgabe 019 ♙ [Strings, scanf]

- Lesen Sie solange Text von der Komandozeile ein, bis der Benutzer "Ende" schreibt.

Aufgabe 020 ♙ [Zeiger, Strings]

- Schreiben Sie eine Funktion `Cut`, die einen String auf eine angegebene Länge kürzt.

Aufgabe 021 ♖ [Zeiger, Strings, Testfälle]

- Schreiben Sie eine Funktion `Reverse`, die einen String "in-place" umdreht (also keine Kopie davon erzeugt).
- Schreiben Sie repräsentative Testfälle.

Aufgabe 022 ♖ [Zeiger, Strings, Testfälle]

- Schreiben Sie eine Funktion `Fork`, die einen String (char*) erhält und zwei identische Strings (Kopien) zurückgibt.
- Schreiben Sie repräsentative Testfälle

Aufgabe 023 ♕ [Zeiger, Strings, Testfälle]

- Schreiben Sie eine Funktion `FindAll`, die alle vorkommenden Muster in einem String findet und diese Menge als Feld von Positionen zurückgibt.
- Schreiben Sie repräsentative Testfälle.

Übungsaufgaben

Aufgabe 024 ♕ [Zeiger, Strings, Testfälle]

- Schreiben Sie eine Funktion `Split`, die einen String in Worte, getrennt durch Leerzeichen oder Tabs, aufteilt und als Feld von Strings zurückgibt.
- Schreiben Sie repräsentative Testfälle.

Aufgabe 025 ♙ [Zeiger]

Zeiger auf Zeiger:

- Definieren Sie zwei `int`s (`n1,n2`) mit beliebigen festen Werten, zwei nicht initialisierte Zeiger auf `int` (`p1,p2`) und einen nicht initialisierten Zeiger auf einen `int`-Zeiger (`pp`).
- Geben Sie den ints neue Werte, aber modifizieren Sie diese nur über pp.

Aufgabe 026 ♙ [Zeiger]

Zeiger aus Funktionen zurückgeben:

- Definieren Sie ein Funktion `AllocCharBuffer`, die dynamisch Speicher für n `char`s anlegt.
- Die Funktion bekommt als ein Argument die Anzahl n und hat keinen Rückgabewert (void).
- Bestimmen Sie einen geeigneten Weg, um die Adresse des dynamischen Speichers an einen Aufrufer der Funktion zurückzugeben.

Aufgabe 027 ♖ [Zeiger]

- Definieren Sie vier Worte *"Dies", "ist", "ein", "Satz"* und dynamisch (!) einen Speicher für vier Zeiger, initialisiert mit den Adressen dieser Worte (quasi ein Feld von Zeigern).
- Übergeben Sie dieses Zeigerfeld einer Funktion, um dort die Worte in umgekehrter Reihenfolge auszugeben.
- Versehen Sie die Worte mit einer Kennzeichnung, dass diese konstant sind.
- Definieren Sie alternativ und analog zu Aufgabe 026 eine Funktion, die den dynamischen Speicher für die Zeiger anlegt.

Aufgabe 028 ♙ [Zeiger, Konstanten]

Zeiger auf Konstanten, konstante Zeiger:

- Definieren Sie einen `float`, einen Zeiger auf diesen `float`, eine Konstante vom Typ `float`, einen Zeiger auf diese Konstante und einen konstanten Zeiger auf `float`.
- Versuchen Sie die `float` Variablen über die entsprechenden Zeiger zu ändern.
- Versuche Sie, den Zeigern einen anderen Wert zuzuweisen.

Was funktioniert und was nicht?

Übungsaufgaben

Aufgabe 029 ♙ [Files]

Dateien lesen und schreiben:

- Öffnen Sie eine Textdatei zum Lesen und eine zum Schreiben.
- Lesen Sie die erste Datei Zeile für Zeile ein und schreiben jede Zeile umgekehrt in die zweite Datei.
- Vergessen Sie nicht, beide Dateien wieder zu schliessen.

Aufgabe 030 ♙ [Files]

- Lesen Sie eine Textdatei Ihrer Wahl ein und schreiben Sie eine Textdatei, die die Häufigkeit aller vorkommenden Buchstaben/Zeichen in der Form
  'Zeichen' (ASCII) #Anzahl
  also beispielsweise
  `'A' (65) #123`
  für 123 mal das Zeichen '`A`' mit ASCII 65, enthält.

Aufgabe 031 ♖ [Files]

- Berechnen Sie die ersten 100 Fibonaccizahlen und schreiben Sie diese leserlich formatiert in eine Datei.

Aufgabe 032 ♖ [Files, Konvertierung]

- Erstellen Sie mit einem Editor eine Textdatei mit dem Inhalt

```
# Dies ist ein Kommentar!
100 200 300
# Jetzt kommt ein Text
"Hallo Leser"
```

- Öffnen Sie die Datei und lesen sie zeilenweise ein. Verarbeiten Sie die Zeilen wie folgt:
- Das Zeichen '`#`' zu Anfang einer Zeile markiert einen Kommentar. Speichern Sie alle Kommentarzeilen und geben Sie sie nach dem Einlesen hintereinander zusammen mit der Zeilennummer (bei 1 beginnend) aus.
- Steht zu Beginn der Zeile eine Zahl, so lesen Sie alle Zahlen der Zeile ein und wandeln diese in `int`s um (siehe auch Aufgabe 033). Speichern Sie diese und geben Sie sie am Ende nach den Kommentarzeilen zusammen mit der Zeilennummer aus.
- Fängt die Zeile mit einem Anführungszeichen '`"`' an, so lesen Sie diesen Text bis zum abschliessenden '`"`' analog zu den Zahlen ein, speichern ihn und geben ihn nach den Zahlen aus.
- Die Ausgabe zu obiger Datei lautet:

```
1, # Dies ist ein Kommentar!
3, # Jetzt kommt ein Text
2, 100
2, 200
2, 300
4, "Hallo Leser"
```

Übungsaufgaben

Aufgabe 033 ♙ [Konvertierung]

- Lesen Sie Text von der Konsole ein und versuchen Sie diesen in `int`, `float` und `double` Zahlen umzuwandeln.

Was passiert bei fehlerhaften Eingaben?

Aufgabe 034 ♙ [struct, union, enum]

Implementieren und Testen Sie eine Struktur, die die Farbwerte eines Pixels beschreibt:

- Der `struct` enthält Farbwerte R,G,B sowie einen Alpha-Wert, jeweils aus dem Bereich 0..255.
- Verpacken Sie diesen `struct` so, dass er auch als vorzeichenloser Integer betrachtet werden kann.
- Geben Sie einen `enum` mit gängigen Farbwerten an.

Aufgabe 035 ♙ [struct, arrays, function pointer]

Implementieren und Testen Sie eine Struktur, die ein Polynom beschreibt. Das Polynom ist durch seine (`double`) Koeffizienten eindeutig beschrieben. Programmieren Sie Funktionen, um

- ein Polynom zu erzeugen (`create`),
- auszuwerten (`eval`) und
- zu einem weiteren Polynom zu addieren (`add`).
- Wenn `P` ein Polynom bezeichne, dann ergänzen Sie die Funktion `eval` um einen Parameter `F`, der statt `P` an der Stelle `x` zu berechnen, zusätzlich noch `F` anwendet, also `F(P(x))` auswertet. Ist `F==NULL`, ist nur `P(x)` zu berechnen.

Aufgabe 036 ♖ [struct, union, enum, Zeiger]

Verkettete Listen:

- Entwerfen Sie eine Datenstruktur, um einen 4-Byte Wert zu realisieren. Ein 4-Byte-Wert, zusammen mit einer Typ-Information, kann als `char`, als kurzes Wort, als `int`-Wert oder `float`-Wert interpretiert werden.
- Entwerfen Sie eine Datenstruktur, um eine doppelt verkettet Liste von 4-Byte-Werten zu realisieren. Implementieren Sie eine Menge von Funktionen, um mit einer solchen Liste zu arbeiten.
- Ziel soll es sein, Zeichen, Worte (bis 4-Zeichen Länge) oder Zahlen in einer solchen Liste abzuspeichern und entsprechend dem jeweiligen Typ der Knoten auszugeben.

Aufgabe 037 ♖ [struct, Zeiger]

- Implementieren Sie einen Stack für `int`s und für Strings mit den entsprechenden Funktionen `pop` und `push`.

Übungsaufgaben

Aufgabe 038 ♖ [function pointer]

Geben Sie in Ihrem Programm eine Funktion an, die bei Beendigung aufgerufen wird. Lesen Sie nach unter `atexit`. Lassen Sie Ihr Programm abstürzen/terminieren - wird Ihre Funktion aufgerufen?

Aufgabe 039 ♖ [function pointer]

Schreiben Sie eine Funktion `Approx`, die einen Startwert `x0`, eine zu iterierende Funktion `f`, sowie ein `Eps` als Parameter erhält:

- `Approx` berechnet `x=f(x)`, mit `x=x0` zu Anfang, solange, bis der neue `x`-Wert sich vom vorhergehenden nicht mehr als `Eps` unterscheidet. Geben Sie den letzten berechneten Wert zurück.
- Schreiben Sie Testfälle für das Heron-Verfahren (Wurzel-Berechnung, siehe Wikipedia; zunächst für feste Werte a).
- Nutzen Sie Typedefs.

Aufgabe 040 ♙ [va_arg]

Funktionen mit beliebiger Anzahl von Argumenten:

- Schreiben Sie eine Funktion, die mit einer beliebigen Anzahl von `double`-Werten aufgerufen werden kann. Summieren Sie sie und geben das Ergebnis zurück.

Aufgabe 041 ♙ [static]

Statische Variablen:

- Implementierung Sie eine Berechnung der Fibonacci-Zahlen, die sich schon einmal berechnete Werte merkt.

Aufgabe 042 ♕ [static, realloc]

Schreiben Sie eine Funktion f zur Berechnung der Fakultät n!:

- Lesen Sie n wiederholt von der Tastatur ein und ermitteln dazu n! (Fakultät). Die Eingabe von n=-1 bricht das Programm ab.
- Ist zuvor schon einmal n! ermittelt worden, so nehmen Sie den Wert aus einem dynamischen Array ohne ihn zu berechnen. Ist er noch nicht abgefragt worden, so berechnen Sie ihn und legen ihn natürlich auch in Ihrem Zwischenspeicher ab.
- Das dynamische Array wird in der Funktion f als statische Pointer-Variable gehalten.
- Wenn Sie einen neuen Wert n! berechnen müssen, fangen Sie nicht bei 1 an sondern suchen sich den letzten berechneten Wert und multiplizieren ab dort.
- Fangen Sie mit einer Größe von 5 für das dynamische Array an und erweitern Ihr Feld bei Bedarf mit `realloc`.

Übungsaufgaben

Aufgabe 043 ♙ [Timing]

Laufzeitmessungen:

- Programmieren Sie eine Primfaktorzerlegung und messen Sie die Laufzeit.
- Vergleichen Sie die Laufzeit einer iterativen Berechnung von Fibonaccizahlen mit einer rekursiven und einer Variante mit statischem Zwischenspeicher.

Aufgabe 044 ♕ [Compile, Link, Library]

Implementieren Sie Ihr Testprogramm (main) und Ihre Methode zur Primfaktorzerlegung aus Aufgabe 043 in separaten Dateien. Compilieren bzw. Linken Sie diese Methode

- als weiteres Objekt-File
- als statische Library
- als dynamische Library

zu Ihrem Testprogramm. Generieren Sie auch ein entsprechendes makefile.

Aufgabe 045 ♙ [Mathematik]

- Schreiben Sie ein Programm, welches die angegebene Berechnungsvorschrift ausführt, bis sich der neue Wert nicht mehr "signifikant" vom vorhergehenden Wert unterscheidet (z.B. Abstand 1e-12). Die Vorschrift lautet:
  $x_{n+1}=1/2 \cos(x_n)$ Startwert $x_0 = 1.0$
- Wie lautet der Wert und wie viele Iterationen werden benötigt?
- Schreiben Sie repräsentative Testfälle.

Aufgabe 046 ♙ [Bit-Ops]

Bit-Operationen:

- Setzen Sie in einer int-Variablen das erste (2^0) und das fünfte Bit (2^4).
- Löschen Sie in einer int-Variablen mit dem Wert 255 die ersten 4 Bits (2^0..2^3).
- Verknüpfen Sie zwei int-Variablen mit xor.

Aufgabe 047 ♙ [Bit-Ops]

Simulieren Sie einen Farbwert in einem Byte mit jeweils 2 Bit für RGB und einem alpha-Kanal.

- Geben Sie diesem Byte einen eigenen Type `color8_t`.
- Schreiben Sie jeweils Funktionen zum Setzen und Auslesen eines Kanals (setR(color8_t col,int value), getR(color8_t col), etc.).
- Definieren Sie ein enum mit gängigen Farbwerten bitweise.

Übungsaufgaben

Aufgabe 101 [cout, cin, vector, push_back, size]

Speichern Sie eingegebene ganze Zahlen:

- Lesen Sie in Ihrem C++-Programm ganze Zahlen via `cin` ein.
- Geben Sie die Eingabe zur Sicherheit direkt noch einmal über `cout` aus.
- Speichern Sie jede Zahl in einem Vektor der Klasse `vector<int>` mittels `push_back` (`#include <vector>`).
- Wenn -1 eingegeben wurde, beenden Sie die Eingabe und geben alle im Vektor gespeicherten Zahlen aus (size gibt die Größe, [] den Zugriff auf die Elemente).

Aufgabe 102 [Referenzen]

Schreiben Sie eine Funktion, die einen String und ein eine ganze Zahl als Referenz übergeben bekommt.

- Geben Sie diesen in der Funktion einen beliebigen Wert.
- Schreiben Sie repräsentative Testfälle.

Aufgabe 103 [class, constructor, operator<<]

Schreiben Sie eine Klasse `Kontakt`, die einen Namen und ein Alter enthält:

- Implementieren Sie einen Default-Constructor.
- Implementieren Sie einen Constructor, der einen Namen und ein Alter zur Initialisierung übergeben bekommt. Benutzen Sie die Member-Initialisierung mit ':'.
- Implementieren Sie einen `friend operator<<` zur Ausgabe.
- Implementieren Sie Getter- und Setter-Methoden für den Namen und das Alter.
- Schreiben Sie repräsentative Testfälle.

Aufgabe 104 [class, constructor, copy constructor, operator=, operator<<]

Schreiben Sie eine Klasse `Punkt` mit zwei ganzzahligen Koordinaten (X, Y):

- Implementieren Sie einen Default-Constructor.
- Implementieren Sie einen Copy-Constructor.
- Implementieren Sie einen Constructor, der zwei Werte (X, Y) zur Initialisierung übergeben bekommt. Benutzen Sie die Member-Init. mit ':'.
- Implementieren Sie einen Constructor, der einen struct zur Initialisierung übergeben bekommt.Benutzen Sie die Member-Init. mit ':'.
- Implementieren Sie einen Destructor (ohne Inhalt).
- Implementieren Sie einen eigenen `operator=`.
- Implementieren Sie einen `friend operator<<` zur Ausgabe.
- Implementieren Sie Getter- und Setter-Methoden für X und Y.
- Implementieren Sie eine Member-Funktion, die die Max-Norm berechnet.
- Schreiben Sie repräsentative Testfälle.

Übungsaufgaben

Aufgabe 105 ♙ [class]

Schreiben Sie eine Klasse, die die Pixel-Struktur aus A034 kapselt, initialisiert sowie geeignete Getter- und Setter-Methoden für RGB und Alpha-Werte bereit stellt.

Aufgabe 106 ♙ [class]

Erweitern Sie die Klasse Punkt aus A104:

- Reservieren Sie für die X- und Y-Koordinate nicht zwei Integer, sondern ein dynamisch angelegtes Integer-Feld der Länge zwei, in dem Sie die X- und Y-Werte ablegen.
- Ändern Sie alle betroffenen Funktionen so ab, dass keine Speicherlecks erzeugt werden.

Aufgabe 107 ♖ [class]

Schreiben Sie eine Klasse, die die Polynom-Struktur aus A035 kapselt und ändern Sie alle betroffenen Funktionen so ab, dass keine Speicherlecks erzeugt werden.

Aufgabe 108 ♙ [iomanip, exceptions]

Formatieren Sie eine Ausgabe:

- Lesen Sie einen string (Beispiel "-> ") und zwei Zahlen (Beispiel 5, 42) ein.
- Geben Sie zeilenweise die folgende Tabelle aus:

```
->   1 * 5 =  5, 42/ 5 = 8.40
->   2 * 5 = 10, 42/10 = 4.20
…
-> 10 * 5 = 50, 42/50 = 0.84
```

  Achten Sie darauf, dass alle Zahlen wie gezeigt formatiert erscheinen.
- Ist eine der Zahlen 0, werfen Sie ein Exception Ihrer Wahl, fangen sie und geben im catch-Block einen Fehlertext aus.

Aufgabe 109 ♙ [fstream]

Programmieren Sie A030 mit file streams und den entsprechenden Member-Funktionen der Klassen.

Aufgabe 110 ♙ [vector, set]

Merken Sie sich eine Menge von Zahlen:

- Lesen Sie den Benutzer nacheinander Zahlen eingeben und fügen Sie diese einer Menge hinzu, wenn sie noch nicht eingegeben wurden.
- Lassen Sie den Benutzer wissen, ob diese Zahl schon in der Menge enthalten war.
- Modellieren Sie die Menge einmal mit `vector` und einmal mit `set`.

Übungsaufgaben

Aufgabe 111 [class, virtual, vector, set]

Schreiben Sie eine Kontakt-Verwaltung:

- Schreiben Sie eine Klasse Kontakt, die eine Anschrift (string) und eine Menge von EMail-Adr. enthält.
- Bereiten Sie eine virtuelle Funktion MonatsReport vor.
- Leiten Sie von Kontakt eine Klasse Kunde ab, die einen Kontostand (int) enthält.
- Leiten Sie von Kontakt eine Klasse Lieferant ab, die eine Bankverbindung (string) enthält.
- Implementieren Sie jeweils für Kunde und Lieferant die Funktion MonatsReport (mit einer einfachen Ausgabe).
- Erstellen Sie einen vector mit Zeigern auf Kontakte und füllen diesen Vektor mit einigen (zwei) fiktiven Kunden und Lieferanten.
- Rufen Sie nun die Funktion MonatsReport für alle Ihre Kontakte in dem Vektor auf.

Aufgabe 112 [abstract class, virtual]

Ändern Sie A111 so ab, dass Kontakt ein "Interface" IReport mit der Funktion MonatsReport erbt und die Klassen Kunde und Lieferant dieses implementieren.

Aufgabe 113 [virtual]

Erklären Sie Ihrem Nachbarn/Ihrer Nachbarin, worin sich ein Objekt einer Klasse mit virtuellen Funktionen von dem einer Klasse ohne unterscheidet.

Aufgabe 114 [virtual]

Schreiben Sie eine Fahrzeug-Verwaltung:

- Schreiben Sie eine Klasse BefoerderungsMittel, die eine Anzahl Sitzplätze enthält.
- Schreiben Sie eine Klasse Fahrzeug, die eine Eigenschaft TopSpeed enthält.
- Leiten Sie eine Klasse Auto von BefoerderungsMittel und von Fahrzeug ab, die eine Anzahl Raeder enthält.
- Leiten Sie eine Klasse Boot von BefoerderungsMittel und von Fahrzeug ab, die eine Anzahl Schiffsschrauben enthält.
- Leiten Sie eine Klasse Amphibie von Auto und von Boot ab, die jedoch nur eine Anzahl Sitzplaetze (gemeinsame Basisklasse BefoerderungsMittel) aber zwei TopSpeeds (zwei Basisklassen Fahrzeug) enthält.
- Legen Sie Amphibien-Objekte an und testen Sie Ihren Code.

Übungsaufgaben

Aufgabe 115 ♙ [operator]

Erweitern Sie die Punktklasse aus A104 um

- Operatoren += -= + - und skalare Multiplikation.
- Schreiben Sie repräsentative Testfälle.

Aufgabe 116 ♙ [operator]

Schreiben Sie eine Klasse Bruch mit ganzzahligem Nenner und Zähler. Implementieren Sie weiterhin

- Operatoren + - * /
- repräsentative Testfälle.

Aufgabe 117 ♖ [operator]

Schreiben Sie eine Klasse Complex mit

- sinnvollen Operatoren Ihrer Wahl
- repräsentativen Testfällen.

Aufgabe 118 ♖ [operator]

Erweitern Sie die Polynomklasse aus A107 um

- sinnvolle Operatoren Ihrer Wahl
- repräsentative Testfälle.

Aufgabe 119 ♙ [initializer-list]

Schreiben Sie eine Klasse (FIFO)Queue für ganze Zahlen. Entwerfen und implementieren Sie:

- Einen Konstruktor, den Sie mit einer initializer-list aufrufen können, etwa so: Queue Q = { 5, 6, 7 }
- Funktionen zum Hinzufügen und zum Entfernen von Einträgen.
- Eine Ausgabe, die alle Elemente der Queue ausgibt.
- Tests für Ihren Code.
- Vergleichen Sie Ihre Klasse mit dem Container std::queue.

Aufgabe 120 ♙ [iterator]

Ergänzen Sie Ihre Queue-Klasse aus A119 um einen Iterator, so dass Sie mit einer Schleife derart

```
for(auto q:Q)
    cout<<q<<endl;
```

Ihre Queue Q durchlaufen können.

Übungsaufgaben

Aufgabe 121 ♙ [rvalue, lvalue]

Erklären Sie Ihrem Nachbarn/Ihrer Nachbarin, was rvalues und was lvalues sind.

Aufgabe 122 ♙ [operator, move]

Erweitern Sie die Punktklasse aus A115 um

- einen move-ctor und einen move-assignment-op

und testen Sie, wann diese aufgerufen werden.

Aufgabe 123 ♖ [operator, move]

Erweitern Sie die Bruchklasse aus A116 um

- einen move-ctor und einen move-assignment-op

und testen Sie, wann diese aufgerufen werden.

Aufgabe 124 ♖ [operator, move]

Erweitern Sie die Complexklasse aus A117 um

- einen move-ctor und einen move-assignment-op

und testen Sie, wann diese aufgerufen werden.

Aufgabe 125 ♙ [explicit, default, delete]

Entwerfen oder erweitern Sie eine Klasse Ihrer Wahl (z.B. Punkt oder Polynom):

- Implementieren Sie nur einen Constructor mit Funktionalität und rufen Sie diesen aus den anderen Constructoren auf.
- Verwenden Sie mind. einen expliziten Constructor und vergewissern Sie sich, welche Definitionen dadurch unmöglich werden.
- Benutzen Sie die Schlüsselworte default und delete sinnvoll.

Aufgabe 126 ♙ [lambda expression]

Schreiben Sie jeweils einen Lambda-Ausdruck, der

- drei double-Werte addiert und zurückgibt.
- testet, ob ein int-Wert in einem Intervall [a,b] liegt (dabei sind a und b lokale Variablen).
- keine Argumente hat, aber eine lokale int-Variable z auf -z setzt.

Aufgabe 127 ♙ [lambda expression]

Schreiben Sie eine Funtion eval, die Sie mit einem Lambda-Ausdruck f und einem double-Wert x aufrufen können, und die f(x) zurückgibt.

Übungsaufgaben

Aufgabe 128 ♕ [lambda expression]

Überlegen Sie sich, wie Sie ein Newtonverfahren (siehe Wikipedia) als

- Funktion implementieren würden, die Sie mit Lambda-Ausdrücken für f und f' aufrufen können.
- selber als Lambda-Ausdruck definieren, den Sie ebenfalls mit Lambda-Ausdrücken für f und f' aufrufen können.

Schreiben Sie Testfälle.

Aufgabe 129 ♙ [threads]

Starten Sie Threads, von denen der erste 1 Sek., der zweite 2 Sek. und der dritte 3 Sek. schlafen. Der Hauptthread wartet auf alle drei.

Aufgabe 130 ♙ [threads]

Summieren Sie die Zahlen 1..N parallel und thread-safe in einer globalen Summenvariablen auf.

Aufgabe 131 ♖ [threads]

Schreiben Sie eine parallele, sichere Variante der summierten (bzw. zusammengesetzten) Sehnentrapezformel (siehe Code-Snippet bzw. Wikipedia→Trapezregel) bei vorgegebener Anzahl Teilintervalle.

Aufgabe 132 ♕ [threads]

Implementieren Sie eine parallele Variante von Mergesort für int-STL-Vektoren. Starten Sie mit einer seriellen Version.

Aufgabe 133 ♕ [threads, mutex, condition_variables]

Starten Sie einen Thread, der auf ein Signal des main-Threads wartet und dieses nach 2 sec vom ihm bekommt.

Aufgabe 134 ♕ [threads, mutex, condition_variables]

Starten Sie drei Threads zu Beginn. Diese warten jeweils auf ein Signal ihres jeweiligen Vorgängers. Das bedeutet, der erste Thread wartet auf ein Signal vom main-Thread, welches dieser 1 sec nach Start des ersten Threads sendet. Der zweite Thread wartet auf ein Signal vom ersten Thread, welches dieser nach Erhalt seines Signals auch nach 1 sec sendet. Der dritte Thread wartet wiederum auf ein Signal des zweiten Threads, welches dieser ebenfalls 1 sec nach dem Erhalt seines Signals sendet.

Aufgabe 135 ♕ [threads, mutex, condition_variables]

Denken Sie sich eine Übungsaufgabe aus, in der Threads gemeinsam mittels notify_all benachrichtigt werden und implementieren Sie sie.

Übungsaufgaben

Aufgabe 136 ♙ [templates]

Schreiben Sie die Punktklasse aus A115 und A122 zu einer Template-Klasse um, so dass sie prinzipiell mit unterschiedlichen Datentypen funktioniert.

Aufgabe 137 ♙ [templates]

Schreiben Sie die Bruchklasse aus A116 und A123 zu einer Template-Klasse um, so dass sie prinzipiell mit unterschiedlichen Datentypen funktioniert.

Aufgabe 138 ♖ [templates]

Schreiben Sie die Complexklasse aus A117 und A124 zu einer Template-Klasse um, so dass sie prinzipiell mit unterschiedlichen Datentypen funktioniert.

Aufgabe 139 ♖ [templates]

Schreiben Sie die Polynomklasse aus A118 und A107 zu einer Template-Klasse um, so dass sie prinzipiell mit unterschiedlichen Datentypen funktioniert.

Aufgabe 140 ♖ [templates]

Schreiben Sie die Queue aus A119 und A120 zu einer Template-Klasse um, so dass sie prinzipiell mit unterschiedlichen Datentypen funktioniert.

Aufgabe 141 ♕ [templates]

Schreiben Sie eine Template-Klasse, die zur Compile-Zeit B hoch N für zwei natürliche Zahlen B und N berechnet.

Aufgabe 142 ♕ [user literals]

Schreiben Sie eine Klasse für Gewichte, so dass Sie mit Gewichts-Objekten arbeiten (addieren, skalar multiplizieren) können und diese mittels user literals mit Einheiten in Gramm (g), in Kilogramm (kg) oder Milligramm (mg) angeben können. Geben Sie ein solches Objekt in einer sinnvollen Einheit (Skalierung) aus.

Aufgabe 143 ♙ [auto-ptr, unique-ptr]

Entwerfen Sie eine beliebige Klasse und füllen (jeweils) einen std::vector mit

- Objekten Ihrer Klasse,
- mit Zeigern auf dynamisch angelegte Objekte Ihrer Klasse, bzw.
- unique- und auto-ptr auf diese.

Werden alle Elemente ordnungsgemäß zerstört oder gibt es Memory-Leaks?

Können Sie alle Elemente in einer Schleife ausgeben?

Aufgabe 144 ♕ [my-smart-ptr]

Entwerfen Sie eine eigene smart-ptr Klasse, die sich wie auto- bzw. unique-ptr verhält.

Übungsaufgaben

Aufgabe 999 ♕ [chocolate]

Formulieren Sie eigene, sinnvolle und machbare Aufgaben jeweils zu einem Thema Ihrer Wahl aus den Bereichen C und C++.

Implementieren und kommentieren Sie jeweils eine Lösung und beschreiben Sie kurz, welche besonderen Erkenntnisse oder Schwierigkeiten zu erwarten sind.

Schicken Sie die Aufgaben inkl. Lösung bis zum Ende des Semesters an a.voss@fh-aachen.de

Die besten/originellsten/schönsten drei Aufgaben werden in der letzten Stunde mit einem kleinen Präsent einer hier ansässigen Schokoladenmanufaktur ausgezeichnet.