

Name, Vorname:

Hilfsmittel:
Prioritätentabelle

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT 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.
- return genau einmal am Ende des Quellcodes einer jeder Funktion (u.a. auch main()) benutzen.
- Programme müssen benutzerfreundlich sein.
- EVA-Prinzip muss benutzt werden.
- Einrücken der entsprechenden Programnteile.
- 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 mit aufgenommen werden.
- Name der Projekte nach folgendem Schema vergeben: Gruppe_Rechnernummer_Nachname_Vorname_Aufgabennr.
Beispiel: A_12_Mustermann_Erika_nr3
- Die Programme müssen ausgedruckt werden.
- Jeder Programnteil (Eingabe, Verarbeitung, Ausgabe) muss als solcher durch einen Kommentar angegeben werden.
- Jedes Projekt (d.h. jeder Projektordner) muß auf den Lehrerstick kopiert werden.

AUFGABEN

- 1) (4 P)
Was ist ein Algorithmus und welche Eigenschaften hat er ?
- 2) (2 P)
Was ist ein Literal ?
- 3) (2 P)
Was bedeutet Priorität in der Programmiersprache C ?
- 4) (2 P)
Was bedeutet Assoziativität in der Programmiersprache C ?
- 5) (2 P)
Kann die Ganzzahl 123456 in einer zwei Bytes grossen integer-Zahl abgespeichert werden ?
Begründen Sie!
- 6) Das Volumen einer Kugel berechnet sich wie folgt: (10 P)
$$V = \frac{4}{3} \cdot \pi \cdot r^3$$

Schreiben Sie das dazu passende C-Programm. Das Volumen muss in einer Variablen mit dem Datentyp float gespeichert werden. Die Konstante pi muss in einer Variablen abgespeichert werden.
- 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)

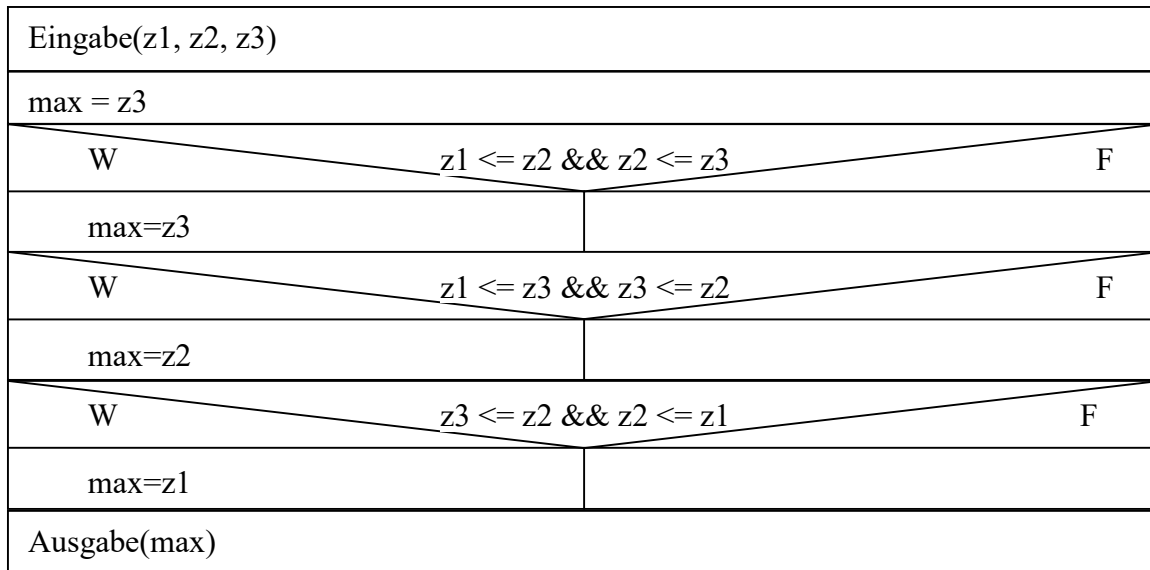
(9P)

Das folgende Struktogramm soll die größte Zahl (dreier Zahlen), also das Maximum max, berechnen und auf dem Bildschirm ausgeben.

Wenn das Programm korrekt ist, schreiben Sie "Der Algorithmus ist korrekt" und geben 5 Beispiele (mit konkreten Werten für z1, z2, z3 und dem berechneten Wert von max) an, wo er korrekt wird.

Wenn das Programm nicht korrekt ist, schreiben Sie "Der Algorithmus ist nicht korrekt" und geben ein **konkretes** Beispiel (mit konkreten Werten für z1, z2, z3 und dem berechneten Wert von max) an, bei dem der Algorithmus falsch wird.

Geben Sie dazu genau an, wie der Fluß (Wert der Variablen in das Programm eintragen) durch das Programm verläuft (einzeichnen).



9)

(12P)

Durch den folgenden Algorithmus (Flußdiagramm) soll die Summe $1 + 2 + 3 + \dots + 100$ berechnet werden.

Dazu wird das Programm immer wieder an der mit <--- bezeichneten Stelle vor der Verzweigung (bei jedem Schleifendurchgang) gedanklich angehalten (Protokoll) und die Werte der Variablen i und sum protokolliert.

a) Tragen Sie dazu in der Tabelle unten die Werte von i und sum bei den ersten 5 Durchgängen ein. Bitte sum **nicht** berechnen, sondern die Summe jeweils darstellen, wie z.B. $7 + 9 + 11$.

b) Welche Werte haben i und sum, wenn das Programm das **letzte** Mal an die mit <--- bezeichnete Stelle kommt?

c) Ist das Programm semantisch korrekt (d.h. wird also durch das Programm die Summe $1 + 2 + 3 + \dots + 100$ berechnet). Bitte Begründung angeben!

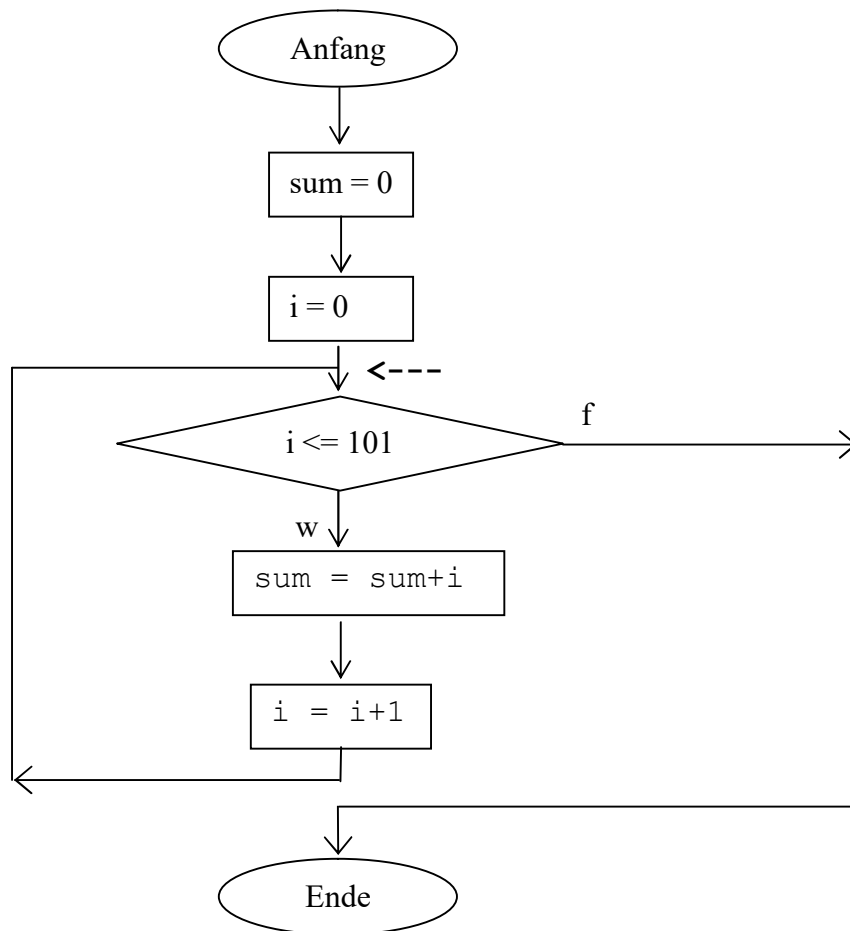


Tabelle (Protokoll):

i							
sum							

Lösung:

1) (4 P)
Ein Algorithmus ist ein Verfahren zur Lösung eines gegebenen Problems. Er ist eindeutig, endlich und schrittweise.

2) (2 P)
Literale sind Bezeichner mit einem festen Wert wie z.B. 9

3) (2 P)
Priorität gibt den Vorrang eines Operators in einem Ausdruck an.
Beispiel: Punkt vor Strich.

4) (2 P)
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.

5) (2 P)
2 Bytes kann $2^{16} \approx 64000$ verschiedene Zustände speichern.
Damit können Ganzzahlen von ungefähr -32000 bis ungefähr 32000 abgespeichert werden.
Also kann 123456 in dieser zwei Bytes grossen Integer-Zahl nicht gespeichert werden.

6) (10 P)
`#include "stdafx.h"`
`#include <stdio.h>`

```
int main(){
    float radius;
    float volumen;
    float pi;

    // Eingabeteil
    printf("Berechnung des Volumens einer Kugel\n");
    printf("Bitte geben sie den Radius ein\n");
    scanf("%lf",&radius);
    fflush(stdin);

    // Verarbeitungsteil
    pi = (float) 3.14;
    volumen = (float)4.0/(float)3.0 *pi *radius*radius*radius;

    //Ausgabeteil
    printf("Kugelvolumen= %f\n",volumen);

    return 0;
}
```

7) (8 P)

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) (9 P)

$z1 = 3 \quad z2 = 1 \quad z3 = 2$

9) (12 P)

a)

i	0	1	2	3	4	...	102
sum	0	0	1	1+2	1+2+3	...	1+...+101

b)

$i = 102$ und $sum = 1+...+101$

c) Da nach das Programm an dieser Stelle beendet wird, haben i und sum die bei b) protokollierten Werte. Das Programm ist also nicht korrekt.

KLAUSUR 1 Programmierpraktikum 2BK11 12.12.2012 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT 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.
- return genau einmal am Ende des Quellcodes einer jeder Funktion (u.a. auch main()) 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 mit aufgenommen werden.
- Name der Projekte nach folgendem Schema vergeben: Gruppe_Rechnernummer_Nachname_Vorname_Aufgabennr.
Beispiel: A_12_Mustermann_Erika_nr3
- Die Programme müssen ausgedruckt werden.
- Jeder Programmteil (Eingabe, Verarbeitung, Ausgabe) muss als solcher durch einen Kommentar angegeben werden.
- Jedes Projekt (d.h. jeder Projektordner) muß auf den Lehrerstick kopiert werden.

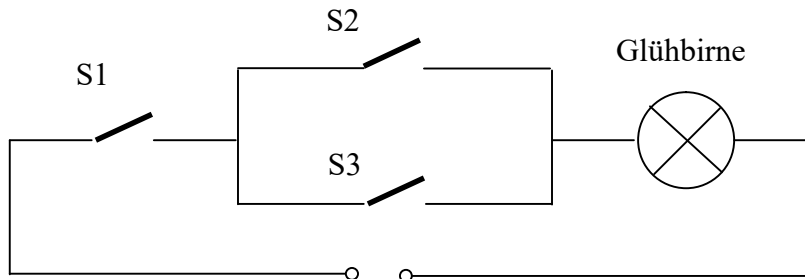
Bemerkungen:

Der Compiler darf keine Warnungen und Fehler erzeugen !!!

AUFGABEN

1) Eine Glühbirne wird mit 3 Schaltern wie folgt verbunden:

25 P



Jeder der 3 Schalter (S1, S2 und S3) ist entweder „aus“ oder „ein“.

Schreiben Sie ein C-Programm, das ausgibt, ob die Glühbirne leuchtet oder dunkel ist.

Die Zustände der Schalter soll über Tastatur eingegeben werden.

Welcher Datentyp für den Zustand (ein / aus) gewählt wird, ist Sache des Programmierers.

Er legt fest, was „ein“ bzw. „aus“ bedeutet und sagt dies dem Anwender durch eine entsprechende Mitteilung auf dem Bildschirm.

(EVA-Prinzip beachten).

2) Es soll über Tastatur eine **ganze** Zahl eingegeben werden.

25 P

Ist diese Zahl eine Schulnote zwischen 1 und 4 (je einschließlich) wird die Meldung "Prüfung bestanden" ausgegeben.

Ist diese Zahl die Schulnote 5 oder 6 wird die Meldung "Prüfung nicht bestanden" ausgegeben.

Sonst wird die Meldung "Diese Zahl ist keine Note" ausgegeben.

Realisieren Sie dies durch ein C-Programm, das genau eine switch- Anweisung (keine if-else- Anweisung, keine if-Anweisung) verwendet.

Lösung:

1)

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[]){
    int schalter1;
    int schalter2;
    int schalter3;
    int zustand;

    // Eingabe
    printf("Schalterstellung (0 / 1) Schalter 1 eingeben\n");
    scanf("%d",&schalter1);
    printf("Schalterstellung (0 / 1) Schalter 2 eingeben\n");
    scanf("%d",&schalter2);
    printf("Schalterstellung (0 / 1) Schalter 3 eingeben\n");
    scanf("%d",&schalter3);

    // Verarbeitung:
    zustand = schalter1 && (schalter2 || schalter3);

    // Ausgabe
    if (zustand == 0){
        printf("Die Lampe ist aus\n");
    }
    else{
        printf("Die Lampe ist ein\n");
    }

    return 0;
}
```

2)

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[]){
    int note;
    printf("Bitte eine ganze Note eingeben\n");
    scanf("%d", &note);

    switch(note){
        case 1:
        case 2:
        case 3:
        case 4:
            printf("Prüfung bestanden \n");
            break;

        case 5:
        case 6:
            printf("Prüfung nicht bestanden \n");
            break;

        default:
            printf("Dies ist keine Note \n");
            break;
    }
    return 0;
}
```

KLAUSUR 1 Programmierpraxis 2BK11 Nachtermin 1 Zeit: 60 Minuten

Name, Vorname:

Hilfsmittel:

keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT 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.
- return genau einmal am Ende des Quellcodes einer jeder Funktion (u.a. auch main()) 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 mit aufgenommen werden.
- Name der Projekte nach folgendem Schema vergeben: Gruppe_Rechnernummer_Nachname_Vorname_Aufgabennr.
Beispiel: A_12_Mustermann_Erika_nr3
- Die Programme müssen ausgedruckt werden.
- Jeder Programmteil (Eingabe, Verarbeitung, Ausgabe) muss als solcher durch einen Kommentar angegeben werden.
- Jedes Projekt (d.h. jeder Projektordner) muß auf den Lehrerstick kopiert werden.

AUFGABEN

1)

50 P

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. Jeden Teil entsprechend kommentieren mit: // Eingabe bzw. // Verarbeitung bzw. //Ausgabe

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.

Lösungen:

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

int main(){
    double zahl1, zahl2;
    double ergebnis;
    char zeichen;
    int zustand;
    // -1 : Programm beenden, -2 : Division durch 0
    // 0 : alles okay

    // E I N G A B E T E I L
    printf("Taschenrechner\n");
    printf("Addieren: A oder a eingeben \n");
    printf("Subtrahieren: S oder s eingeben \n");
    printf("Multiplikizieren: M oder m eingeben \n");
    printf("Dividieren: D oder d eingeben \n");
    printf("Programmende: irgendein anderes Zeichen eingeben \n");
    scanf("%c", &zeichen);

    switch(zeichen){
        case 'A':
        case 'a':
        case 'S':
        case 's':
        case 'M':
        case 'm':
        case 'D':
        case 'd':
            zustand = 0;
            break;
        default:
            zustand = -1;
    }

    if(zustand==0){
        printf("Bitte Zahl1 eingeben\n");
        scanf("%lf",&zahl1);
        fflush(stdin);

        printf("Bitte Zahl2 eingeben\n");
        scanf("%lf",&zahl2);
        fflush(stdin);
    }

    // V E R A R B E I T U N G S T E I L
    if(zustand == 0){
        if(zeichen=='A' || zeichen=='a'){
            ergebnis = zahl1 + zahl2;
        }

        else if(zeichen=='S' || zeichen=='s'){
            ergebnis = zahl1 - zahl2;
        }

        else if(zeichen=='M' || zeichen=='m'){
            ergebnis = zahl1 * zahl2;
        }

        else { // Division
            if(zahl2!=0){
                ergebnis = zahl1 / zahl2;
            }
            else{
                zustand = -2;
            }
        }
    }
}
```

```
        // A U S G A B E T E I L
        if(zustand==0){
            printf("Ergebnis=%f", ergebnis);
        }
        else if(zustand==-1){
            printf("Programmende\n");
        }
        else{ //Division durch 0
            printf("Durch 0 darf nicht dividiert werden\n");
        }
    }
    return 0;
}
```

Name, Vorname:

Hilfsmittel:
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT 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.
- return genau einmal am Ende des Quellcodes einer jeder Funktion (u.a. auch main()) 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 mit aufgenommen werden.
- Name der Projekte nach folgendem Schema vergeben: Gruppe_Rechnernummer_Nachname_Vorname_Aufgabennr.
Beispiel: A_12_Mustermann_Erika_nr3
- Die Programme müssen ausgedruckt werden.
- Jeder Programmteil (Eingabe, Verarbeitung, Ausgabe) muss als solcher durch einen Kommentar angegeben werden.
- Jedes Projekt (d.h. jeder Projektordner) muß auf den Lehrerstick kopiert werden.

AUFGABEN

1) 3P
Was ist eine Endlosschleife?

2) 8P
a) Was ist allgemein der Hauptunterschied (bzgl. der Anzahl der Durchgänge) zwischen einer while-Anweisung und einer do-while Anweisung ?
b) Geben Sie dazu jeweils (mitAnzahl der Durchgänge) ein **konkretes** Beispiel (Programmausschnitt) an, wobei der Schleifenkörper der gleiche sein muß.

3) 9P
Wie oft wird in den folgenden Programmausschnitten jeweils die Meldung "Hallo Welt" auf dem Bildschirm ausgegeben ?

a)

```
x = 2;
while (x<=2) {
    printf("Hallo Welt\n");
}
```

b)

```
do{
    x=3;
    printf("Hallo Welt\n");
}while (3<x);
```

c)

```
while (5>5) {
    printf("Hallo Welt\n");
}
```

4) 10P
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.

5) Gegeben ist der folgende syntaktisch korrekte Programmteil:

8P

```
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 !

6)

13P

Durch den folgenden Algorithmus (Flußdiagramm) soll die Summe $1 + 2 + 3 + \dots + 100$ berechnet werden.

Dazu wird das Programm immer wieder an der mit <--- bezeichneten Stelle vor der Verzweigung (bei jedem Schleifendurchgang) gedanklich angehalten (Protokoll) und die Werte der Variablen i und sum protokolliert.

a) Tragen Sie dazu in der Tabelle unten die Werte von i und sum bei den ersten 5 Durchgängen ein.

b) Welche Werte haben i und sum, wenn das Programm das **letzte** Mal an die mit <--- bezeichnete Stelle kommt?

c) Ist das Programm semantisch korrekt (d.h. wird also durch das Programm die Summe $1 + 2 + 3 + \dots + 100$ berechnet). Bitte Begründung angeben!

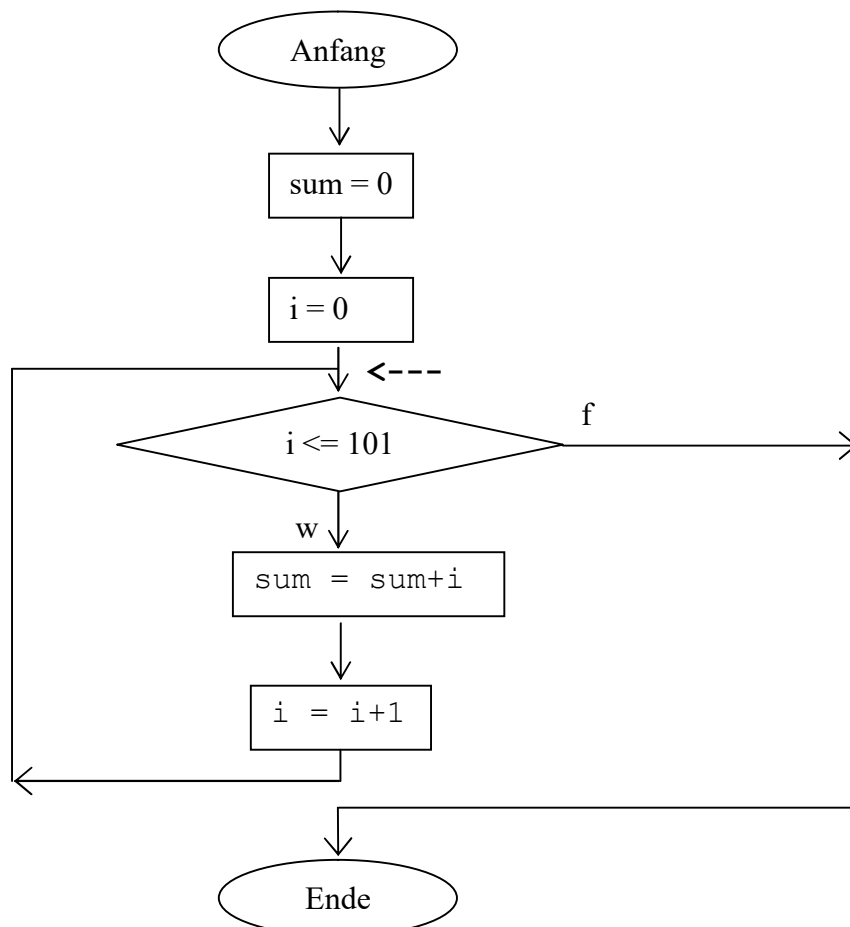


Tabelle (Protokoll):

i							
sum							

Lösungen:

1) 3P
Eine Endlosschleife ist eine Schleife, die nie verlassen wird.

2) 8P

while-Anweisung: 0 - 00

do-while Anweisung: 1 - 00

```
while (1<1) {  
    x=5;  
}
```

0 Durchgänge

```
do {  
    x=5;  
}while (1<1)
```

1 Durchgang

3) 9P

00, 1, 0

4) 10P

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

5) 8P

Ein Mal, weil

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

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

6) 13P

a)

i	0	1	2	3	4	...	102
sum	0	0	1	1+2	1+2+3	...	1+...+101

b)

i = 102 und sum = 1+...+101

c) Da nach das Programm an dieser Stelle beendet wird, haben i und sum die bei b) protokollierten Werte. Das Programm ist also nicht korrekt.

KLAUSUR 2 Programmiertheorie 2BKI1 Nachtermin 1 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:

keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT 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.
- return genau einmal am Ende des Quellcodes einer jeder Funktion (u.a. auch main()) 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 mit aufgenommen werden.
- Name der Projekte nach folgendem Schema vergeben: Gruppe_Rechnernummer_Nachname_Vorname_Aufgabennr.
Beispiel: A_12_Mustermann_Erika_nr3
- Die Programme müssen ausgedruckt werden.
- Jeder Programmteil (Eingabe, Verarbeitung, Ausgabe) muss als solcher durch einen Kommentar angegeben werden.
- Jedes Projekt (d.h. jeder Projektordner) muß auf den Lehrerstick kopiert werden.

AUFGABEN

1)

50P

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 C-Programm, das die ersten 100 Primzahlen auf dem Bildschirm ausgibt.

EVA muß nicht eingehalten werden.

Name, Vorname:

Hilfsmittel:
Prioritätentabelle

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT 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.
- return genau einmal am Ende des Quellcodes einer jeder Funktion (u.a. auch main()) 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 mit aufgenommen werden.
- Name der Projekte nach folgendem Schema vergeben: Gruppe_Rechnernummer_Nachname_Vorname_Aufgabennr.
Beispiel: A_12_Mustermann_Erika_nr3
- Die Programme müssen ausgedruckt werden.
- Jeder Programmteil (Eingabe, Verarbeitung, Ausgabe) muss als solcher durch einen Kommentar angegeben werden.
- Jedes Projekt (d.h. jeder Projektordner) muß auf den Lehrerstick kopiert werden.

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.

- a) Welchen Wert hat der Inhalt der Variablen zahl und quad an der Stelle 1 im Programm ?
- b) Welchen Wert hat z und zq beim Aufruf von quadrat(zahl, &quad) ?
- c) Was bewirkt die Anweisung *zq = z * z an welcher Adresse im Arbeitsspeicher ?
(konkreten Wert der Adresse und deren Inhalt angeben!)
- d) 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 3 Programmierpraktikum 2BK11 26.4.2013 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:

keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT 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.
- return genau einmal am Ende des Quellcodes einer jeder Funktion (u.a. auch main()) 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 mit aufgenommen werden.
- Name der Projekte nach folgendem Schema vergeben: Gruppe_Rechnernummer_Nachname_Vorname_Aufgabennr.
Beispiel: A_12_Mustermann_Erika_nr3
- Die Programme müssen ausgedruckt werden.
- Jeder Programmteil (Eingabe, Verarbeitung, Ausgabe) muss als solcher durch einen Kommentar angegeben werden.
- Jedes Projekt (d.h. jeder Projektordner) muß auf den Lehrerstick kopiert werden.

Bemerkungen:

Der Compiler darf keine Warnungen und Fehler erzeugen !!!

AUFGABEN

1) Die ersten anz2 Zahlen des 2. Feldes sollen an die ersten anz1 Zahlen des 1. Feldes angehängt werden. Dies soll mit der Funktion anfüegen(...) realisiert werden.

a) 20P
Machen Sie zuerst eine Leistungsbeschreibung (Dokumentation) dieser Funktion anfüegen(...) nach dem im Unterricht verwendeten Schema. Wo nötig (bzw. von Vorteil) den Bezeichner const verwenden.

b) 20P
Implementieren Sie die Funktion "anfüegen (...)"

c) 10P
Schreiben Sie ein Programm, in dem ein Aufruf der Funktion "anfüegen(...)" verwendet wird (keine Eingabe über scanf(), sondern Aufruf mit konkreten Parametern).
Testen Sie diese Funktion, indem die Elemente des Feldes (die zwei zusammengesetzten Felder) auf dem Bildschirm ausgegeben werden.

Lösung:

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

void anfüegen(const int feld1[], int anz1, const int feld2[], int anz2,
              int feld3[]);

int main() {
    int feld1[10]={1,2,3,4,5};
    int feld2[20]={6,7,8,9,10,11};
    int feld3[30];
    int i;
    anfüegen(feld1, 5, feld2, 6, feld3);
    for(i=0;i<11;i++){
        printf(" %d",feld3[i]);
    }
    return(0);
}

/*****
**
** void anfüegen(const int feld1[], int anz1,
**               const int feld2[], int anz2, int feld3[]) **
**
**#*****
*/
Parameter:
    (i) const int feld1[] : das 1. Feld
    (i) int anz1          : die Anzahl der zu betrachtenden Elemente
                           des 1. Feldes
    (i) const int feld2[] : das 2. Feld
    (i) int anz2          : die Anzahl der zu betrachtenden Elemente
                           des 2. Feldes
    (o) int feld3[]       : das 3. Feld = anz1 Elemente von feld1 +
                           anz2 Elemente von feld2

Return:
    kein

Beschreibung:
    Fügt die ersten anz2 Elemente von feld2 an
    die ersten anz1 Elemente von feld1 an.
    Der Anwender dieser Funktion, muss sicherstellen, dass die
    Felder genügend gross sind.
*/

void anfüegen(const int feld1[], int anz1, const int feld2[], int anz2,
              int feld3[]){
    int i;
    int j;

    for(i=0;i<anz1;i++){
        feld3[i]=feld1[i];
    }
    j=0;
    for(i=anz1;i<anz1+anz2;i++){
        feld3[i]=feld2[j];
        j++;
    }
}
```

Name, Vorname:

Hilfsmittel:
keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT 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.
- return genau einmal am Ende des Quellcodes einer jeder Funktion (u.a. auch main()) 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 mit aufgenommen werden.
- Name der Projekte nach folgendem Schema vergeben: Gruppe_Rechnernummer_Nachname_Vorname_Aufgabennr.
Beispiel: A_12_Mustermann_Erika_nr3
- Die Programme müssen ausgedruckt werden.
- Jeder Programmteil (Eingabe, Verarbeitung, Ausgabe) muss als solcher durch einen Kommentar angegeben werden.
- Jedes Projekt (d.h. jeder Projektordner) muß auf den Lehrerstick kopiert werden.

AUFGABEN

1) 4P

Eine Funktion f(...) berechnet Wetterdaten und liefert diese in einem Feld zurück.

Da der aufrufende Programmierer, der diese Funktion benutzt, nicht weiß wie groß das Feld ist, das für die Wetterdaten verwendet wird, muß er zur Sicherheit immer Speicherplatz für ein großes Feld verwenden. Deshalb schreibt der Entwickler der Funktion f eine neue Funktion fNeu(...). In dieser wird nun dynamisch Speicher allokiert (mit malloc).

Da nach dem Aufruf dieser Funktion dieser Speicher bis zum Ende des Hauptprogramms zur Verfügung steht, gibt er ihn kurz vor Ende der Funktion (also eine Befehlszeile vor return) mit free wieder frei. Was ist an dieser diese Überlegung falsch?

2) 6P

In einer Funktion wird

- Speicherplatz für eine lokale Variable reserviert,
- dynamisch Speicher reserviert,
- Festplattenspeicher reserviert (eine Datei angelegt und beschrieben).

Welcher Speicher steht jeweils

- a) während des Aufrufs der Funktion,
- b) nach dem Aufruf der Funktion im Hauptprogramm und
- c) nach dem Beenden des Hauptprogramms zur Verfügung?

	Während des Aufrufs	Nach dem Aufruf der Funktion	Nach Beenden des Hauptprogramms
Speicherplatz für lokale Variable			
Dynamischer Speicher			
Festplattenspeicher (Datei anlegen)			

Ein Kreuz x bedeutet : Speicher steht zur Verfügung

Kein Kreuz x bedeutet : Speicher steht nicht zur Verfügung

3) Zu Beginn eines Zeitabschnitts wird ein Kapital zum Zinssatz p angelegt.

Welchen Wert hat es nach n Zeitabschnitten, wenn die Zinsen auf dem Sparbuch bleiben ?

Beispiel:

Zinssatz: 10%

Anfangskapital: 1000 Euro

Zeitabschnitte	Kontostand
0	1000
1	$1000 + 10/100 * 1000 = 1000 + 100 = 1100$
2	$1100 + 10/100 * 1100 = 1100 + 110 = 1210$
3	$1210 + 10/100 * 1210 = 1210 + 121 = 1331$
...	...

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

a) 20P

Entwerfen Sie für den "Programmierknecht" eine **Beschreibung** (Leistungsbeschreibung) der Funktion **vermehrten (...)** nach dem im Unterricht verwendeten Schema. (Kein Programm !!!)

Überlegen Sie sich zuerst genau, welche **Parameter** nötig sind und welche **Zusicherungen** Sie machen (möglichst viel erlauben).

Wo nötig (bzw. von Vorteil) den Bezeichner `const` verwenden.

b) 20P

Da der sogenannte "Programmierknecht" urplötzlich über Nacht Ihre Vorgesetzte wurde, muss die Funktion **vermehrten (...)** von Ihnen selbst implementiert werden. Implementieren Sie die Funktion **vermehrten (...)**.

Dazu dürfen keine von der Entwicklungsumgebung bereitgestellten Funktionen (wie z.B. `pow(...)`, usw.) verwendet werden.

Lösungen:

1) Nach dem Beenden von return in der Funktion, wurde der Speicherplatz schon freigegeben.

Er steht also nach dem Beenden der Funktion nicht mehr zur Verfügung.

2)

a) während des Aufrufs der Funktion: lokale Variable, dynamischer Speicher , Datei

b) nach den Aufruf im Hauptprogramm: dynamischer Speicher, Datei

c) nach dem Beenden des Hauptprogramms: Datei

	Während des Aufrufs	Nach dem Aufruf der Funktion	Nach Beenden des Hauptprogramms
Speicherplatz für lokale Variable	X		
Dynamischer Speicher	X	X	
Datei	X	X	X


```

3)
/*****
/**
/**  double vermehren(const double anfangskapital,      **/
/**                      const double zinssatz, const int n)  **/
/**                                     **/
/*#*****/
/*

```

Parameter:

- (i) const double anfangskapital>=0 : Anfangskapital
- (i) const double zins>=0: Zins
- (i) const int n>=0: Anzahl der Zeitabschnitte

Return:

berechnetes Endkapital nach n Jahren

Beschreibung:

Zu Beginn eines Zeitabschnitts wird das Anfangskapital "anfangskapital" zum Zinssatz "zinssatz" n Zeitabschnitte angelegt, Daraus ergibt sich dann das Endkapital, wenn die Zinsen auf dem Sparbuch bleiben.

5)

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

```
double vermehren(const double anfangskapital, const double
                zins, const int n);
```

```
int main(int argc, char* argv[]){
    double anfangskapital = 100;
    double endkapital;
    double zins = 10;
    int n = 0;

    endkapital=vermehren(anfangskapital, zins, n);
    printf("Endkapital=%f\n",endkapital);
    return 0;
}
```

```
double vermehren(const double anfangskapital, const double
                zins, const int n){
    int i=0;
    double kapital;
    kapital = anfangskapital;

    for(i=1; i<=n; i++){
        kapital=kapital+kapital*zins/100;
    }
    return kapital;
}
```

KLAUSUR 4 Programmierpraktikum 2BK11 7.6.2013 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:

keine

Hinweise (unbedingt beachten):

- Alle Aufgaben müssen bearbeitet werden.
- Der Name und Vorname muß in DRUCKSCHRIFT 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.
- return genau einmal am Ende des Quellcodes einer jeder Funktion (u.a. auch main()) 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 mit aufgenommen werden.
- Name der Projekte nach folgendem Schema vergeben: Gruppe_Rechnernummer_Nachname_Vorname_Aufgabennr.
Beispiel: A_12_Mustermann_Erika_nr3
- Die Programme müssen ausgedruckt werden.
- Jeder Programmteil (Eingabe, Verarbeitung, Ausgabe) muss als solcher durch einen Kommentar angegeben werden.
- Jedes Projekt (d.h. jeder Projektordner) muß auf den Lehrerstick kopiert werden.

Bemerkungen:

Der Compiler darf keine Warnungen und Fehler erzeugen !!!

AUFGABEN

1)

a)

40P

Eines morgens in aller Frühe finden Sie auf Ihrem Schreibtisch die folgende Leistungsbeschreibung (Dokumentation) der Funktion "ersetzen (...)" vor (siehe Rückseite). Implementieren Sie diese Funktion.
Wo nötig (bzw. von Vorteil) den Bezeichner const verwenden.

b)

5P

Schreiben Sie ein Programm, in dem ein Aufruf der Funktion "ersetzen (...)" verwendet wird (keine Eingabe über scanf(), sondern Aufruf mit konkreten Parametern). Testen Sie diese Funktion, indem die Elemente der neuen Folge auf dem Bildschirm ausgegeben werden.

c)

5P

Welchen Nachteil hätte es, wenn man in der Funktion ersetzen(...) dynamisch Speicherplatz für "neueFolge" belegt?
Man könnte ja die Größe dieses Speicherplatzes innerhalb der Funktion ersetzen(...) berechnen lassen und dann genau so viel Speicherplatz für "neueFolge" allokieren, wie "neueFolge" benötigt.

```

/*****
/**
/** void ersetzen(char zeichen, char folge[],
/** char ersetzung[], char neueFolge[])
/**
/**
/*****
/*

```

Parameter:

- (i) char zeichen : zu ersetzendes Zeichen.
- (i) char folge[] : dort wird das "zeichen" ersetzt.
- (i) char ersetzung[]: Das Zeichen "zeichen" wird durch die Zeichenfolge "ersetzung" ersetzt.
- (o) char neueFolge[] : Die durch die Ersetzung entstehende neue Zeichenfolge

Return:

kein

Beschreibung:

In der Zeichenfolge "folge" wird beim erstmaligen Auftauchen des Zeichens "zeichen" dieses Zeichen durch die Zeichenfolge "ersetzung" ersetzt. Dadurch entsteht die neue Zeichenfolge "neueFolge".

Der Programmierer, der diese Funktion benutzt muß dafür Sorge tragen, dass beim Aufruf dieser Funktion genügend Speicherplatz für "neueFolge" bereitgestellt wird.

Beispiel 1:

zeichen: a
 folge: "raav"
 ersetzung: "xy"
 Nach dem Aufruf:
 neueFolge: "rxyav"

Beispiel 2:

zeichen: s
 folge: "raav"
 ersetzung: "xy"
 Nach dem Aufruf:
 neueFolge: "raav"

*/

Lösungen:

```
#include "stdafx.h"
#include <string.h>
#include <malloc.h>

void ersetzen(const char zeichen, const char folge[],
              const char ersetzung[], char neueFolge[]);

int main(int argc, char* argv[]){
    char folge[4]="raf";
    char ersetzung[3]="xy";
    char neueFolge[4];
    char zeichen='a';

    ersetzen(zeichen, folge, ersetzung, neueFolge);
    printf("neueFolge=%s\n",neueFolge);
    return 0;
}

void ersetzen(const char zeichen, const char folge[],
              const char ersetzung[], char neueFolge[]){
    int i=0;
    int j=0;
    int index=0;

    // Kopiere alle Elemente von folge bis zum Auftreten
    // des gefundenen Zeichens in neueFolge
    while(folge[i]!='\0' && folge[i]!=zeichen){
        neueFolge[i]=folge[i];
        i++;
    }
    // Zeichen wurde nicht gefunden
    if(i==strlen(folge)){
    }
    else{
        index=i;
        // Füge ersetzung an neueFolge an
        while(ersetzung[j]!='\0'){
            neueFolge[i]=ersetzung[j];
            i++;
            j++;
        }
        j = index+1;
        // Füge von j=index an die folge an die neue Folge an.
        while(folge[j]!='\0'){
            neueFolge[i]=folge[j];
            i++;
            j++;
        }
    }
    neueFolge[i]='\0';
}
```

c)

Da der Speicher nicht innerhalb der Funktion freigegeben werden könnte, müsste bei jedem Aufruf neuer Speicher reserviert werden (Speicher kann ausgehen).

Oder man müsste die Funktion so implementieren, daß sie die Anfangsadresse des reservierten Speichers zurückgibt, so daß man den Speicherplatz dann in main wieder freigeben könnte.