Algorithmen und Datenstrukturen I

Blatt 2

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 18.03.2018 20:05

Aufgabe 1 — `struct`, Parameterübergabe und `const`

Das Repository für diese Aufgabe bekommen Sie unter https://classroom.github.com/a/DHabOGin.

In Java ist das ja klar geregelt. Die sog. primitiven Datentypen werden *by-value* übergeben und alle anderen, die Referenztypen, *by-reference*. Wenn man sich daran gewöhnt hat, kommt man ganz gut damit zurecht.

In C/C++ ist aber alles möglich. Es gibt nicht einmal die Unterscheidung zwischen diesen Typen. Ein `int` kann z.B. als Referenz übergeben werden und ein komplexes Objekt so, dass automatisch eine Kopie erzeugt wird. Und um die Verwirrung noch komplett zu machen, können auch noch Pointer übergeben werden. Und diese auch *by-value* oder *by-reference* und darauf auch noch Pointer und …

Also kurzum, alles ist möglich. Natürlich ist das eine in einem bestimmten Kontext sinnvoller als das andere und je nach verfügbarer Ressourcen kann es wieder anders herum sein. Komplexere Stukturen, wie Werte einer `struct` oder Objekte, werden natürlich fast immer so wie in Java übergeben. Aber das müssen wir dem Compiler extra sagen.

Wir wollen einen Typ `color` nutzen, der aber eigentlich nur ein `string` ist. Dazu können wir (außerhalb der `main`-Funktion oder auch innerhalb) mit `typedef` ein Typ-Alias definieren. Wir müssen dazu die `string`-Library einbinden.

```
typedef std::string color;
```

Nun können wir eine Struktur für farbige, zweidimensionale Punkte definieren:

```
struct point {
    int x;
    int y;
    color c;
};
```

Auch die Struktur können wir innerhalb von der `main`-Funktion definieren (und dann natürlich auch nur dort nutzen) oder außerhalb.

Eine Struktur in C ist eine Zusammenfassung verschiedener Werte. In C++ ist eine Struktur eine Klasse mit öffentlichen Elementen. D.h. ein C++-Compiler akzeptiert Methoden in einer `struct`, ein C-Compiler nicht. Die `string`-Library kennt C übrigens nicht. Dort müssten Sie mit einem null-terminierten `char`-Array arbeiten.

Sie können jetzt Punkte auf verschiedene Weise erzeugen. Sehen Sie sich den folgenden Code an und versuchen Sie jede einzelne Code-Zeile zu verstehen.

```
point p1;
p1.x = 12;
p1.y = 13;
p1.c = "red";

point p2 = { 11, 13, "red" };

point *pp3;
pp3 = &p1;

point *pp4 = new point;
pp4->x = 12;
pp4->y = 13;
pp4->c = "red";
```

Haben Sie bemerkt, dass `new` in C++ eine Semantik hat, die im ersten Moment überrascht. Wenn Sie es aber mit Java vergleichen, macht es Sinn, dass es so funktioniert, oder?

Fügen Sie Ihrer point-Struktur folgenden Konstruktor hinzu und probieren Sie Ihr Programm wieder aus:

```
point(int xval, int yval, color colorval) :
    x(xval), y(yval), c(colorval) {}
```

Warum jetzt die ganzen Fehler? Fixen Sie den Code der die Punkte erzeugt, so dass es wieder läuft. Die Typen müssen aber gleich bleiben, also `p1` und `p2` Punkte und `p3` und `p4` Pointer auf Punkte.

Wir wollen nun unser Programm um eine Prozedur erweitern. Jetzt ist es notwendig, dass Sie die Struktur `point` außerhalb der `main`-Funktion definiert haben, denn Funktionen

oder Prozeduren können Sie nicht direkt in Funktionen oder Prozeduren definieren. (Es geht über λ-Ausdrücke).

Schreiben Sie also eine Prozedur mit folgendem Prototyp (Signatur):

```
void printPoint(point p);
```

Die Prozedur soll den Punkt so ausgeben

```
((x-Wert, y-Wert), Farbe)
```

(Wir werden demnächst den Operator `<<` überladen, dann brauchen wir keine "print"-Prozedur mehr.)

Probieren Sie Ihre Prozedur aus. Versuchen Sie einen Punkt direkt als Parameter anzugeben und nicht erst einer Variablen zuzuweisen.

Eine Sache an `printPoint` ist nicht ganz ideal gelöst: Der Punkt der als Parameter dient, wird als Wert (*by value*) übergeben. Das heisst er wird kopiert und auf den Stack gelegt. Das geht, weil C++ automatisch einen Kopierkonstruktor generiert. Probieren Sie doch mal aus:

```
point p5(p1);
```

Das erzeugt einen neuen Punkt `p5`, der eine Kopie von `p1` ist. Natürlich können Sie den Copy-Constructor auch selbst schreiben.

Jetzt aber zurück zu `printPoint`. Besser wäre doch, wenn der Punkt nicht kopiert wird, sondern eine Referenz übergeben wird. Das könnten wir mit einem Pointer auf einen Punkt erreichen:

```
void printPoint(point *p)
```

Was da jetzt aber etwas nervig ist, ist die Tatsache, dass wir nun

1. den Code in `printPoint` ändern müssen, und
2. die Aufrufe ändern müssen.

Ist Ihnen klar warum? Ansonsten ändern Sie mal die Signatur und sehen Sie sich die Fehlermeldungen an.

Wir können das aber vermeiden, denn es gibt die Möglichkeit dem C++-Compiler zu sagen, dass der Punkt ein Punkt bleiben, aber als Referenz übergeben werden soll. Nachdem das irgendwas mit der Adresse zu tun hat, wird dabei in Anlehnung an den Adressoperator im Typ ein `&` am Ende angefügt. Ändern Sie die Signatur auf

```
void printPoint(point &p)
```

Aufruf und Rumpf können so bleiben wie ohne `&` und `*`.

Jetzt haben wir aber ein anderes Problem. Vorher hätte in der Prozedur `printPoint` so unsinniger Code wie,

```
p.x = -23;
```

stehen können und es hätte uns höchstens bei der Ausgabe gestört. Nachdem `printPoint` eine eigene Kopie hatte, hat es unseren Punkt nicht tangiert. Mit Übergabe als Referenz, kann so eine Prozedur aber unseren Punkt tatsächlich ändern.

Was könnten Sie denn da in Java tun. Z.B. den Parameter `final` setzen. Aber nützt das wirklich was? `final` heißt ja nur, das ich dem Bezeichner nichts anderes zuweisen darf. Also nein. Wenn wir eine Klasse hätten, könnten wir alles `private` machen.

In C++ gibt es eine sehr elegante Lösung. Wir können dem Compiler sagen, dass die Prozedur `printPoint` nur lesend auf `p` zugreift. Dies geht mit dem Schlüsselwort `const`. Ändern wir also den Kopf der Prozedur auf

```
void printPoint(const point &p)
```

wird der Code mit der Anweisung `p.x = -23` gar nicht mehr übersetzt, denn `p` ist jetzt eine *read-only variable*.

Wir könnten die Prozedur `printPoint` jetzt auch noch zu einer Methode der Struktur machen. Damit ändert sich die Signatur auf

```
void printPoint()
```

denn der Punkt muss ja nicht mehr übergeben werden, `printPoint` ist ja jetzt Teil des Punktes. (Als Name wäre jetzt sicherlich `print` sinnvoller, denn der Aufruf ist ja jetzt `p.printPoint()`)

Wie sollen wir denn jetzt zusichern, dass `printPoint` den Punkt nicht verändert. Das geht in dem wir `const` hinter die Parameterliste schreiben, also

```
void printPoint() const
```

Dieses `const` bedeutet, dass die Methode `printPoint` keine Nebeneffekte (*side effects*) auf dem Objekt selbst hat. Alle Objektvariablen sind in `printPoint` nur *read-only* verfügbar.

Aufgabe 2 — Zeiger und Vektoren

Das Repository für diese Aufgabe bekommen Sie unter https://classroom.github.com/a/UvZB1AaQ.

Sie haben in Ihrem Unternehmen das letzte Jahr ausschließlich Java-Entwicklung gemacht. Nun hat die Geschäftsleitung beschlossen, dass einiges umstrukturiert werden soll. Unter anderem soll bei Neuentwicklungen C++ statt Java genutzt werden. Deshalb gab es in der letzten Woche einen kurzen Workshop und nun stehen Sie schon mittendrin in der C++-Programmierung.

Ein Kollege von Ihnen hatte den Auftrag eine Prozedur zu schreiben, mit der eine Reihe von `int`s eingesammelt werden sollen. Er hat im Workshop gut aufgepasst und weiß, dass der Prozedur am Besten ein Pointer auf einen `vector` übergeben werden soll und dieser dann befüllt werden kann. Ein `vector` ist eine Art Array, aber er kann z.B. dynamisch wachsen und noch viel viel mehr.

Er weiß auch das es Pointer gibt und hat gemerkt, dass das ja alles ganz einfach ist. Interessant fand er, dass ein Aufruf von new einen Zeiger auf das neu geschaffene Objekt zurück gibt. Also eigentlich wie in Java, außer das man das in Java irgendwie gar nicht sieht. Absolut abgefahren fand er, dass mit `new int(15)` ein Zeiger auf einen `int` erzeugt wird und so einiges mehr.

Mit der C++-Referenz und einigen anderen Quellen die er im Internet findet, macht er sich an die Arbeit und wird pünktlich zu seinem Urlaubsantritt fertig. Nachdem Sie auf dem Urlaubsschein unterschrieben haben, dass Sie ihn vertreten, haben Sie nun die Verantwortung für seinen Code. Kein Problem, denken Sie. Er hat zu seiner Prozedur `fillvector()` ja noch eine Ausgabeprozedur `printvector` geschrieben und beides sogar in einer `main`-Funktion verbunden. Compiliert, gepusht und, ach ja, ausprobieren hat er nicht mehr ganz geschafft, da er vor seinem Urlaub noch kurz beim Chef vorbei musste.

Und genau dieser Chef kommt jetzt wutschnaubend aus seinem Büro. Reißt die Tür von Ihrem Büro auf und ist stinksauer weil der Code nicht funktioniert. Sie haben bis heute Abend Zeit das Problem zu finden. Als erstes ziehen Sie sich mal den Code und schauen ihn an. Sieht doch einwandfrei aus, oder?

Weil Sie leider nicht ganz so gut aufgepasst haben, müssen Sie noch ein bisschen in der C++-Doku und im Internet nachschauen, aber finden immer noch, dass das schon alles passen müsste. Also mal compilieren und ausprobieren:

```
./main
nächster Int (abbrechen mit 0): 1
nächster Int (abbrechen mit 0): 2
nächster Int (abbrechen mit 0): 3
nächster Int (abbrechen mit 0): 4
nächster Int (abbrechen mit 0): 0
{ 0, 0, 0, 0 }
```

Und nachdem der Jenkins nur überprüft ob es compilier- und ausführbar ist, bekommen Sie nicht einmal eine Fehlermeldung von ihm.

Irgendwas passt da wirklich nicht. Finden Sie den Fehler und fixen Sie den Code.