RA FS 12 Repetitionsserie: Teil C

Samuel Bucheli, Gian Calgeer, Judith Fuog

Die Repetitionsserie dient der individuellen Prüfungsvorbereitung und muss nicht abgegeben werden. Für Fragen steht im ILIAS jederzeit ein Forum zur Verfügung.
Viel Spass!

1 C-Programmierung

1.1 Zusammengesetzte Datentypen

Wir möchten in unserem C-Programm folgende Zuweisung verwenden können:

```
person.alter = 79;
```

Geben Sie eine `struct` an, die diese Zuweisung ermöglicht.

Zusätzlich soll der Person noch Vor- und Nachnamen zugewiesen werden können. Der Vorname soll maximal 10 Zeichen lang sein können (etwa Rantanplan), der Nachname 11 Zeichen (etwa Eschenmoser).

Geben Sie auch einen Codeausschnitt an, in dem Sie eine Beispielsperson namens *Joe Darling (32)* erstellen.

1.2 Pointer (1 Punkt)

Angenommen `x` werde im untenstehenden Programm an der Adresse `0xbffff97c` initialisiert (`0x` bedeutet, dass die Adresse `bffff97c` hexadezimal ist).

Auf welche Adresse im Speicher zeigt `px` nach Ausführung des Programms? Gehen Sie von einem 32-Bit Prozessor aus.

```
1 main() {
2   short x, *px;
3   px = &x;
4   px++;
5 }
```

1.3 StringCopy (1 Punkt)

Der folgende Code-Ausschnitt zeigt eine typische Implementation der `strcpy`-Funktion. Welche Probleme können auftreten?

```
1 char *strcpy(char *s1, const char *s2) {
2   char *dst = s1;
3   const char *src = s2;
4   while( (*dst++=*src++)!='\0');
5   return s1;
6 }
```

1.4 Dynamischer Speicher

Wo liegt der Fehler in der Funktion `iSOE` in folgendem Codebeispiel? Beheben Sie dort den Fehler und beschreiben Sie, welchen Zweck die Funktion `hp` erfüllt:

```
1 int* iSOE(int limit) {
2        int soe[limit+1], i;
3        for(i=0; i<limit+1; i++) {
4               soe[i]=i;
```

```
5         }
6         return soe;
7   }
8   int hp(int limit) {
9         int cp,j,i;
10        int * soe = iSOE(limit);
11        soe[1] = 0;
12        for(i=0; i<limit+1; i++) {
13              if(soe[i]) {
14                    cp=soe[i];
15                    for(j=cp*cp; j<limit+1; j+=cp) {
16                          soe[j]=0;
17                    }
18              }
19        }
20        return cp;
21  }
```

1.5 Typedefs

Erklären Sie, wofür man `typdef` in C braucht.

1.6 Printf

Fülle die fehlenden Elemente der folgenden Tabelle ein. Die Werte werden wie folgt der `printf`-Funktion übergeben:

```
printf(ARGUMENT, EINGABE);
```

Für die erste Zeile der Tabelle lautet der Funktionsaufruf also

```
printf("%i", 10)
```

Eingabe	Argument	Ausgabe
10	%i	10
12	%x	
	%X	F2
8.7561		8.76
	|%-4s|	|abc |
	%o	200

1.7 Function Pointer

Gibt es in Java Function Pointers? Was könnten die Gründe (dafür bzw. dagegen) sein?

1.8 Struct Alignment (1 Punkt)

Dieses Programmstück gibt `12 8` aus, erklären Sie warum:

```
1  typedef struct {
2    char a1[1];
3    long b;
4    char a2[3];
5  } A;
6
7  typedef struct {
8    char a1[1];
9    char a2[3];
10   long b;
11 } B;
12
13 int main() {
14   printf("%i %i \n", sizeof(A), sizeof(B));
15   return EXIT_SUCCESS;
16 }
```

1.9 Dynamische Speicherallozierung

Gegeben sei folgende `struct`:

```
typedef struct _LinkedElement{
  char *name;
  int size;
  struct _LinkedElement *prev;
  struct _LinkedElement *next;
} LinkedElement;
```

Implementieren Sie die folgenden drei Methoden:

(a) `LinkedElement * create_object(char*, int, LinkedElement*, LinkedElement*)` Erstellt eine Neues `LinkedElement`, dabei kann ein Name, eine Grösse und ein vorheriges und ein nächstes Element angegeben werden.

(b) `LinkedElement * clone_object(LinkedElement *)` Erstellt eine Kopie eines `LinkedElement` wobei die `prev` und `next` Element nicht kopiert werden müssen (shallow copy)

(c) `void free_object(LinkedElement *)` Gibt den Speicher eines `LinkedElement` rekursiv frei. D.h. alle `prev` und `next` müssen explizit freigegeben werden.

Schreiben sie ein lauffähiges Testprogramm, welches folgende `main`-Funktion ausführen kann:

```
int main (int argc, const char * argv[]) {
  LinkedElement * el1 = create_object("el1", 2, NULL, NULL);
  LinkedElement * el12 = create_object("el12", 2, NULL, NULL);
  LinkedElement * el11 = create_object("el1", 2, el1, el12);
  LinkedElement * el2 = clone_object(el11);
  LinkedElement *el3 = create_object("el3", 3, el1, el2);
  free_object(el3);
  return 0;
}
```

1.10 Sizeof

Welche Ausgabe erzeugt das folgende Codestück?

```
1  char a[10];
2  char* b = a;
3  printf("%i %i\n",sizeof(a), sizeof(b));
```

1.11 Arrays initialisieren

Wieso kann in C eine Array-initialisierende Funktion nicht folgendes machen?

```
1 int* init() {
2   int i[32];
3   return i;
4 }
```

1.12 Pointerarithmetik

Wies gibt `int *a = address; a++;` und `char *a = address; a++;` nicht das selbe Resultat?.

1.13 Variablentypen

Welchen Typ haben die Variablen aus `int* a, b`?

1.14 Operatorenpräzedenz

In welcher Reihenfolge werden die Ausdrücke in `*&a++` evaluiert?

1.15 Best coding practices

Warum ist `if (3 == a) {}` besser als `if (a == 3) {}`?

1.16 Preprocessor Fun

Welche Ausgabe erzeugt das folgende Codestück?

```
1 #define MIN(a,b) ((a) < (b) ? (a) : (b))
2
3 int a=3;
4 int b=4;
5 int m=MIN(a++,b++);
6 printf("min(%i, %i) = %i\n", a, b, m);
```

1.17 Datenstrukturen in C

Um was für eine Struktur handelt es sich bei `tr`? Stellen Sie `tr* a` graphisch dar und beschreiben Sie Output und Ablauf des folgenden Programms

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  typedef struct tr_ {
5    char n;
6    struct tr_ *l;
7    struct tr_ *r;
8  } tr;
9
10 tr* ct(char n, tr *l, tr *r) {
11    tr *t = malloc(sizeof(t));
12    t->n = n; t->l = l; t->r = r;
13    return t;
14 }
15
16 void rec(tr *t, void (*o)(tr *)) {
17    if (t) {
18       tr *l = t->l;
19       tr *r = t->r;
20       o(t);
21       rec(l, o);
22       rec(r, o);
23    }
```

```
24 }
25
26 void p(tr *t) {
27   printf("%c", t->n);
28 }
29
30 void s(tr *t) {
31   tr* tmp = t->l;
32   t->l = t->r;
33   t->r = tmp;
34 }
35
36 void fr(tr *t) {
37   free(t);
38 }
39
40 int main() {
41   tr* g = ct('g',NULL,NULL);
42   tr* f = ct('f',NULL,NULL);
43   tr* e = ct('e',NULL,NULL);
44   tr* d = ct('d',NULL,NULL);
45   tr* c = ct('c',f,g);
46   tr* b = ct('b',d,e);
47   tr* a = ct('a',b,c);
48   rec(a, &p);
49   printf("\n");
50   rec(a, &s);
51   rec(a, &p);
52   printf("\n");
53   rec(a, &fr);
54   return EXIT_SUCCESS;
55 }
```

1.18 Fehlersuche

Das folgende Programm soll mittels der Methode `init_ptr` für ein Stringarray der Länge 2 Speicher allozieren und mit den Werten "Test1" bzw. "Test2" initialisieren, anschliessend sollen diese beiden Werte mittels einer For-Schlaufe ausgegeben werden.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main(int argc, char** argv) {
6          char *ptr[2]={NULL, NULL};
7          init_ptr(ptr);
8          for (int i=0; i=2; i++) {
9                  printf("%s\n", *ptr+i);
10         }
11         return EXIT_SUCCESS;
12 }
13
14 int init_ptr(**data) {
15         *data=malloc(5);
16         *(data+1)=malloc(5);
17         strcpy(*data,"Test1");
18         strcpy(*data+1,"Test2");
19 }
```

Der Versuch, das Programm mittels `gcc -ansi -pedantic -Wall program.c` zu kompilieren, ergibt folgende Meldungen:

```
program.c: In function ‘main’:
```

```
program.c:7: warning: implicit declaration of function ‘init_ptr’
program.c:8: error: ‘for’ loop initial declaration used outside C99 mode
program.c:8: warning: suggest parentheses around assignment used as truth value
program.c: At top level:
program.c:14: error: expected declaration specifiers or ‘...’ before ‘*’ token
program.c: In function ‘init_ptr’:
program.c:15: error: ‘data’ undeclared (first use in this function)
program.c:15: error: (Each undeclared identifier is reported only once
program.c:15: error: for each function it appears in.)
program.c:19: warning: control reaches end of non-void function
```

(a) Schreiben Sie den Code so um, dass er ohne Fehler und Warnungen kompiliert werden kann. Versuchen Sie, so wenig Änderungen wie möglich vorzunehmen und halten Sie sich an Empfehlungen des Compilers.

(b) Nachdem der Code durch die Anpassungen kompilierbar gemacht wurde, stellen Sie fest, dass es bei der Ausführung zu einer Endlosschleife kommt, dabei wird andauernd eine Zeile mit dem String “est2” ausgegeben. Mittels eines Debuggers stellen Sie zudem fest, dass nach der Ausführung von `init_ptr` die Speicherbelegung `ptr = ‘‘Ttest2’’, ‘‘’’` gilt. Geben Sie an, welche Zeilen im kompilierbaren Code wie zu verändern sind.

(c) Überprüfen Sie Ihren Code auf weitere allfällige Fehler und geben Sie Korrekturen an.

1.19 More Preprocessor Fun

Das folgende Programm gibt den Output 3. Wo liegt der Fehler und wie kann man ihn beheben, so dass das Programm den gewünschten Output 4 liefert?

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MULT(a,b) a*b
4
5  int main(int argc, char** argv) {
6    int a=1; int b=1;
7    int c=1; int d=1;
8    printf("%i\n", MULT(a+b, c+d));
9    return EXIT_SUCCESS;
10 }
```

1.20 Mehr Fehlersuche

Das folgende Programm gibt den Output `(null) (null), 0`. Wo liegt der Fehler? Korrigieren Sie die Fehler, so dass das Programm die intendierte Funktion erbringt.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #define MAX_STRING_LENGTH 40
5
6  typedef struct {
7    char *firstName;
8    char *lastName;
9    unsigned int age;
10 } Person;
11
12 void assignValues(Person p, const char *firstName, const char *lastName, unsigned int age) {
13   p.firstName = firstName;
14   p.lastName = lastName;
15   p.age = age;
16 }
17
18 int main(int argc, char** argv) {
19   Person p = *(Person*)malloc(sizeof(Person));
20   assignValues(p, "Sandra", "Muster", 33);
```

```
21    printf("%s %s, %i\n", p.firstName, p.lastName, p.age);
22    return EXIT_SUCCESS;
23  }
```