

5.1.V Einführung und Klassen/Neue Möglichkeiten für structs – Versuch

5.1.V.1 Ziele

Hier soll ein Programm mit einem C-Record umgebaut werden in ein Programm mit einer C++-Klasse.

5.1.V.2 Fragen

Was muss man am Programm ändern?

5.1.V.3 Grundlage

Hier ist wieder das Beispiel mit dem Person-Record:

```
person_a.c
1 #include <stdio.h>
2 #include <string.h>
3 //=====
4 struct persontyp
5 {
6     char nachname[30];
7     char vorname[30];
8     long gebdat;
9 };
10 //=====
11 void print(struct persontyp pe)
12 {
13     printf("Nachname: %s\nVorname: %s\nGeb.-Dat.: %02i.%02i.%04i\n",
14         pe.nachname, pe.vorname, pe.gebdat%100, pe.gebdat/100%100,
15         pe.gebdat/10000);
16 }
17 //=====
18 struct persontyp init(const char *nn, const char *vn, long gd)
19 {
20     struct persontyp pe={"", "", 0};
21     strncat(pe.nachname, nn, sizeof(pe.nachname)-1);
22     strncat(pe.vorname, vn, sizeof(pe.vorname)-1);
23     pe.gebdat=gd;
24     return pe;
25 }
26 //=====
27 void aendere_nachname(struct persontyp *ppe, const char *nn)
28 {
29     ppe->nachname[0]='\0';
30     strncat(ppe->nachname, nn, sizeof(ppe->nachname)-1);
31 }
32 //=====
33 int main(void)
34 {
35     struct persontyp mitarb;
36     mitarb=init("Meier", "Willi", 19601020);
37     aendere_nachname(&mitarb, "Meyer");
38     print(mitarb);
39     return 0;

```

40 }

5.1.V.4 Schritt 1: Weg mit den Details

Besonderes Augenmerk ist auf die Funktionen zu richten, die mit dem Record `struct persontyp` zu tun haben. Das sind alle die Funktionen, die diesen Datentyp als Parameter oder als Rückgabewert haben:

- `print()`
- `init()`
- `aendere_nachname()`

Diese Funktionen müssen zu Methoden werden. `main()` ist auf andere Art betroffen: Dort werden diese Funktionen benutzt. Dazu später.

Zunächst ist es sinnvoll, dass man sich auf die Struktur konzentriert. Alle Details können weggelassen werden. Daher sollte man zunächst die Funktionsrümpfe weglassen:

person_b.c

```

1 #include <stdio.h>
2 #include <string.h>
3 //=====
4 struct persontyp
5 {
6     char nachname[30];
7     char vorname[30];
8     long gebdat;
9 };
10 //=====
11 void print(struct persontyp pe){}
12 struct persontyp init(const char *nn, const char *vn, long gd){}
13 void aendere_nachname(struct persontyp *ppe, const char *nn){}
14 //=====
15 int main(void)
16 {
17     struct persontyp mitarb;
18     mitarb=init("Meier", "Willi", 19601020);
19     aendere_nachname(&mitarb, "Meyer");
20     print(mitarb);
21     return 0;
22 }
```

5.1.V.5 Schritt 2: Funktionen werden zu Methoden

Im nächsten Schritt werden die Funktionen in das Record hinein verschoben. Damit werden sie zu Methoden. Das kann der C-Compiler nicht mehr verarbeiten, daher ändert man die Endung von `.c` auf `.cpp`.

person_c.c

```

1 #include <stdio.h>
2 #include <string.h>
3 //=====
4 struct persontyp
```

```

5  {
6  char nachname[30];
7  char vorname[30];
8  long gebdat;
9  void print(struct persontyp pe){}
10 struct persontyp init(const char *nn, const char *vn, long gd){}
11 void aendere_nachname(struct persontyp *ppe, const char *nn){}
12 };
13 //=====
14 int main(void)
15 {
16     struct persontyp mitarb;
17     mitarb=mitarb.init("Meier", "Willi", 19601020);
18     mitarb.aendere_nachname(&mitarb, "Meyer");
19     mitarb.print(mitarb);
20     return 0;
21 }

```

Wie man in `main()` sieht, sind auch dort Änderungen nötig: Die Methoden gehören zu einem Objekt, müssen dort also beim Aufruf zusammen mit einem Objektnamen aufgerufen werden. Aus `print()` wird `mitarb.print()`.

5.1.V.6 Schritt 3: Methoden an das Objekt binden

Der entscheidende Vorteil einer Methode ist, dass sie Zugriff auf das Objekt hat, in dem sie sich befindet; die Attribute sind von der Methode aus sichtbar.

Damit können alle Parameter und Rückgabewerte, die vom Typ `struct persontyp` sind, einfach wegfallen:

person_d.c

```

1  #include <stdio.h>
2  #include <string.h>
3  //=====
4  struct persontyp
5  {
6      char nachname[30];
7      char vorname[30];
8      long gebdat;
9      void print(void){}
10     void init(const char *nn, const char *vn, long gd){}
11     void aendere_nachname(const char *nn){}
12 };
13 //=====
14 int main(void)
15 {
16     struct persontyp mitarb;
17     mitarb.init("Meier", "Willi", 19601020);
18     mitarb.aendere_nachname("Meyer");
19     mitarb.print();
20     return 0;
21 }

```

Man sieht, wie einfach plötzlich die Aufrufe werden.

5.1.V.7 Schritt 4: Details wieder einbauen

Zuletzt kann man die Funktionsrümpfe wieder hinzufügen; allerdings sind ein paar Änderungen nötig. Wir haben keine Parameter vom Typ `struct persontyp` mehr, stattdessen werden die Attribute des aktuellen Objekts direkt verändert:

```

                                     person_e.c
1  #include <stdio.h>
2  #include <string.h>
3  //=====
4  struct persontyp
5  {
6      char nachname[30];
7      char vorname[30];
8      long gebdat;
9      void print(void)
10     {
11         printf("Nachname: %s\nVorname: %s\nGeb.-Dat.: %02i.%02i.%04i\n",
12             nachname, vorname, gebdat%100, gebdat/100%100,
13             gebdat/10000);
14     }
15     void init(const char *nn, const char *vn, long gd)
16     {
17         nachname[0]=vorname[0]='\0';
18         strncat(nachname, nn, sizeof(nachname)-1);
19         strncat(vorname, vn, sizeof(vorname)-1);
20         gebdat=gd;
21     }
22     void aendere_nachname(const char *nn)
23     {
24         nachname[0]='\0';
25         strncat(nachname, nn, sizeof(nachname)-1);
26     }
27 };
28 //=====
29 int main(void)
30 {
31     struct persontyp mitarb;
32     mitarb.init("Meier", "Willi", 19601020);
33     mitarb.aendere_nachname("Meyer");
34     mitarb.print();
35     return 0;
36 }

```