PR3 - Lösungen zu freiwilligen Übungsaufgaben

WS 2020/21 (Stand 2020-12-29 20:59)

Prof. Dr. Holger Peine
Hochschule Hannover
Fakultät IV – Abteilung Informatik
Raum 1H.2.60, Tel. 0511-9296-1830
Holger.Peine@hs-hannover.de

Inhalt

Blatt A.1

5 Deklarationen und Definitionen

a)

1. `int i;` => Definition: Speicher für 1 int-Variable wird in dieser Objektdatei angelegt
2. `extern int summe(int a, int b);` => Deklaration
3. `int summe(int a, int b) { return a + b; }` => Definition
4. `extern int k;` => Deklaration
5. `typedef int Entfernung;` => Deklaration
   (eine "Definition" für Typen gibt es streng genommen gar nicht, denn ein Typ belegt niemals Speicher)
6. `Entfernung hannoverNachHamburg;` => Definition (einer Variablen vom Typ `Entfernung`)

   Hinweisen: Leider ist bei Funktionsdeklarationen das Schüsselwort `extern` optional (denn es ist schon durch das Fehlen eines Funktionskörpers klar, dass es keine Funktionsdefinition sein kann). Es wird aber empfohlen, `extern` immer zu verwenden, weil die Deklaration so leichter erkennbar und konsistent mit der Syntax für die Deklaration von Variablen ist.

   Hinweisen: Achtung, bei Typdeklarationen (kommen später, z.B. `typedef int entfernung`) und Makro-Deklarationen (in 2c behandelt: `#define MAX_ENTFERNUNG 100`) spricht man oft schlampigerweise ebenfalls von "Definitionen" – eigentlich sollte man aber auch dort "Typ- bzw. Makrodeklaration" sagen.

b) Eine Header-Datei darf nur Deklarationen enthalten.

   Nur bei Fragen hinweisen: Trotzdem dürfen Deklarationen desselben structs (s.Kapitel 4) nicht mehrfach beim Kompilieren ein und derselben Quelldatei auftauchen (wird später in der Vorlesung noch behandelt); andere Deklarationen (für Funktionen, Variablen, Makros und typedef's) dürfen durchaus mehrfach in derselben Datei auftauchen, solange sie identisch sind.

c) Die Größe der Objektdatei wächst nicht: Daran sieht man, dass aus Deklarationen (und nur solche dürfen in Header-Dateien enthalten sein) weder Code noch Variablen (die beide Speicher in der Objektdatei und zur Laufzeit belegen würden) entsteht.

6 Inhalt von Objektcode-Dateien (0 Punkte)

```
peine> nm main.o
0000000000000000 T main
                 U printf
                 U quadrat
peine> nm quadrat.o
0000000000000000 T quadrat
```

Offensichtlich wird ist der Maschinencode der Funktion main() in der Datei main.o enthalten; außerdem werden in dieser Datei noch die externen Funktionen quadrat und printf benutzt. Der Maschinencode für quadrat ist offensichtlich in der Datei quadrat.o enthalten, die selbst keine externen Funktionen benutzt.

Das Verdopplen der Zeile #include "quadrat.h" in main.c ändert nichts an main.o: Daran sieht man, dass Deklarationen keinen Code erzeugen (sonst wäre der entsprechende Code nun zweimal in main.o enthalten), denn Header-Dateien (wie hier quadrat.h) sollten immer nur Deklarationen enthalten, aber keine Definitionen.

Hinweisen:
`U printf` bedeutet, dass in `main` u.a. die externe Funktion `printf` benutzt wird. Der Maschinencode für diese Funktion steht nicht in einer der Objektdateien, sondern in der Standard-Bibliothek, die beim Linken immer mitdurchsucht wird; dort gefundener vom Programm benötigter Code wird dann ins die ausführbare Programmdatei hineinkopiert (bzw. bei dynamischem Linken wird in der Programmdatei ein Verweis auf die Bibliothek eingetragen).
Die Ausgabe von `nm /usr/lib/x86_64-linux-gnu/libc.a | grep ' printf$'` zeigt, dass `printf` in der Datei `…/libc.a` (das ist die C-Standard-Bibliothek) definiert ist:
`… T printf`

Hiweisen: Leider sind in der Objektdatei keine Typinformationen enthalten: Für Funktionen und Variable ist nur ihr Name enhalten (und die Größe ihres Objektcodes bzw. ihr Speicherbedarf, jeweils in Bytes). Dies ist eine Schäche der Programmiersprache C (bzw. ihres Compilers und Linkers), in C++ ist diese Schwäche beseitigt. In C ist es deshalb besonders wichtig, dass in einer Quelldatei für jede darin benutzte externe Funktion eine Deklaration dieser Funktion steht (die man normalerweise durch das Inkludieren der entsprechenden Header-Datei erhält).

Blatt A.2

2 Inhalt von Objektdateien

```
nm math.o summe.o differenz.o input.o

math.o:
0000000000000000 T berechne
                 U differenz
0000000000000068 T main
                 U printf
                 U puts
                 U scanf
                 U summe

summe.o:
                 U get_input
0000000000000000 T summe

differenz.o:
0000000000000000 T differenz
                 U get_input

input.o:
0000000000000000 T get_input
                 U printf
                 U scanf
```

T steht für die Definition einer Funktion, D (kommt hier nicht vor) für die Definition einer Variable und U für zwar benutzte, aber nicht in dieser Datei definierte Funktion. Demgemäß bedeutet die obige nm-Ausgabe Folgendes:

Modul	definiert bzw. implementiert	benutzt, aber nicht definiert
math.o	berechne, main	differenz, printf, puts, scanf, summe
summe.o	summe	get_input
differenz.o	differenz	get_input
imput.o	get_input	printf, scanf

Auf die Korrespondenz dieser Tabelle mit den Definitionen und #include's des Quelltexts hinweisen!

8 Fehler in Makefile finden

Hier die fehlenden und die ~~falschen~~ Bestandteile; außerdem muss das Target `prog` statt `program` heißen

```
prog: m.o f.o g.o
      gcc -o prog m.o f.o g.o

f.o: g.h f.c
      gcc -c f.c   # Ohne –c würde auch der Linker aufgerufen und schlüge fehl

g.o: ~~stdio.h~~  g.c # Abhängigkeit von Systemdateien nicht angeben, da sie sich nicht ändern
      gcc -c g.c

m.o: f.h g.h m.c
      gcc -c m.c

~~m.c: f.h g.h~~                  # *.c-Dateien können nicht automatisch gebaut werden,
   ~~gcc -o m.c f.h g.h~~         # sondern müssen vom Programmierer geschrieben werden
```

9 Portierung von Java nach C

Musterlösung:

```c
/* vielfache.c */

#include <stdio.h>

int main(void) {
    int a;
    int i, j;
    do {
        printf("Bitte positive Zahl eingeben: ");
        scanf("%d", &a);
    } while (a <= 0);
    for (i=0; i<10; i++) {
        for (j=0; j<10; j++) {
            if (i==0) {
                printf(" ");
```

```
            }
            printf("%d", i*10+j);
            if ((i*10+j)%a==0) {
                printf("# ");
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }
    return 0;
}
```

Achten auf: Parameter void (nicht einfach weglassen, da das bedeuten würde, dass die Deklaration der Parameter vertagt wird).

Blatt A.3

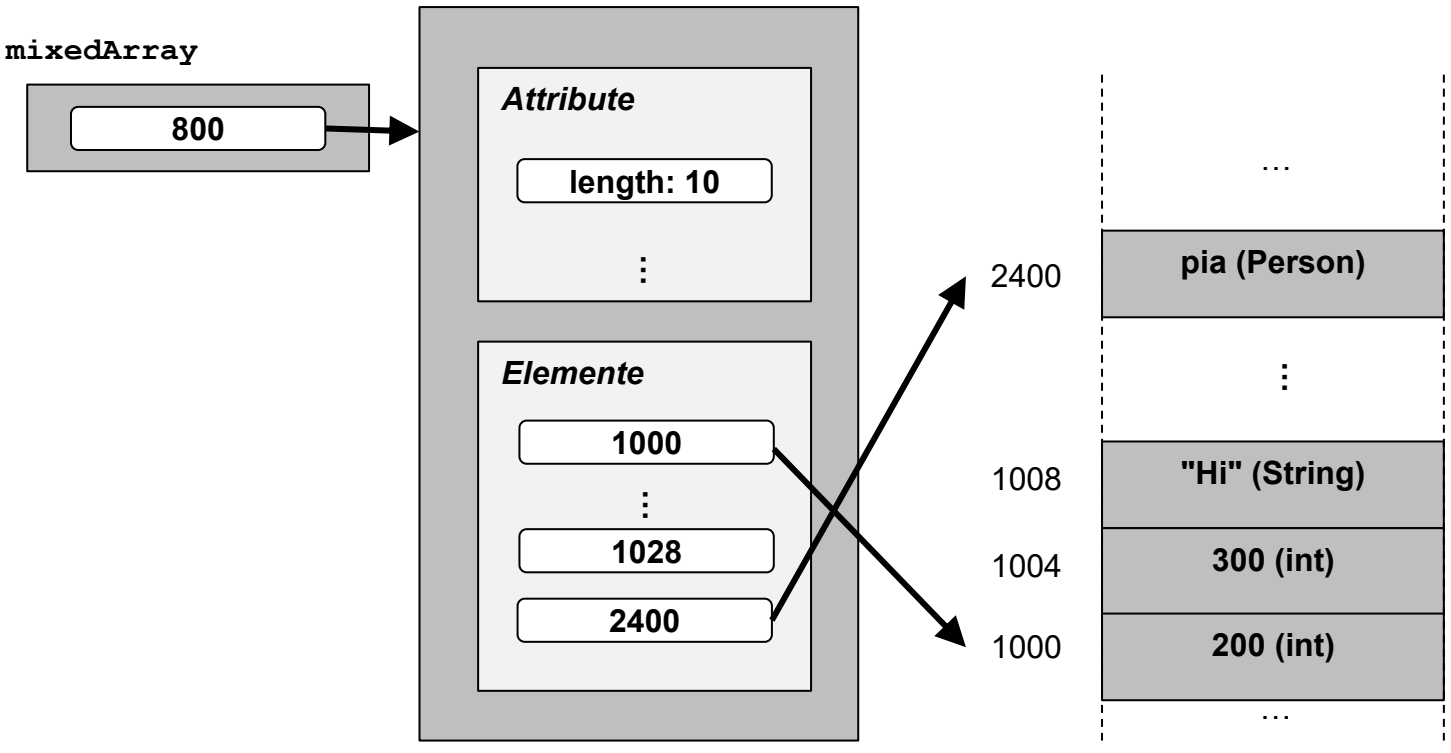
1 Speichergrößen skalarer Datentypen

z. B. mittels `printf("sizeof(double) = %ld\n", sizeof(double));`

`%ld` ist nötig, weil der Typ eines `sizeof`-Ausdrucks meistens `long` ist, nicht `int`

3 Gemischte Arrays in Java und C (0 Punkte)

a) In Java wären gemischte Arrays möglich, weil das Array-Objekt für jedes Element einen eigenen Zeiger auf dessen Speicherbereich enthält, so dass es für das Finden der Speicheradresse eines Elements nicht nötig ist, seine Größe (oder die Größe der Elemente vor ihm) zu kennen, etwa so:



Java erlaubt aber trotzdem keine gemischten Arrays und bietet auch keine Syntax dafür, weil dann bei einem variablen Array-Index (z.B. a[i]) nicht klar wäre, von welchem Typ das Element wäre. Man könnte als Ersatz für alle Typen Object wählen,müsste aber beim Zugriff immer passend Downcasten.

In C wären gemischte Arrays unmöglich, weil für den Zugriff auf ein Element (z.B. a[5]) seine Adresse aus der Adresse des ersten Elements (hier: &a[0] bzw. a) plus den Index des Elements multipliziert mit der Elementgröße errechnet wird (also die Adresse von a[0] + 5*4 bei einer Elementgröße von z.B.) – das funktioniert aber nur, weil alle Elemente dieselbe feste Größe (hier: 4) haben.

b) In Java sind unterschiedlich lange Sub-Arrays möglich, da jeder Sub-Array sein eigenes Längen-Attribut hat (s. (a)), z.B. so:

```
int[][] a = new int[][] { {1, 2}, {1, 2, 3} };
```

In C ist dies nicht möglich, da die Position jedes Sub-Arrays aus seinem Index innerhalb des Haupt-Arrays und der Länge der Elemente des Haupt-Arrays berechnet wird – das funktioniert aber nur, wenn diese Elemente (also die Sub-Arrays) alle gleich groß sind.

4 Speicherbedarf von Arrays in Java und C (0 Punkte)

C: Es werden nur die Element selbst gespeichert (ohne jede Verwaltungsinformation), also 5 x 10 x 4 Bytes = 200 Bytes.

Java: Zusätzlich zu den (wie bei C berechneten) 200 Bytes für die Nutzdaten müssen noch die Verwaltungsdaten von 5 Arrays zu je 10 `int`'s und die von 1 Array zu 5 Objekten (nämlich die 5 10er-Arrays) hinzugezählt werden. Pro Array umfassen diese Vewaltungsdaten mindestens ein `int` (4 Bytes) für die Anzahl der Elemente und pro Element von nichtelementarem Typ eine Objektreferenz zu (auf 64-Bit-Rechnern) 8 Bytes. Das sind in diesem Fall mindestens 5*(4 + 10*0) + (4 + 5*8) = 20+44 = 64 Bytes Verwaltungsdaten.

5 strcat nachprogrammieren (0 Punkte)

Bitte noch keine (Lauf-) Zeiger verwenden, da dies in der Vorlesung erst später behandelt wird, sondern nur Indizes in s1 und s2:

```
char* mystrcat(char s1[9, char s2[9) {
  int sourceOffset = 0;
  int destOffset  = strlen(s1);
  while (s2[sourceOffset] != '\0')
    s1[destOffset++] = s2[sourceOffset++];
  s1[destOffset] = '\0';
  return s1;
}
```

Hinweisen: C-typisch prüft die Funktion nicht, ob s1 genügend Platz bietet – das ist Sache des Programmierers. Falls zuwenig Platz da ist, gibt es keine Exception (wie etwa in Java die `ArrayIndexOutOfBoundsException`), sondern das Programm überschreibt ungerührt den Speicher unbeteiligter Variablen mit den überschüssigen Zeichen.

Blatt A.4

1 Alignment und Padding (0 Punkte)

a) Mit einem Beispielprogramm lässt sich die Struktur im Speicher untersuchen:

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

#include "printmemory.h"

struct beispiel
{
  short i;
  int j;
  char s[10];
};

int main(void)
{
  struct beispiel bsp;
  bsp.i = 0x0102;
  bsp.j = 0x11121314;
  strcpy(bsp.s, "XXXXXXXXX");
  printmemory(&bsp, sizeof(bsp));

  return 0;
}
```

Man erhält so beispielsweise folgende Ausgabe:

```
0x7ffc329837a0: 02 01 cf 93 14 13 12 11 58 58 58 58 58 58 58 58
0x7ffc329837b0: 58 00 98 32
```

Im Beispiel belegt `short i` die ersten beiden Bytes (`02 01`). Es folgen zwei Padding Bytes (`cf 93`). Die nächsten vier Bytes werden durch `int j` belegt (`14 13 12 11`). Das `char`-Array belegt die nächsten 10 Bytes (`58` hexadezimal ist der ASCII-Code von `X`; beachten Sie auch das `00`-Byte, das das String-Ende markiert) gefolgt von weiteren zwei Padding Bytes (`98 32`).

b) Das folgende Beispiel benötigt statt 20 nur noch 16 Bytes:

```
struct kompakt
{
  int j;
  char s[10];
  short i;
};
```

Da die short-Variable so bereits an einer durch zwei teilbaren Adresse liegt, werden keine Padding Bytes benötigt.

c) Für diese Aufgabe muss man sich überlegen, wie die einzelnen Elemente der Struktur im Speicher repräsentiert werden, und muss diesen entsprechend ihres Offsets vom Start des struct initialisieren. Für das struct aus Teil (b) sieht das so aus:

```
void fill(char u[])
{
  *(int*)u = 32168;
  *(short*)(u + sizeof(int)) = 89;
  strcpy(u + sizeof(int) + sizeof(short), "Rosi");
}
```

Mit einem kleinen Hilfsprogramm konnte man sich bei der letzten Teilaufgabe viel Arbeit ersparen:

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

```c
#include <string.h>

struct beispiel
{
  short i;
  int j;
  char s[10];
};

void print_char(char u[], int n)
{
  int i;
  printf("print_char:\n");
  for(i = 0; i < n; ++i)
  {
    printf("u[%d] = %d;\n", i, u[i]);
  }
}

void print_int(int u[], int n)
{
  int i;
  printf("print_int:\n");
  for(i = 0; i < n; ++i)
  {
    printf("u[%d] = %d;\n", i, u[i]);
  }
}

int main(void)
{
  struct beispiel bsp;
  bsp.i = 89;
  bsp.j = 32168;
  strcpy(bsp.s, "Rosi");
  print_char((char*)&bsp, sizeof(bsp));
  print_int((int*)&bsp, sizeof(bsp)/sizeof(int));
  return 0;
}
```

2 Opake Datentypen (0 Punkte)

Damit das Schlüsselwort `struct` weggelassen werden kann, könnte man folgende Zeile einfügen:

```c
typedef struct user user;
```

So wird `user` zu einem Alias für `struct user`.
Alternativ wäre auch folgende Deklaration der Struktur möglich, bei der ein namenloser Strukturtyp deklariert wird und den alias `user` erhält:

```c
typedef struct
{
  char name[10];
  int admin;
} user;
```

5 Array-Funktion erkennen (0 Punkte)

`char* foo(char* s1, char* s2)` liefert einen Zeiger auf das erste Vorkommen von `s2` innerhalb von `s1` oder `NULL`, falls `s2` nirgends in `s1` enthalten ist. Diese Funktion ist unter dem Namen `strstr(char* s1, char* s2)` in der Standard-Bibliothek enthalten.
Die gezeigte Implementierung ist die einfachste, aber nicht die effizienteste.

10 Funktion zum Finden eines Zeichens in einem String (0 Punkt)

```
#include <string.h>

char* find(char s[], char c) {
  char* p;
  if (*s == '\0')
    return NULL;
  for (p = s + strlen(s) - 1;
       *p != c && p >= s;
       --p)
    ;
  if (*p == c)
    return p;
  else
    return NULL;
}
```

Diese Funktion ist unter dem Namen `strrchr` in der Standard-Bibliothek enthalten.

13Zeiger (0 Punkte)

a) 2x die Adresse des ersten Elements des Arrays
b) 2x die Adresse der Zeichenkette
c) `H a`
d) `l a`
e) `a a ( 1[p]` entspricht `*(1+p) = *(p+1) = p[1] )`
f) `10 1`
g) `2`
h) `2` und undefinierter Wert (uninitialisierter Speicherinhalt genau ein `int` hinter dem Array)

14 Operationen auf Zeichenketten (0 Punkte)

Eine mögliche Lösung könnte so aussehen:

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

/*
 * The strlen() function calculates the length of the string pointed to by s,
 * excluding the terminating null byte ('\0').
 */
size_t mystrlen(const char *s)
{
  size_t i = 0;
  while (*(s++) != '\0')
    ++i;
```

```
  return i;
}

/*
 * The strcpy() function copies the string pointed to by src, including the
 * terminating null byte ('\0'), to the buffer pointed to by dest.
 * The strings may not overlap, and the destination string dest must be
 * large enough to receive the copy.
 * The strcpy() function returns a pointer to the destination string dest.
 */
char *mystrcpy(char *dest, const char *src)
{
  char *ret = dest;
  while (*src != '\0')
  {
    *dest = *src;
    dest++; src++;
  }
  *dest = 0;
  return ret;
}

/*
 * The strcat() function appends the src string to the dest string,
 * overwriting the terminating null byte ('\0') at the end of dest, and then
 * adds a terminating null byte. The strings may not overlap, and the dest
 * string must have enough space for the result.
 * The strcat() function returns a pointer to the resulting string dest.
 */
char *mystrcat(char *dest, const char *src)
{
  char *ret = dest;
  while(*dest != '\0')
    dest++;
  while (*src != '\0')
  {
    *dest = *src;
    dest++; src++;
  }
  *dest = 0;
  return ret;
}

/*
 * The strrchr() function returns a pointer to the last occurrence of the
 * character c in the string s. Here "character" means "byte".
 * The strrchr() function returns a pointer to the matched character or NULL if
 * the character is not found. The terminating null byte is considered part of
 * the string, so that if c is specified as '\0', these functions return a
 * pointer to the terminator.
 */
char *mystrrchr(char *s, int c)
{
  char *last = NULL;
  while (*s)
  {
    if (*s == c)
      last = s;
    s++;
  }
  if (c == '\0')
    last = s;
  return last;
}
```

```c
/*
 * The strstr() function finds the first occurrence of the substring needle in
 * the string haystack. The terminating null bytes ('\0') are not compared.
 * These functions return a pointer to the beginning of the located substring,
 * or NULL if the substring is not found.
 */
char *mystrstr(char *haystack, char *needle)
{
  char *last = NULL;
  while (*haystack != '\0')
  {
    char *it1 = haystack;
    char *it2 = needle;
    while (*it2 != '\0')
    {
      if (*it2 != *it1)
        break;
      it1++; it2++;
    }
    if (*it2 == '\0')
      last = haystack;
    haystack++;
  }
  return last;
}


int main(void)
{
  char s1[] = "Dies ist ein Teststring!";
  char s2[] = "Und dies noch einer.";
  char buffer1[2000];
  char buffer2[2000];

  printf("mystrlen:  %s\n", mystrlen(s1) == strlen(s1) ? "Richtig" : "Falsch");

  mystrcpy(buffer1, s1);
  printf("mystrcpy:  %s\n", strcmp(buffer1, s1) == 0 ? "Richtig" : "Falsch");

  mystrcpy(buffer1, s1);
  mystrcat(buffer1, s2);
  strcpy(buffer2, s1);
  strcat(buffer2, s2);
  printf("mystrcat:  %s\n", strcmp(buffer1, buffer2) == 0 ? "Richtig" : "Falsch");

  printf("mystrrchr: %s\n",
      mystrrchr(s1, 'i') == strrchr(s1, 'i') ? "Richtig" : "Falsch");

  printf("mystrstr:  %s\n",
      mystrstr(s1, "Test") == strstr(s1, "Test") ? "Richtig" : "Falsch");

  return 0;
}
```

Blatt A.5

3 Strukturen mit Zeigern auf Strukturen

Musterlösung:

```c
/* graph.c */

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


#define FNAME "graph.txt"
#define MAX_KNOTEN 100
#define MAX_NAME_LENGTH 101

struct knoten {
  char name[MAX_NAME_LENGTH];
  struct knoten *parent;
};


int main(void) {


    int read, numlines= 0, i;
    struct knoten baum[MAX_KNOTEN];  /* Alle Knoten */
    char parentname[MAX_KNOTEN][MAX_NAME_LENGTH];  /* Namen der parents */

    do {
      read= scanf("%s %s", baum[numlines].name, parentname[numlines]);
      numlines++;
    } while (read > 0);
    numlines--;

    for (i=0; i<numlines; i++) {
      /* Wir wollen die Komponente baum[i].parent initialisieren */
      int j;
      char* p= parentname[i]; /* Das ist der Name des Parent */
      /* Wir suchen nun die Adresse des Knotens mit Name p */
      for (j= 0; j<numlines; j++) {
            if (strcmp(p, baum[j].name) == 0) {
                  baum[i].parent= &(baum[j]);
                  break;
            }
      }
      if (baum[i].parent == NULL && strcmp(p, "<none>") != 0) {
            printf("Fehler: übgeordneter Knoten zu %s nicht gefunden: %s\n",
baum[i].name, p);
      }
    }

    printf("Anzahl knoten: %d\n", numlines);
    for (i=0; i<numlines; i++) {
      char* parentname= "-";
      if (baum[i].parent != NULL) {
            parentname= baum[i].parent->name;
            printf(" ");
      } else {
            printf("*");
      }
      printf("%s -> %s\n", baum[i].name, parentname);
    }



    return 0;
}
```

Inhalt der Datei `graph.txt`:

```
B K
X K
K M
M <none>
J U
U M
L K
```

16 Buffer Overflow minimal

a) Es müssen mindestens 13 Zeichen eingegeben werden, damit in `u.admin` ein Wert ungleich 0 steht. Da hinter `name` noch zwei Padding Bytes eingefügt werden, damit `admin` an einer durch 4 teilbaren Speicheradresse liegt, müssen auch diese beiden Bytes überschrieben werden. Deshalb sind 11 Zeichen nicht ausreichend.

b) Das geänderte Programm könnte wie folgt aussehen. Beachten Sie, dass `fgets` auch das Zeichen für das Zeilenende einliest, welches anschließend überschrieben werden muss.

```c
#include <stdio.h>

struct user
{
  char name[10];
  int admin;
};

int main(void)
{
  int i;
  struct user u = { "", 0 };
  printf("Name: ");

  fgets((char*)&u.name, 10, stdin);
  for (i = 0; i < 10; ++i)
    if (u.name[i] == '\n')
      u.name[i] = '\0';

  printf("Hallo %s!\n", u.name);
  if (u.admin)
        printf("Gratulation! Sie sind Admin!\n");
  return 0;
}
```

17 Zeichenkette als verkettete Liste

Eine mögliche Lösung könnte so aussehen:

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

struct character
{
  char c;
```

```
  struct character* next;
};
typedef struct character string;

string* create_string(char* str)
{
  string* first = NULL;
  struct character* current = NULL;
  struct character* previous = NULL;
  while (*str != '\0')
  {
    current = (struct character*)malloc(sizeof(struct character));
    current->c = *(str++);
    current->next = NULL;
    if (previous != NULL)
      previous->next = current;
    previous = current;
    if (first == NULL)
      first = current;
  }
  return first;
}

void delete_string(string* str)
{
  while (str != NULL)
  {
    struct character* next = str->next;
    free(str);
    str = next;
  }
}

void print_string(string* str)
{
  while (str != NULL)
  {
    printf("%c", str->c);
    str = str->next;
  }
}

int main(void)
{
  string* test = create_string("Hallo String!\n");
  print_string(test);
  return 0;
}
```

Blatt A.6

3 Probleme mit Zeigern

d) Die Funktion `func1` gibt einen Zeiger auf eine lokale Variable zurück, welche nach dem Beenden der Funktion keine Gültigkeit mehr hat.
Die Funktion `func2` verwendet einen nicht initialisierten Zeiger.
Die Funktion `func3` ist prinzipiell in Ordnung. Die Verantwortung für den allokierten Speicher wird über den zurückgegebenen Zeiger an die aufrufende Funktion

weitergegeben, d.h. der Aufrufer von `func3` muss daran denken, diesen Speicher irgendwann wieder mittels `free` freizugeben.

e) Die Funktion `func` erhält zwar den Wert der Zeigervariablen `a`, kann diese aber nicht verändern (Call by Value). Soll die Zeigervariable `a` verändert werden, muss ein Zeiger auf diese übergeben werden (Call by Reference):

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

void func(int** p)
{
  *p = (int*)malloc(sizeof(int));
}

int main()
{
  int *a;
  func(&a);
  *a = 42;
  printf("%d\n", *a);
  free(a);
  return 0;
}
```

Außerdem kann der in func reservierte Speicher nie mehr freigegeben werden, weil der Zeiger darauf (nicht aber der Speicherblock selbst) verloren geht, wenn `func` endet.

f) Zu Beginn der Funktion `f` zeigt `p` auf `i` und `q` auf `j`. Nach der Anweisung `p = q;` zeigt auch auf `p` auf `j`. Durch die Anweisung `*p = 2;` wird also der Wert von `j` verändert. Die Ausgabe ist also `0 2`.

18Stackframes

Es ist zu beachten, dass die folgende Lösung nur ein idealisiertes Beispiel sein kann. Die tatsächliche Struktur des Stacks hängt von vielen Faktoren (z. B. Prozessor, Betriebssystem, Compiler, …) ab. In diesem Beispiel wird angenommen, dass Zeiger und int jeweils vier Bytes und short zwei Bytes belegen.

Stackframe am Punkt A:

Adresse	Name	Wert
1998	s[1].i	undefiniert
1997	Padding Byte	undefiniert
1994	s[1].c	undefiniert
1992	s[0].i	52
1991	Padding Byte	undefiniert
1988	s[0].c	„Hi“

Stackframe am Punkt B:

Adresse	Name	Wert
1998	s[1].i	99
1997	Padding Byte	undefiniert
1994	s[1].c	„Ho“
1992	s[0].i	52
1991	Padding Byte	undefiniert
1988	s[0].c	„Hi“
1984	s (lokale Variable in f)	1994

Stackframe am Punkt C:

Adresse	Name	Wert
1998	s[1].i	99
1997	Padding Byte	undefiniert
1994	s[1].c	„Ho“
1992	s[0].i	52
1991	Padding Byte	undefiniert
1988	s[0].c	„Hi“
1984	s (lokale Variable in f)	1994
1980	p (lokale Variable in g)	1998
1976	i (lokale Variable in g)	79

Stackframe am Punkt D:

Adresse	Name	Wert
1998	s[1].i	79
1997	1 Padding Byte	undefiniert
1994	s[1].c	„Ho“
1992	s[0].i	52
1991	1 Padding Byte	undefiniert
1988	s[0].c	„Hi“

19Speicherorte

Die Variable `i` befindet sich im Datensegment. Die Variablen `j` und `k` befinden sich auf dem Stack, wobei `k` auf einen Bereich im Heap zeigt.

21qsort-Bibliotheksfunktion benutzen

Gemeinsame Lösung für (a) und (b):

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

#define N_ELEMS 100

int randomNumber(int hi) { /* Rückgabe in {0,1,...,hi-1} */
    const double scale = rand()/((double)RAND_MAX + 1.0);
    int i = (int)(scale * hi);
    /* falls wegen Rundungsfehlern der gewÃ¼nschte Wertebereich
       ueberschritten wurde: eingrenzen */
    return (i >= hi ? hi - 1 : i);
}

int intCompare(const void* i1p, const void* i2p) {
  return *(int*)i1p - *(int*)i2p;
}

typedef struct {
  char name[15];  /* Platz fuer Schmidt_1234 */
```

```
  int gehalt;
} angestellter;

int angCompare(const void* a1p, const void* a2p) {
  return ((angestellter*)a1p)->gehalt - ((angestellter*)a2p)->gehalt;
}

int main(void) {
  int numbers[N_ELEMS];
  angestellter angs[N_ELEMS];
  int i;
  char nAsString[] = "0000";

  srand( (unsigned int)time(NULL) );

  for (i = 0; i < N_ELEMS; ++i) {
    numbers[i] = 1 + randomNumber(1000);
  }
  qsort(numbers, N_ELEMS, sizeof(int), &intCompare);
  for (i = 0; i < N_ELEMS; ++i) {
    printf("%d\n", numbers[i]);
  }

  for (i = 0; i < N_ELEMS; ++i) {
    strcpy(angs[i].name, "Schmidt_");
    sprintf(nAsString,  "%d", randomNumber(1000));
    strcat(angs[i].name, nAsString);
    angs[i].gehalt = 2000 + randomNumber(4001);
  }
  qsort(angs, N_ELEMS, sizeof(angestellter), &angCompare);
  for (i = 0; i < N_ELEMS; ++i) {
    printf("%s mit Gehalt %d\n",angs[i].name, angs[i].gehalt);
  }

  return 0;
}
```

22Opaker Typ für Vektoren

a)

```
/* DVektor.h */

struct DVektor_s;
typedef struct DVektor_s* DVektor;

extern DVektor create(double data[], int nElems);

extern DVektor copy(DVektor original);

extern void    delete(DVektor v);

extern int     add(DVektor destination, DVektor source);

extern void    process(DVektor v, void (*f)(double* elemPtr));


/* DVektor.c */

#include "DVektor.h"
```

```c
#include <stdlib.h>

struct DVektor_s {
  int nElems;
  double* data;
};

DVektor create(double data[], int nElems) {
  int i;
  DVektor this = (DVektor)malloc(sizeof(struct DVektor_s));
  this->nElems = nElems;
  this->data = (double*)malloc(nElems * sizeof(double));
  for (i = 0; i < nElems; ++i)
    this->data[i] = data[i];
  return this;
}

DVektor copy(DVektor original) {
  return create(original->data, original->nElems);
}

void delete(DVektor v) {
  free(v->data);
  v->data = NULL;  /* Just to make sure */
  free(v);
}

int add(DVektor destination, DVektor source) {
  int i;
  if (destination->nElems != source->nElems)
    return 0;
  for (i = 0; i < destination->nElems; ++i)
    destination->data[i] += source->data[i];
  return 1;
}

void process(DVektor v, void (*f)(double* elemPtr)) {
  int i;
  for (i = 0; i < v->nElems; ++i)
    (*f)(&v->data[i]);
}
```

b)

```c
/* DVektor_sum.c */

#include <stdio.h>
#include "DVektor.h"

void printDouble(double* elemPtr) {
  printf("%f ", *elemPtr);
}

double runningSum = 0;
void sumElem(double* elemPtr) { runningSum += *elemPtr; }
double sum(DVektor v) {
   runningSum = 0;
   process(v, sumElem);
   return runningSum;
}


int main(void) {
  double data[] = {1.1, 2.2, 3.3};
  DVektor v = create(data, 3);
```

```
  sum(v);
  printf("Summe von ");
  process(v, printDouble);
  printf(" = %f\n", runningSum);
  return 0;
}
```

c)

```
/* Vektor.h */

struct Vektor_s;
typedef struct Vektor_s* Vektor;

extern Vektor create(char data[], int nElems, int elemSize);

extern Vektor copy(Vektor original);

extern void   delete(Vektor v);

extern void   process(Vektor v, void (*f)(char* elemPtr));
```

```
/* Vektor.c */

#include "Vektor.h"
#include <stdlib.h>
#include <string.h>

struct Vektor_s {
  int nElems;
  int elemSize;
  char* data;
};

Vektor create(char data[], int nElems, int elemSize) {
  Vektor this = (Vektor)malloc(sizeof(struct Vektor_s));
  this->nElems = nElems;
  this->elemSize = elemSize;
  this->data = (char*)malloc(nElems * elemSize);
  memcpy(this->data, data, nElems * elemSize);
  return this;
}

Vektor copy(Vektor original) {
  return create(original->data, original->nElems, original->elemSize);
}

void delete(Vektor v) {
  free(v->data);
  v->data = NULL;  /* Just to make sure */
  free(v);
}

void process(Vektor v, void (*f)(char* elemPtr, int elemSize)) {
  int i;
  for (i = 0; i < v->nElems; ++i)
    (*f)(&v->data[i * v->elemSize], v->elemSize);
}
```

d) Der Typ von `process` 's Parameter `f` enthält nicht den Elementtyp (denn `char` steht ja nur als "Vertreter" für das erste Byte des Elements), sondern nur die Elementgröße.

Daher kann der Compiler nicht verhindern, dass man `process(v, f)` auf einem `v` mit einem anderen Elementtyp aufruft als `f` erwartet.

Blatt A.7

3 Textdatei lesen, stdout und stderr

Musterlösung:

```c
/* gedicht.c */

#include <stdio.h>
#include <errno.h>
#include <string.h>

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

    FILE* infile;
    int c;

    if (argc != 2) {
        printf("Benutzung: %s <dateiname>     oder\n", argv[0]);
        printf("Benutzung: %s -\n", argv[0]);
        return -1;
    }


    if (strcmp(argv[1], "-") == 0) {
        infile= stdin;
    } else {
        errno= 0;
        infile= fopen(argv[1], "r");

        if (infile == NULL) {
            fprintf(stderr, "%s\n", strerror(errno));
            return -1;
        }
    }

    while ((c= fgetc(infile)) != EOF) {
        if (c == 0xF6) {
            printf("oe"); /* beispielhafter Vergleich von ints */
            continue;
        }
        char ch= (char)c;
        switch (ch) {
            case '\xE4': printf("ae"); break; /* Angabe des Codes */
            case 'ü': printf("ue"); break;
            case 'ß': printf("ss"); break;
            case 'Ä': printf("Ae"); break;
            case 'Ö': printf("Oe"); break;
            case 'Ü': printf("Ue"); break;
            default: putchar(c);
        }
    }

    return 0;
}
```

Testen, ob wirklich die Standardfehlerausgabe gewählt wurde durch Umlenken, in der bash z. B. so:

```
./a.out erlkoenig.txt  >output.txt 2> error.txt
```

5 Einfach verkettete Liste

a)

```
void anwenden(struct knoten* kopf, void (*f)(int)) {
    struct knoten* laufzeiger = kopf;
    while (laufzeiger != NULL) {
        (*f)(laufzeiger->wert);
        laufzeiger = laufzeiger->next;
    }
}
```

b) Ja:

```
void ausgeben(int i) {
   printf("%d", i);
}
anwenden(liste, &ausgeben);
```

c)

```
void verketten(struct knoten** kopf1Ptr, struct knoten* kopf2) {
    struct knoten *laufzeiger;
    if (kopf1Ptr == NULL)
      return; /* Falsches Argument! */
    if (kopf2 == NULL)
      return;  /* Nichts zu tun */
    if (*kopf1Ptr == NULL) {
      *kopf1Ptr = kopf2;  /* An leere Liste anhaengen ist Ersetzen */
      return;
    }
    laufzeiger = *kopf1Ptr;
    while (laufzeiger->next != NULL) {
        laufzeiger = laufzeiger->next;
    }
    laufzeiger->next = kopf2;
}
```

Blatt B.1

5 Konstruktion und Übergabe von Objekten (0 Punkte)

Der entscheidende Punkt ist die Parameterübergabe bei foo und bar: a1 wird jeweils als Kopie übergeben; dabei wird der globale Zähler der Objekt-IDs inkrementiert, und Änderungen an a1 wirken sich nicht auf main() aus. a2 dagegen wird jeweils als Referenz übergeben; dabei wird zwar durch die Übergabe noch nichts geändert, aber weitere Änderungen an a2 wirken sich auf main() aus.

```
#include <iostream>
```

```
using namespace std;

class A {
public:
  int data;
  int id;
  static int counter;

  A() {
    data = 0;
    id = counter++;
  }

  A(const A& other) {
    data = other.data;
    id = counter++;
  }
};

int A::counter {0};

void foo(A a1, A& a2) {  // a1 = (0, 2), a2 = (0, 1)
    ++a2.data;           // a1 = (0, 2), a2 = (1, 1)
    a1 = a2;             // a1 = (1, 1), a2 = (1, 1)
    cout << a1.data << a1.id << a2.data << a2.id << endl;
}

void bar(A a1, A& a2) { // a1 = (0, 3), a2 = (1, 1)
    ++a1.data;          // a1 = (1, 3), a2 = (1, 1)
    a2 = a1;            // a1 = (1, 3), a2 = (1, 3)
    cout << a1.data << a1.id << a2.data << a2.id << endl;
}

int main(void) {
  A x, y;    // x = (0, 0), y = (0, 1)
  foo(x, y); // x = (0, 0), y = (1, 1)
  bar(x, y); // x = (0, 0), a2 = (1, 3)
  cout << x.data << x.id << y.data << y.id << endl;

  return 0;
}
```

Zeitpunkt	Wert von `x` bzw. `a1`		Wert von `y` bzw. `a2`	
	`data`	`id`	`data`	`id`
Nach Konstruktion von x und y	0	0	0	1
Nach Eintritt in `foo` (direkt hinter `{`)	0	2	0	1
Vor Rückkehr aus `foo` (direkt vor `}`) (entspricht 1. Ausgabe)	1	1	1	1
Nach Eintritt in `bar`	0	3	1	1
Vor Rückkehr aus `bar` (entspricht 2. Ausgabe)	1	3	1	3
Nach Rückkehr aus `bar` (entspricht 3. Ausgabe)	0	0	1	3

6 Unerwartete Objekte

Der Kredit wird zweimal ausgezahlt, weil zwei `Kredit`-Objekte erzeugt werden, und jedes der beiden zahlt in seinem Destruktor den Kredit zurück. Das erste Kreditobjekt namens `meinKredit` wird sichtbar im Quelltext von `main()` erzeugt. Bei der Übergabe von `meinKredit` an `nutze()` wird aber `meinKredit` kopiert, da der Parameter vom Typ `Kredit` (also call-by-value) ist und nicht etwa vom Typ `Kredit&` ist. Beim Kopieren wird aus `meinKredit` ein zweites `Kredit`-Objekt namens `kredit` mit Hilfe des Default-Kopierkonstruktors von `Kredit` erzeugt (der Default-Kopierkonstruktor kopiert einfach sämtliche Attribute).

Der Fehler liegt also im falschen Parametertyp: Da beim Nutzen eines Kredits kein neuer Kredit erzeugt werden soll, sondern der übergebene Kredit genutzt werden soll, muss der Parametertyp `Kredit&` lauten. Zusätzlich könnte man zur Sicherheit noch verhindern, dass `Kredit`-Objekte kopiert werden können, indem man den Copy-Konstruktor entweder `private` definiert (dann kann er in `main()` oder in `main()` nicht mehr aufgerufen werden):

```
private:
   Kredit(const Kredit& other) { }
```

oder ihn explizit als nicht vorhanden ("`= delete`") definiert (dieses Merkmal von C++ wurde nicht in der Vorlesung erwähnt):

```
public:
   Kredit(const Kredit& other) = delete;
```

Die `delete`-Variante ist noch sicherer als die `private`-Variante, da so der Copy-Konstruktor nirgends (auch nicht innerhalb von `Kredit`) aufgerufen werden kann.

7 C++ und Speicherklassen

a) MyInt(1) constructed.
Right before MyInt local
MyInt(2) constructed.
MyInt(3) constructed.
MyInt(2) constructed.
foo invoked
MyInt(2) destructed.
MyInt(4) constructed.
foo invoked
MyInt(4) destructed.
Cleaning up now …
MyInt(3) destructed.
MyInt(2) destructed.
MyInt(1) destructed.

Die beiden foo-Aufrufe unterscheiden sich dadurch, dass beim ersten der Copy-Konstruktor aufgerufen wird (um local nach m zu kopieren) und beim zweiten nicht (weil das anonym konstruierte Objekt direkt an m gebunden wird: Ein Kopieren wäre unnötig, da das anonyme Objekt – anders als local – nirgendwo außerhalb von foo verwendet werden kann; deshalb wird es "direkt" in foo übernommen).

b) Die Typen passen nicht: foo erwartet ein MyInt, aber new MyInt liefert einen MyInt*. Man sieht am Vergleich mit der Zeile darüber, dass es durchaus möglich ist, auch auf

dem Stack als anonymes Objekt zu erzeugen (so wie new ein anonymes Objekt auf dem Heap erzeugt).

8 Verbesserung der Ort-Klasse aus der Vorlesung

```
#include <cstring>
#include <cstdlib>
#include <iostream>
using namespace std;

class Ort {
private:
  double breite; // Breitengrad
  double laenge; // Laengengrad
  double hoehe;  // Hoehe ueber NN in m
  string name;   // Ortsname
public:
  double getBreite()     { return breite; }
  double getLaenge()     { return laenge; }
  double getHoehe()      { return hoehe; }
  const string getName() { return name; }
  Ort(double breite, double laenge, double hoehe, const string& name) {
    this->breite= breite; this->laenge= laenge; this->hoehe= hoehe;
    this->name= name;
  }
  Ort(double breite, double laenge, const string& name) {
    // Alternativ mit Initialisierungsliste:
    // breite {breite}, laenge {laenge}, hoehe(0.0), name {name} { }
    this->breite= breite; this->laenge= laenge; this->hoehe= 0.0;
    this->name= name;
  }

  // Default Copy-Konstruktor (ruft string::string(const string& s)
  // auf) genuegt jetzt :-)

  // Default-Destruktor (ruft string::~string() auf) genuegt jetzt :-)

  void print() {
    cout << name << " (" << breite << "°B, " << laenge << "°L, " << hoehe <<
         " m ü. NN)" << endl;
  }
};

int main() {
  Ort hannover {52.3667, 9.71667, 55.0, "Hannover"};
  Ort hannover2 {hannover};
  Ort rio {-22.9, -43.23333, 32.0, "Rio de Janeiro"};
  hannover.print();
  hannover2.print();
  rio.print();
}
```

Blatt B.2

4 Vector

Musterlösung:

```
// IntBig.cpp

#include <cstring>
#include <iostream>
#include <vector>
using namespace std;


class IntBig {
private:
  vector<char> c;  // Einerstelle an Index 0
public:
  IntBig(const char* val);
  // IntBig(const IntBig& other);             // unnoetig: siehe unten
  // IntBig& operator=(const IntBig& other);  // unnoetig: siehe unten
  size_t size() const;                 // size_t überall, da Typ von vector::size()
  char get_digit(size_t pos) const;
  void set_digit(size_t pos, char digit);
  IntBig& operator+=(const IntBig& summand2);
  void print(ostream& os = cout) const;
};


IntBig::IntBig(const char* val) {
  int len = strlen(val);
  for (int i = 0; i < len; i++) {
    c.push_back(val[len - i - 1]);
  }
}

/*
 * NICHT noetig, da die Default-Copykonstruktor den von vector aufruft
 * (also vector::vector(const vector&), und das ist alles, was IntBig benoetigt:
 * IntBig::IntBig(const IntBig& other) {
 * *this = other;
}
*/

/*
 * NICHT noetig, da die Default-Zuweisung vector::operator=() aufruft,
 * und das ist alles, was IntBig benoetigt:
 * IntBig& IntBig::operator=(const IntBig& other) {
 *   c = other.c;
 *   return *this;
 * }
*/

// liefert eine Zahl zwischen 0 und 9
char IntBig::get_digit(size_t pos) const {
  if (pos >= c.size())
    return 0;
  return (c.at(pos) - (char) '0');
}

void IntBig::print(ostream& os) const {
  for (int i= c.size()-1; i>=0; i--) {
    os << c.at(i);
```

```
  }
}

IntBig& IntBig::operator+=(const IntBig& summand2) {
  if (this == &summand2)  return *this;
  size_t len = size();
  if (len < summand2.size())
    len = summand2.size();

  int uebertrag = 0;
  for (size_t i = 0; i < len; i++) {
    int sum = get_digit(i) + summand2.get_digit(i) + uebertrag;
    if (sum > 9)
      uebertrag = 1;
    else
      uebertrag = 0;
    set_digit(i, sum % 10);
  }
  return *this;
}

IntBig operator+(const IntBig& summand1, const IntBig& summand2) {
  IntBig result(summand1);
  result += summand2;
  return result;
}

ostream& operator<<(ostream& os, const IntBig& i) {
  i.print(os);
  return os;
}
```

```
bool operator<(const IntBig& a, const IntBig& b) {
  size_t len = a.size();
  if (len < b.size())
    len = b.size();
  for (size_t i = len - 1;
       /* i >= 0 would be useless, since a size_t is always >= 0 */;
       i--) {
    if (a.get_digit(i) != b.get_digit(i)) {
      return a.get_digit(i) < b.get_digit(i);
    }
    if (len == 0)
      return false;
  }
  return false;
}
```

```
// liefert 20 für eine 20-stellige Zahl
size_t IntBig::size() const {
  return c.size();
}
```

```
void IntBig::set_digit(size_t pos, char digit) {
  // ggf. 0en auffüllen
  while (c.size() <= pos) {
    c.push_back('0');
  }
  c[pos] = digit + (char) '0';

  // Führende 0en entfernen
  while (c.size() > 0 && c.at(c.size()-1) == '0') {
    c.pop_back();
```

```
  }
}
```

```
int main() {
  IntBig a {"10000000000000000000000000000000000000000"};
  IntBig b {"100"};
  IntBig c {"0"};
  cout << a << endl;
  cout << b << endl;
  a += b;
  cout << a << endl;
  cout << (a+b) << endl;
  cout << c << endl;
  cout << (c+b) << endl;
  cout << "a < b = " << (a < b) << endl;
  cout << "b < a = " << (b < a) << endl;
  cout << "a < a = " << (a < a) << endl;
  return 0;
}
```

5 Einfache String-Klasse implementieren

```
#include <iostream>
#include <cstring>
#include <stdexcept>

using namespace std;

class MyString {
public:
  MyString();
  MyString(const MyString& other);
  MyString(const char chars[]);
  friend ostream& operator << (ostream& out, const MyString& ms);
  int length() const { return len; }
  bool operator == (const MyString& other) const;
  char& operator [] (int index) const throw (out_of_range);
  MyString& operator += (const MyString& other);
  MyString& operator = (const MyString& other);
  MyString operator * (int n) const;
  ~MyString();
private:
  char* ptr;
  int len;
};

MyString::MyString() {
  len = 0;
  ptr = new char {'\0'};
}

MyString& MyString::operator = (const MyString& other) {
  delete[] ptr;
  len = other.len;
  ptr = new char[len + 1];
  strcpy(ptr, other.ptr);
  return *this;
}

MyString::MyString(const MyString& other) {
  ptr = nullptr;
```

```
  *this = other;
}

MyString::~MyString() {
  delete[] ptr;
}

MyString::MyString(const char chars[]) {
    len = strlen(chars);
    ptr = new char[len + 1];
    strcpy(ptr, chars);
}

ostream& operator << (ostream& out, const MyString& ms) {
  for (int i = 0; i < ms.len; ++i) {
    out << ms.ptr[i];
  }
  return out;
}

bool MyString::operator == (const MyString& other) const {
  if (len != other.len)
    return false;
  for (int i = 0; i < len; ++i) {
    if (ptr[i] != other.ptr[i])
      return false;
  }
  return true;
}

char& MyString::operator [] (int index) const throw (out_of_range) {
  if (index < 0 || index >= len)
    throw out_of_range("MyString index out of bounds");
  return ptr[index];
}

MyString operator + (const MyString& ms1, const MyString& ms2) {
  MyString result(ms1);
  result += ms2;
  return result;
}

MyString& MyString::operator += (const MyString& other) {
  int newLen = len + other.len;
  char* newPtr = new char[newLen];
  strcpy(newPtr, ptr);
  delete[] ptr;
  strcpy(newPtr + len, other.ptr);
  len = newLen;
  ptr = newPtr;
  return *this;
}

MyString MyString::operator * (int n) const {
  MyString result;
  for (int i = 0; i < n; ++i) {
    result += *this;
  }
  return result;
}

int main() {
  MyString empty {};
  cout << "The following should be empty: " << empty << endl;  // Nothing
```

```
  MyString abc {"abc"};
  cout << abc << endl;                                    // abc
  MyString abc2 {abc};
  cout << abc2 << endl;                                   // abc
  cout << abc.length() << endl;                           // 3
  cout << boolalpha << (abc == abc2) << endl;             // true
  abc[2] = 'd';
  cout << abc2 << endl;                                   // abc
  cout << boolalpha << (abc == abc2) << endl;             // false
  cout << abc << endl;                                    // abd
  abc = abc2;
  abc2[0] = 'X';
  cout << abc << endl;                                    // abc
  MyString def {"def"};
  MyString abcdef {abc + def};
  cout << abcdef << endl;                                 // abcdef
  abcdef += abc;
  cout << abcdef << endl;                                 // abcdefabd
  MyString defdefdef {def * 3};
  cout << defdefdef << endl;                              // defdefdef

  return 0;
}
```

6 Virtuelle Aufrufe in Konstruktoren

Beispielprogramm zur Verdeutlichung:

```
// virtual.cpp

#include <iostream>
using namespace std;


class A {
public:
      A();
      virtual ~A() {}
      virtual char f();
};

char A::f() {
      return 'A';
}

A::A() {
      cout << f();
}

class B : public A {
public:
      B() : A() {}
      virtual ~B() {}
      virtual char f() { return 'B'; }
};



int main(void) {
      B b;
}
```

Ausgabe ist: `A`, d. h. es wird nicht die Methode der Subklasse aufgerufen, da das Subobjekt noch gar nicht existiert.

Wenn man die Basismethode `f` pure virtual ohne Definition macht, gibt es eine Warnung und einen Linker-Fehler:

```
$ g++ -std=c++14 -pedantic-errors -Wall virtual.cpp
virtual.cpp: In constructor »A::A()«:
virtual.cpp:18: Warnung: abstract virtual »virtual char A::f()« called from constructor
virtual.cpp:(.text+0x8c): undefined reference to `A::f()'
virtual.cpp:(.text+0xbc): undefined reference to `A::f()'
```

Ein äquivalentes Java-Programm zeigt druckt dagegen `B`, d.h. die Methode `f` verhält sich auch bei Aufruf schon im Konstruktor polymorph:

```
// Virtual.java

class A {
	public A() {
		System.out.println(f());
	}

	public char f() {
		return 'A';
	}
}

class B extends A {

	public B() {
		super();
	}

	@Override
	public char f() {
		return 'B';
	}
}

public class Virtual {
	public static void main(String[] args) {
		B b = new B();
	}
}
```

Ein äquivalentes Java-Programm mit Zugriff auf ein Attribut in der Subklasse zeigt aber trotzdem ein Problem:

```
// VirtualSubclassAttribute.java

class A {
	public A() {
		System.out.println(f());
	}

	public char f() {
		return 'A';
	}
}
```

```
class B extends A {
	private char c;

	public B() {
		super();
		c = 'B';
	}

	@Override
	public char f() {
		return c;
	}
}
public class VirtualSubclassAttribute {
	public static void main(String[] args) {
		B b = new B();
	}
}
```

In Java wird zwar ein Aufruf einer Methode polymorph ausgeführt, da aber bei zum Zeitpunkt eines Aufrufs aus einem Superklassenkonstruktor das Subobjekt noch nicht vollständig konstruiert ist (lediglich die Standardinitialisierung mit 0 ist bereits erfolgt), wird das noch nicht initialisierte Subklassenattribut `B.c` ausgegeben, also das nicht druckbare Zeichen \0.

7 Polymorphie

Musterlösung:

```
// Teilnehmer.h

#ifndef TEILNEHMER_H
#define TEILNEHMER_H

class Teilnehmer {
private:
  int pos;
protected:
  void setPos(int pos);
public:
  int getPos() const;
  bool istAmZiel() const;
  Teilnehmer();
  virtual ~Teilnehmer();
  virtual void weiter() = 0;
  virtual char asChar() const = 0;
};

#endif
```

```
// Teilnehmer.cpp

#include "Teilnehmer.h"

Teilnehmer::Teilnehmer() {
  pos = 0;
}

int Teilnehmer::getPos() const {
  return pos;
}

void Teilnehmer::setPos(int pos) {
```

```
  this->pos= pos;
}

bool Teilnehmer::istAmZiel() const {
  return (pos == 99);
}

Teilnehmer::~Teilnehmer() {
}
```

```
// Igel.h
#ifndef IGEL_H
#define IGEL_H
#include "Teilnehmer.h"
class Igel : public Teilnehmer {
public:
  void weiter() override;
  ~Igel();
  char asChar() const override;
};
#endif
```

```
// Igel.cpp

#include "Igel.h"
#include <cstdlib>
#include <cmath>
#include <iostream>
using namespace std;

void Igel::weiter() {
  if (istAmZiel()) return;
  setPos(getPos()+1);
}

Igel::~Igel() {
}

char Igel:: asChar() const  {
  return 'I';
}
```

```
// Hase.h
#ifndef HASE_H
#define HASE_H
#include "Teilnehmer.h"
class Hase : public Teilnehmer {
public:
  void weiter() override;
  ~Hase();
  char asChar() const override;
};
#endif
```

```
// Hase.cpp

#include "Hase.h"
#include <random>
```

```
#include <iostream>
using namespace std;

const int WEITE {10};

void Hase::weiter() {
  if (istAmZiel()) return;

  // Wähle sprung zufällig zwischen -WEITE und WEITE
  random_device r;
  default_random_engine engine(r());
  uniform_int_distribution<int> uniform_dist(0, 2*WEITE);
  int sprung = -WEITE + uniform_dist(engine);

  int neuPos = getPos() + sprung;
  if (neuPos < 0) { neuPos = 0; }
  else if (neuPos > 99) { neuPos = 99; }
  setPos(neuPos);
}

Hase::~Hase() {
}

char Hase:: asChar() const {
  return 'H';
}
```

```
// main.cpp

#include "Teilnehmer.h"
#include "Igel.h"
#include "Hase.h"
#include <vector>
#include <iostream>
using namespace std;

void out(Teilnehmer* t) {
  for (int pos = 0; pos < 100; pos++) {
     if (t->getPos() == pos) {
       cout << t->asChar();
     } else {
      cout << '-';
    }
  }
  cout << endl;
}

void out(const vector<Teilnehmer*>& teilnehmer) {
   for (auto t : teilnehmer) {
     out(t);
  }
}

int main() {
  vector<Teilnehmer*> teilnehmer;
  teilnehmer.push_back(new Igel()); // 1 Igel gegen …
  for (int i=0; i<2; i++) {         // 2 Hasen
    teilnehmer.push_back(new Hase());
  }

  cout << "Start:" << endl;
  out(teilnehmer);
  bool ende= false;
```

```
  for (int i = 1; !ende && i <= 100; i++) {
    for (auto t : teilnehmer) {
      t->weiter();
      ende = ende || (t->istAmZiel());
    }
    cout << "Schritt " << i<< ":" << endl;
    out(teilnehmer);
  }


  for (auto t : teilnehmer) {
    delete t;
  }
  return 0;
}
```