Programmieren C++

Aufgaben

Markus Jetzer
Starenstrasse 5
4142 Münchenstein

markus.jetzer@freesurf.ch

Inhaltsverzeichnis

1 Aufgabe 1

1.1 Einleitung

Dieses Skript soll Ihnen beim Erlernen der Programmiersprache C++ helfen. Im wesentlichen werden hier praktische Übungen vorgestellt, die Sie zu Hause als Aufgaben oder im Unterricht in der Gruppe lösen können. Die Übungen sind analog zum Unterricht aufgebaut. Nach jeder Lektion werde ich Ihnen mitteilen, welche Aufgaben Sie laut dem Unterrichtsstand lösen können.

1.2 Softwareinstallation

Zuerst sollten Sie Ihre Entwicklungsumgebung in Betrieb nehmen. Im Unterricht werden wir vorerst mit der Programmierumgebung (Visual C++) von Microsoft arbeiten. Es gibt sehr viele leistungsfähige Kompiler und Entwicklungsumgebungen für C++ Programme. Unter anderem kann mit einer Linuxumgebung ein opensource Entwicklungstool mit dem ich persönlich sehr gute Erfahrungen gemacht habe eingesetzt werden. Dieses Tool nennt sich KDevelop und wurde im Umfang der KDE Oberfläche entwickelt. Diese Software ist in den Standarddistributionen (SuSe, Redhat, Mandrake) enthalten.

Ihre Aufgabe: Installieren sie die Programmieroberfläche Ihrer Wahl auf Ihrem PC. Eine Anleitung zur Verwendung der Visual C++ Oberfläche können Sie denn erhaltenen Unterlagen entnehmen.

Das Ziel: Eine lauffähige Entwicklungsoberfläche einzurichten.

1.3 Hello World!

Die meisten Programmierkurse beginnen mit dem "Hello World!" Programm. Deshalb werden wir uns auch an dieses ungeschriebene Gesetz halten.

Ihre Aufgabe: Schreiben Sie ein C++ Programm, welches auf der Konsole den Text "Hello World!" ausgibt.

Das Ziel: Die Ausgabe von Text auf der Konsole kennen lernen.

1.4 Dokumentation

Normalerweise wird geschriebener Kode fortlaufend ausdokumentiert. Dadurch wird eine bessere Übersicht im Programmkode erreicht.

Ihre Aufgabe: Dokumentieren Sie den Kode der Aufgabe 1.3 aus. Schreiben Sie die Bedeutung jeder Kode-Zeile direkt in die Kode-Datei.

Das Ziel: Die Dokumentation von Kode-Zeilen kennen lernen.

2 Aufgabe 2

2.1 Addition

Ihre Aufgabe: Schreiben Sie ein Programm, welches zwei Zahlen (Variablen) addiert und das Resultat auf der Konsole ausgibt.

Bedingung: Das Programm soll drei Variablen vom Typ *double* deklarieren und initialisieren. In der dritten Variable wird das Resultat gespeichert. Der Wert dieser Varialble muss auf der Konsole ausgegeben werden.

Zusatz: Erweitern Sie das Programm mit einer Multiplikation, Subtraktion und Division.

Das Ziel: Sich mit Variablen vertraut machen.

2.2 Eingabe von Daten

Ihre Aufgabe: Schreiben Sie ein Programm, welches eine Zahl vom Typ *double* einliest. Diese Zahl soll in eine andere Variable vom gleichen Typ gespeichert und ausgegeben werden.

Bedingung: Das Programm soll zwei Variablen enthalten.

Zusatz: Speichern Sie den *double* Wert in einer *integer* Variable und geben Sie die Varaible auf der Konsole aus. Verwenden Sie dazu das "Casting-Prinzip" welches im Skript beschrieben wird.

Das Ziel: Dateneingabe, Casting und Datenausgabe kennen lernen.

2.3 Zylinder

Ihre Aufgabe: Schreiben Sie ein Programm, welches das Inhaltsvolumen eines Zylinders berechnet und auf dem Bildschirm ausgibt.
Eingabedaten:

- Radius, Hoehe

Berechnung:

- $V = r^2 \cdot \pi \cdot h$

Bedingung: π (Pi) muss als Konstante deklariert werden.

Zusatz: Das Programm soll die Daten eines zweiten Zylinders einlesen und das Durchschnittsvolumen der beiden Zylinder berechnen und auf dem Bildschirm ausgeben.

Das Ziel: Die Anwendung von Formeln kennen lernen.

3 Aufgabe 3

3.1 Drei Additionen

Ihre Aufgabe: Schreiben Sie ein Pogramm, welches die Variable x mit dem Wert 1 addiert.

Bedingung: Verwenden Sie für die Addition drei verschiedene Operatoren.

Das Ziel: Unterschiedliche Operatoren kennen lernen.

3.2 Inkrementierung

Ihre Aufgabe: Welchen Wert gibt folgendes Programm aus?

```
1 a=12;
2 a+=++a+a++;
3 a=a+a;
4 cout<<a<<endl;
```

Bedingung: Überlegen Sie sich, welchen Wert die Variable a am Schluss enthält. Wenn Sie sich sicher sind, implementieren Sie das Programm und vergleichen Sie die Resultate.

Das Ziel: Inkrementieren mit der Post- und Präfixversion kennen lernen.

3.3 sizeof

Ihre Aufgabe: Schreiben Sie ein Programm, welches das Speichervolumen folgender Datentypen in Byte ausgibt:

- int
- long int
- float
- double
- long double
- char

Verwenden Sie den Datentypnamen für drei Ausgaben Ihrer Wahl. Erzeugen Sie für die restlichen Punkte je eine Variable, welche Sie für die Ausgabe des Wertes einsetzen.

Das Ziel: Datentypen und die Funktion *sizeof()* kennen lernen.

4 Aufgabe 4

4.1 Freier Fall

Beim freien Fall befindet sich ein Körper zunächst in Ruhe und bewegt sich unter dem Einfluss der Erdanziehungskraft aus einer bestimmten Höhe h_0 nach unten. Die Dauer des Falls kann mit der folgenden Formel berechnet werden:

$$T = \sqrt{\frac{2h_0}{g}}$$

g ist die Erdbeschleunigungskonstante. Ihr Wert beträgt:
$g = 9.807$

Ihre Aufgabe: Schreiben Sie ein Programm, welches die Höhe h_0 einliest und den Wert der Dauer des Falls T ausgibt.

Bedingung: Beachten Sie die richtige Verwendung der Datentypen und Konstanten. Binden Sie im Kopfteil folgende Anweisung ein:

```
1 #include<cmath>
```

Die Funktion zur Berechnung der Wurzel lautet:
sqrt()

Das Ziel: Umgang mit mathematischen Formeln kennen lernen.

4.2 if - else

Ihre Aufgabe: Schreiben Sie ein Programm, welches eine Zahl einliest und übeprüft. Falls die Zahl gerade ist, soll der Text "Die Zahl x ist gerade" sonst "Die Zahl x ist ungerade" ausgegeben werden. Falls die Zahl ungerade ist, müssen Sie zusätzlich überprüfen ob die Zahl kleiner oder grösser als 0 ist.

Bildschirmausgabe:
Geben sie eine beliebige Zahl ein: *5* Die Zahl 5 ist ungerade! Die Zahl 5 ist grösser als 0!

Bedingung: Verwenden Sie eine Zahl vom Typ Integer.

Zusatz: Schreiben Sie das gleiche Programm mit einer *switch* Anweisung.

Das Ziel: Abfragen kennen lernen.

5 Aufgabe 5

5.1 Struktogramme

Ihre Aufgabe: Zeichnen Sie das Struktogramm der Aufgabe 4.2. Sie können es von Hand oder mit Hilfe einer Software zeichnen.

Bedingung: Verwenden Sie den Nassi-Schneidermann Standard.

Das Ziel: Ein Programm grafisch darstellen zu können.

5.2 Schleifen

Ihre Aufgabe: Erstellen Sie ein Programm zur Berechnung von Potenzen mit natürlichen Exponenten. Dabei muss die Basis und der Exponent von der Tastatur eingelesen werden.

Eingabedaten:

- Basis = 3.45
- Exponent = 6

Bildschirmausgabe:
Basis = 3.45 Exponent = 6 3.45 ^6 = 1686.221

Zeichnen Sie zuerst ein Struktogramm des Programms.

Bedingung:

- Verwenden Sie eine for-Schleife

Zusatz:

- Schreiben Sie das gleiche Programm mit einer *while* Schleife.
- Schreiben Sie das gleiche Programm mit einer *do/while* Schleife.

Das Ziel: Die Schleifen kennen lernen.

6 Aufgabe 6

6.1 Gravitationsgesetz von Newton

Ihre Aufgabe: Zwischen den zwei Massen *m1* und *m2* wirkt im gegenseitigen Abstand r stets eine Anziehungskraft. (Gravitationskraft) Diese kann mit der folgenden Formel berechnet werden:

$$F_G = f \cdot \frac{m1 \cdot m2}{r^2}$$

f ist die Gravitationskonstante. Ihr Wert beträgt:

$f = 6.673 \cdot 10^{-11}$

Das Programm soll *m1*, *m2* und r einlesen und den Wert der Anziehungskraft ausgeben.

Zeichnen Sie zuerst ein Struktogramm des Programms.

Bedingung: f muss als Konstante deklariert werden.

Das Ziel: Umgang mit mathematischen Formeln kennen lernen.

6.2 Zahlenfolge

Ihre Aufgabe: Ein Programm soll eine Zahlenfolge auf dem Bildschirm ausgeben.

Eingabedaten:

- Startwert = 2.5
- Endwert = 20
- Schrittweite = 2.5

Bildschirmausgabe:
Startwert = 2.5 Endwert = 20 Schrittweite 2.5 2.5 5.0 7.5 10.0 12.5 15.0 17.5 20.0

Zeichnen Sie zuerst ein Struktogramm des Programms.

Bedingung: Schreiben Sie das Programm mit einer *while* Schleife.

Zusatz: Schreiben Sie das gleiche Programm mit einer *for* Schleife.

Das Ziel: Schleifen kennen lernen.

7 Aufgabe 7

7.1 Muster

Ihre Aufgabe: Schreiben Sie ein Programm, welches folgende Ausgabe erzeugt:

Bildschirmausgabe:
+ ++ +++ ++++

Zeichnen Sie zuerst ein Struktogramm des Programms.

Bedingung: Verwenden Sie zur Ausgabe eine oder mehrere Schleifen Ihrer Wahl.

Das Ziel: Schleifen kennen lernen.

7.2 Taschenrechner

Ihre Aufgabe: Erstellen Sie einen einfachen Taschenrechner, der jeweils 2 Zahlen und einer der vier Grundoperatoren (+, −, ∗, /) einliest und das Resultat auf dem Bildschirm ausgibt. Eingabedaten:

- 1. Operand = 12
- 2. Operand = 45
- Operator = ∗
- Abbruchbedingung

Bildschirmausgabe:
1. Operand = 12 2. Operand = 10 Operator = * Resultat = 120 Moechten Sie eine weitere Rechenoperation durchfuehren (y, n)? y

Zeichnen Sie zuerst ein Struktogramm des Programms.

Zusatz: Integrieren Sie zusätzlich die Modulofunktion (%) in das Programm.

Das Ziel: Schleifen und Abfragen kennen lernen.

8 Aufgabe 8

8.1 Kleines EinMalEins

Ihre Aufgabe: Entwerfen Sie ein Programm, welches das kleine EinMalEins auf dem Bildschirm ausgibt.

Bildschirmausgabe:									
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Zeichnen Sie zuerst ein Struktogramm des Programms.

Bedingung: Verwenden Sie zwei verschachtelte *for* Schleifen.

Das Ziel: Verschachtelte Schleifen kennen lernen.

8.2 Kleinste Zahl

Ihre Aufgabe: Schreiben Sie eine Funktion, welcher Sie drei Werte übergeben. Die Funktion soll den kleinsten Wert an die main-Funktion zurückgeben. Der Wert muss in der Konsole angezeigt werden.

Bildschirmausgabe:
Geben Sie 3 Zahlen ein:
Die 1. Zahl: 2
Die 2. Zahl: 4
Die 3. Zahl: 6
Die kleinste Zahl ist : 2

Zeichnen Sie zuerst ein Struktogramm des Programms.

Das Ziel: Funktionen kennen lernen.

9 Aufgabe 9

9.1 Vorzeichen erkennen

Schreiben Sie eine Funktion, welcher Sie einen positiven, negativen oder neutralen (0) Wert übergeben können.
Folgende Möglichkeiten müssen berücksichtigt werden:

- Wenn Sie der Funktion einen negativen Wert übergeben, muss sie den Wert -1 zurückgeben.
- Wenn Sie der Funktion einen neutralen Wert übergeben, muss sie den Wert 0 zurückgeben.
- Wenn Sie der Funktion einen positiven Wert übergeben, muss sie den Wert 1 zurückgeben.

Zeichnen Sie zuerst ein Struktogramm des Programms.

Ihre Aufgabe: Schreiben Sie ein komplettes Programm, welches eine Zahl einliest und der Funktion übergibt. Der Rückgabewert und ein kurzer Text müssen dem Benutzer sichtbar gemacht werden.

Bedingung: Die Ein- und Ausgabe der Zahlen erfolgt ausschliesslich in der main-Funktion. Alle "Logikelemente" müssen in die Funktion ausgelagert werden.

Zusatz: Schreiben Sie zwei zusätzliche Funktionen, die sich mit der Ein- und Ausgabe der Zahlen und Texte befassen. Die main-Funktion enthält dadurch nur noch die Funktionsaufrufe.

Das Ziel: Funktionen kennen lernen.

9.2 Summe

Einer Funktion namens *sum()* soll ein Wert x übergeben werden. Diese Funktion soll alle Zahlen von 0 bis und mit x addieren und das Resultat an die main-Funktion zurückliefern.

Zeichnen Sie zuerst ein Struktogramm des Programms.

Ihre Aufgabe: Schreiben Sie ein komplettes Programm, welches die *sum()* Funktion implementiert.

Bedingung: Verwenden Sie Integer Werte.

Das Ziel: Funktionen kennen lernen.

10 Aufgabe 10

10.1 Überladene Funktion

Ihre Aufgabe: Schreiben Sie eine Funktion die drei verschiedene Datentypen übernehmen kann. Folgende Variablen müssen zur Auswahl stehen:

- *char*
- *int*
- *double*

Die Funktionen soll folgende Ausgaben erzeugen:

- Für das *char* Zeichen:
  Es wurde eine *char* Variable eingegeben
- Für den *int* Wert:
  Es wurde eine *int* Variable eingegeben
- Für den *double* Wert:
  Es wurde eine *double* Variable eingegeben

Zeichnen Sie zuerst ein Struktogramm des Programms.

Bedingung: Programmieren Sie eine überladene Funktion, welche drei verschiedene Implementierungen enthält.

Das Ziel: Überladene Funktionen kennen lernen.

10.2 Anzahl Aufrufe

Ihre Aufgabe: Schreiben Sie eine Funktion die einen Wert zurückgibt. Der Wert soll anzeigen, wie oft die Funktion aufgerufen wurde.

Bildschirmausgabe:
Wie oft soll die Funktion aufgerufen werden? *5* Die Funktion wurde 5 mal aufgerufen!

Zeichnen Sie zuerst ein Struktogramm des Programms.

Bedingung: Verwenden Sie eine statische lokale Variable zur Lösung des Problems. Die Funktions Aufrufe müssen aus einer Schleife heraus erfolgen. Am Ende des Programms muss der Wert der *static* Variable in der main-Funktion ausgegeben werden. (Eine Kopie der Variable)

Das Ziel: *static* kennen lernen. Repetition von Funktionen und Schleifen.

11 Aufgabe 11

11.1 Vier Zahlen einlesen und ausgeben

Ihre Aufgabe: Schreiben Sie ein Programm, welches vier Zahlen einliest und in umgekehrter Reihenfolge wieder ausgibt:

Bildschirmausgabe:
z[0] = 2.5 z[1] = 23.5 z[2] = 99.9 z[3] = 12.0 Ausgabe: 12.0 99.9 23.5 2.5

Die Zahlen müssen vom Benutzer eingegeben und vom Programm wieder ausgegeben werden.

Zeichnen Sie zuerst ein Struktogramm des Programms.

Bedingung: Verwenden Sie einen Array vom Typ *double* mit vier Speicherstellen. Die Ein- und Ausgabe muss in einer Schleife erfolgen.

Das Ziel: Arrays kennen lernen.

11.2 Swap

Ihre Aufgabe: Schreiben Sie ein Programm, welches zwei Werte in der main-Funktion einliest und speichert. Die Werte müssen in einen Array mit zwei Feldern gespeichert werden. Übergeben Sie die Werte (bzw. die Adresse) einer Swap-Funktion, welche die Daten vertauscht. Die main-Funktion soll die vertauschten Werte auf der Konsole ausgegeben.

Bildschirmausgabe:
Eingabe der Werte: a[0] = 11 a[1] = 99 Die Werte nach Aufruf der Funktion: a[0] = 99 a[1] = 11

Zeichnen Sie zuerst ein Struktogramm des Programms.

Bedingung: Die Funktion soll *void* an den Aufrufer zurückgeben.

Das Ziel: Kombination von Arrays mit Funktionen kennen lernen.

12 Aufgabe 12

12.1 String addieren

Stringvariablen verwenden eine spezielle überladene Version des + Operators.

Ihre Aufgabe: Schreiben Sie ein Programm, welches zwei Strings vom Benutzer einliest. Diese beiden Variablen sollen aufaddiert und wieder ausgegeben werden. Sie können das gesamte Programm in der main-Funktion ausführen.

Bildschirmausgabe:
Geben Sie den 1. Teilstring ein: *Programmieren ist* Geben Sie den 2. Teilstring ein: *super.* Der Zusammengesetzte String lautet: Programmieren ist super.

Zeichnen Sie zuerst ein Struktogramm des Programms.

Tipp: Vergessen Sie nicht die *string* Library einzubinden.

Das Ziel: Strings kennen lernen.

12.2 Zeichen zählen

Mittels einer Schleife können die Anzahl Zeichen, die eine Stringvariable enthält ermittelt werden.

Ihre Aufgabe: Schreiben Sie ein Programm, welches vom Benutzer ein String einliest. Dieser String soll einer Funktion übergeben werden, welche die Anzahl Zeichen ermittelt und an die main-Funktion zurück gibt.

Bildschirmausgabe:
Geben Sie einen String ein: *Programmieren ist super.* Der String enthält 24 Zeichen!

Zeichnen Sie zuerst ein Struktogramm des Programms.

Tipp: Beachten Sie das Beispiel im Skript.

Das Ziel: Kombination von Strings und Funktionen kennen lernen.

12.3 Cäsar-Chiffre

Die Cäsar Verschlüsselung ist eine der einfachsten Verschlüsselungsalgorithmen. Jeder Buchstabe im Wort wird um eine bestimmte Anzahl Stellen im Alphabet noch hinten geschoben. Nachdem das `z` erricht wurde, muss das Alphabet zum weiterzählen nocheinmal wiederholt werden.

Ihre Aufgabe: Schreiben Sie ein Programm, welches einen Schlüssel und ein Wort einliest welches verschlüsselt und wieder entschlüsselt werden kann.
Eingabedaten:

- ein Wort
- Schlüssel

Das Programm soll je eine Funktion zum verschlüsseln und entschlüsseln des Wortes beinhalten.

Bildschirmausgabe:
Schluessel eingeben: 6 Wort eingeben: programmieren Der Schlüssel lautet: 6 Der verschlüsselte Text lautet: vxumxgssokxkt Der entschlüsselte Text lautet: programmieren

Zeichnen Sie zuerst ein Struktogramm des Programms.

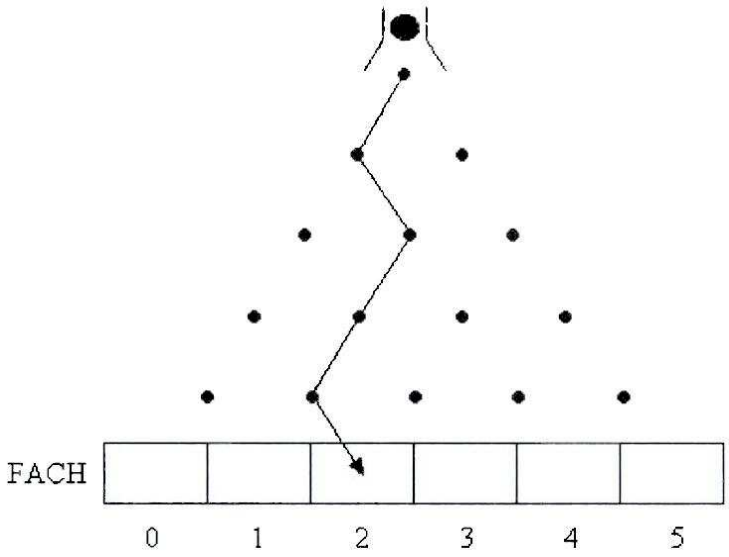
Bedingung: Verwenden Sie einen char-Array zum speichern des Wortes und eine Integer Variable zum speichern des Schlüssels.

Das Ziel: Funktionsaufrufe mit Adressen kennen lernen. Die ASCII Zeichen kennen lernen.

13 Aufgabe 13

13.1 Galtonsches Brett

Das galtonsche Brett besteht aus einer vertikalen Fläche mit Nägeln. Durch diese Nägel werden herunterfallende Kugeln abgelenkt. Die Nägel sind in 5 horizontalen Reihen angeordnet, sodass eine fallende Kugel bei jeder Reihe zufällig nach links oder rechts abgelenkt wird, bis sie nach der letzten Reihe in ein Fach fällt.



Ihre Aufgabe: Schreiben Sie ein Programm (ohne Grafik), das berechnet in welches Fach eine Kugel nach einem Durchgang geleitet wird. Das Programm soll automatisch 100 Kugeln berechnen und am Schluss die Anzahl Kugeln pro Fach anzeigen. Die Fächer sollen als Array implementiert werden.
Zeichnen Sie zuerst ein Struktogramm des Programms.

Tipp: Durch die Anzahl Rechtsablenkungen können Sie erkennen, in welches Fach die Kugel geleitet wird. Verwenden Sie Schleifen und Abfragen.

Das Ziel: Kleine Applikationen entwickeln lernen.

13.2 Fakultät

Die Fakultätsfunktion $n!$ ist das Produkt aller natürlichen Zahlen von 1 bis n.
$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \ldots \cdot 2 \cdot 1$
Ergänzend definiert man: $0! = 1$

Ihre Aufgabe: Schreiben Sie ein Programm, welches die Fakultätsfunktion implementiert. Die Berechnung muss in eine Funktion ausgelagert werden.

Das Ziel: Kleine Algorithmen entwickeln lernen.

14 Aufgabe 14

14.1 Zahlen raten

Dem Spieler stehen 6 Versuche zum erraten einer Zahl zwischen 1 und 100 zur Verfügung.

Ihre Aufgabe: Schreiben Sie ein Programm, welches eine Zufallszahl zwischen 0 und 100 erzeugt. Der Benutzer soll aufgefordert werden eine Zahl einzugeben. Das Programm muss die eingegebene Zahl überprüfen. Falls die Zahl grösser oder kleiner ist als die erzeugte Zufallszahl soll der Benutzer über diesen Zustand informiert werden. Falls die Zahl gefunden wurde soll das Programm beendet werden. Dem Benutzer stehen maximal 6 Versuche zur Verfügung. Falls die Zahl nicht innerhalb dieser 6 Versuche gefunden wurde, hat der Benutzer das Spiel verloren.
Es muss die Möglichkeit bestehen das Programm nach einem Durchgang zu widerhohlen oder abzubrechen.

Bildschirmausgabe:
Geben Sie die 1. Zahl ein: 50
Die gesuchte Zahl ist kleiner
Geben Sie die 2. Zahl ein: 25
Die gesuchte Zahl ist grösser
Geben Sie die 3. Zahl ein: 35
Die gesuchte Zahl ist grösser
Geben Sie die 4. Zahl ein: 40
Die gesuchte Zahl ist grösser
Geben Sie die 5. Zahl ein: 48
Die gesuchte Zahl ist kleiner
Geben Sie die 6. Zahl ein: 46
Gratulation! Sie haben die Zahl 46 im 6. Versuch erraten.

Zeichnen Sie zuerst ein Struktogramm des Programms.

Tipp: Verwenden Sie Schleifen und Abfragen.

Zusatz: Wenn der Benutzer die Zahl sehr früh herausfindet soll ein spezieller Text ausgegeben werden.

Das Ziel: Repetition von Schleifen und Abfragen.

14.2 Berechnung des Osterdatums eines Jahres

Das Osterdatum eines Jahres ist astronomisch wie folgt definiert:
Wenn d das erste Vollmond-Datum nach oder am 21. März ist, dann ist Ostern der erste Sonntag nach oder an diesem Tag d.
Es gibt verschiedene Algorithmen zur Berechnung des Osterdatums eines Jahres y. Der folgende von T.H. O'Beirne funktioniert für alle Jahre y im Bereich:

$1900 <= y <= 2099$

Algorithmus (T.H. O'Beirne)

1. Setzen Sie $n = y - 1900$
2. Führen Sie die folgenden 5 ganzzahligen Divisionen mit Rest durch (Tipp: Modulo Operator %):

Schritt	Dividend	Divisor	Quotient	Rest
(1)	n	19	not used	a
(2)	$7a + 1$	19	b	not used
(3)	$11a + 4 - b$	29	not used	m
(4)	n	4	q	not used
(5)	$n + q + 31 - m$	7	not used	w

3. Setzen Sie $k = 25 - m - w$
4. Es gilt: Osterdatum = k-ter April. Dabei kann k auch $<= 0$ sein. Verwenden Sie dazu folgenden Interpredation:

   k = 0: 31.März
   k = -1: 30.März
   k = -2: 29.März
   u.s.w.

Ihre Aufgabe: Schreiben Sie ein Programm, welches ein Jahr einliest und das Osterdatum dieses Jahres berechnet und anzeigt. Die Berechnung muss in einer Funktion erfolgen, welcher Sie das Jahr übergeben.

Bildschirmausgabe:
Berechnen des Osterdatums zwischen 1900 und 2099: Geben Sie das Jahr ein: *2005* Im Jahr 2005 sind die Ostern am 27. Maerz.

Tipp: Programmieren Sie Schritt für Schritt nach Vorgabe.

Das Ziel: Kleine Algorithmen entwickeln lernen.

14.3 Zinsberechnung

Es soll ein Zinsberechnungsprogramm für eine Bank entwickelt werden.

Ihre Aufgabe: Schreiben Sie ein Programm, welches folgende Daten vom Benutzer einliest:

- Schuldbetrag (`double`)
- Zins pro Jahr (`double`)
- Laufzeit (`int`)
- Amortisation (Rückzahlung) pro Jahr (`double`)

Schreiben Sie eine Funktion, welche die vier Werte übernimmt und dem Bankangestellten die Einnahmen zurück gibt. Geben Sie den Gewinn für die Bank und den totalen Rückzahlungsbetrag für den Bankkunden in der main-Funktion aus. In der Funktion muss der Schuldbetrag für das abgelaufene Jahr und der daraus resultierende Zinsbetrag ausgegeben werden.

Bildschirmausgabe:
Kreditbetrag in SFr: *200* Laufzeit in Jahren: *8* Zins pro Jahr in %: *12* Rückzahlung pro Jahr in SFr: *10* Schuldbetrag 1. Jahr: 200 SFr. Ergibt: 24.00 SFr Zins Schuldbetrag 2. Jahr: 190 SFr. Ergibt: 22.80 SFr Zins ... Schuldbetrag 7. Jahr: 140 SFr. Ergibt: 16.80 SFr Zins Schuldbetrag 8. Jahr: 130 SFr. Ergibt: 15.60 SFr Zins Der totale Zins über die gesamte Laufzeit beträgt: 158.4 Der totale Rückzahlungsbetrag des Kunden beträgt: 358.4

Tipp: Entwickeln Sie den Algorithmus bevor Sie mit der Programmierung beginnen.

Das Ziel: Programme nach einer Beschreibung entwickeln lernen.

14.4 Zinsberechnung mit einem Zeiger auf einen Array

Schreiben Sie das Zinsberechnungsprogramm so um, dass statt den einzelnen Variablen ein Array (double) mit drei Feldern verwendet wird. Nehmen Sie folgende Änderungen vor:

1. übergeben Sie der Funktion den Array (Schuldbetrag, Zins, Amortisation)
2. übergeben Sie der Funktion die Laufzeit (call by Referenz) des Vertrages

Das Ziel: Den Umgang mit Arrays und Funktionen kennen lernen. Referenzen als Funktionsparameter kennen lernen.

15 Aufgabe 15

15.1 Swap mit Referenzen

Es sollen zwei Werte vom Typ *double* in der main-Funktion vom Benutzer eingegeben werden. Schreiben Sie eine Funktion, welcher Sie die zwei Werte per Referenz übergeben können. Der Rückgabetyp der Funktion ist *void*. Die beiden Werte müssen ausgetauscht werden.

Bildschirmausgabe:
1. Wert eingeben: 12.9 2. Wert eingeben: 1.1 1. Wert vor dem Funktionsaufruf: 12.9 2. Wert vor dem Funktionsaufruf: 1.1 1. Wert nach dem Funktionsaufruf: 1.1 2. Wert nach dem Funktionsaufruf: 12.9

Zeichnen Sie zuerst ein Struktogramm des Programms.

Tipp: Beachten Sie das entsprechende Kapitel im Skript.

Das Ziel: Funktionsaufrufe mit Referenzen kennen lernen.

15.2 Programm analysieren

Analysieren Sie folgendes Programm. Überlegen Sie sich, welche Werte die Variablen a und b einnehmen werden. Schreiben Sie das Resultat auf und überprüfen sie es indem Sie den Kode ausführen.

```
1  int main(){
2   int a(6), b(2);
3   int *pa, *pb;
4   pa = &a;
5   *pa +=1;
6   cout<<"a = ? "<<a<<endl;
7
8   pb = &b;
9   *pb =*pb**pa;
10  b=b++ + ++a;
11  cout<<"b = ? "<<b<<endl;
12  return 0;
13 }
```

Das Ziel: Pointer verstehen und Kodesegmente analysieren lernen.

15.3 Programm analysieren

Analysieren Sie folgendes Programm. Überlegen Sie sich, welche Werte die Variablen, Pointer und Referenzen einnehmen werden. Schreiben Sie das Resultat auf und überprüfen Sie es indem Sie den Kode ausführen.

```
#include<iostream>
using namespace std;
int main(){
  float *f1, f2, f3(1), *f4;
  f2 = 23;
  float &f5 = f2;
  f1 = &f5;
  *f1 = 5;
  f1 = NULL;
  cout<<"f1: "<<f1<<endl;
  cout<<"f2: "<<f2<<endl;
  f4 = f1 = &f2;
  *f1 = 23;
  f4 = &f3;
  f3 = 17;
  cout<<"f1: "<<f1<<endl;
  cout<<"f2: "<<f2<<endl;
  cout<<"f3: "<<f3<<endl;
  cout<<"++*f4: "<<++*f4<<endl;
  return 0;
}
```

Tipp: Verwenden Sie eine Tabelle als Hilfsmittel.

Das Ziel: Pointer und Referenzen verstehen und Kodesegmente analysieren lernen.

15.4 Kleines EinMalEins mit zweidimensionalem Array - 1.Teil

Verändern Sie die Aufgabe mit dem kleinen EinMalEins so, dass die Werte in einen statischen zweidimensionalen Array gespeichert werden. Die Ausgabe der Werte muss in einer Funktion erfolgen. Übergeben Sie das Array der Funktion.

Das Ziel: Zweidimensionale Arrays kennen lernen.

15.5 Kleines EinMalEins mit zweidimensionalem Array - 2.Teil

Verwenden Sie statt dem statischen Array ein dynamisches zweidimensionales Array. Die Ausgabe der Werte muss in einer Funktion erfolgen. Übergeben Sie das Array der Funktion. Vergessen Sie nicht den Speicher wieder frei zu geben.

Das Ziel: Dynamische zweidimensionale Arrays kennen lernen.

16 Aufgabe 16

16.1 Struktur die einen Bruch enthält - 1. Teil

Schreiben Sie ein Programm, welches eine Struktur (*struct*) enthält. Die Daten eines Bruches sollen darin gespeichert werden. Die Daten werden in der main-Funktion vom Benutzer eingelesen und müssen nach folgender Vorgabe verändert werden:

- Der Bruchstrich muss durch ein Multiplikationszeichen ersetzt werden
- der Nenner muss durch den Zähler und der Zähler durch den Nenner ersetzt werden

Bildschirmausgabe:
Den Zähler eingeben: *1* Den Nenner eingeben: *5* Eingegebener Wert: 1/5 Ausgabe nach der Berechnung: 5 * 1

Das Ziel: *struct* kennen lernen.

16.2 Struktur die einen Bruch enthält - 2. Teil (Funktionen)

Die Veränderung der Daten (Zähler, Nenner, Operator) soll nun in eine Funktion ausgelagert werden. Diese Funktion besitzt den Rückgabetyp *void*. Die Struktur muss per Referenz übergeben werden.

Tipp: Der Name der Funktion darf auf keinen Fall die Zeichenkette *struct* enthalten, da es sonst Probleme beim Linken des Programmes geben könnte.

Das Ziel: *struct* in Kombination mit Funktionen kennen lernen.

16.3 Struktur die einen Bruch enthält - 3. Teil (Pointer)

Verwenden Sie für den 3. Teil dieser Aufgabe wieder die main-Funktion aus dem 1. Teil. Erzeugen Sie einen Pointer der Struktur. Dieser Pointer soll auf ein initialisiertes Strukturobjekt zeigen. Nachdem Sie die Adresszuweisung durchgeführt haben, dürfen Sie nur noch via Pointervariable auf die Strukturelemente zugreifen.

Tipp: Verwenden Sie den Pfeiloperator (`->`) zum lösen dieser Aufgabe.

Das Ziel: *struct* in Kombination mit Pointern kennen lernen.

17 Aufgabe 17

17.1 Datenbankprojekt - Teil 1

Erzeugen Sie eine Struktur (*struct*), welche vom User seinen Namen, Vornamen und sein Alter speichern kann. Verwenden Sie dazu die passenden Datentypen, jedoch keine String-Objekte. Stattdessen können Sie *char* Arrays anlegen. Die Eingabe und Ausgabe muss in einer Funktion erfolgen, welche solange Aufgerufen wird bis der Benutzer entscheidet, dass er keinen weiteren Datensatz eingeben oder Anzeigen möchte. Nach der Eingabe der Daten muss der Benutzer gefragt werden, welchen Datensatz er Anzeigen möchte.

Bildschirmausgabe:
Den 1. Datensatz eingeben: Name: *Muster* Vorname: *Hans* Alter: *99* Moechten Sie einen weiteren Datensatz eingeben? (y, n) *n* Welchen Datensatz moechten Sie anzeigen? *1* Name: Muster Vorname: Hans Alter: 99 Moechten Sie einen weiteren Datensatz anzeigen? (y, n) *n*

Tipp: Erzeugen Sie einen globalen Array, welcher Strukturelemente aufnehmen kann. Übergeben Sie der Funktion lediglich den Index des Arrays und nicht die Strukturvariablen. Verwenden Sie eine Schleife, welche den Funktionsaufruf enthält.

Das Ziel: *struct* kennen lernen.

17.2 Datenbankprojekt - Teil 2

Verwenden Sie den Kode aus dem 1. Teil der Aufgabe. Modularisieren Sie die Aufgabe. Verwenden Sie folgende drei Dateien zum speichern des Kodes:

- ein Headerfile
- ein Mainfile
- eine zusätzliche *.cpp-Datei für die Funktionen

Das Ziel: Die Modularisierung von Programmen kennen lernen.

18 Aufgabe 18

Es ist an der Zeit, dass wir uns mit den Standard-Libraries beschäftigen. Es gibt viele Libraries die zum lösen von Problemen verwendet werden können. Deshalb können viele Aufgaben an diese Bibliotheken delegiert werden. Das folgende Programm beschäftigt sich hauptsächlich mit der *ctime* Library.

18.1 Rechnen auf Zeit

Entwerfen Sie ein Programm, welches zwei Zufallszahlen zwischen 0 und 9 generiert. Der Benutzer muss aufgefordert werden, die beiden Zahlen im Kopf zu multiplizieren und das Resultat auf der Konsole einzugeben. Zeigen Sie danach in der Konsole an, wielange der Benutzer für die Berechnung benötigte.

Bildschirmausgabe:
Aufgabe starten? (y, n) *y* 5 * 4 = ? *20* Korrekt! Sie benoetigten 2 Sekunden.

Tipp: Zum lösen dieser Aufgabe müssen Sie die folgenden zwei Standard-Libraries einbinden:

- ctime
- cstdlib

Verwenden Sie die bereits vorhandene Datentypen und Funktionen der ctime Library.

Das Ziel: Vorgefertigte Elemente aus den Standard-Libraries kennen lernen.

18.2 Zahlen raten auf Zeit

Schreiben Sie die Aufgabe "Zahlen raten" so um, dass dem Benutzer 10 Sekunden zum erraten der Zahl zur Verfügung gestellt wird.

Bildschirmausgabe:
Geben Sie die 1. Zahl ein: 50 Die gesuchte Zahl ist kleiner Geben Sie die 2. Zahl ein: 46 Gratulation! Sie haben die Zahl 46 im 2. Versuch nach 3 Sekunden erraten.

Das Ziel: Repetition von Abfragen, Schleifen und C-Libraries.

19 Aufgabe 19

Zum Abschluss des Teils der strukturierten Programmierung sollten wir noch eine kurze Repetitionsphase einlegen. Lösen Sie dazu folgende Aufgaben.

19.1 Rekorde

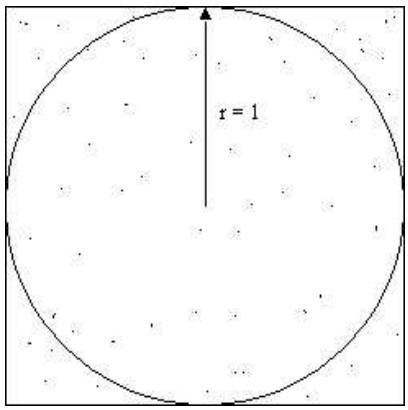
Ein Programm soll eine Folge von n = 100 Zufallszahlen erzeugen. $x_1, x_2, x_3, ..., x_n$.
Eine Zahl x_i (i = Index) der Folge ist ein Rekord wenn sie grösser ist als alle vorangegangenen Zahlen. Die Anzahl Rekorde ist in einer solchen Folge eine beliebige Zahl im Bereich 1...n. Wir interessieren uns für die zu erwartende Anzahl Rekorde. Um dies zu erreichen muss das Programm in eine Äussere Schleife eingebettet werden die es z.B. 100 mal wiederholt.
Erstellen Sie ein Pogramm zur Berechnung des Mittelwertes.

Tipp: Mit der Methode der Wahrscheinlichkeit kann der Theoretische Mittelwert (Erwartungswert) für die Anzahl Rekorde berechnet werden. Man erhält den Wert 5.187.

Das Ziel: Repetition von Abfragen und Schleifen.

19.2 Pi berechnen

In einem Quadrat werden zufällig erzeugte Punkte plaziert. In einem Kreis mit dem Radius 1 beträgt die Fläche π. Die Fläche können Sie durch das Verhältnis der Punkte innerhalb und ausserhalb (im umgebenden Quadrat) des Kreises berechnen.
Stellen Sie Sich folgende Grafik vor:



Ihre Aufgabe: Schreiben Sie eine Funktion zu Berechnung von π. Erzeugen Sie einen zufälligen Punkt (bestehend aus x und y Koordinaten) im Bereich zwischen -1 und 1. Verwenden Sie zur Berechnung die Formel des Pythagoras ($c = \sqrt{a^2 + b^2}$). Falls $c <= 1$ ist, dann befindet sich der Punkt innerhalb des Kreises. Erzeugen Sie z.B. 1000 Punkte. Zählen Sie alle Punkte innerhalb des Kreises. Danach multiplizieren Sie die Fläche des Quadrates mit der Anzahl Punkten im Kreis und dividieren es dann mit der Anzahl aller erzeugten Punkten.

Das Ziel: Repetition von Abfragen und Schleifen.

20 Aufgabe 20

Es ist nun an der Zeit, dass wir mit der Objektorientierten Programmierung (OOP) beginnen. Es ist sehr wichtig, dass Sie regelmässig Aufgaben zu diesem Thema lösen, da man sonst sehr leicht den Faden verliert.

20.1 Klassen - 1. Teil

Schreiben Sie ein Programm, welches eine Klasse enthält. Diese Klasse soll eine *private* Variable und zwei *public* Methoden enthalten.
Die *private* Variable soll folgender massen definiert werden:

- vom Typ Integer
- mit dem Namen *_a*

Die *public* Methoden müssen nach folgender Vorgabe bezeichnet werden:

- *void setValue(int x)*
- *void showValue()*

In der *private* Variable soll ein Wert gespeichert werden. Dafür ist die erste Methode (*setValue(int x)*) zuständig. Sie setzt die *private* Variable, indem sie einen vom Benutzer eingegebenen Wert übernimmt und in der Variable speichert. Die zweite Methode *(showValue())* hat keinen Übergabe- und keinen Rückgabewert. Sie muss lediglich den Wert der *private* Variable anzeigen.

Tipp: Überlegen Sie sich genau wie Sie auf die *private* Variable zugreifen können.

Das Ziel: Klassen, Methoden, Objekte und Attribute kennen lernen.

20.2 Klassen mit verschiedenen Dateien - 2. Teil

Verwenden Sie den 1. Teil der Aufgabe. Teilen Sie das Programm in drei verschiedene Dateien auf. Erzeugen Sie folgende Dateien:

- ein Headerfile (z.B. myTest.h)
- ein Mainfile (z.B. main.cpp)
- ein Implementationsfile (z.B. myTest.cpp)

Bedingung: Das Programm muss ohne Warnungen kompilier-, und ausführbar sein.

Tipp: Teilen Sie das Programm zuerst in eine Header-Datei (mit allen Headerinformationen und der Klasse) und eine main-Datei (mit allen Methoden und der Mainfunktion) auf.

Das Ziel: Die Aufteilung von Headerinformationen und Quellcode in Verbindung mit OOP kennen lernen.

21 Aufgabe 21

21.1 Rechteck

Schreiben Sie ein objektorientiertes Programm, welches die Fläche eines Rechteckes berechnet. Es soll nach der Länge der Seiten a und b gefragt werden. Diese beiden Werte werden in der main-Funktion eingegeben und von einer Methode (*setValues(double x, double y)*) in die zwei *private* Variablen gespeichert. Nach dem setzen der Werte soll eine Methode (*calculateArea()*) aufgerufen werden, welche die Fläche des Rechteckes berechnet. Zum Schluss muss die Fläche in der main-Funktion ausgegeben werden. Dazu benötigen Sie die Methode getArea(), welche das Resultat zurückgibt.
Verwenden Sie als Hilfe folgende Klasse:

```
1  class MyRectangle
2  {
3    public:
4      void setValues(double x, double y);
5      void calculateArea();
6      double getArea();
7
8    private:
9      double _a;
10     double _b;
11     double _area;
12 };
```

Implementieren Sie die drei Methoden und die main-Funktion. Verwenden Sie die vorgegebene Klassendeklaration. Teilen Sie den Kode in drei Dateien auf. (zwei *.cpp und ein *.h File)

Bildschirmausgabe:
Seite a eingeben: *3* Seite b eingeben: *4* Die Flaeche des Rechteckes lautet: *12*

Zusatz: Schreiben Sie eine zusätzliche Methode, die überprüft ob es sich bei dem Rechteck um ein Quadrat handelt.

Das Ziel: Klassen und Datenkappselung kennen lernen.

22 Aufgabe 22

22.1 Konstruktor

Erweitern Sie die Klasse der Aufgabe 21 mit einem Konstruktor, welcher die drei *private* Variablen initialisiert.
Führen Sie an der Aufgabe 21 folgende Änderungen durch:

- entfernen Sie die Methode setValues(double x, double y)
- implementieren Sie einen Konstruktor, welcher die *private* Variablen initialisiert
- konstruieren Sie das Objekt, nachdem der Benutzer die zwei Werte eingegeben hat.

Die zwei Werte sollen eingelesen werden bevor das Objekt erzeugt wird. Dem Konstruktor müssen diese beiden Variablen übergeben werden. Dadurch kann bei der Erzeugung eines Objektes sofort die Initialisierung der *private* Variablen durchgeführt werden. Das Attribut *_area* soll im Konstruktor mit 0 initialiesiert werden.

Das Ziel: Standard Konstruktoren kennen lernen.

22.2 Kopierkonstruktor

Erweitern Sie die Klasse der Aufgabe 22.1 mit einem Kopierkonstruktor. Der Kopierkonstruktor kopiert alle Attribute eines Objektes in ein Objekt, welches neu erzeugt wird. Erzeugen Sie nach der Berechnung der Fläche ein neues Objekt, welches eine Kopie des ersten Objektes darstellt. Geben Sie zur Kontrolle die Fläche der beiden Objekte auf der Konsole aus.

Das Ziel: Standard Kopierkonstruktoren kennen lernen.

22.3 Kopierkonstruktor mit dynamischen Attributen

Kopieren Sie die Aufgabe 22.2. Verändern Sie die *private* Variable *_b*. Sie soll als Pointervariable deklariert werden. Dazu müssen Sie die Variable im Standard- und Kopierkonstruktor dynamisch erzeugen. Überlegen Sie sich genau, wie Sie die Werte kopieren.
Berücksichtigen Sie folgende Punkte bei der Programmierung des Standard Konstruktors:

- das Attribut darf nicht auf die Speicherstelle einer Variable der main-Funktion zeigen
- das Attribut darf nicht auf eine lokale Variable des Konstruktors zeigen

Berücksichtigen Sie folgende Punkte bei der Programmierung des Kopierkonstruktors:

- die Attribute der Objekte dürfen nicht auf die gleiche Speicherstelle zeigen

Zusatz: Schreiben Sie den passenden Destruktor dazu.

Das Ziel: Kopierkonstruktoren mit dynamischen Attributen kennen lernen.

23 Aufgabe 23

23.1 Koordinatensystem

Punkte sollen gesetzt, verschoben und angezeigt werden können.

```
class Point{
  public:
    Point(double x, double y);
    Point(const Point &P);
    void setPoint(double x, double y);
    void showPoint();
    double getX();
    double getY();
    void shiftAxis(double value);
  private:
    double _x;
    double _y;
};
```

Implementieren Sie die Methoden nach folgender Vorgabe:

- *setPoint(double x, double y)* setzt die *_x* und *_y* Variable
- *showPoint()* zeigt die Werte der *_x* und *_y* Variable an
- *getX()*, *getY()* gibt den Wert der *_x*, bzw. *_y* Variable zurück
- *shiftAxis(double value)* schiebt die Achsen in den positiven Bereich (*axValue*)

Die main-Funktion soll nach folgender Vorgabe definiert werden:

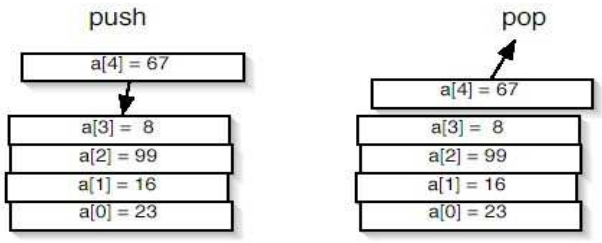
```
int main(){
  double dx(0),dy(0),axValue(2);
  Point p1(0,0);
  cout<<"x und y Werte eingeben: \n"; cin>>dx; cin>>dy;
  p1.setPoint(dx,dy);
  cout<<"Die Achse um "<<axValue<<" verschieben"<<endl;
  p1.shiftAxis(axValue);
  cout<<"\nPosition von p1:  x= "<<p1.getX()<<" y= "<<p1.getY()<<endl;
  Point p2(p1);
  cout<<"\nPosition von p2: "<<endl;
  p2.showPoint();
  return 0;
}
```

Das Ziel: Klassen mit Konstruktoren und Methoden kennen lernen.

24 Aufgabe 24

24.1 Stack - 1. Teil

Stack ist Englisch und bedeutet Stapel. Der Stack ist eine einfache Datenstruktur. Zu speichernde Elemente (z.B. Variablen) werden gestapelt. Ein neues Element wird immer auf das vorangegangene Element gelegt. Entfernt wird immer nur das oberste Element. D.h., dass das zuletzt gespeicherte Element immer zuerst entfernt werden muss. Diese Art von Speicherorganisation nennt man LIFO-Struktur. (**L**ast **I**n **F**irst **O**ut)



Verwenden Sie folgende Klasse als Vorlage:

```
1  class Stack{
2    public:
3      Stack(int s = 5);
4      ~Stack();
5      bool push(const int &element);
6      bool pop(int &element);
7      bool top(int &element);
8    private:
9      int *_pa;
10     int _top;
11     int _count;
12 };
```

Implementieren Sie die Methoden nach folgender Vorgabe:

- Konstruktor und Destruktor
- *bool push(const int &element)* überprüft, ob ein weiteres Element auf den Stack gelegt werden darf. Wenn der Stack noch nicht voll ist, wird der Wert gespeichert.
- *bool pop(int &element)* überprüft, ob der Stack schon leer ist. Wenn der Stack nicht leer ist, wird der oberste Wert der Referenz zugewiesen, welche nachher in der main-Funktion ausgelesen wird. Nach dem auslesen wird das oberste Element im Stack gelöscht.
- *bool top(int &element)* überprüft, ob der Stack schon leer ist. Wenn der Stack nicht leer ist, wird der oberste Wert der Referenz zugewiesen, welche nachher in der main-Funktion ausgelesen wird.

Bildschirmausgabe:
5 Zufallszahlen werden auf den Stack gelegt: Das 1. Element mit dem Wert 72 wurde auf den Stack gelegt. ... Das 5. Element mit dem Wert 97 wurde auf den Stack gelegt. Das oberste Element hat den Wert: 97 Die 5 Elemente werden vom Stack geholt: Das 5. Element hat den Wert 97. ... Das 1. Element hat den Wert 72.

Das Ziel: Klassen, Konstruktoren und Methoden kennen lernen.

24.2 Stack - 2. Teil

Erweitern Sie die Aufgabe mit dem Stack. Fügen Sie dem Programm folgende Punkte hinzu:

- Deklarieren Sie die `top()` Methode als konstant.
- Ersetzen Sie das statische Objekt durch einen Array, welcher drei dynamisch erzeugte Objekte enthält. Verwenden Sie das Objekt im zweiten Feld für die Ausführung der Aufgabenstellung vom 1. Teil.

Das Ziel: Konstante Methoden und die Erzeugung von dynamischen Objekten kennen lernen.

25 Aufgabe 25

25.1 Bubble Sort

Schreiben Sie ein Programm, welches mit dem *Bubble Sort* Algorithmus acht Zahlen sortiert. Die Zahlen sind beliebig und sollen in einem Array gespeichert werden. Geben Sie den Inhalt des Arrays vor und nach der Sortierung aus.
Implementieren Sie eine passende *Bubble Sort* Funktion. (Achtung Pointer!) Das Programm muss nicht objektorientiert aufgebaut werden.

Bildschirmausgabe:
Die Zahlen vor der Sortierung: 45, 34, 91, 23, 12, 67, 2, 36 Die Zahlen nach der Sortierung: 2, 12, 23, 34, 36, 45, 67, 91

Tipp: Überlegen Sie sich genau wie viele Schleifen Sie benötigen.

Das Ziel: Sortieralgorithmen kennen lernen.

25.2 Selection Sort

Schreiben Sie ein Programm, welches die Vorgaben der vorhergehenden Aufgabe erfüllt. Verwenden Sie jedoch den *Selection-*, statt den *Bubble Sort* Algorithmus.

Tipp: Überlegen Sie sich, ob Sie das *Bubble Sort* Programm als Basis für das *Selection Sort* Programm verwenden können.

Das Ziel: Sortieralgorithmen kennen lernen.

25.3 Insertion Sort

Schreiben Sie ein Programm, welches die Vorgaben der vorhergehenden Aufgabe erfüllt. Verwenden Sie jedoch den *Insertion Sort*, statt den *Bubble-*, oder *Selection Sort* Algorithmus.

Tipp: Überlegen Sie sich, ob Sie eines der vorhergehenden Programme als Basis für das *Insertion Sort* Programm verwenden können.

Das Ziel: Sortieralgorithmen kennen lernen.

26 Aufgabe 26

26.1 *this*-Zeiger

this verweist auf die Adresse des aufrufenden Objektes. Wenn in einer Methode eine Instanzvariable den selben Namen hat wie eine lokale Variable, muss man den this Zeiger verwenden. Dadurch kann man auf die Instanzvariablen des aufrufenden Objektes verweisen.
Gegeben ist die folgende Klassendeklaration:

```
1 class MyThis{
2   public:
3     MyThis();
4     int getX();
5     void showX(int x);
6
7   private:
8     int x;
9 };
```

Entwickeln Sie die main-Funktion und die deklarierten Methoden. Achten Sie darauf, dass Sie zwei Variablen mit dem selben Namen x in der Methode *setX()* ansprechen können.
Die Methoden haben folgende Aufgaben:

- *int getX()* soll die Instanzvariable des aufrufenden Objektes zurück geben
- *void showX(int x)* soll den Wert der lokalen- und Instanzvariable ausgeben

Initialisieren Sie die zu übergebende Variable x in der main-Funktion mit dem Wert 99 und konstruieren Sie die Instanzvariable x im Konstruktor mit dem Wert 100.

Bildschirmausgabe:
In der Methode: lokales x = 99 Instanzvariable x = 100 Rueckgabewert der getX()-Methode im main(): 100

Das Ziel: Den *this*-Zeiger kennen lernen.

26.2 Fragen zum *this*-Zeiger

1. Wie können Sie das Problem mit dem *this* Zeiger umgehen?
2. Könnten Sie in jeder "Instanz-" Methode den *this* Zeiger verwenden wenn Sie auf ein Attribut zugreifen?

27 Aufgabe 27

27.1 static

Implementieren Sie die Methoden der Klasse *MyStatic*. Achten Sie darauf, dass die Methode *static int getA()* eine statische Klassenmethode ist. Die Variable *a* ist ebenfalls statisch. Erzeugen Sie in der main-Funktion 3 neue Objekte. Geben Sie den Wert der Variable *a* vor Erzeugung des ersten Objektes, nach Erzeugung des zweiten Objektes und am Schluss der main-Funktion auf der Konsole aus. Die *static* Variable soll bei jedem Konstruktor Aufruf inkrementiert werden. Dadurch können Sie die Anzahl erzeugter Objekte ermitteln. Der Zugriff auf die *static* Methode *getA()* muss die ersten zwei mal mit dem Klassen Namen und das letzte mal mit einem Objektnamen erfolgen.

```
1 class MyStatic{
2   public:
3     MyStatic();
4     static int getA();
5
6   private:
7     int x;
8     static int a;
9 };
```

Bildschirmausgabe:
Rueckgabewert der getA()-Methode im main(): 0
1. Objekt erzeugen.
2. Objekt erzeugen.
Rueckgabewert der getA()-Methode im main(): 2
3. Objekt erzeugen.
Rueckgabewert der getA()-Methode im main(): 3

Tipp: Überlegen Sie sich mit welchem Operator Sie auf *static* Elemente zugreifen können.

Das Ziel: *static* kennen lernen.

27.2 Fragen zu *static*

1. Kann man auf private *static* Variablen aus der main-Funktion heraus direkt zugreifen?
2. Kann man auf *static* Elemente zugreifen bevor ein Objekt einer Klasse erzeugt wurde? Begründen Sie Ihre Antwort.
3. Ist es möglich, eine Instanzvariable aus einer *static* Methode zu ändern?
4. Wann brauchen Sie welchen Operator zum zugreifen auf ein *static* Elemente?
5. Was ist der Sinn einer *static* Variable?

27.3 Pferderennen 1.Teil

Ein Pferderennen besteht aus drei Runden. Jede Runde eintspricht einem Objekt, welches die Rundenlänge und die Rundenzeit speichern kann. Es sollen drei Runden Objekte erzeugt werden. Wenn das Rennen beendet wird, müssen die Runden (Zeit und Länge) addiert werden. Dazu müssen Sie den + Operator überladen.

Ihre Aufgabe: Schreiben Sie ein Programm, welches drei Objekte erzeugen und addieren kann. Verwenden Sie folgende Klassen Vorgabe:

```
class Lap{
  public:
    Lap(int l = 400 );
    double randomTime();
    double delay(double );
    //Deklaration des überladenen + Operators
    double getTime();
    int getLength();
  private:
    int roundLength;
    double roundTime;
};
```

Hinweise zur Implementierung:

- Deklarieren Sie den Konstruktor wie deklariert. Der Parameter entspricht der Rundenlänge. Die Erste Runde beträgt nur 200m, die Restlichen 400m.
- `randomTime()` gibt eine Zufallszeit x zwischen 5 und 10 Sekunden zurück.
- `delay()` übernimmt beim Aufruf die Zufallszeit und unterbricht das Programm für x Sekunden pro Runde. Die Zufallszeit x wird zurück gegeben.
- Der überladene Operator muss die Rundenlängen und Rundenzeiten addieren.

Verwenden Sie die folgende main-Funktion:

```
int main(){
  srand(time(NULL));
  cout<<"Das Rennen wurde gestartet..."<<endl;
  Lap obj1(200);
  cout<<"Zeit fuer 1. Runde: "<<obj1.delay(obj1.randomTime())<<"s"<<endl;
  Lap obj2(400);
  // ...
  Lap objRes;
  objRes = obj3 + obj2 + obj1;
  cout<<"Gesamtzeit nach 3 Runden("<<objRes.getLength()<<...;
  return 0;
}
```

Bildschirmausgabe:
Das Rennen wurde gestartet...
Zeit fuer 1. Runde: 6s
Zeit fuer 2. Runde: 5s
Zeit fuer 3. Runde: 8s
Gesamtzeit nach 3 Runden(1000m): 19s

Tipp: Bauen Sie das Programm Schritt für Schritt auf.

Das Ziel: Überladen von Operatoren kennen lernen.

27.4 Pferderennen 2.Teil

Da ein Pferderennen aus drei Runden besteht wäre es sinnvoll, wenn man eine Klasse `Race` verweden könnte, welche die drei Runden enthält.

Ihre Aufgabe: Fügen Sie dem Pferderennen 1. Teil eine die Klasse `Race` hinzu, welche die Runden als Attirutte enthält.

Das Ziel: Agregation kennen lernen.

28 Aufgabe 28

28.1 Vererbung

Implementieren Sie ein Programm, welches drei Klassen enthält. Die erste Klasse ist die Superklasse (Basisklasse) welche *Person* heisst. Die Subklassen *Man* und *Woman* erben von der Superklasse. Betrachten Sie folgende Klassendeklaration:

```
1  class Person{
2    public:
3      void showElements();
4      void setAge(int );
5      void setSize(int );
6    protected:
7      int _size;
8    private:
9      int _age;
10 };
```

Leiten Sie die Klasse *Man* und *Woman* ab. Diese Klassen besitzen jeweils eine eigene Implementation der Methode *drawShape()*. Die Methoden der Superklasse müssen nur als Bestandteile der Superklasse implementiert werden, können jedoch von den Subklassen verwendet werden. Die Methoden beinhalten folgende Funktionalität:

- *void showElements()* zeigt die Attribute an
- *void setAge(int)* und *void setSize(int)* setzten die Attribute
- *void drawShape()* zeichnet den Körper der Person

Die Methode *drawShape()* muss Zugriff auf die Grösse der Person haben. Deshalb darf *_size* nicht als *private* Variable in der Superklasse deklariert werden. Betrachten Sie die folgende Klassendeklaration der Subklasse *Man*:

```
1  class Man : public Person{
2    public:
3      void drawShape();
4  };
```

Deklarieren Sie die gleiche Subklasse mit den Namen *Woman*.
Die main-Funktion soll vom Benutzer die Daten für die Frau und den Mann einlesen. Die Werte müssen danach wieder mit der Methode *showElements()* ausgegeben werden. Der Körper muss mit der Methode *drawShape()* gezeichnet werden.
Die Berechnung der Grösse der Person erfolgt jeweils in der *drawShape()* Methode. Die Einheit ist Millimeter. Nachdem die Methode die Grösse ermittelt hat, soll sie durch 10 dividiert werden und in eine lokale Variable gespeichert werden. Das Resultat ergibt die Anzahl Zeilen die gezeichnet werden müssen.

Betrachten Sie die folgende Ausgabe:

Bildschirmausgabe:
Einen Mann erzeugen: Die Groesse eingeben: 50 Das Alter eingeben: 33 Die Daten vom Mann anzeigen: Age: 33 Size: 50 ********* ******* ***** *** * Einen Frau erzeugen: Die Groesse eingeben: 40 Das Alter eingeben: 30 Die Daten der Frau anzeigen: Age: 30 Size: 40 * *** ***** *******

Tipp: Erstellen Sie zuerst das Grundgerüst des Programms mit kurzen *cout* Befehlen. Dadurch können Sie erkennen, ob die gewünschten Methoden aufgerufen werden.

Zusatz: Zeichnen Sie das UML (Unified Modeling Language) Diagramm der Aufgabe.

Das Ziel: Vererbung kennen lernen.

28.2 Fragen zur Vererbung

1. Was unterscheidet *protected* von *private*?
2. Was unterscheidet *protected* von *public*?
3. Worin bestehen die zwei grössten Vorteile wenn man mit Vererbung arbeitet?
4. Durch welche Erweiterung wird eine Klasse als Subklasse deklariert?
5. Kann eine Subklasse eine Superklasse einer anderen Klasse sein?