Übungspaket 3

Der Datentyp `struct`

Übungsziele:

1. Organisation von `structs` im Arbeitsspeicher
2. Problemangepasste Verwendung von `structs`.

Literatur:

C-Skript[1], Kapitel: 53

Semester:

Wintersemester 2017/18

Betreuer:

Kevin, Peter und Ralf

Synopsis:

Strukturen (`structs`) sind nach den Arrays die zweite Form komplexer Datenstrukturen. Ein wesentliches Element von Strukturen ist, dass sie *unterschiedliche* Variablen zu einem neuen (komplexen) Datentyp zusammenfügen können. Dieser neue Datentyp kann anschließend wie jeder andere verwendet werden. Zu einem sind `structs` ein hervorragendes Strukturierungsmittel. Zum anderen sind sie aufgrund ihrer Eigenschaft des Zussammenfügens *unterschiedlicher* Datentypen eine wesentliche Voraussetzung dynamischer Datenstrukturen.

[1] www.amd.e-technik.uni-rostock.de/ma/rs/lv/hopi/script.pdf

Teil I: Stoffwiederholung

Aufgabe 1: Strukturen vs. Arrays

Strukturen und Arrays gehören zu den komplexen Datentypen. Für den Programmieranfänger ist es anfangs oft schwer, diese beiden Strukturierungsmethoden auseinander zu halten, obwohl dies für die weitere Arbeit von besonderer Bedeutung ist. Erkläre in eigenen Worten, was `struct`s (Strukturen) sind und was der Sinn dahinter ist.

Strukturen und Arrays unterscheiden sich vor allem in zweierlei Hinsicht. Stelle diese in der folgenden Tabelle einander gegenüber.

Aspekt	Arrays	Strukturen
Datentypen der Elemente bzw. der Komponenten		
Zugriff auf die einzelnen Elemente bzw. Komponenten		

Lassen sich Arrays und `struct`s kombinieren

Aufgabe 2: Beispiele

Zeige anhand dreier Beispiele, wie `struct`s definiert werden:

Zeige anhand einiger Beispiele, wie man auf die einzelnen Komponenten eines `structs` zugreift. Berücksichtige dabei auch mindestens ein Array:

Natürlich müssen wir auch Zeiger auf `structs` können. Nehmen wir an, wir haben einen Zeiger `p` auf einen `struct s`, in dem sich eine Komponente `i` vom Typ `int` befindet. Auf welche beiden Arten kann man auf die Komponente `i` desjenigen `structs` `s` zugreifen, auf das `p` zeigt?

1. 2.

Zeichne für folgende `struct`-Definition ein Speicherbild:

```
1 struct two_strings { char *first, *second; };
```

Wie viele Bytes belegt ein derartiges `struct`, wenn ein Zeiger vier Bytes belegt?

Nun nehmen wir folgende Variablendefinition an:

```
1 struct two_strings example = { "Enrico", "Johanna" };
```

Wie viele Bytes belegt eine derartige Variable?

Zeichne hierfür ein Speicherbildchen:

Aufgabe 3: Rekursive `struct`-Definitionen

Zur Erinnerung: *Rekursion* bedeutet, sich selbst wieder aufzurufen. Das haben wir bereits bei Funktionen kennengelernt und dort auch deren Vorzüge gesehen. Ein wichtiger Aspekt bei rekursiven Funktionsaufrufen ist, dass man sie irgendwann terminieren muss.

Nun zu den `structs`: Nehmen wir an, wir hätten folgende Definition:

```
1 struct rekursion {
2                     int i;
3                     struct rekursion noch_was;
4                   };
```

Erkläre mit eigenen Worten, weshalb eine derartige rekursive `struct`-Definition in C *nicht* mögllich ist:

Was wäre aber, wenn wir folgende Definition hätten?

```
1 struct rekursion {
2                     int i;
3                     struct rekursion *noch_mehr;
4                   };
```

Von welchem Datentyp ist die Komponente `i`?

Von welchem Datentyp ist die Komponente `noch_mehr`?

Dies ist in der Tat in C erlaubt. Wie viele Bytes belegt eine derartige Struktur, wenn ein `int` und ein Zeiger jeweils vier Bytes brauchen (`sizeof(struct rekursion)`)?

Diskutiere mit den Kommilitonen bei einem Kaffee, Bier oder sonstwas, was man damit machen könnte.

Teil II: Quiz

So ein „übliches“ Quiz ist uns zu diesem Thema nicht eingefallen, da die Sachverhalte schlicht zu einfach sind. Daher beschäftigen wir uns diesmal mit Typen und Werten im Rahmen von `structs`.

Aufgabe 1: `structs`, Typen, Zeiger, Werte

Nehmen wir an, wir haben folgendes C-Programm:

```
 1  struct combo {
 2            char c;
 3            int  a[ 2 ];
 4          };
 5
 6  int main( int argc, char **argv )
 7      {
 8          struct combo test;
 9          struct combo a[ 2 ];
10          struct combo *ptr;
11          test.a[ 0 ] = 4711;
12          test.a[ 1 ] = 815;
13          ptr = & test; ptr->c = 'x';
14      }
```

Dann können wir am Ende von Programmzeile `10` folgendes Speicherbildchen erstellen, in dem wie immer alle Adressen frei erfunden sind und wir davon ausgehen, dass Variablen vom Typ `int` sowie Zeiger immer vier Bytes belegen.

Adresse	Var.	Komponente	Wert
0xFE2C		int a[1] :	
0xFE28		int a[0] :	
0xFE24	test	char c :	
0xFE20	ptr	:	

Adresse	Var.	Komponente	Wert
0xFE44		int a[1] :	
0xFE40		int a[0] :	
0xFE3C	a[1]	char c :	
0xFE38		int a[1] :	
0xFE34		int a[0] :	
0xFE30	a[0]	char c :	

Ergänze zunächst die Effekte der Programmzeilen `11` bis `13` im obigen Speicherbildchen.

Vervollständige nun die folgende Tabelle. Die Ausdrücke werden im Verlaufe des Quiz immer schwieriger. Im Einzelfalle lohnt es sich, entweder ein kurzes Testprogramm zu scheiben und/oder mit den Betreuern zu diskutieren.

Ausdruck	Type	Wert	Anmerkung
`test`	.....	.....	.....
`sizeof( test )`	.....	.....	.....
`& test`	.....	.....	.....
`ptr`	.....	.....	.....
`sizeof( ptr )`	.....	.....	.....
`*ptr`	.....	.....	.....
`sizeof( *ptr )`	.....	.....	.....
`test.c`	.....	.....	.....
`ptr->a[0]`	.....	.....	.....
`ptr->c`	.....	.....	.....
`& ptr`	.....	.....	.....
`test.a[0]`	.....	.....	.....
`&(test.a[0])`	.....	.....	.....
`sizeof( a )`	.....	.....	.....
`&(a[ 1 ].a[ 0 ])`	.....	.....	.....
`*(a + 1)`	.....	.....	.....

Die folgenden Quiz-Fragen sind sehr schwer!

Ausdruck	Type	Wert	Anmerkung
`sizeof( test.a )`	.....	.....	.....
`test.a`	.....	.....	.....
`a`	.....	.....	.....

Betrachte noch die folgenden Anweisungen. Trage die Auswirkungen der einzelnen Anweisung in das Speicherbildchen der vorherigen Seite ein.

```
1  a[ 0 ].c = 'a';
2  (*(a + 0)).a[ 0 ] = -273;
3  (a + 0)->a[ 1 ] = 24;
4  a[ 1 ] = a[ 0 ];
```

Teil III: Fehlersuche

Aufgabe 1: Definition und Verwendeung von `structs`

Nach der Vorlesung hat Dr. Strucki versucht, ein paar `structs` zu programmieren. Offensichtlich benötigt er eure Hilfe, da ihm nicht alles klar geworden ist.

```
 1 struct cpx double re, im;                    // a complex number
 2 struct ivc { double len; cpx cx_nr; };              // plus len
 3
 4 int main( int argc, char **argv )
 5     {
 6         struct cpx cx, *xp1, *xp2;
 7         struct ivc vc, *v_p;
 8         struct ivc i_tab[ 2 ];
 9
10         cx.re = 3.0; im = -4.0;
11         vc.cx_nr = cx; vc.len = 5.0;
12         vc.re = 3.0; vc.cx_nr.im = 4.0;
13
14         xp1 = & cx; xp2 = vc.cx_nr;
15         xp1->re = 6.0; xp1.im = -8.0;
16         *xp2 = *xp1; vc.len = 10.0;
17
18         cx *= 2;
19         vc.cx_nr += cx;
20
21         (*(i_tab + 0)) = vc;
22         (i_tab + 1)->cx_nr = cx;
23     }
```

Teil IV: Anwendungen

Der Anwendungsteil dieses Übungspaketes dient nur dazu, folgende Aspekte kurz zu wiederholen: `structs`, `sizeof()`, Adressberechnungen und die implizite Arraydefinition mittels Initialisierung. In Zweifelsfällen helfen die Skriptkapitel 33, 37, 46, 47, und 54 weiter. Schaut euch im Nachgang die Musterlösungen an und besprecht diese mit dem Betreuer.

Aufgabe 1: Ein einfaches Programm mit `structs`

Schreibe ein Programm, das einen neuen Datentyp `STRUCT` definiert, der platz für je eine `int` und `double`-Variable hat. Im Hauptprogramm (`main()`) soll ein Array mit vier Elementen dieses Datentyps definiert werden. Lese die Werte der entsprechenden Komponenten von der Tastatur ein, sortiere das Array aufsteigend bezüglich des `int`-Wertes und gebe die Daten des Arrays aus. Verwende keine weiteren Funktionen ausser `main()`.