

# KLAUSUR 1 Programmierpraktikum 2BK11 24.11.2009 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:

keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.  
Bei praktischen Arbeiten am Computer bitte folgendes beachten:
- NICHT **ablauffähige** C-Programme werden mit 6 bewertet ! **Jede** Aufgabe als eigenes Projekt (keine Warnungen vom Compiler).
- Der Name, Vorname und die Aufgabennummer (z.B: Aufgabe 1) muß als **Kommentar** in jedes Programm mitaufgenommen werden.
- Name der Quelldatei nach folgendem Schema vergeben: Gruppe\_Rechnernummer\_Nachname\_Vorname\_Aufgabennr.  
Beispiel: A\_12\_Mustermann\_Erika\_nr3
- Die Programme müssen ausgedruckt werden.
- Die entsprechenden Quelldateien müssen auf Diskette abgespeichert und abgegeben werden.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punktabzug !!!!

Bemerkungen:

Der Compiler darf keine Warnungen und Fehler erzeugen !!!

## AUFGABEN

1) Es soll ein Taschenrechner programmiert werden.

Zuerst muss dazu ein Zeichen über Tastatur eingegeben werden:

Bei Eingabe des Zeichens A oder a wird eine Addition durchgeführt,

bei Eingabe des Zeichens S oder s wird eine Subtraktion durchgeführt,

bei Eingabe des Zeichens M oder m wird eine Multiplikation durchgeführt,

bei Eingabe des Zeichens D oder d wird eine Division durchgeführt,

bei Eingabe eines anderen als der oben beschriebenen Zeichen, muß das Programm **sofort** beendet werden und eine entsprechende Meldung auf den Bildschirm ausgegeben werden. (insbesondere dürfen dann nicht mehr weitere Zahlen eingegeben bzw. etwas gerechnet werden).

Dann müssen - falls das Programm nicht beendet werden soll - 2 Zahlen eingegeben werden und die entsprechende Rechenoperation ausgeführt werden.

Bemerkungen:

1) Dies muß mit dem **EVA-Prinzip** realisiert werden.

2) Es widerspricht nicht dem EVA-Prinzip, wenn z.B. im Eingabeteil in bestimmten Variablen bestimmte Zustände abgespeichert werden (z.B: ob das Programm beendet werden soll) und diese dann im Ausgabeteil abgeprüft werden.

3) return darf nur einmal (am Ende des Programms) benutzt werden.

Programm darf nicht durch exit(...) oder sonstige Befehle verlassen bzw. beendet werden.

4) Durch 0 darf nicht dividiert werden.

# KLAUSUR 1 Programmierpraktikum 2BK11 25.11.2009 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:  
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.  
Bei praktischen Arbeiten am Computer bitte folgendes beachten:
- NICHT **ablauffähige** C-Programme werden mit 6 bewertet ! **Jede** Aufgabe als eigenes Projekt (keine Warnungen vom Compiler).
- Der Name, Vorname und die Aufgabennummer (z.B: Aufgabe 1) muß als **Kommentar** in jedes Programm mitaufgenommen werden.
- Name der Quelldatei nach folgendem Schema vergeben: Gruppe\_Rechnernummer\_Nachname\_Vorname\_Aufgabennr.  
Beispiel: A\_12\_Mustermann\_Erika\_nr3
- Die Programme müssen ausgedruckt werden.
- Die entsprechenden Quelldateien müssen auf Diskette abgespeichert und abgegeben werden.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punktabzug !!!!

Bemerkungen:

Der Compiler darf keine Warnungen und Fehler erzeugen !!!

## AUFGABEN

1) Es soll ein Rechner für Digitalaltungen programmiert werden.

Zuerst muss dazu eine Zahl über Tastatur eingegeben werden:

Bei Eingabe der Zahl 1 wird eine NOR Schaltung berechnet,

bei Eingabe der Zahl 2 wird eine NAND Schaltung berechnet,

bei Eingabe einer anderen als der oben beschriebenen Zahlen, muß das Programm **sofort** beendet werden und eine entsprechende Meldung auf den Bildschirm ausgegeben werden. (insbesondere dürfen dann nicht mehr weitere Eingangswerte eingegeben bzw. etwas gerechnet werden).

Dann müssen - falls das Programm nicht beendet werden soll - 2 Eingangswerte (die nur die Werte 0 oder 1 haben dürfen) eingegeben werden und der Ausgangswert der entsprechenden Schaltung berechnet werden.

Bei Eingabe eines von 0 oder 1 verschiedenen Eingangswertes, muß das Programm **sofort** beendet werden und eine entsprechende Meldung auf den Bildschirm ausgegeben werden. (insbesondere dürfen dann nicht mehr weitere Eingangswerte eingegeben bzw. etwas gerechnet werden).

Bemerkungen:

1) Dies muß mit dem **EVA-Prinzip** realisiert werden.

2) Es widerspricht nicht dem EVA-Prinzip, wenn z.B. im Eingabeteil in bestimmten Variablen bestimmte Zustände abgespeichert werden (z.B: ob das Programm beendet werden soll) und diese dann im Ausgabeteil abgeprüft werden.

3) return darf nur einmal (am Ende des Programms) benutzt werden.

Programm darf nicht durch exit(...) oder sonstige Befehle verlassen bzw. beendet werden.

4) Durch 0 darf nicht dividiert werden.

*Name, Vorname:*

Hilfsmittel:

Tabelle mit Prioritäten und Assoziationen

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form auf jeder Seite unten rechts durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Programme dürfen beim Kompilieren keine Warnungen und keine Fehlermeldungen bringen.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

- 1) Was ist ein Algorithmus und welche Eigenschaften hat er? (5 P)
- 2) Was ist die ANSI Norm ? (2 P)
- 3) Was ist ein Literal ? (2 P)
- 4) Was bedeutet Priorität in der Programmiersprache C ? (2 P)  
Geben Sie ein Beispiel (Ausdruck) an.
- 5) Was bedeutet Assoziativität in der Programmiersprache C ? (3 P)  
Geben Sie ein Beispiel (Ausdruck) an.
- 6) Kann die Ganzzahl 29999 in einer zwei Bytes grossen Integer-Zahl abgespeichert werden ?  
Begründen Sie! (4 P)
- 7) (8 P)
  - a) Erscheint beim Kompilieren des folgenden syntaktisch korrekten Programmausschnitts eine Warnung ? Begründen Sie genau!
  - b) Welchen Wert hat f ?

```
...  
float f;  
f = 3/5*(3.5 + 4 * 2.5);  
...
```

8) Vergleichen Sie jeweils die folgenden Ausdrücke (10 P)  
Haben diese (bei jeweils gleichem Wert der Variablen) Ausdrücke immer den gleichen Wert?  
Wenn nein, geben Sie bitte Werte für x und y, so dass die Ausdrücke verschiedene Werte haben. Geben Sie auch die Werte der Ausdrücke an:

-----  
Beispiel:

$(x+y)*z$        $x=1, y=2, z=3$ : Wert 9

$x+y*z$        $x=1, y=2, z=3$ : Wert 7  
-----

x und y haben den Datentyp int.

a)  $!(x < y)$       bzw.  $!x < y$

b)  $4 == x \ || \ 4 < x$       bzw.  $4 == x \ || \ 4$

9) (15 P)

Was wird auf dem Bildschirm ausgegeben ? Begründen Sie !

```
...
int x, y ;
x=0;
y=0;
if(x==y==0)
    printf("Ausgabe1\n");
if(y==(x=0))
    printf("Ausgabe2\n");
if(y=x=0)
    printf("Ausgabe3\n");
...
```

Lösungen:

1) Ein Algorithmus ist ein Verfahren zur Lösung eines gegebenen Problems. Er ist eindeutig, endlich und schrittweise.

2) Die ANSI Norm ist eine Vorschrift, die festlegt, nach welchen Regeln ein C-Programm aufgebaut sein muss.

3) Literale sind Bezeichner mit einem festen Wert wie z.B. 9

4) Priorität gibt den Vorrang eines Operators in einem Ausdruck an.  
Beispiel: Punkt vor Strich.

5) Assoziativität gibt an, in welcher Reihenfolge (von links nach rechts oder von rechts nach links) Operatoren mit der gleichen Priorität abgearbeitet werden.

Beispiel: \* wird von links nach rechts abgearbeitet.

6) 2 Bytes kann  $2^{16} \approx 64000$  verschiedene Zustände speichern.

Damit können Ganzzahlen von ungefähr -32000 bis ungefähr 32000 abgespeichert werden.

Also kann 29999 in dieser zwei Bytes grossen Integer-Zahl gespeichert werden.

7)

a) Es wird zuerst  $3/5$  berechnet (Assoziativität von links nach rechts). Der Wert ist 0.

4 ist integer, 2.5 ist double. Also wird 4 in double 4.0 umgewandelt. Das Ergebnis 10.0 ist damit double. 3.5 ist double und damit ist  $3.5 * 10.0$  auch double. Dieser Wert wird mit integer 0 multipliziert, wobei deshalb vorher die integer 0 in double 0.0 umgewandelt wird und den Wert double 0.0 ergibt.

Da double in float abgespeichert wird, wird die double Zahl 0.0 in float umgewandelt. Dabei kann ein Datenverlust entstehen. Deshalb gibt der Compiler eine Warnung aus.

b) 0.0

8)

a)  $!(x < y)$       bzw.     $!x < y$

$x=1, y=2$  ;

$!(x < y)$  hat den Wert 0

$!x < y$  hat den Wert 1

b)  $x=2$  :

$4 == x \parallel 4 < x$  hat den Wert 0

$4 == x \parallel 4$  hat den Wert 1

9)

a) Da x und y den Wert 0 haben, hat der Ausdruck  $x == y$  den Wert 1. Damit hat der Ausdruck  $x == y == 0$  den Wert 0.

Also gibt es keine Ausgabe auf dem Bildschirm.

b) Der Ausdruck  $x=0$  hat den Wert 0. Damit hat der Ausdruck  $y==(x=0)$  den Wert 1.

Also wird « Ausgabe1 » auf dem Bildschirm ausgegeben.

c) Der Ausdruck  $x=0$  hat den Wert 0. Damit hat der Ausdruck  $y=x=0$  den Wert 0.

Also gibt es keine Ausgabe auf dem Bildschirm.

*Name, Vorname:*

Hilfsmittel:

Tabelle mit Prioritäten und Assoziationen

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form auf jeder Seite unten rechts durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Programme dürfen beim Kompilieren keine Warnungen und keine Fehlermeldungen bringen.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

1) Was ist allgemein der Hauptunterschied (bzgl. der Anzahl der Durchgänge) zwischen einer While-Anweisung und einer Do-Anweisung ?

Geben Sie dazu jeweils ein **konkretes** Beispiel (Programmteil) an. (10P)

2) Gegeben ist der folgende syntaktisch korrekte Programmteil: (10P)

```
i=10;
for(i=0;i<20;i=i+1);{
    printf("Hallo Welt\n");
}
```

Wie oft gibt dieser Programmteil die Meldung "Hallo Welt" auf dem Bildschirm aus ?  
Begründen Sie !

3) a) Was ist eine Endlosschleife? (5P)

b) Schreiben Sie einen Programmausschnitt (in der Programmiersprache C) mit einer Endlosschleife.

4) Ein Anwender soll eine ganze Zahl zwischen 1 und 5 (je einschließlich) über Tastatur eingeben. Schreiben Sie dazu einen Programmausschnitt in C, in der der Anwender gezwungen wird, so lange eine Zahl einzugeben, bis diese Bedingung erfüllt ist. Erst dann soll im Programm die nächste Anweisung erreicht werden. (10P)

5) Gegeben ist der folgende Programmteil:

(15P)

```
sum = 0;  
i = 1;  
  
while(1==1) {  
    i = i+1;  
    sum = sum +i;  
-->  
}
```

a)

Tragen Sie die Werte der Variablen (an der Stelle -->) i und sum bei den ersten drei Schleifendurchgängen in die Tabelle ein (ohne Berücksichtigung der Schleifenbedingung )

i					
sum					

b)

Verändern Sie den Programmausschnitt an genau 2 Stellen so, daß der Wert der Variablen sum nach Verlassen der Schleife gleich der Summe  $5 + 6 + 7 + \dots + 13 + 14 + 15$  ist.

## Lösungen:

1) Anzahl Durchgänge der do-while-Anweisung:  $\geq 1$

Anzahl Durchgänge der while-Anweisung:  $\geq 0$

```
i = 10;
do{
    printf("%d ", i);
}
while(i < 10);
```

```
i = 10;
while(i < 10){
    printf("%d ", i);
}
while(i < 10);
```

2)

Ein Mal, weil

```
printf("Hallo Welt\n");
```

nicht zum Körper der for-Anweisung (Semikolon beachten !) gehört.

3)

a) Eine Endlosschleife wird nie verlassen

b)

```
while(1==1){
}
```

4)

...

```
do {
    printf("ganze Zahl zwischen 1 und 5 eingeben\n");
    scanf("%d", &i);
} while(!(i <= 10) && i <= 5);
```

5)

a)

i	2	3	4		
sum	2	2 + 3	2 + 3 + 4		

b)

```
sum = 0;
i = 4;
```

```
while(i < 15){
    i = i + 1;
    sum = sum + i;
}
```



# KLAUSUR 1 Programmiertheorie 2BKI1 Nachtermin 1 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:

Tabelle mit Prioritäten und Assoziationen

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form auf jeder Seite unten rechts durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Programme dürfen beim Kompilieren keine Warnungen und keine Fehlermeldungen bringen.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

- 1) 10P
- a) Geben Sie das Struktogramm eines Programms (mit einer while Anweisung realisiert) an, das die Zahlen 10, 11, ..., 19, 20 auf dem Bildschirm ausgibt.
- b) Geben Sie das Struktogramm eines Programms (mit einer do - while Anweisung realisiert) an, das die Zahlen 10, 11, ..., 19, 20 auf dem Bildschirm ausgibt.
- c) Geben Sie das Struktogramm eines Programms (mit einer for Anweisung realisiert) an, das die Zahlen 10, 11, ..., 19, 20 auf dem Bildschirm ausgibt.

- 2) (40P)

Definition:

Die Quersumme einer ganzen Zahl ist die Summe seiner Ziffern.

Beispiele:

Die Quersumme von 367 ist:  $3 + 6 + 7 = 16$

Die Quersumme von 1996 ist:  $1 + 9 + 9 + 6 = 25$

Die Quersumme von -13 ist: 4 (Quersumme ist immer  $\geq 0$ )

- a) Erstellen Sie ein **Struktogramm**, das von einer ganzen Zahl (Datentyp integer) die Quersumme berechnet.

Beachten Sie:

Die Quersumme ist immer  $\geq 0$

- b) Machen Sie dazu verschiedene Tests (dokumentierte Protokolle mit Testdaten). Geben Sie jeweils an, ob die Tests erfolgreich waren oder nicht.

*Name, Vorname:*

Hilfsmittel:  
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form auf jeder Seite unten rechts durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Programme dürfen beim Kompilieren keine Warnungen und keine Fehlermeldungen bringen.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

Bemerkung:

1) For-Schleifen sind bei allen Aufgaben **nicht** erlaubt. Bei Benutzung einer For-Schleife gibt es 0 Punkte.

2) Bei Feldern Konstante benutzen

1) Geben Sie die syntaktischen Fehler und die Fehler, die zur Laufzeit entstehen können, an.

8P

```
#include "stdafx.h"
#include <stdio.h>

int main() {
    int i = 3;
    int v[3]={10,9,15,8};
    double w[i];

    v[2] = 10;
    w[i+1]=v[3];
    printf("r=%d\n",v[i-4]);
    return 0;
}
```

2) In einem Feld sollen die Mathenoten einer Schulklasse abgespeichert werden. 10P

a) Schreiben Sie ein C-Programm, in dem der Anwender die Noten über Tastatur eingibt und diese im kompletten Feld abgespeichert werden (das komplette Feld "ausfüllen")

b) Ein Anwender, kann bei der Eingabe einen Fehler machen, indem er z.B. keine Note eingibt, sondern eine Zahl wie z.B. 6.5 oder -3. 14P

Ergänzen Sie das obige, angefangene Programm so, daß es in alle Zellen des Feldes, in denen keine Note eingegeben wurde, eine 0 schreibt.

c) Ergänzen Sie das obige, angefangene Programm so, daß es den Mittelwert berechnet. (Existiert der immer?) 18P

## Lösungen:

1)

```
int main(){
    int i = 3;
    int v[3]={10,9,15,8};    // zu viele Elemente
    double w[i];             // i muss Konstante sein

    v[2] = 10;
    w[i+1]=v[3];             // v[3] nicht reservierter Speicher
    printf("r=%d\n",v[i-4]); // v[i-4] nicht reservierter Speicher
    return 0;
}
```

2)

```
#include "stdafx.h"
#include <stdio.h>

const int LEN = 4;
int main(){
    int i;
    double noten[LEN];
    double mittelwert;
    double summe;
    int anzahl;

    // Eingabeteil
    i=0;
    printf("Bitte geben sie die Noten ein\n");
    do {
        scanf("%lf",&noten[i]);
        i++;
    }while(i<LEN);

    // Markieren der falschen Noten mit 0
    i=0;
    do {
        if(noten[i]<1 || noten[i]>6){
            noten[i]=0;
        }
        i++;
    }while(i<LEN);

    // Mittelwert berechnen
    i=0;
    anzahl=0;
    summe=0;
    do {
        if(noten[i]!=0){
            summe = summe + noten[i];
            anzahl++;
        }
        i++;
    }while(i<LEN);
```

```
    if(anzahl>0){
        mittelwert = summe / anzahl;
    }
    else{
        mittelwert = -1;
    }

    // Ausgabeteil
    printf("Mittelwert= %f\n", mittelwert);
    i=0;
    do {
        printf("%f ", noten[i]);
        i=i+1;
    }
    while(i<LEN);

    return 0;
}
```

## KLAUSUR 2 Programmiertheorie 2BKI1 Nachtermin 1 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:

Tabelle mit Prioritäten und Assoziationen

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form auf jeder Seite unten rechts durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Programme dürfen beim Kompilieren keine Warnungen und keine Fehlermeldungen bringen.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

### AUFGABEN

2) In einem Feld sollen die Mathenoten einer Schulklasse abgespeichert werden.

Die letzte Note (muß nicht notwendig das letzte Element des Feldes sein) erkennt man daran, dass die darauffolgende Zelle den Wert -1 hat.

Beispiel: Mittelwert ist 3.5

3.5	4.5	2.5	3.5	-1			
-----	-----	-----	-----	----	--	--	--

Voraussetzungen:

v1) Der Anwender gibt Noten ein (also Zahlen zwischen 1 und 6), außer er will die Eingabe beenden, dann gibt er -1 ein.

v2) Der Anwender gibt mindestens eine Note ein.

v3) Das Feld hat mindestens die Länge LEN = 5

a) Schreiben Sie ein C-Programm, in dem der Anwender die Noten über Tastatur eingibt und diese - wie oben beschrieben - abgespeichert werden.

b) Ergänzen Sie das obige Programm, so dass es den Mittelwert berechnet und in der 0. Zelle abspeichert.

Der Mittelwert (Durchschnitt) soll in der 0-ten Zelle abgespeichert werden.

Es müssen die Umsätze (größer oder gleich Null) einer Firma in ein Feld eingelesen werden.

Die Umsätze müssen über Tastatur eingegeben werden. Wenn für einen Umsatz ein Wert kleiner als Null eingegeben wird, wird die Eingabe beendet. Dann wird der Gesamtumsatz (die Summe dieser Umsätze) berechnet und auf dem Bildschirm ausgegeben.

Erstellen Sie dazu ein **Struktogramm**, kein C-Programm !!

Bemerkung:

Es dürfen keine nicht reservierten Zellen eines Feldes überschrieben werden.

# KLAUSUR 3 Programmiertheorie 2BKI1 Nachtermin 1 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:  
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Programme dürfen beim Kompilieren keine Warnungen und keine Fehlermeldungen bringen.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

1) Was ist ein Pointer ?

2P

2)

10P

```
...  
float e = 3.1;  
float f = 2.7;  
float *p;  
p = &e;  
*p = f + *p;
```

	Adresse	Wert
e	01350	
f	02460	
p	03570	

Welchen Wert hat e und p nachdem alle die obigen Anweisungen ausgeführt wurden?  
Tragen Sie die Werte in die Tabelle ein.

3) In einem C-Programm wurde folgende Variable deklariert:

6P

```
char w[4];
```

In dem Programm stehen genau die folgenden Anweisungen:

```
w[0] = 'w';  
w[1] = 'i';  
w[2] = 'r';
```

Es soll ein Programm(ausschnitt) geschrieben werden, in dem der Inhalt der Variablen w (also die Zeichenkette "wir") ausgegeben werden soll.

- Realisieren Sie dies durch die Verwendung der Formatierung "%s" in printf
- Realisieren Sie dies durch die Verwendung der Formatierung "%c" in printf

4) Wieviel Speicherplatz (in Bytes) benötigen die Variablen v1, v2, v3 des folgenden Programmausschnitts ?

3P

```
int v1[10];  
int v2[10][10];  
int v3[10][10][10];
```

5)

14P

Ein Text ist in einem zweidimensionalen Feld `ff` (Quadrat) abgespeichert. Um ihn vor neugierigen Blicken zu schützen wird er so codiert, dass die 1. Zeile mit der 1. Spalte, die 2. Zeile mit der 2. Spalte, usw. vertauscht werden.

Realisieren Sie das mit Hilfe eines Struktogramms.

Zeilenanzahl: `LEN`

Spaltenanzahl: `LEN`

(Als Zeilenanzahl und Spaltenanzahl keine feste Zahl wählen!!)

Tipp: Überlegen Sie sich, wo das Element `ff[i][j]` nach seiner Vertauschung steht!

6)

5P

Gegeben ist ein zweidimensionales Feld `ff` mit der Zeilenanzahl `ANZZ` und der Spaltenanzahl `ANZS`.

Die 1. Zeile dieses zweidimensionalen Feld `ff` wird in das eindimensionale Feld `f` (am Anfang des Feldes `f` beginnend) kopiert.

Gleich dahinter wird die 2. Zeile des zweidimensionalen Feldes `ff` angefügt.

Gleich dahinter wird die 3. Zeile des zweidimensionalen Feldes `ff` angefügt, usw.

a) Welche Feldlänge hat `f` ?

b) Das zweidimensionale Feld `ff` wird zerstört.

Durch welchen Zugriff kann man jetzt im eindimensionalen Feld `f` auf ein Element des Feldes `ff` zugreifen, das vor der Zerstörung an der Stelle `ff[i][j]` abgespeichert wurde ?

Bemerkungen:

In den obigen Aufgaben ist unter *i-ter Zeile* bzw. *i-ter Spalte* natürlich **programmtechnisch** die *i-1-te Zeile* bzw. *i-1-te Spalte* gemeint.

Das heißt zum Beispiel, dass mit 1. Zeile bzw. 1. Spalte programmtechnisch die 0. Zeile bzw. 0. Spalte gemeint ist

## Lösungen

1) Ein Pointer ist eine Variable, deren Wert die Adresse eines Speicherplatzes (z.B. einer Variable) ist.

2)

```
...  
float e = 3.1;  
float f = 2.7;  
float *p;  
p = &e;  
*p = f + *p;
```

	Adresse	Wert
e	01350	5.8
f	02460	2.7
p	03570	01350

3) a)

```
w[3] = '\\0';  
printf("%s", w);
```

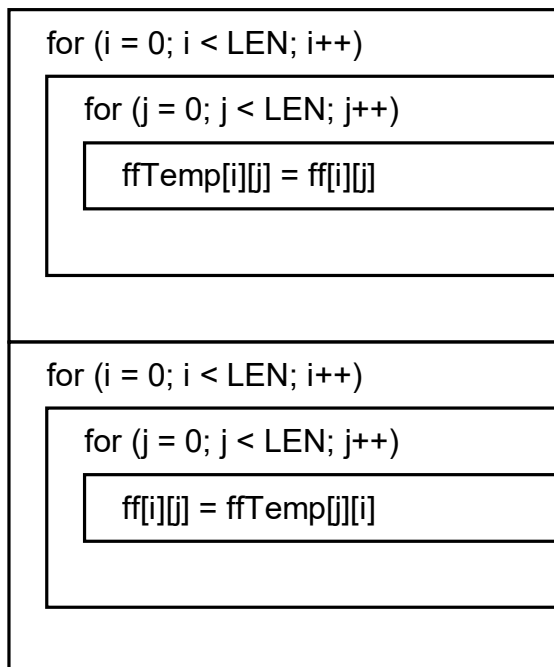
b)

```
for(i=0; i<3; i++)  
    printf("%c", w[i]);
```

4)

Speicherplatzverbrauch von v1 = 10 \* Speicherbedarf(int);  
Speicherplatzverbrauch von v2 = 100 \* Speicherbedarf(int);  
Speicherplatzverbrauch von v3 = 1000 \* Speicherbedarf(int);

5)





6)

a) (2P)

Feldlänge  $f = \text{ANZZ} * \text{ANZS}$

b) (3P)

$\text{ff}[i * \text{ANZS} + j] = \text{ff}[i][j]$

**KLAUSUR 3 Programmierpraktikum 2BK11 8.6.2010 Zeit: 45 Minuten**  
**Gruppe A Name, Vorname:**

Hilfsmittel:  
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punktabzug !!!!
- Bei praktischen Arbeiten am Computer bitte folgendes beachten:
- NUR **ablauffähige** C-Programme werden bewertet ! **Jede** Aufgabe als eigenes Projekt (keine Warnungen vom Compiler).
- Der Name, Vorname und die Aufgabennummer (z.B: Aufgabe 1) muß als **Kommentar** in jedes Programm mitaufgenommen werden.
- Name der Quelldatei nach folgendem Schema vergeben: Gruppe\_Rechnernummer\_Nachname\_Vorname\_Aufgabennr.  
Beispiel: A\_12\_Mustermann\_Erika\_nr3
- Die Programme müssen ausgedruckt werden.
- Die entsprechenden Quelldateien müssen auf Diskette abgespeichert und abgegeben werden.

Bemerkungen:

Der Compiler darf keine Warnungen und Fehler erzeugen !!!

## AUFGABEN

1)a) Programmieren Sie anhand der folgenden Beschreibung die dazugehörige Funktion. **40 P**

b) Im Hauptprogramm muss der Anwender eine Basis und einen Exponenten eingeben. **10 P**  
können. Das Programm berechnet dann mit Hilfe der Funktion power(...) das Ergebnis und gibt es auf dem Bildschirm aus.

```
/*  
**  
**  int power(double basis, int exponent, double *erg)  **  
**  
*#*****  
*/
```

Parameter:

- (i) double basis: Grundzahl (Basis)
- (i) int exponent: Hochzahl (Exponent)
- (o) double \*erg : basis ^ exponent

Return:

- 1: 0^0
- 1: sonst

Beschreibung:

Berechnet: basis hoch exponent, also basis^exponent  
wobei 0^0 nicht definiert ist (return: -1).

Beispiel:

```
erg = pow(2, -4);  --> erg = 2^-4 = 1/16  
erg = pow(-2, 3); --> erg = (-2)^3 = -8  
erg = pow(0, 0);  --> erg undefiniert --> return -1  
*/
```

**KLAUSUR 3 Programmierpraktikum 2BK11 8.6.2010 Zeit: 45 Minuten**  
**Gruppe B** *Name, Vorname:*

Hilfsmittel:  
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!
- Bei praktischen Arbeiten am Computer bitte folgendes beachten:
- NUR **ablauffähige** C-Programme werden bewertet ! **Jede** Aufgabe als eigenes Projekt (keine Warnungen vom Compiler).
- Der Name, Vorname und die Aufgabennummer (z.B: Aufgabe 1) muß als **Kommentar** in jedes Programm mitaufgenommen werden.
- Name der Quelldatei nach folgendem Schema vergeben: Gruppe\_Rechnernummer\_Nachname\_Vorname\_Aufgabennr.  
Beispiel: A\_12\_Mustermann\_Erika\_nr3
- Die Programme müssen ausgedruckt werden.
- Die entsprechenden Quelldateien müssen auf Diskette abgespeichert und abgegeben werden.

Bemerkungen:

Der Compiler darf keine Warnungen und Fehler erzeugen !!!

## AUFGABEN

1) Eine natürliche Zahl  $n$  ( $n \geq 2$ ) heißt Primzahl, wenn sie nur durch 1 und sich selbst teilbar ist. Beispiele für Primzahlen: 2, 3, 5, 7, 11, 13, ....

Schreiben Sie ein Programm, das folgendes macht:

a) In dem Programm muss sich eine Funktion `istPrimzahl(...)` befinden, die von einer ganzen Zahl  $\geq 2$  feststellt, ob diese eine Primzahl ist.

b) In dem Hauptprogramm sollen mit Hilfe der Funktion `istPrimzahl(...)` die ersten 100 Primzahlen ausgegeben werden.

# KLAUSUR 4 Programmiertheorie 2BKI1 8.7.2010 Zeit: 60 Minuten

Name, Vorname:

Hilfsmittel:

Keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

1)

12P

Betrachten Sie den folgenden Programmausschnitt:

```
...
int main() {
    int zahl = 4;
    int quad = 8;
    → 1
    quadrat(zahl, &quad);
    → 2
    printf("%d hoch 2 = %d\n", zahl, quad);
    return 0;
}

void quadrat(int z, int *zq) {
    *zq = z * z;
}
```

Durch die Deklaration der Variablen zahl und quad werden die folgenden Zellen

	Adresse	Inhalt
zahl	08151	?
quad	04711	?

im Arbeitsspeicher reserviert.

- Welchen Wert hat der Inhalt der Variablen zahl und quad an der Stelle 1 im Programm ?
- Welchen Wert hat z und zq beim Aufruf von quadrat(zahl, &quad) ?
- Was bewirkt die Anweisung `*zq = z * z` an welcher Adresse im Arbeitsspeicher ?  
(konkreten Wert der Adresse und deren Inhalt angeben!)
- Welchen Wert hat der Inhalt der Variablen zahl und quad an der Stelle 2 im Programm ?

2)

13P

Sie wollen ein Programm schreiben, das abhängig von der Eingabe entweder die Summe oder die Differenz oder das Produkt oder den Quotienten zweier Zahlen liefert.

Sie wollen dazu die Funktion `tr` (wie Taschenrechner) benutzen, die Sie aber wegen Arbeitsüberlastung von einem "Programmierknecht" implementieren (programmieren) lassen. Entwerfen Sie für den "Programmierknecht" eine **Beschreibung** (Leistungsbeschreibung) dieser Funktion mit dem im Unterricht verwendeten Schema. (Kein Programm !!!)

3)

12P

Betrachten Sie den folgenden Programmausschnitt:

```
...
int main() {
    float r;
    float u;
    r = 2;
    u = 3;
    → 1
    berechne_umfang (r, u);
    → 2
    printf("Radius= %f, Umfang= %f", r, u);
}

void berechne_umfang(float radius, float umfang) {
    umfang = 2 * 3.14 * radius ;
}
```

Durch die Deklaration der Variablen `r` und `u` werden die folgenden Zellen

	Adresse	Inhalt
<code>r</code>	0120	?
<code>u</code>	0130	?

im Arbeitsspeicher reserviert.

a) Welchen Wert hat der Inhalt der Variablen `r` und `u` an der Stelle 1 im Programm ?

b) Welchen Wert hat `radius` und `umfang` beim Aufruf von `berechne_umfang (r, u)` ?

c) Was bewirkt die folgende Anweisung im Arbeitsspeicher an der Adresse 0130 ?

`umfang = 2 * 3.14 * radius;`

d) Welchen Wert hat der Inhalt der Variablen `r` und `u` an der Stelle 2 im Programm ?

4)

13P

Programmieren Sie anhand der folgenden Beschreibung die dazugehörige Funktion:

```

/*****
/**
/**  int ersatz(double r1, double r2, int mod, double *rg)  **/
/**
/**
/**
/*#*****/
/*

```

Parameter:

```

(i) double r1>0:    erster Widerstandswert
(i) double r2>0:    zweiter Widerstandswert
(i) int mod:         10: Parallelschaltung
                    20: Reihenschaltung
(o) double *rg:     Gesamtwiderstand

```

Return:

```

(o) 0: Parallelschaltung oder Reihenschaltung wurde
      berechnet (mod ist 10 oder 20)
-1: mod ist weder 10 noch 20

```

Beschreibung:

Berechnet in Abhängigkeit vom Modus mod (10 bedeutet eine Parallelschaltung, 20 bedeutet eine Reihenschaltung), den Widerstandswerten r1 und r2 den Ersatzwiderstand (Gesamtwiderstand) rg der Widerstandsschaltung.

\*/

## Lösungen

- 1) 12P  
a) zahl: 4, quad: 8 1P + 1P  
b) z: 4, zq: 04711 1P + 2P  
c) In den Inhalt der Adresse 04711 wird der Wert 16 geschrieben. 4P  
d) zahl: 4, quad: 16 1P + 2P

2) 13P  
/\*\*\*\*\*/  
/\*\* \*\*/  
/\*\* double tr (double z1, double z2, int mod) \*\*/  
/\*\* \*\*/  
/\*#\*\*\*\*\*/

Parameter:

- (i) double z1: erste Zahl
- (i) double z2!=0, wenn mod=4: zweite Zahl
- (i) int mode{1;2;3;4}:
  - 1: berechnet Summe z1+z2
  - 2: berechnet Differenz z1-z2
  - 3: berechnet Produkt z1\*z2
  - 4: berechnet Quotient z1/z2

Return:

- (o) Ergebnis der gewünschten Operation (Summe, oder Differenz oder Produkt oder Quotient).

Beschreibung:

Berechnet in Abhängigkeit vom Modus mod:

Summe z1+z2, wenn mod = 1

Differenz z1-z2, wenn mod = 2

Produkt z1\*z2, wenn mod = 3

Quotient z1/z2, wenn mod = 4

\*/

Jede fehlende Zusicherung: -2 P

z2!=0 ist keine richtige Zusicherung, da dann z.B. auch 3 \* 0 verboten würde: -2 P

Bemerkung zum Begriff Zusicherung:

Die Angabe beim Parameter z1:

z2!=0, wenn mod=4

und die Angabe:

mode{1;2;3;4}

nennt man Zusicherung. Das bedeutet, daß unter diesen Voraussetzungen der Programmierer dieser Funktion für die Korrektheit der Berechnungen dieser Funktion **garantiert**.

Je weniger Zusicherungen der Programmierer macht, desto größer wird der programmtechnische Aufwand für ihn, desto mehr "Intelligenz" muss er in die Funktion packen.

- 3) 12P  
a)  $r = 2, u = 3$  1P + 1P  
b) radius = 2, umfang = 3 1P + 2P  
c) nichts 4P  
d)  $r = 2, u = 3$  3P

4) 13P  

```
int ersatz(double r1, double r2, int mod, double *rg){
    int r;
    if(mod==10){
        *rg=1/(1/r1+1/r2);
        r=0;
    }
    else if(mod==20){
        *rg=r1+r2;
        r=0;
    }
    else
        r=-1;
    return(r);
}
```



# KLAUSUR 4 Programmiertheorie 2BKI1 Nachtermin 1 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:

Keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

1) Es soll eine Zahlenfolge kodiert bzw. dekodiert werden.

Die Kodierung geschieht dadurch, dass zu jeder ganzen Zahl einer ganzzahligen Zahlenfolge eine bestimmte, konstante ganze Zahl dazu addiert wird.

Beispiel:

10, 7, -23, 19 ---+3---> 13, 10, -20, 22

Sie sollen dazu eine möglichst "luxuriös" gestaltete **Funktion** benutzen, die Sie aber wegen Arbeitsüberlastung im Fach Mathematik von einem "Programmierknecht" implementieren (programmieren) lassen.

a) Entwerfen Sie für den "Programmierknecht" eine **Beschreibung** (Leistungsbeschreibung) der Funktion "kodieren" mit dem im Unterricht verwendeten Schema. (Kein Programm !!!)  
Überlegen Sie sich zuerst genau, welche **Parameter** nötig sind

b) Da sich der Programmierknecht zur Zeit in einem sogenannten "Tschill-Urlaub" befindet und deshalb aktuell (und wohl auch zukünftig) nicht ansprechbar ist, muss die Funktion "kodieren" von Ihnen selbst implementiert werden. Implementieren Sie die Funktion "kodieren".

c) Schreiben Sie ein Programm mit einem Aufruf der Funktion "kodieren" (keine Eingabe über scanf(), sondern Aufruf mit konkreten Parametern).

d) Rufen Sie die Funktion "kodieren" (mit geeigneten Parametern) direkt nach dem Aufruf in c) nochmals so auf, dass die veränderte Zahlenfolge wieder ihre ursprünglichen Werte bekommt (dekodieren).

## Lösungen:

1a)

```
/* **** */
/**                                     **/
/**  kodieren (int zahlen[], int anzahl, int wert)  **/
/**                                     **/
/*# **** */
/*
```

Parameter:

(i/o) int zahlen[]: zu kodierende Zahlenenfolge  
(i) int anzahl: Anzahl der Zahlen der Zahlenenfolge  
(i) int wert: Zahl, die zu der Zahlenfolge dazuaddiert wird

Return:

kein

Beschreibung:

Kodiert anzahl Zahlen einer Zahlenfolge, indem ein bestimmter Wert dazuaddiert wird.

Beispiel:

```
zahlen[5]: {1, 2, 3, 5, 6}
anzahl:    3
wert:      100
Nach dem Aufruf:
zahlen[5]: {101, 202, 203, 5, 6}
*/
```

b)

```
#include "stdafx.h"
```

```
void kodieren(int zahlen[], int anzahl, int wert);
```

```
int main(int argc, char* argv[]){
```

```
    int i;
    int zahlen[5]={1, 2, 3, 5, 6};
    kodieren(zahlen, 4, 100);
    for(i=0; i<4; i++){
        printf("%d ", zahlen[i]);
    }
```

```
    kodieren(zahlen, 4, -100);
    printf("\n");
    for(i=0; i<4; i++){
        printf("%d ", zahlen[i]);
    }
    return 0;
}
```

```
void kodieren(int zahlen[], int anzahl, int wert){
```

```
    int i;

    for(i=0; i<anzahl; i++){
        zahlen[i]=zahlen[i]+wert;
    }
}
```

# KLAUSUR 3 Programmierpraxis 2BKI1 Nachtermin 1 Zeit: 60 Minuten

Name, Vorname:

Hilfsmittel:

keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß auf jedes Aufgabenblatt und auf jedes Lösungsblatt geschrieben werden.
- Aufgabenblätter bitte auch abgeben.
- Die Lösungsblätter müssen in folgender Form durchnummeriert werden. Beispiel: 1/4 2/4 3/4 4/4
- Die rote Farbe darf nicht benutzt werden.
- Lassen Sie bitte auf der linken Seite einen mindestens 3cm breiten Rand.
- Selbsterklärende Variablennamen benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programmteile.
- Programme dürfen beim Kompilieren keine Warnungen und keine Fehlermeldungen bringen.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punkteabzug !!!!

## AUFGABEN

Bearbeiten Sie genau **EINE** der zwei folgenden Aufgaben (Es wird nur eine Aufgabe gewertet):

1)

Ein Glied  $a(i)$  einer Folge berechnet sich aus der Summe der beiden vorhergehenden Folgenglieder  $a(i-1)$  und  $a(i-2)$ :  $a(i) = a(i-1) + a(i-2)$

Die beiden Anfangsglieder der Folge sind z.B.

$$a(1) = 5$$

$$a(2) = 3$$

Das ergibt die Folge:

$$a(1) = 5$$

$$a(2) = 3$$

$$a(3) = a(2) + a(1) = 3 + 5 = 8$$

$$a(4) = a(3) + a(2) = 8 + 3 = 11$$

$$a(5) = a(4) + a(3) = 11 + 8 = 19$$

...

also:

5, 3, 8, 11, 19, ...

Erstellen Sie eine Funktion, die in Abhängigkeit von den 2 Anfangswerten und  $n$  das Folgendglied  $a(n)$  berechnet.

2)

a) Die vier Zahlen 13, 7, 19, 3 müssen hintereinander in einer verketteten Liste abgespeichert werden.

b) Geben Sie diese Zahlen hintereinander auf dem Bildschirm (mit Hilfe einer Schleife) aus.