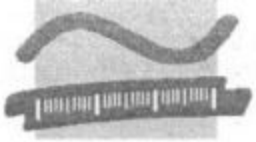
Dipl.Phys. Gerald Kempfer
Technische Fachhochschule Berlin
- University of Applied Sciences -
Fachbereich VI – Informatik und Medien
Studiengang Technische Informatik Bachelor



TFH Berlin

Klausur zur Lehrveranstaltung TB PR2 (Programmieren in C) 02. Februar 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 16.00 Uhr und endet – ebenfalls pünktlich – um 17.30 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: _____ 3. Prüfungsversuch: ☐

Bewertung:

Aufgabe	Mögliche Punktzahl	Erreichte Punktzahl
1	10	9
2	15	14
3	20	20
4	25	25
5	30	28
Summe	**100**	96

Note Klausur 1,3

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>
#inklude <stdlib.h>

int main()
{
   struct TArtikel
   {
      int ArtNr;
      int Status;
      struct TArtikel *NextArtikel;
   } *Artikel, *Start;
   int i;

   Artikel = malloc(sizeof(struct TArtikel));
   Start = Artikel;
   for (i = 0, i < 10; )
   {
      Artikel->ArtNr = ++i;
      Artikel->Status = 0x1fg2 | i;
      if (i < 10)
         Artikel->NextArtikel = malloc(sizeoff(struct TArtikel));
      else
         Artikel->NextArtikel = NULL;
      Artikel = Artikel.NextArtikel;
   }

   Artikel = Artikel->Start;
   for (i = 20; i >= 11; i--)
   {
      printf("ArtNr : %i\n", Artikel->ArtNr);
      printf("Status: %i\n", Artikel->Status);
      Artikel ->= NextArtikel;
   }

   while (Start != Null)
   {
      Artikel = (*Start).NextArtikel;
      free(Start);
      Start = Artikel;
   }

   return 0;
}
```

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->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 verweise, 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? (20 / 20)
Schreiben Sie die Ergebnisse der vorgegebenen Ausdrücke (Zwischenergebnisse) sowie die Bildschirm-Ausgabe des Programms auf das folgende Blatt!

```
#include <stdio.h>

int Fkt1(int);
int Fkt2(int);
int Fkt3(int);

int main()
{
   int (*FktArray[3])(int) = {Fkt1, Fkt2, Fkt3};

   printf("Datum: %02i.",  FktArray[Fkt2(Fkt1(3))](3) - 4  );
   printf("%02i.",         FktArray[Fkt2(32)     ](Fkt3(7)));
   printf("%04i\n", 2001 + FktArray[Fkt3(4) - 12 ](0) / 2  );
}

int Fkt1(int Wert)
{
   return (Wert + 3) * (Wert + 4);
}

int Fkt2(int Wert)
{
   return (Wert - 2) / 10 - 2;
}

int Fkt3(int Wert)
{
   return Fkt1(Wert - 4);
}
```

Zwischenergebnisse:

erste `printf`-Zeile:

`Fkt1(3)` = 42 ✓

`Fkt2(Fkt1(3))` = 2 ✓

`FktArray[Fkt2(Fkt1(3))]` zeigt auf die Funktion Fkt3 ✓

`Fkt3(3)` = 6 ✓

`Fkt3(3) - 4` = 2 ✓

zweite `printf`-Zeile:

`Fkt2(32)` = 1 ✓

`FktArray[Fkt2(32)]` zeigt auf die Funktion Fkt2 ✓

`Fkt3(7)` = 42 ✓

`Fkt2(Fkt3(7))` = 2 ✓

dritte `printf`-Zeile:

`Fkt3(4)` = 12 ✓

`Fkt3(4) - 12` = 0 ✓

`FktArray[Fkt3(4) - 12)` zeigt auf die Funktion Fkt1 ✓

`Fkt1(0)` = 12 ✓

`Fkt1(0) / 2` = 6 ✓

`2001 + Fkt1(0) / 2` = 2007 ✓

Bildschirm-Ausgabe des oben stehenden Programms:

Datum: 02.02.2007 ✓

Aufgabe 4: Schreiben Sie die passende Funktion `fcopyChars` zum vorgegebenen Hauptprogramm, die alle Buchstaben ('a' ... 'z' und 'A' ... 'Z') aus der Quelldatei in die Zieldatei kopiert. Die Funktion erhält zwei Zeichenketten, in denen die Namen von Quell- und Zieldatei stehen. Als Funktionsergebnis soll die Anzahl der kopierten Buchstaben; im Fehlerfalle sollen negative Werte (siehe Hauptprogramm) zurückgegeben werden.
Weitere Headerdateien sind nicht erlaubt! (25 / 25)

```
#include <stdio.h>

// Platz für Ihre Funktionsdeklaration:
int fcopyChars(char*,char*);


int main()
{
  int Erg;

  Erg = fcopyChars("Datei1.txt", "Datei2.txt");
  switch(Erg)
  {
    case -1:
      printf("Die Quell-Datei konnte nicht geoeffnet werden!\n");
      break;
    case -2:
      printf("Die Ziel-Datei konnte nicht geoeffnet werden!\n");
      break;
    case 0:
      printf("Quell-Datei leer / beinhaltet keine Buchstaben!\n");
      break;
    default:
      printf("%i Buchstaben von Quell- nach Zieldatei!\n", Erg);
      break;
  }
  return 0;
}

// Platz für Ihre Funktionsdefinition:
int fcopy(chars(char*q, char*z)
{
  int b, Erg=0;
  File *quelle=NULL, *Ziel=Null;
  quelle=fopen(q, "r");
  if(!quelle)
    return -1;
  Ziel=fopen(z, "w");
  if(!Ziel) <fclose(quelle);
    return-2;
```

Fortsetzung der Funktionsdefinitionen für Aufgabe 4:

```
fseek(quelle, 0, SEEK_SET);
fseek(quelle, 0, SEEK_SET);

while (!feof(quelle))
{
   b = fgetc(quelle);
   if((b>='a' && b<='z')||(b>='A' && b<='Z'))
   {
      fputc(b, Ziel);
      Erg++;
   }
}
fclose(quelle);
fclose(Ziel);
return Erg;
}
```

Aufgabe 5: Schreiben Sie die drei fehlenden Funktionen zum vorgegebenen Hauptprogramm.
Die Funktion `CreateEMail` soll den notwendigen Speicher für eine EMail reservieren und anschließend unter Verwendung der Funktion `GetMem` für die Felder der EMail den entsprechenden Speicher reservieren und die Parameter-Werte in die neue EMail kopieren. Die erzeugte EMail soll zurückgegeben werden.
Die Funktion `DestroyEMail` soll die reservierten Speicherbereiche der angegebenen EMail wieder freigeben
Die Funktion `InsertEMail` soll die übergebene EMail in die angegebene Mailbox einfügen.
(28 / 30)

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

typedef struct
{
   char *From, *To, *Subject, *Body;
} EMail;

typedef struct
{
   char *User, *Adresse; // Benutzer und Absender-EMail-Adresse
   int AnzEMails;        // Anzahl der im Array gespeicherten EMails
   EMail *EMails[100];
} Mailbox;

// Platz für Ihre Funktionsdeklarationen:
EMail *CreateEMail(char*, char*, char*, char*);
void InsertEMail(Mailbox*, EMail*);
void DestroyEMail(EMail*);

int main()
{
   EMail *EZ = NULL;
   Mailbox MyMailbox;
   int i;

   MyMailbox.User = "Martin Mustermann";
   MyMailbox.Adresse = "martin@mustermann.de";
   MyMailbox.AnzEMails = 0;

   EZ = CreateEMail(MyMailbox.Adresse, "to@xyz.de", "Test", "Hallo Welt!");
   InsertEMail(&MyMailbox, EZ);
   EZ = CreateEMail(MyMailbox.Adresse, "susi@sorge.de", "Hallo", "Eine EMail!");
   InsertEMail(&MyMailbox, EZ);

   for (i = 0; i < MyMailbox.AnzEMails; i++)
      DestroyEMail(MyMailbox.EMails[i]);
   MyMailbox.AnzEMails = 0;
}

void GetMem(char **TZ, char *T)
{
   (*TZ) = malloc(strlen(T) + 1);
   if (*TZ != NULL)
      strcpy(*TZ, T);
}
```

Funktionsdefinitionen für Aufgabe 5:

```
EMail *(reate EMail(char *f, char *t, char *s, char *b)
{   EMail *e = NULL;

    e = (EMail*) calloc(1, sizeof(EMail));
-1  if (!e)
    { fatal Mem &(e->From), 1);
-1    fatal Mem &(e->To), e);
      fatal Mem &(e->Subject), s);
      fatal Mem &(e->Body), b);
    }
    return e;
}                                   (✓)

void InsertEMail(MailBox *m, EMail *e)

so. { if (m)
    { m->EMails[m->AnzEMails] = e;
      m->AnzEMails++;
    }
}                                   (✓)

void DestroyEMail(EMail *e)
s.o. { if (e)
    { free(e->From);
      free(e->To);
      free(e->Subject);
      free(e->Body);
    }
    free(e);
}                                   (✓)
```