FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Einführung in C++
Übungen

Prof. Dr. rer. nat. Alexander Voß

FH Aachen
University of applied Sciences

06.10.2020

Allgemeine Informationen

Achtung

In jedem Kapitel gibt es repräsentative Übungsaufgaben und auch Beispiellösungen.

Machen Sie nicht den Fehler und gucken sich nur die Lösungen an, nur um dann festzustellen, dass das ja einfach erscheint.

Es ist nicht das unbedingte Ziel, alle Aufgaben zu bearbeiten, sondern sich lieber mit den jeweils unbekannten oder unsicheren Gebieten zu beschäftigen.

Schließen Sie die Aufgaben zu Hause ab und bringen Sie offene Fragen zur nächsten Veranstaltung bzw. zum Chat mit.

0x01 – Übungen

’West Rose Cliff’

- Lesen Sie eine ganze Zahl ’n’ vom Typ ’int’ und ein Zeichen ’c’ vom Typ ’char’ ein.
- Testen Sie, ob die Zahl ’n’ größer als 0 und ob das Zeichen ’c’ ein Großbuchstabe ist.
- Definieren Sie bool’sche Variablen mit den Testergebnissen und geben diese aus.

Erweiterung:

- Nutzen Sie einmal, wenn möglich, den ’?’-Operator anstelle einer if-Abfrage.

0x01 – Übungen

’Blue Canyon’

Berechnung von b^n:

- Lesen Sie Variablen ’b’ und ’n’ eines geeigneten Typs ein. Berechnen Sie ’b hoch n’ in einer Schleife und geben Sie das Ergebnis aus.

Erweiterung:

- Schreiben Sie eine Funktion ’pot’, die ’b’ und ’n’ übergeben bekommt und das Ergebnis zurückgibt.
- Definieren Sie die Funktion ’pot’ erst hinter main und geben Sie vor ’main’ nur die Signatur an.
- Formulieren Sie die Schleife einmal als ’for’ und einmal als ’while’-Schleife.

0x01 – Übungen

’Beverly Hollow’

- Berechnung der Fibonacci-Folge (f_i):
- Lesen Sie eine ganze Zahl ’n’ von der Konsole ein.
- Berechnen Sie die ersten ’n’ Fibonaccizahlen f_i iterativ, d.h. in einer Schleife (also nicht rekursiv) und geben Sie sie aus.
  Die Vorschrift lautet: $f_i = f_{i-1} + f_{i-2}$ mit $f_0 = f_1 = 1$.

Erweiterung:

- Berechnen Sie die Zahlen f_i jeweils rekursiv.
- Überlegen Sie: Welcher Ansatz ist (vermutlich) schneller und warum?

0x01 – Übungen

’Sanborn Cliff’

- Legen Sie ein Array der Länge 3 vom Typ ’double’ an und initialisieren Sie die ersten beiden Elemente mit den Werten 1.0 und 2.0.
- Speichern Sie die Summe in dem dritten Element.
- Geben Sie alle Elemente aus.
- Legen Sie ein weiteres Array der gleichen Größe an und kopieren Sie das erste in das zweite Array.

Erweiterung:

- Übergeben Sie das Feld einer Funktion zur Ausgabe. Beachten Sie, dass Sie die Länge mit übergeben müssen.

0x01 – Übungen

’Pine Point’

- Primzahlberechnung:
- Lesen Sie eine ganze Zahl ’n’ von der Konsole ein.
- Bestimmen Sie in einer Schleife über alle Zahlen 2 bis ’n’, ob die jeweilige Zahl eine Primzahl ist oder nicht. Geben Sie die jeweilige Zahl aus, wenn sie eine Primzahl ist.

Erweiterung:

- Schreiben Sie eine Funktion ’isPrim’, die eine Zahl übergeben bekommt und ’true’ oder ’false’ zurück gibt.
- Definieren Sie die Funktion ’isPrim’ nach ’main’ und geben Sie vor ’main’ nur die Signatur an.

0x01 – Übungen

Hausübung

'work_int8', 'work_string', 'work_switch', 'work_for'.

Selbstkontrolle

- Ich habe alle Codes und Übungsthemen verstanden. Ich kann ein C++-Programm erstellen und ausführen.
- Ich kenne unterschiedliche Datentypen und kann sie definieren, deklarieren, einlesen, verarbeiten und ausgeben.
- Ich bin mit den Kontrollstrukturen 'if-else', 'do-while', 'while' und 'for' vertraut.
- Ich weiß, wie man eine Funktion aufruft, sie definiert oder deklariert.
- Ich kenne globale und lokale Variablen. Ich kenne den Namensraum 'std'.

0x02 – Übungen

’Baker River’

- Schreiben Sie eine Funktion ’cut’, die bei zwei per Referenz übergebenen ’double’-Zahlen jeweils die Nachkommastellen auf 0 setzt. Nutzen Sie die Funktion ’floor’ aus ’cmath’.
- Schreiben Sie eine Funktion ’shift’, die drei per Referenz übergebene Strings ’der Reihe nach’ tauscht, also der erste String bekommt den Inhalt des zweiten, der zweite den Inhalt des dritten und der dritte den des ersten.

0x02 – Übungen

’Elkford’

Beispielstruktur ’polynom’:

- Definieren Sie eine globale Variable ’dim’ mit Wert 3.
- Definieren Sie eine Struktur ’polynom’, die ein Feld der Größe ’dim’ von Koeffizienten ’coeffs’ enthält.
- Programmieren Sie eine globale Funktionen ’eval’, um ein Polynom ’p’ an einer Stelle ’x’ auszuwerten. Übergeben Sie das Polynom (’call-by-ref’) und die Stelle der Funktion.
- Schreiben Sie aussagekräftigen Testcode.

Erweiterung:

- Implementieren Sie einen operator ’≪’ zur Ausgabe (Nachlesen).

0x02 – Übungen

’Harshire’

Beispielstruktur ’stack’:

- Informieren Sie sich, wie ein ’Stack’ funkt. (Wikipedia).
- Definieren Sie eine Struktur ’stack’, die ein Feld fester Länge (z.B. 3) von ’int’ enthält sowie eine Variable ’next’ für die nächste freie Position im Feld.
- Implementieren Sie Funktionen ’pop’ und ’push’, die Daten vom Stack holen bzw. dort ablegen (wenn Platz ist).
- Schreiben Sie aussagekräftigen Testcode.

Erweiterung:

- Werfen Sie eine eigene Exception, wenn der Stack voll ist.

0x02 – Übungen

Hausübung

'work_stoi_stod'. Nur ansehen.

Selbstkontrolle

- Ich habe alle Codes und Übungsthemen verstanden.
- Ich weiß, was eine Referenz ist.
- Ich kenne den Unterschied zwischen 'call-by-value' und 'call-by-ref '.
- Ich kann Ausnahmen werfen und fangen.
- Ich kann 'auto' anwenden.

0x03 – Übungen

’Ravencastle’

Eine Klasse ’bruch’ (’fraction’) entwerfen.

- Implementieren Sie einen default-ctor und einen dtor.
- Implementieren Sie einen ctor, der Zähler und Nenner übergeben bekommt. Nutzen Sie ’:’ zur Initialisierung.
- Implementieren Sie ’getter’ und ’setter’ für (ganzzahlige) Zähler bzw. Nenner.
- Schreiben Sie aussagekräftigen Testcode.

Erweiterung:

- Implementieren Sie einen copy-ctor.
- Implementieren Sie einen operator ’«’ zur Ausgabe.
- Verwenden Sie so oft wie möglich ’const’.

0x03 – Übungen

’Stone Ridge’

Eine Klasse ’punkt’ mit ’x’- und ’y’-Koordinate entwerfen.

- Implementieren Sie einen default-ctor und einen dtor.
- Implementieren Sie einen weiteren ctor, der einen x- und einen y-Wert vom Typ ’double’ übergeben bekommt. Benutzen Sie ’:’ zur Initialisierung.
- Implementieren Sie ’getter’ und ’setter’ für ’x’ und ’y’.
- Schreiben Sie aussagekräftigen Testcode.

Erweiterung:

- Implementieren Sie einen copy-ctor.
- Implementieren Sie einen operator ’«’ zur Ausgabe.
- Verwenden Sie so oft wie möglich ’const’.

0x03 – Übungen

’Lucky Rock’

Eine Klasse ’kontakt’ mit Alter und Namen entwerfen.

- Implementieren Sie einen default-ctor und einen dtor.
- Implementieren Sie einen ctor, der ein ganzzahliges Alter vom Typ ’unsigned int’ und einen Namen vom Typ ’string’ übergeben bekommt. Nutzen Sie ’:’ zur Initialisierung.
- Implementieren Sie ’getter’ und ’setter’.
- Schreiben Sie aussagekräftigen Testcode.

Erweiterung:

- Implementieren Sie einen copy-ctor.
- Implementieren Sie einen operator ’«’ zur Ausgabe.
- Verwenden Sie so oft wie möglich ’const’.

0x03 – Übungen

’Meadow River’

- Überführen Sie die Struktur ’polynom’ aus Übung ’Elkford’ in eine Klasse ’polynom’.
- Überführen Sie die Funktion ’eval’ in eine Memberfunktion.
- Implementieren Sie eine Memberfunktion ’at’, die einen Index ’i’ übergeben bekommt und den i’ten Koeffizienten zurückgibt. Werfen Sie eine Ausnahme, falls ’i’ ungültig ist.

Erweiterung:

- Implementieren Sie eine globale Funktion ’add’, um zwei Polynome zu addieren und geben Sie das Ergebnis zurück.
- Nutzen Sie Ihren Testcode aus ’Elkford’ und testen Sie zusätzlich mit der Dimension 4.

0x03 – Übungen

Hausübung

'work_class_car'. Nur ansehen.

Selbstkontrolle

- Ich habe alle Codes und Übungsthemen verstanden.
- Ich kenne den Unterschied zwischen 'struct' und 'class'.
- Ich weiß, wie man eine Klasse definiert.
- Ich kann Konstruktoren, Destruktoren und 'getter' und 'setter' definieren.
- Ich kenne die Bedeutung von 'const' im Zusammenhang mit Memberfunktionen.
- Ich weiß, wann ein Kopierkonstruktor aufgerufen und wie er verwendet wird.
- Ich kann einen eigen Operator '«' für meine Klasse schreiben.

0x04 – Übungen

’Oakberg’

Datencontainer ’range-based-for’ durchlaufen und ’auto’ verwenden.

- Definieren Sie ein ’int’-Feld ’a’, initialisiert mit den Werten 2, 3, 5, 7, sowie einen ’vector’ ’v’, initialisiert mit den selben Werten.
- Durchlaufen Sie ’a’ und ’v’ jeweils in einer ’range-based-for’-Schleife mit ’auto’ und geben Sie das jeweilige Element aus.

0x04 – Übungen

’Brickgate’

Erweitern Sie Übung ’Oakberg’.

- Nutzen Sie ’auto&’ in der ’range-based-for’-Schleife über den ’vector’ ’v’ und verdoppeln Sie den Wert jedes Elements.
- Definieren einen eigenen Typ ’it_type’ als ’const_iterator’ des Vektors und geben Sie in einer Schleife über einen const_iterator ’it’ das jeweilige Element des Iterators aus. Beachten Sie: ’*it’ ist der Wert des Elements und ’cbegin’ und ’cend’ geben einen const_iterator zurück.

0x04 – Übungen

’Hicks Bluff’

Suchen und löschen in Datencontainern mit Iteratoren.

- Suchen Sie in dem Vektor ’{1,2,3,4,5}’ das Element ’2’ und geben Sie die nächsten drei Elemente (inkl.) aus (wenn vorhanden).
- Legen Sie folgende ’unordered_map’ an
  ’{ 1→’Eins’, …, 5→’Fünf’ }’ und suchen Sie dort nach dem Schlüssel ’2’.
- Löschen Sie in obiger Map alle Elemente, deren Schlüssel größer ist als ’2’.

Erweiterung:

- Nutzen Sie ’auto’ so oft wie möglich.

0x04 – Übungen

’Ashfield’

Mapping von ISBN-Nummern zu Büchern.

- Implementieren Sie ein ’struct’ ’buch’, welches (vereinfacht) einen Autor und einen Titel enthält.
- Definieren Sie einen eigenen Typ ’katalog_t’ durch eine ’unordered_map’, die einen ’string’ (ISBN) auf ein ’buch’ abbildet. Legen Sie eine Variable ’katalog’ dieses Typs an.
- Füllen Sie ’katalog’ mit drei echten Büchern (und ISBN) Ihrer Wahl, z.B. ’44245381X’ → {’Walter Moers’,’Die 13 1/2 Leben des Käpt’n Blaubär’}, …
- Suchen Sie in einer Schleife mit ’const auto&’ alle Bücher, deren Titel länger als 30 Zeichen ist und geben Sie diese aus.

0x04 – Übungen

’Sparrow Town’

Klasse zu Template erweitern.

- Erweitern Sie die Klasse ’bruch’ bzw. ’fraction’ aus Übung ’Ravencastle’ zu einem Template.
- Testen Sie Ihre generische Klasse mit unterschiedlichen Datentypen.

0x04 – Übungen

’Bakeropolis’

Klasse zu Template erweitern.

- Erweitern Sie die Klasse ’punkt’ aus Übung ’Stone Ridge’ zu einem Template.
- Testen Sie Ihre generische Klasse mit unterschiedlichen Datentypen.

0x04 – Übungen

Hausübung

Sehen Sie sich die Member der STL-Templates ’vector’, ’set’, ’array’ und ’unordered_map’ an, hier insbesondere die Funktionen zum Einfügen (’insert’), zum Löschen (’erase’, ’clear’) und zum Ersetzen (’emplace’).

Selbstkontrolle

- Ich habe alle Codes und Übungsthemen verstanden.
- Ich kann eine Klasse zu einer generischen Klasse, einem Template, erweitern (wenn es sinnvoll ist!).
- Ich kann Elemente aus Datencontainern lesen, suchen und verändern.
- Ich kann Iteratoren verwenden.

0x05 – Zeiger/Pointer – Hintergrund Stack

Push und Pop auf einem (rückwärts wachsendem) Stack (Prinzip, 16 Bit):

Die Variable (=Register) SP (stack pointer) bezeichnet eine Position im Speicher, an der Daten auf dem Stack abgelegt (push, SP wächst rückwärts) bzw. von dort wiedergeholt (pop, SP steigt) werden. Die Variable IP (instruction pointer) bezeichnet die nächste Position im Programm, an der ein Befehl ausgeführt wird. A ist eine beliebige Variable.

Asm-Programm	Befehl	IP danach	SP danach	A danach
		1	0x2304	
1	A=0x1234	2	-	0x1234
2	push A	3	0x2302	-
3	A=0x5678	4	-	0x5678
4	push A	5	0x2300	-
5	A=0x9abc	6		0x9abc
6	pop A	7	0x2302	0x5678
7	pop A	8	0x2304	0x1234
8	...			

Speicher 0x2300-0x2307	+0x00	+0x02	+0x04	+0x06
"	0x0000	0x0000	0x0000	0x0000
"	0x0000	0x0000	0x0000	0x0000
"	0x0000	0x0000	0x1234	0x0000
	0x0000	0x0000	0x1234	0x0000
	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x5678	0x1234	0x0000

0x05 – Zeiger/Pointer – Hintergrund lokale Variablen

Lokale Variablen auf dem Stack (Prinzip 16 Bit):

Die Variable (=Register) BP bezeichnet eine Position im Speicher, zu der relativ lokale Variablen abgelegt werden. BP wird zu Beginn mit SP initialisiert und danach wächst SP (-=), damit künftige Calls die Daten nicht zerstören. Das muss am Ende wieder korrigiert werden (+=). Die Speicheradressen von n1 und n2 sind hier: 0x2304 und 0x2302.

Rückgabewerte werden z.B. in der Variablen A (=Register) zurückgegeben.

	C++ Programm	Asm-Programm
1	{	BP = SP (0x2306)
2		SP -= 0x06
3	int n1 = 0x1234	[BP-2] = n1
4	int n2 = 0x5678	[BP-4] = n2
5	return 0xaabb	A = 0xaabb
		SP += 0x06
6	}	ret

Speicher 0x2300-0x2307	+0x00	+0x02	+0x04	+0x06
"	0x0000	0x0000	0x0000	0x0000
"	0x0000	0x0000	0x0000	0x0000
"	0x0000	0x0000	0x1234	0x0000
	0x0000	0x5678	0x1234	0x0000
	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x1234	0x1234	0x0000

0x05 – Zeiger/Pointer – Hintergrund lokale Variablen

Lokale Variablen auf dem Stack (konkret, 32 bzw. 64 Bit):

C++ vs. Assembler

```
// Argument x und lokale Variablen auf dem Stack
int f(int x) {
    int n1=0x11223344, n2=0x55667788, n3=0x6699cc
    g();
    return 0x12345678;
}
```

```
0x10b759f60 <+0>:  55                    pushq  %rbp
0x10b759f61 <+1>:  48 89 e5              movq   %rsp, %rbp
0x10b759f64 <+4>:  48 83 ec 10           subq   $0x10, %rsp
```

```
        int n1=0x11223344, n2=0x55667788, n3=0x6699ccff;

0x10b759f6b <+11>: c7 45 f8 44 33 22 11  movl   $0x11223344, -0x8(%rbp)
0x10b759f72 <+18>: c7 45 f4 88 77 66 55  movl   $0x55667788, -0xc(%rbp)
0x10b759f79 <+25>: c7 45 f0 ff cc 99 66  movl   $0x6699ccff, -0x10(%rbp)
```

```
0x10b759f85 <+37>: b8 78 56 34 12        movl   $0x12345678, %eax
0x10b759f8a <+42>: 48 83 c4 10           addq   $0x10, %rsp
0x10b759f8e <+46>: 5d                    popq   %rbp
0x10b759f8f <+47>: c3                    retq
```

0x05 – Zeiger/Pointer – Hintergrund

Der Code zeigt ausserdem, dass lokale Variablen

- eine absolute Position im Speicher haben (Speicheradresse), auch wenn sie relativ angesprochen werden.
- einen zufälligen Wert besitzen (eben der an ihrer Position auf dem Stack), wenn sie nicht initialisiert werden.
- keine Typinformation im Speicher besitzen – es sind immer nur Bytes, die dem Typ entsprechend interpretiert werden.
- hintereinander liegen, wenngleich das nicht sein muss.
- illegal verändert werden können, wenn man nur die Speicheradresse einer kennt.

Zeiger bzw. Pointer sind i.d.R. Speicheradressen auf andere Variablen! Über Zeiger kann man diese lesen und auch verändern.

0x05 – Übungen

’Copper View’

Variablen über Zeiger modifizieren.

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

Erweiterung:

- Legen Sie einen Zeiger auf den Zeiger auf ’int’ an, initialisieren Sie ihn mit der Adresse Ihres ’int’-Zeigers und geben Sie den Wert der ’int’-Variablen hierüber aus.

0x05 – Übungen

’Kengate’

Swap.

- Schreiben Sie eine Funktion ’swap’, die zwei übergebene ’double’-Zahlen tauscht. Implementieren Sie die Funktionen einmal über Zeiger und einmal über Referenzen.

Erweiterung:

- Schreiben Sie eine weitere Funktion ’swap_ptr’, die zwei übergebene ’double’-Zeiger tauscht (Zeiger und Referenzen).
- Testen Sie Ihren Code entsprechend.

0x05 – Übungen

’McAllen Spring’

Illegal – beachten Sie auch das Snippet zum Thema…

- Versuchen Sie, durch Verwendung von Zeigern einzelne Bytes von lokalen ’int’-Variablen zu manipulieren. Nutzen Sie dafür casts in ’char*’.
- Verändern Sie einzelne Bytes eines Texts der Form: char* p=”huhu”;
  Was passiert?

Erweiterung:

- Versuchen Sie, über Zeiger-Manipulationen lokale Variablen gezielt auf dem Stack zu verändern, ohne diese direkt zu adressieren.
  Tipp: Ermitteln Sie zunächst die Adresse einer lokalen Variablen. Die anderen liegen ’in der Nähe’.

0x05 – Übungen

Hausübung

- Überlegen Sie sich, wofür Sie Zeiger verwenden könnten? Gibt es Anwendungsfälle, in denen Referenzen nicht möglich sind, aber Zeiger durchaus?

Selbstkontrolle

- Ich habe alle Codes und Übungsthemen verstanden.
- Ich weiß, was ein Zeiger ist.
- Ich kann Zeiger initialisieren, verwenden und mit ihnen rechnen.
- Ich kann sowohl Zeiger als auch Referenzen für ’call-by-reference’ verwenden.
- Ich kenne spezielle Ausdrücke wie ’nullptr’ oder ’void*’.
- Ich weiß, was Felder und Indexzugriffe mit Zeigern zu tun haben.