5.1.A Einführung und Klassen/Neue Möglichkeiten für structs – Arbeitsblatt

Aufgabe 1: Klasse `punkttyp`

Die Klasse `punkttyp` soll einen beliebigen Punkt auf einer zweidimensionalen Ebene beschreiben. Es gibt eine x- und eine y-Koordinate.

Mit der Methode `print()` (ohne Parameter) sollen die beiden Attribute auf den Bildschirm ausgegeben werden (in der Form `x=3.5, y=-2.7`).

Mit der Methode `scan()` (auch ohne Parameter) sollen die beiden Attribute nacheinander (jeweils mit `scanf()`) von der Tastatur eingelesen werden.

a) Welcher Datentyp ist sowohl für die x- als auch für die y-Komponente sinnvoll?

b) Zeichnen Sie das UML-Diagramm für die Klasse!

c) (`punkttyp1.cpp`)
Schreiben Sie die Klasse! Im Hauptprogramm soll ein Punkt angelegt werden, und mit `scan()` sollen die Attribute eingelesen werden. Die x-Koordinate soll verdoppelt werden, und zum Schluss sollen die Attribute mit `print()` ausgegeben werden.

Aufgabe 2: Klasse `tageszeittyp`

Die Klasse `tageszeittyp` soll einen Zeitpunkt an einem Tag sekundengenau beschreiben. Hier ist der C-Code:

```
1 struct tageszeittyp
2 {
3     unsigned char std;
4     unsigned char min;
5     unsigned char sek;
6     // ...
7     // hier fehlen noch die Methoden!
8     // ...
9 };
```

Mit der Methode `print()` (ebenfalls ohne Parameter) soll die Uhrzeit in der Form `01:23:45 Uhr` auf den Bildschirm ausgegeben werden. Mit der Methode `scan()` (auch ohne Parameter) soll die Uhrzeit getrennt nach Stunde, Minute und Sekunden von der Tastatur eingelesen werden.

a) Zeichnen Sie das UML-Diagramm für die Klasse!

b) (`tageszeittyp1.cpp`)
Schreiben Sie die fehlenden Methoden der Klasse sowie das Hauptprogramm! Im Hauptprogramm soll ein Zeitpunkt angelegt werden, mit `scan()` sollen die Attribute gesetzt werden. Danach soll die Zeit um eine Stunde erhöht werden. Zum Schluss soll die Uhrzeit mit `print()` ausgegeben werden.

Hinweis: Der Platzhalter für die Ausgabe von Zahlen des Datentyps `unsigned char` lautet `%hhu`.

Aufgabe 3: Klasse `ztartikeltyp`

Die Klasse `ztartikeltyp` soll einen Zeitungs- oder Zeitschriftenartikel beschreiben. Attribute sind:

- Name der Zeitschrift
- Jahrgang (z. B. 2020)

- Ausgabe (z. B. 21 für die 21. Ausgabe des Jahres)
- Seitennummer (beginnt auf Seite Nr.)
- Seitenzahl (wie lang der Artikel ist)
- Überschrift des Artikels
- Autor
- Schlagworte, durch Komma getrennt

Für die numerischen Attribute soll in diesem Fall `int` gewählt werden. Für die Zeichenketten soll in diesem Fall `char[80]` gewählt werden.

Auch hier soll eine Methode `print()` die Daten auf den Bildschirm ausgeben; eine Methode `scan()` soll Daten von der Tastatur in ein Objekt der Klasse einlesen.

a) Zeichnen Sie das UML-Diagramm für die Klasse!

b) (`ztartikeltyp1.cpp`)
Schreiben Sie die Klasse, dazu das Hauptprogramm! Im Hauptprogramm soll ein Artikel angelegt werden. Anschließend soll er mit `print()` ausgegeben werden.

Aufgabe 4: Klasse `bruch`

In der Datei `src/bruch.cpp` wird die Klasse `bruch` definiert. Jedes Objekt dieser Klasse besitzt zwei Attribute (`zaehler` und `nenner`), die den Zähler und den Nenner eines Bruchs darstellen. Mit `zaehler=4` und `nenner=5` wird so die gebrochene Zahl $\frac{4}{5} = 0,8$ dargestellt.

```
1  #include <iostream>
2  using namespace std;
3  struct bruch
4  {
5     int zaehler;
6     int nenner;
7     bool is_ok(void)
8     {
9        if(nenner!=0) return true;
10       return false;
11    }
12    void display(void)
13    {
14       if(is_ok()) cout << zaehler << "/" << nenner;
15       else        cout << "Bruch fehlerhaft!";
16    }
17    void plus1(void)
18    {
19       if(is_ok()) zaehler=zaehler+nenner;
20       else        display();
21    }
22    void plusx(int x)
23    {
24       if(is_ok()) zaehler=zaehler+zaehler*nenner;
25       else        display();
26    }
27    void malx(int x)
28    {
29       if(is_ok()) zaehler=zaehler*x;
30       else        display();
31    }
32    double dblwert()
33    {
34       return (float)zaehler/(float)nenner;
35    }
36 };
37
38 int main(void)
39 {
40    bruch zahl={4,5}; // vier fuenftel
41    zahl.plus1();
42    zahl.malx(10);
43    zahl.display();
44    cout << endl << zahl.dblwert() << endl;
45    return 0;
46 }
```

a) Geben Sie die UML zu `bruch` an!

b) Warum ist es nötig, zu Beginn jeder Methode zu testen, ob das aktuelle Objekt gültig ist?

c) Erstellen Sie ein Aufrufdiagramm der Methoden!

d) Erweitern Sie die Klasse um die Funktionen `void minus1(void)`, `minusx(int x)` und `durchx(int x)`!

e) Erweitern Sie die Klasse um die Funktion `int ggT(void)`, die den größten gemeinsamen Teiler von Zähler und Nenner ermittelt und um die Funktion `void kuerzen(void)`, die diese Funktion ausnutzt.

f) Ergänzen Sie die Funktion `kuerzen()` überall dort, wo es zweckmäßig ist!

Aufgabe 5: Klasse `datumtyp`

Es soll eine Klasse angelegt werden, mit der ein Datum, bestehend aus Tag, Monat und Jahr verwaltet werden kann. Abbildung 1 zeigt das zugehörige Klassendiagramm. Sie können dabei die

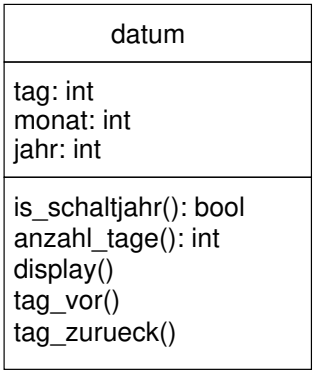

Abbildung 1: Klassendiagramm zu `datum`

Klasse `bruch` aus `src/bruch.cpp` zur Orientierung benutzen.

a) Erstellen Sie die Klasse zunächst lediglich mit der Methode `display()`!

b) Legen Sie im Hauptprogramm ein Objekt an, geben Sie den Attributen Werte und rufen Sie `display()` auf!

c) Ergänzen Sie die restlichen Methoden!

Aufgabe 6: Klasse `person`

Die Klasse `person` soll folgende Attribute haben: Vorname (41 Zeichen), Nachname (41 Zeichen) und Geburtsdatum (`int`, Format: JJJJMMTT). Als Methoden sind vorgesehen: `print1()` (Ausgabe im Format `vorname␣nachname`) und `print2()` (Ausgabe im Format `nachname,␣vorname`).

a) Geben Sie die UML an!

b) Erstellen Sie die Klasse!

c) Testen Sie die Klasse mit einem Beispiel!

Aufgabe 7: Vom Record zur Klasse

Gegeben ist folgendes C-Programm:

aufzeichnung.c

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

struct aufzeichnungstyp
{
    time_t zeitpunkt;
    int dauer_in_sec;
    char ort[200];
    char thema[200];
};

struct aufzeichnungstyp init(int neu_dauer, const char *neu_ort,
                             const char *neu_thema)
{
   struct aufzeichnungstyp erg={0};
   erg.dauer_in_sec=neu_dauer;
   strncat(erg.ort, neu_ort, sizeof(erg.ort)-1);
   strncat(erg.thema, neu_thema, sizeof(erg.thema)-1);
   return erg;
}

void setze_zeitpunkt(struct aufzeichnungstyp *paufz,
                     int tag, int mon, int jahr)
{
   struct tm zeit_rec={0,0,0,tag,mon-1,jahr-1900};
   (*paufz).zeitpunkt=mktime(&zeit_rec);// paufz->zeitpunkt=...
}

void print(struct aufzeichnungstyp aufz)
{
   printf("Thema....: %s\n", aufz.thema);
   printf("Zeitpunkt: %s", ctime(&aufz.zeitpunkt));
   printf("Ort......: %s\n", aufz.ort);
   printf("Dauer....: %i Sekunden\n", aufz.dauer_in_sec);
}

int main(void)
{
   struct aufzeichnungstyp aktuelle_doku;
   aktuelle_doku=init(40, "Bremen", "Weser-Hochwasser Dez. 2023");
   setze_zeitpunkt(&aktuelle_doku, 29, 12, 2023);
   print(aktuelle_doku);
   return 0;
}
```

Dieses Programm rund um das Record `struct aufzeichnungstyp` soll so geändert werden, dass es objektorientiert arbeitet. Das Record soll dazu in eine gleichnamige Klasse umgearbeitet werden.

a) Sorgen Sie dafür, dass aus den Funktionen, die mit `struct aufzeichnungstyp` zu tun haben, Methoden dieser Klasse werden! Die Rümpfe der Methoden sollen zunächst leer bleiben.

b) Bearbeiten Sie die Parameterlisten und Rückgabetypen der Methoden so, dass jetzt unnötige Parameter und Rückgabewerte entfernt werden!

c) Füllen Sie die Rümpfe der Methoden so, dass sie das Gleiche tun wie die bisherigen Funktionen!

d) Ändern Sie die `main()`-Funktion so ab, dass jetzt statt der Funktionen die Methoden aufgerufen werden!