

Name, Vorname:

Hilfsmittel:
Prioritätentabelle

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) (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) (4 P)

Kann die Ganzzahl 12345 in einer zwei Bytes grossen integer-Zahl abgespeichert werden ?
Begründen Sie!

6) (2 P)

Die Strecke s eines fallenden Steins in Abhängigkeit der Zeit t und der Erdbeschleunigung g wird wie folgt berechnet. $s = \frac{1}{2} \cdot g \cdot t^2$

Setzen Sie diese Formel in einen Ausdruck in C um.

7) (10P)

Erstellen Sie ein Struktogramm zu dem Programm, das zu der in einer Prüfung erreichten Punktezahl p , (die der Lehrer über Tastatur eingibt), folgende Meldung ausgibt:

p zwischen 0 und 36 Punkte (je einschließlich):	" Prüfung nicht bestanden"
p größer 36 Punkte und kleiner (oder gleich) 100 Punkte:	" Prüfung bestanden"
p nicht im Bereich zwischen 0 und 100:	"unzulässige Punktezahl"

Bemerkung: Die Punktezahl kann auch nicht ganzzahlig sein!

8)

(14P)

Gegeben sind die 3 folgenden Programmausschnitte: (x1,x2,x alle vom Datentyp integer)

Programmausschnitt1:

```
if((x==x1) || (x==x2))
    printf("Ausgabe1\n");
else
    printf("Ausgabe2\n");
```

Programmausschnitt2:

```
if(x==x1 || x2)
    printf("Ausgabe1\n");
else
    printf("Ausgabe2\n");
```

Programmausschnitt3:

```
if(x== (x1 || x2))
    printf("Ausgabe1\n");
else
    printf("Ausgabe2\n");
```

- Geben Sie die syntaktischen Fehler in den Programmausschnitten an (falls es diese gibt).
- Kann man die Klammern bei `(x==x1) || (x==x2)` im Programmausschnitt1 weglassen (damit das Programm das Gleiche macht)? Begründen Sie!
- Kann man die Klammern bei `(x1 || x2)` im Programmausschnitt3 weglassen (damit das Programm das Gleiche macht)? Begründen Sie!
- Geben Sie Werte für x1, x2 und x an, so daß (für diese Werte) Programmausschnitt1 und (für diese Werte) Programmausschnitt2 verschiedene Ausgaben auf den Bildschirm bringen (und zeigen dadurch, daß die Programmausschnitte nicht semantisch gleichwertig sind).
- Geben Sie Werte für x1, x2 und x an, so daß (für diese Werte) Programmausschnitt1 und (für diese Werte) Programmausschnitt3 verschiedene Ausgaben auf den Bildschirm bringen (und zeigen dadurch, daß die Programmausschnitte nicht semantisch gleichwertig sind).

9)

(11P)

Der folgende Programmausschnitt soll feststellen, ob 3 über Tastatur eingegebene ganze Zahlen gleich sind, oder nicht.

- Ist der Programmausschnitt syntaktisch korrekt?
- Geben Sie 3 **konkrete**, ganzzahlige Werte für z1, z2 und z3 an, für die das Programm eine falsche Ausgabe erzeugt. Begründen Sie.

```
...
if(z1==z2==z3)
    printf("alle Zahlen sind gleich\n");
else
    printf("nicht alle Zahlen sind gleich\n");
...
```

- Ändern Sie den Programmausschnitt so ab, dass es semantisch korrekt wird (das oben Angegebene macht). Benutzen Sie dazu möglichst wenig Klammern.

Lösung:

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

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

3) Priorität gibt den Vorrang eines Operators in einem Ausdruck an. (2P)

4) Assoziativität gibt an, in welcher Reihenfolge (von links nach rechts oder von rechts nach links) Operatoren mit der gleichen Priorität abgearbeitet werden. (2P)

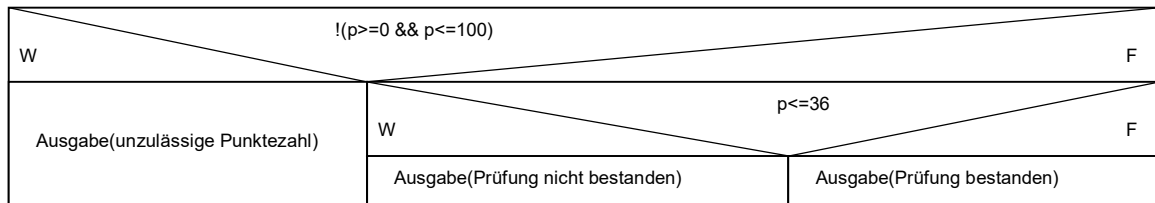
5) 2 Bytes kann $2^{16} \approx 64000$ verschiedene Zustände speichern. (4P)

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

Also kann 12345 in dieser zwei Bytes grossen Integer-Zahl gespeichert werden.

6) $s = 0.5 * g * t * t;$ (2P)

7) (10P)



8) (14P)

a) Es gibt keine syntaktischen Fehler (2P)

b) ja, weil == höhere Priorität hat als || (2P)

c) nein, weil == höhere Priorität hat als || (2P)

d) $x=0, x1=100, x2=0$ (4P)

oder

$x=1, x1=2, x2=3$

e) $x=1, x1=2, x2=3$ (4P)

9) (11P)

a) ja (2P)

b) (5P)

Eingabe: $z1=z2=z3=0$

$z1==z2==z3$, also:

$0 == 0 == 0$

$\underbrace{\quad\quad\quad}_1$
 $\underbrace{\quad\quad\quad}_0$

c) (4P)

Ersetze $z1==z2==z3$ durch $z1==z2 \ \&\& \ z2==z3$

KLAUSUR 1 Programmierpraktikum 2BK11 15.11.2011 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.

Lösung:

```
#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;
}

```

KLAUSUR 1 Programmierpraxis 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 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) Schreiben Sie ein C-Programm, das die Vereinigungsmenge zweier Zahlenmengen berechnet:

Über Tastatur werden die zwei Elemente (müssen jeweils verschieden sein) der Menge A eingegeben. Dann werden über Tastatur die zwei Elemente (müssen jeweils verschieden sein) der Menge B eingegeben.

Wurden für eine Menge zwei gleiche Zahlen eingegeben, muß das Programm **sofort** beendet werden und eine entsprechende Meldung auf den Bildschirm ausgegeben werden.

(insbesondere darf dann nicht mehr eine weitere Zahl eingegeben und der Durchschnitt und die Vereinigung berechnet werden).

Dies muß mit dem EVA-Prinzip realisiert werden.

Außerdem muß in der Ausgabe noch angegeben werden, in welcher Menge

(z.B. 2. Menge) 2 gleiche Zahlen eingegeben wurde (und welcher Wert eingegeben wurde).

Erstellen Sie das dazugehörige C-Programm.

Bemerkungen:

a) Es widerspricht nicht dem EVA-Prinzip, wenn z.B. im Eingabeteil in bestimmten Variablen bestimmte Zustände abgespeichert werden (z.B: ob in der 1. Menge zwei gleiche Zahlen eingegeben wurden) und diese dann im Ausgabeteil abgeprüft werden.

b)

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

c) Im Ergebnis (Vereinigungsmenge bzw. Durchschnitt) darf ein Element auch nur einmal vorkommen, also z.B: {1; 2; 3} und nicht {1; 2; 3; 2}

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 Punktabzug !!!!

AUFGABEN

1) 8P

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

```
#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) 6P

Wie oft wird in den folgenden Programmausschnitten jeweils die Meldung "Hallo" auf dem Bildschirm ausgegeben ?

a)

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

b)

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

c)

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


3)

10P

```

sum = 1;
i = 1;

--> do
{
    sum = sum + i;
    i = i+1;
}
while .....

```

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)

sum					
i					

b) Ergänzen Sie den Programmausschnitt (Bedingung der while-Schleife) so, daß der Wert der Variablen sum nach Verlassen der Schleife gleich der folgenden Summe ist:

$2 + 2 + 3 + 4 + 5 + \dots + 101 + 102$

4)

26P

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 5

5	4	5	6	-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.

Die erste eingegebene Note muss in der 1. Zelle, die zweite eingegebene Note muss in der 2. Zelle, usw. abgespeichert werden. Die 0. Zelle bleibt zunächst frei.

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

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) a) unendlich oft

b) 1

c) 0

3)

10P

a)

7P

sum	1	1+1	1+1+2	1+1+2+3	1+1+2+3+4
i	1	2	3	4	5

b)

3P

```
sum = 1;
i = 1;

--> do
{
    sum = sum + i;
    i = i+1;
}
while (i<=102);
```

4)

```
int _tmain(int argc, _TCHAR* argv[]){
    const int LEN = 5;
    double noten[LEN];
    double note;
    // zeigt auf letzte Note
    int zeiger=1;
    int beenden=0; // nicht beenden;
    double sum,mittel;
    int i;

    for(i=0;i<LEN;i++){
        noten[i]=-1;
    }

    do{
        if(zeiger>=LEN-1){
            // Programm beenden
            beenden=1;
        }
        else{
            printf("Bitte Note eingeben\n");
            scanf("%lf", &note);
            if(note>=1 && note <=6){
                noten[zeiger]=note;
                zeiger++;
            }
            else{
                beenden=1;
            }
        }
    }
    while(beenden==0);

    // Durchschnitt berechnen
    i=1;
    sum=0;
    while(noten[i]!=-1){
        sum=sum+noten[i];
        i++;
    }
    mittel=sum/(i-1);
    noten[0]=mittel;

    printf("Alle Zahlen im Feld\n");
    for(i=0;i<LEN;i++){
        printf("%lf ", noten[i]);
    }

    return 0;
}
```

KLAUSUR 2 Programmierpraxis 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 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 Punkteabzug !!!!

Bemerkungen:

Der Compiler darf keine Warnungen und Fehler erzeugen !!!

AUFGABEN

- 1) 50P
Schreiben Sie ein C-Programm, das ein Feld von Zahlen einer bestimmten Länge (Konstante nehmen) der Größe nach sortiert.

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 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)

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 in ein Feld der Länge 100 (Konstante nehmen) schreibt.

KLAUSUR 3 Programmierpraktikum 2BK11 24.4.2012 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 !!!!
- 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) 40P

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

b) 10P

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

Hinweis zum Algorithmus:

Um von einer Zahl z festzustellen, ob sie eine Primzahl ist, versucht man, diese durch alle Zahlen 1, 2, 3, 4, ..., z zu teilen. Wenn die Anzahl der erfolgreichen (Zahl ist teilbar) Versuche gleich 2 ist, hat man eine Primzahl.

KLAUSUR 3 Programmiertheorie 2BKI1 10.5.2012 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 3 Programmiertheorie 2BKI1 Nachtermin 1 Zeit: 45 Minuten

Name, Vorname:

Hilfsmittel:

Prioritätentabelle

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)

Die Funktion `split2(...)` teilt eine Zeichenfolge in 2 Teilzeichenfolgen, wobei das erstmalige Vorkommen eines bestimmten Zeichens die erste Teilzeichenfolge begrenzt.

Beispiel:

Zeichenfolge: "abcasdfgfdgdf"

Begrenzer: 'f'

Die zwei Teilzeichenfolgen:

"abcasd" und "gfdgdf"

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

a)

15P

Entwerfen Sie für den "Programmierknecht" eine **Beschreibung** (Leistungsbeschreibung) der Funktion "split2" mit dem im Unterricht verwendeten Schema. (Kein Programm !!!)

Überlegen Sie sich zuerst genau, welche **Parameter** nötig sind

b)

20P

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 "split2" von Ihnen selbst implementiert werden. Implementieren Sie die Funktion "split2".

c)

5P

Schreiben Sie ein Programm mit einem Aufruf der Funktion "split2"

(keine Eingabe über `scanf()`, sondern Aufruf mit konkreten Parametern).

d)

10P

Was würde sich ändern, wenn man bei a) dynamischen Speicher verwenden würde.

Geben Sie die Änderungen genau an.

Lösungen:

1a)

```
/* **** */
/**
/** void split2(char str1[], char begrenzer, char str2[]) **/
/**
/*# **** */
/*
```

Parameter:

- (i/o) char str1[]: String, der in 2 Strings aufgeteilt werden soll. Außerdem wird dort auch der 1. String links des Begrenzers abgespeichert (als output)
- (i) char begrenzer: Zeichen, des 1. Vorkommen die Zerlegung angibt.
- (i) char str2[] : Der String rechts des Begrenzers.

Return:

kein

Beschreibung:

Zerlegt den String str1 an der Stelle des Begrenzers (erstmaliges Vorkommen) in die 2 Teilstrings str1 und str2
Kommt der Begrenzer nicht in str1 vor, bleibt str1 unverändert und str2 = '\0'

Beispiel:

```
str1[]: "abcfxyzffz"
begrenzer: 'f'
str2[]: "wwwwwwwwwwww"
split2(str1, begrenzer, str2);
str1[]: "abc"
begrenzer: 'f'
str2[]: "xyzffz"
*/
```

b)

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

```
void split2(char str1[], char begrenzer, char str2[]);
```

```
int main(){
    char str1[]="abdexgijkf";
    char str2[]="xxxxxxxxxxxxxxxxxx";
    char begrenzer = 'f';
    printf("str1= %s\n",str1);
    split2(str1, begrenzer, str2);
    printf("str1= %s\n",str1);
    printf("str2= %s\n",str2);

    return 0;
}
```

c)

```
void split2(char str1[], char begrenzer, char str2[]){
    int len;
    int i;
    int index;

    index=-1;
    len=0;

    // Länge der zu splittenden Zeichenkette bestimmen
    for(i=0;str1[i]!='\0';i++){
        len++;
    }

    // Index des Begrenzers bestimmen
    for(i=0; str1[i]!='\0'; i++){
        if(str1[i]==begrenzer){
            index=i;
            break;
        }
    }

    if(index==-1)
        str2[0]='\0';
    else{
        str1[index]='\0';
        for(i=0;str1[index+i+1]!='\0';i++)
            str2[i]=str1[index+1+i];
        str2[i]='\0';
    }
}
```

d)

Vor dem Aufruf der Funktion müßte (z.B. in main) Speicher für str1 und str2 reserviert werden (mit malloc).

Nach dem Aufruf der Funktion müßte (z.B. in main) der reservierte Speicher für str1 und str2 wieder mit free freigegeben werden.

KLAUSUR 4 Programmiertheorie 2BKI1 28.6.2012 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 Programnteile.
- Bei Nichtbeachtung dieser Hinweise gibt es einen Punktabzug !!!!

AUFGABEN

1) 16P

- a) Welchen Sinn hat die von der C-Entwicklungsumgebung "spendierte" Funktion `malloc(...)` ?
- b) Geben Sie ein Beispiel an (kein Program, sondern verbale Beschreibung mit Begründung), wo `malloc(...)` unbedingt benötigt wird?
- c) Welchen Sinn hat die von der C-Entwicklungsumgebung "spendierte" Funktion `free(...)` ?
- d) Geben Sie ein Beispiel an (kein Program, sondern verbale Beschreibung mit Begründung), wo `free(...)` unbedingt benötigt wird?

2) 12P

- a)
- Erstellen Sie die Funktion
- ```
void set(int *p, int anz, int wert)
```
- die eine bestimmte Anzahl sich hintereinander befindlicher Elemente eines Integer-Feldes auf den Wert "wert" setzt.

- b)
- Gegeben ist das folgende Feld v der Länge 9

|   |   |   |   |   |          |          |          |   |
|---|---|---|---|---|----------|----------|----------|---|
| 3 | 2 | 7 | 8 | 3 | <b>9</b> | <b>7</b> | <b>1</b> | 4 |
|---|---|---|---|---|----------|----------|----------|---|

Rufen Sie `set` so auf, daß die fett markiertn Elemente dieses Feldes mit 17 belegt werden.

- c) Rufen Sie `set(...)` so auf, daß dieser Aufruf zwar syntaktisch korrekt ist, aber es zur Laufzeit Schwierigkeiten gibt.

3) 12P

- Das 1. Folgenglied einer Zahlenfolge ist 3 und das 2. Folgenglied ist 5.  
Die weiteren Folgenglieder berechnen sich aus den letzten 2 Folgengliedern.  
Also:  
3, 5, 8, 13, 21, 34, ...  
(z.B.  $21 = 13 + 8$ )

Schreiben Sie eine C-Funktion

```
int fib(int z)
```

das das n. Folgenglied dieser Folge berechnet.

Z.B. liefert `fib(3)` den Wert 8 zurück, `fib(4)` den Wert 13), usw.

4)

12P

Erstellen Sie zu der unten beschriebenen Funktion anfüegen(...) das zugehörige Struktogramm.

**Bemerkungen:**

```
p = (int *) malloc (3*sizeof(int)); // Speicher holen
free(p); // Speicher freigeben
```

```
/*
**
** int* anfüegen(int *v, int zahl)
**
**
**#
**
*/
```

Parameter:

- (i) int\* v : dynamisches Feld, an das die Zahl "zahl" angefügt werden soll.
- (i) int zahl: Zahl, die an das Feld "feld" angefügt wird.

Return:

Adresse des neuen Feldes.

**Beschreibung:**

Fügt an das letzte Element des Feldes "feld" die Zahl "zahl" an.  
 Dies geschieht dadurch, dass Speicher für ein neues Feld allokiert wird, das um 1 größer ist, als das alte Feld. Das neue Feld besteht dann aus dem alten Feld (kopieren der Elemente des alten Feldes in das neue Feld)  
 Dann wird die Zahl "zahl" als letztes Element an das neue Feld angefügt.  
 Mit return wird die Anfangsadresse dieses neuen Felds zurückgegeben.  
 Im 0. Element des Feldes wird die Feldlänge verwaltet.

**Beispiel:**

```
int *feld=NULL;
int *feldNeu;
int i;

for(i=0;i<10;i++){
 feldNeu=anfüegen(feld, i+10);
 feld=feldNeu;
}
*/
```



Lösungen:

1) Jede Teilaufgabe 4P

a) Damit kann man während des Programmlaufs Speicher reservieren.

b) Wenn ein Anwender während des Programmlaufs Zahlen eingibt, deren Anzahl er bestimmen kann (z.B. über Tastatureingabe).

c) Damit kann der reservierte Speicher wieder freigegeben werden.

d) Der Anwender reserviert während des Programmlaufs den maximal vom Betriebssystem zur Verfügung gestellten Speicher (Wetterdaten für Montag).

Diese Daten werden ausgewertet und dann nicht mehr benötigt. Wenn dieser Speicher nicht freigegeben wird, kann der Anwender im **gleichen** Programm keine neuen Wetterdaten für Dienstag mehr eingeben (weil er dafür vom Betriebssystem mehr keinen Speicher mehr bekommen kann).

Bem: Wenn das Programm beendet wird, wird automatisch der ganze reservierte Speicher freigegeben (dieser muß nicht mehr vorher durch free(...) freigegeben werden).

2) a) 6P

```
void set(int *p, int anz, int wert){
 int i;
 for(i=0; i<anz; i++){
 *(p+i)=wert;
 }
}
```

b) 3P

```
set(&v[5], 3, 17);
```

c) 3P

```
set(&v[100], 3, 17);
```

3) 12P

```
int fib(int n){
 int erg;
 int i;
 int letzte=5;
 int vorletzte=3;

 if(n==1)
 erg=3;
 else if(n==2)
 erg=5;
 else{
 erg=0;
 for(i=1; i<=n-2; i++){
 erg=letzte+vorletzte;
 vorletzte=letzte;
 letzte=erg;
 }
 }
 return erg;
}
```

4)

12P

```
int* anfüegen(int *v, int zahl){
 int i;
 int len;
 int *p;
 if(v==NULL){
 len=2;
 p = (int *) malloc (2*sizeof(int));
 p[0]=len;
 p[1]=zahl;
 }
 else{
 // Länge des alten Feldes
 len=v[0];
 // Neues Feld erzeugen, das um 1 länger ist.
 p = (int *) malloc ((len+1)*sizeof(int));
 // altes Feld in neues Feld kopieren
 for(i=0;i<len;i++)
 p[i]=v[i];
 // Element anfüegen
 p[len]=zahl;
 // Länge setzen im neuen feld
 p[0]=len+1;
 // Altes Feld freigeben
 free(v);
 }
 return p;
}
```

## KLAUSUR 4 Programmierpraktikum 2BK11 4.7.2012 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 !!!!
- 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)

50P

a) Implementieren Sie die unten dokumentierte Funktion contains.

b) Rufen Sie innerhalb von main() die Funktion "contains(...)" auf.  
(keine Eingabe über scanf(), sondern Aufruf mit konkreten Parametern).

Bemerkung:

Es darf die Funktion strlen (siehe Entwicklungsumgebung) benutzt werden.

Diese muß mit

#include "string.h"

im include-Teil des Programms eingebunden werden

```

/*****
/**
/** int contains(const char str[], const char string[]) **/
/** **/
/*#*****/
/*
Parameter:
 (i) const char str[] != "" : Zeichenkette, die darauf geprüft wird,
 ob sie in string enthalten ist.
 (i) const char string[] != "" : Zeichenkette

Return:
 1: str ist Teilzeichenkette von string
 0: str ist keine Teilzeichenkette von string

Beschreibung:
 Prüft ab, ob str Teilzeichenkette von string ist

Beispiele:
 str = "abcdefg"
 string = "abc"
 contains(str, string) liefert 0 zurück

 str = "abc"
 string = "xefabcdef"
 contains(str, string) liefert 1 zurück

 str = "cba"
 string = "xefabcdef"
 contains(str, string) liefert 0 zurück
*/

```

# KLAUSUR 4 Programmierpraxis 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 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.

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 ersetze(...) dynamisch Speicherplatz für "neueFolge" belegt?

Man könnte ja die Größe dieses Speicherplatzes innerhalb der Funktion ersetze(...) berechnen lassen und dann genau so viel Speicherplatz für "neueFolge" allokalieren, 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';
}
```

## KLAUSUR 4 Programmierpraxis 2BKI1 Nachtermin 2 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 !!!!
- 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\_Vormane\_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 Lehrerstück kopiert werden.

Bemerkungen:

Der Compiler darf keine Warnungen und Fehler erzeugen !!!

### AUFGABEN

1)

50P

a) Schreiben Sie die Funktion

`int primSum(int n)`

(unter Verwendung entsprechender selbstgeschriebener Funktionen), die die Summe der ersten n Primzahlen bildet.

b) Berechnen Sie in `main()` mit Hilfe der obigen Funktion die Summe der ersten 100 Primzahlen.



### Lösung:

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

int summeVonPrimzahlen(int n);
int istPrimzahl(int zahl);

int main(){
 int erg;

 erg = summeVonPrimzahlen(4);
 printf("Summe = %d \n",erg);
 return 0;
}

int summeVonPrimzahlen(int n){
 int erg;
 int zahl;
 int anzahl;
 int summe=0;

 zahl=2;
 anzahl=0;

 do{
 erg=istPrimzahl(zahl);
 if(erg==1){
 anzahl++;
 summe=summe+zahl;
 }
 zahl++;
 }
 while(anzahl<n);
 return summe;
}

int istPrimzahl(int zahl){
 int zaehler;
 int i;
 zaehler=0;

 for(i=1;i<=zahl;i++){
 if(zahl%i==0){
 zaehler++;
 }
 }
 if(zaehler==2)
 return 1;
 else
 return 0;
}
```