

Nachklausur zur Lehrveranstaltung TB PR2 (Programmieren in C) 21. September 2007

Hinweise:

Die Teilnahme an der Klausur ist nur bei Bestehen der Übungsaufgaben zulässig!

Zum Bestehen der Klausur benötigen Sie 50 Punkte von 100 möglichen Punkten.

Die Bearbeitungszeit der Klausur beginnt pünktlich um 10.00 Uhr und endet – ebenfalls pünktlich – um 11.30 Uhr. Sie haben also 90 Minuten Zeit.

Folgende Hilfsmittel in Papierform sind zugelassen: Skripte, Mitschriften und Bücher. **Nicht** zugelassen sind elektronische Geräte (Handys, Notebooks, PDAs, usw.).

Schreiben Sie mit Kugelschreiben oder Füller, **nicht** mit Bleistift! Verwenden Sie **nicht** die Farbe Rot. Schreiben Sie leserlich – was ich nicht lesen kann, ist grundsätzlich falsch! Beschreiben Sie nur die Vorderseiten!

Jeder Austausch mit anderen Personen wird als Täuschungsversuch gewertet und führt dazu, dass die Klausuren aller Beteiligten als „nicht bestanden“ gewertet werden.

Erläuterungen sollten kurz, aber dennoch präzise und vollständig sein. Wenn möglich ist eine stichpunktartige Beantwortung zu wählen, sofern die Verständlichkeit gegeben ist. Im Zweifelsfall können ganze Sätze Klarheit schaffen.

Kennzeichnen Sie jedes Blatt, das Sie abgeben, mit Ihrem Namen und / oder Ihrer Matrikelnummer.

Viel Erfolg!

Name: _____

Matrikelnummer: _____

letzter Prüfungsversuch: ☐

Bewertung: _____

Aufgabe	Mögliche Punktzahl	Erreichte Punktzahl
1	10	3
2	15	14
3	20	14
4	30	30
5	25	23
Summe	100	90

Note Klausur

1,7

Aufgabe 1: Finden Sie im folgenden Programm alle Syntaxfehler. (9 / 10)
Markieren Sie die Fehler und schreiben Sie hinter bzw. unter die Zeile, wie die Zeile richtig aussehen muss.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE* Datei;
    int AnzZeichen[256];
    int i;
    int Zeichen;

    for (i = 0; i < 256; i++) // Array Array zuruecksetzen
        AnzZeichen[i] = 0;
    Datei = fopen("aufg01.c", "r");
    if (FILE (Datei))
    {
        while (!feof(Datei))
        {
            Zeichen = fgetc(Datei); // &Z
            if ((Zeichen >= 0) and (Zeichen < 256))
                AnzZeichen[Zeichen]++;
        }
        fclose(Datei);
        printf("Anzahl gelesener Zeichen:\n");
        printf("ASCII | Anzahl\n");
        printf("-----\n");
        for (i = 0; i < 256; i++)
            if (AnzZeichen[i] > 0)
                printf(" %03i | %6i\n", i, AnzZeichen[i]);
        printf("-----\n");
    }
}
```

Aufgabe 2: Multiple-Choice-Fragen.

(14 / 15)

Kreuzen Sie an, wie der Satz richtig heißen muss. Es gibt immer nur eine richtige Antwort.

Mit dem Schlüsselwort `struct` wird eine

- ☒ Datenstruktur definiert.
- ☐ Programmstruktur definiert.
- ☐ Computerstruktur definiert.

Mit dem Präprozessorbefehl `#define` wird eine

- ☐ eine Variable definiert.
- ☒ eine Konstante definiert.
- ☐ eine Funktion definiert.

Eine andere Schreibweise für $z \rightarrow F$ (z ist ein Zeiger auf eine Struktur, die das Feld F beinhaltet) ist

- ☐ `*(Z) . F`
- ☒ `(*Z) . F`
- ☐ `*(Z . F)`

Eine Variable namens `FktPointer` vom Typ Zeiger auf eine Funktion, die zwei `int`-Parameter erhält und einen `int`-Zeiger zurückgibt, wird folgendermaßen definiert:

- ☐ `int ** (FktPointer(int, int))`
- ☐ `int (*FktPointer*)(int, int)`
- ☒ `int * (*FktPoiner)(int, int)`

Reservierte Speicherbereiche werden freigegeben mit der Funktion

- ☐ `realloc`
- ☒ `free`
- ☐ `remove`

Eine Struktur darf

- ☒ beliebig viele Unterstrukturen enthalten.
- ☐ nur eine Unterstruktur enthalten.
- ☒ keine Unterstruktur enthalten.

Eine Variable vom Typ `int **` belegt

- ☐ viermal so viel Speicher wie eine Variable vom Typ `int *`.
- ☐ doppelt so viel Speicher wie eine Variable vom Typ `int *`.
- ☒ genau so viel Speicher wie eine Variable vom Typ `int *`.

Eine `make`-Datei

- ☐ kann wie eine Header-Datei eingebunden werden.
- ☐ kann vom C-Compiler kompiliert werden.
- ☒ beschreibt Abhängigkeiten.

Kann ein dynamisch reservierter Speicherbereich nicht mehr freigegeben werden, weil z.B. kein Zeiger mehr auf diesen Speicherbereich verweist, wird dieses

- ☐ Computerleck genannt.
- ☒ Speicherleck genannt.
- ☐ Programmleck genannt.

Der Unterschied zwischen Text- und Binär-Modus bei der Datei-Ein- und Ausgabe liegt u.a.

- ☒ in der Erkennung des Dateiendes.
- ☐ in der Erkennung der deutschen Umlaute.
- ☐ in der Schreib- und Lesegeschwindigkeit.

Mit `int * const Zeiger;` wird folgendes definiert:

- ☐ Ein unveränderbarer Zeiger auf eine unveränderbare Variable.
- ☐ Ein veränderbarer Zeiger auf eine unveränderbare Variable.
- ☒ Ein unveränderbarer Zeiger auf eine veränderbare Variable.

Ein mit `#define` definiertes Makro wird wieder entfernt mit dem Befehl

- ☐ `#undefine`
- ☒ `#undef`
- ☐ `#delete`

Wenn mit der `fopen`-Funktion eine Datei nicht geöffnet werden kann, gibt die Funktion

- ☒ einen `NULL`-Zeiger zurück.
- ☐ den Wert `EOF` zurück.
- ☐ gar nichts zurück.

Wird ein Speicherbereich mit `malloc` reserviert, ist der Inhalt dieses Speicherbereiches

- ☐ mit 0 initialisiert.
- ☐ mit 255 initialisiert.
- ☒ undefiniert.

Die `fclose`-Funktion

- ☒ gibt eine 0 zurück, wenn die Datei geschlossen werden konnte.
- ☐ gibt nichts zurück.
- ☐ gibt den Parameter zurück.

Aufgabe 3: Was gibt das folgende Programm aus? (14 / 20)
Schreiben Sie die Ergebnisse der vorgegebenen Ausdrücke (Zwischenergebnisse) sowie die Bildschirm-Ausgabe des Programms auf das folgende Blatt!

```
#include <stdio.h>

int Addiere1(int);
int Halbiere(int);
int MalDrei(int);

int main()
{
    printf("%02i.", MalDrei (Halbiere(Addiere1(13))));
    printf("%02x.", Addiere1(Halbiere(Addiere1(MalDrei(5)))));
    printf("%02d", Halbiere(Addiere1(MalDrei (13))));
    printf("%02o", Addiere1(MalDrei (Halbiere( 4))));
}

int Addiere1(int i)
{
    return i + 1;
}

int Halbiere(int i)
{
    return i / 2;
}

int MalDrei(int i)
{
    return i * 3;
}
```

Zwischenergebnisse:

erste printf-Zeile:

Addiere1(13) = 14 ✓

Halbiere(Addiere1(13)) = 7 ✓

MalDrei(Halbiere(Addiere1(13))) = 21 ✓

int

zweite printf-Zeile:

MalDrei(5) = 15 ✓

Addiere1(MalDrei(5)) = 16 ✓

Halbiere(Addiere1(MalDrei(5))) = 8 ✓

Addiere1(Halbiere(Addiere1(MalDrei(5)))) = 9 ✓

HEX

dritte printf-Zeile:

MalDrei(13) = 39 ✓

Addiere1(MalDrei(13)) = 40 ✓

Halbiere(Addiere1(MalDrei(13))) = 20 ✓

dezimal

vierte printf-Zeile:

Halbiere(4) = 2 ✓

MalDrei(Halbiere(4)) = 6 ✓

Addiere1(MalDrei(Halbiere(4))) = 7 ✓

octal

Bildschirm-Ausgabe des oben stehenden Programms:

21(✓)
9
70100
7

21.09.2007

(-6)

Aufgabe 4: Schreiben Sie die drei fehlenden Funktionen zum vorgegebenen Hauptprogramm.

Die Funktion `LiesDatei` soll MAX Zeichenketten (maximale Länge: MAXLEN Zeichen) aus der Textdatei mit dem angegebenen Dateinamen einlesen und in dem angegebenen Array von Zeichenketten (übergeben wird die Adresse von einem Zeiger auf `char *`) ablegen. Der Speicherbereich für die einzelnen Zeichenketten muss zuvor noch reserviert werden. Es kann von einer korrekt formatierten Textdatei ausgegangen werden.

Die Funktion `Vergleich` soll zwei Zeichenketten vergleichen, die als Parameter übergeben werden. Das Funktionsergebnis soll größer 0 sein, wenn die erste größer als die zweite Zeichenkette, kleiner 0, wenn die erste kleiner als die zweite Zeichenkette und gleich 0, wenn beide Zeichenketten gleich sind (dazu kann die Funktion `strcmp` verwendet werden!).

Die Funktion `Sortiere` soll das angegebene Array von Zeichenketten mittels der BubbleSort- und der Vergleichsfunktion sortieren. (30 / 30)

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define MAX 10
#define MAXLEN 100

typedef char* String;

void BubbleSort(String *Array, int Anzahl, int (*Vergleich)(String, String));
void SchreibeDatei(String Name, String *Array);
```

// Platz für Ihre Funktionsdeklarationen:

```
void LiesDatei(char *, char ***); ✓
int Vergleich(char *, char *); ✓
void Sortiere(char **); ✓
```

```
char** TextArray int main()
{ String *TextArray;

  TextArray = malloc(MAX * sizeof(String));
  if (TextArray != NULL)
  { LiesDatei("unsortiert.txt", &TextArray);
    Sortiere(TextArray);
    SchreibeDatei("sortiert.txt", TextArray);
    free(TextArray);
  }
}

void BubbleSort(String *Array, int Anzahl, int (*Vergleich)(String, String))
{ char temp[MAXLEN];

  for (int i = 1; i < Anzahl; i++)
    for (int j = Anzahl - 1; j >= i; j--)
      if (Vergleich(*(Array + j), *(Array + j - 1)) < 0)
      { strcpy(temp, *(Array + j));
        strcpy(*(Array + j), *(Array + j - 1));
        strcpy(*(Array + j - 1), temp);
      }
}
```

```
void SchreibeDatei(String Name, String *Array)
{ FILE *D;

  D = fopen(Name, "w");
  if (D != NULL)
    for (int i = 0; i < MAX; i++)
      { fprintf(D, "%s\n", *(Array + i));
        free(*(Array + i));
      }
}
```

// Platz für Ihre Funktionsdefinitionen:

void hiesDatei(char *N; char **ZK)

{ FILE *D;

int i;

i = 0;

D = fopen(N, "w");

if (D != NULL)

~~{ *ZK = malloc(sizeof(MAXLEN));~~

~~while (i < MAX)~~

~~{ fprintf(D, "%s\n", *(ZK + i));~~

~~free(*(ZK + i));~~

{ while (i < MAX)

{ *ZK + i = malloc(sizeof(MAXLEN));

if (*ZK + i)

fscanf(D, "%s", *(ZK + i));

i++;

} fclose(D);

}

return i;

}



Fortsetzung der Funktionsdefinitionen für Aufgabe 4:

```
int Vergleich(char * T1, char * T2) {
```

```
    return strcmp(T1, T2);
```

```
}
```

```
void sortiere(char ** ZK)
```

```
{ bubbleSort(ZK, MAX, Vergleich);
```

```
return; }
```

Aufgabe 5: Schreiben Sie die passenden Funktionen zum vorgegebenen Hauptprogramm.

Die erste Funktion `Trim` soll alle Leerzeichen am Ende einer Zeichenkette löschen (d.h. mit dem ASCII-Wert 0 überschreiben). Als Parameter wird die Zeichenkette übergeben. Die Funktion gibt nichts zurück.

Die zweite Funktion `Leerzeichen` soll alle Leerzeichen in einem Array von 10 Zeichenketten mit je maximal 50 Zeichen zählen. Als Parameter erhält es ein zweidimensionales Array von Zeichen. Das Funktionsergebnis ist die Anzahl der Leerzeichen in allen Zeichenketten des Arrays von Zeichenketten. (23 / 25)

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

```
// Platz für Ihre Funktionsdeklarationen:
```

```
void Trim (char *);  
int AnzahlLeerzeichen (char **);
```

```
int main()
```

```
{  
    char Text[10][50] = {"Dies ist ein langer",  
                        "Text, der in mehreren",  
                        "Zeilen untergebracht",  
                        "ist und der auch noch",  
                        "in manchen Zeilen mehrere",  
                        "Leerzeichen am Ende",  
                        "beinhaltet! Die Leer-",  
                        "zeichen dieses Textes",  
                        "sollen gezählt werden!",  
                        ""};
```

```
    int i = 0;
```

```
    for ( ; i < 10; i++)  
        Trim(Text[i]);
```

```
    printf("\nIn dem Text sind %i Leerzeichen enthalten!\n",  
           AnzahlLeerzeichen(Text));
```

```
}
```

```
// Platz für Ihre Funktionsdefinitionen:
```

Fortsetzung der Funktionsdefinitionen für Aufgabe 5:

void Trim (char x[2])

{ while (i > 0) { { int i
i = 50;

if ((* (z+i) == '#' ||
* (z+i) == "0") {
i --;

else return;

if (* (z+i) > ' ')

}

-1

somit wird sofort abgebrochen,
da das letzte Zeichen kein
Leerzeichen ist!

int AnzahlZeichen (char x[k])

{ ~~int i;~~ ~~int e;~~

while (i < 10) { i = 0;
e = 0;
leer = 0;

{ while (e < 50)

{

if (x[i][e] == '#' || " ")

e++; leer++;

}

i++;

} return leer;

}

