Der Vorteil von Klassen, der wohl jedem Programmierer am schnellsten einleuchtet, ist die Möglichkeit, Funktionen zu definieren, die bei der Entstehung der Objekte automatisch aufgerufen werden und so garantieren können, dass ein Objekt immer korrekt initialisiert ist. Analog können Sie eine Funktion schreiben, die immer bei der Auflösung des Objekts aufgerufen wird und die dann angeforderte Ressourcen wieder freigeben kann. Da diese Aufgaben nur einmal bei der Definition der Klasse erledigt werden, entfallen viele Flüchtigkeitsfehler, die durch vergessene Initialisierungen entstehen.

**Konstruktor und Destruktor**

Die Elementfunktion, die beim Erzeugen eines Objekts aufgerufen wird, nennt man Konstruktor. In dieser Funktion können Sie dafür sorgen, dass alle Elemente des Objekts korrekt initialisiert sind.

**Konstruktordefinition**

Der Konstruktor trägt immer den Namen der Klasse selbst und hat keinen Rückgabetyp, auch nicht void. Der Standardkonstruktor hat keine Parameter.

**Destruktordefinition**

Das Gegenstück zum Konstruktor ist der Destruktor. Er wird ausgeführt, wenn ein Objekt zerstört wird. Der Destruktor ist vor allem dann wichtig, wenn das Objekt im Laufe seiner Existenz Ressourcen angefordert hat. Durch den Destruktor kann gewährleistet werden, dass sie wieder freigegeben werden. Der Name des Destruktors wird gebildet, indem eine Tilde (~) dem Klassennamen vorangestellt wird. Wie der Konstruktor hat auch der Destruktor keinen Rückgabetyp, also auch nicht void. Der Destruktor hat niemals Parameter.

**Beispiel**

Im Falle einer Datumsklasse wäre es sinnvoll, dass der Konstruktor alle Elemente auf 0 setzt. Daran kann jede Elementfunktion leicht erkennen, dass das Datum noch nicht festgelegt wurde. Sie könnten alternativ das aktuelle Datum ermitteln und eintragen. Im Beispiel ist auch ein Destruktor definiert worden, obwohl er im Falle eines Datums keine Aufgabe hat.

[Konstruktor und Destruktor]

class tDatum

{

public:

tDatum();

~tDatum();

...

};

tDatum::tDatum()

{

Tag=0; Monat=0; Jahr=0;

}

tDatum::~tDatum()

{

}

**Zeitpunkt der Ausführung**

Wann Konstruktor und Destruktor aufgerufen werden, hängt davon ab, wann das Objekt erzeugt und zerstört werden. Globale Objekte werden beim Programmstart angelegt und zum Programmende aufgelöst. Lokale Objekte rufen ihren Konstruktor bei der Definition auf und werden bei Verlassen ihres Geltungsbereichs entfernt. Schließlich kann die Erzeugung und Zerstörung explizit im Programm mit den Operatoren new und delete erfolgen. Wird mit dem Befehl new ein Array angelegt, wird für jedes einzelne Element der Konstruktor aufgerufen. Entsprechend wird beim Aufruf von delete[] für jedes Element dann wieder der Destruktor aufgerufen.

[Konstruktor- und Destruktoraufrufe]

{

tDatum heute;

tDatum \*morgen; // kein Konstruktoraufruf!

tDatum \*Urlaub; // auch kein Konstruktoraufruf

morgen = new tDatum; // aber hier wird er aufgerufen

Urlaub = new tDatum[14]; // 14 Konstruktoraufrufe

delete morgen; // hier Destruktoraufruf

...

delete[] Urlaub; // 14 Destruktoraufrufe

} // hier Destruktor von heute

**Sonderform der Initialisierung**

Ein Konstruktor wird in den meisten Fällen aus einigen Zuweisungen bestehen, das die Elementvariablen des Objekts initialisiert. In bestimmten Fällen braucht man eine andere Form der Initialisierung. So können Konstanten der Klasse nicht per Zuweisung vorbelegt werden.

**Initialisierung statt Zuweisung**

Anstatt die Elementvariablen des Objekts im Rumpf des Konstruktors per Zuweisung zu belegen, können sie auch initialisiert werden. Dazu werden zwischen dem Kopf und dem Rumpf der Konstruktordefinition ein oder mehrere Initialisierer aufgezählt. Die Initialisierer sind durch einen Doppelpunkt von dem Konstruktorkopf abgesetzt. Ein Initialisierer besteht aus dem Variablen- oder Konstantennamen und einer Klammer, in der sich der Initialisierungswert befindet.

[Alternative Initialisierung]

tDatum::tDatum() : Tag(0),Monat(0),Jahr(0)

{

}

In diesem Fall werden die Elementvariablen **Tag**, **Monat** und **Jahr** auf 0 gesetzt. Der Konstruktorkörper ist leer. Die Initialisierung erfolgt bereits vor dem Ausführen des Funktionsrumpfes. Es gibt einen entscheidenden Unterschied zur Zuweisung der Werte an die Elementvariablen: Im Körper eines Konstruktors kann nur eine Zuweisung stattfinden, während diese Form eine Initialisierung ist. In den meisten Fällen ist der Unterschied unerheblich, aber wenn die Klasse Referenzvariablen oder Konstanten enthält, können diese nur durch eine Initialisierung vorbelegt werden. Alle Versuche, solche Elemente durch eine Zuweisung vorzubelegen, werden scheitern.[[1]](http://www.willemer.de/informatik/cpp/construct.htm" \l "fuss1)

**Konstruktor und Parameter**

**Vorgabewerte**

Konstruktoren können auch Parameter entgegennehmen. Die übergebenen Werte werden im Normalfall vom Konstruktor verwendet, um Elementvariablen zu initialisieren.

**Überladen**

Konstruktoren können genauso überladen werden wie normale Funktionen auch. Es kann neben dem Standardkonstruktor auch mehrere weitere Konstruktoren mit verschiedenen Parametern geben. Der Compiler wird anhand der Aufrufparameter unterscheiden, welcher Konstruktor verwendet wird.

**Beispiel**

Das folgende Beispiel zeigt die Klasse **tDatum** mit einem Konstruktor mit drei Parametern.

[Konstruktor mit Parametern]

class tDatum

{

public:

tDatum(int Tag, int Monat, int Jahr=-1);

...

};

tDatum::tDatum(int Tag, int Monat, int Jahr)

{

this->Tag=Tag;

this->Monat=Monat;

this->Jahr=Jahr;

if (Jahr<0)

{

// setze das aktuelle Jahr ein

...

}

}

tDatum Start(1,1,1970);

tDatum Silvester(31,12);

tDatum \*HeiligAbend = new tDatum(24,12);

Das Objekt **Start** wird durch den Konstruktor auf den 1.1.1970 gesetzt. Das Objekt **Silvester** erhält als Parameter den 31.12. ohne eine Angabe des Jahres. Da der dritte Parameter in diesem Fall --1 vorgibt, wird dieser Wert angenommen. Innerhalb des Konstruktors wird im Falle eines negativen Jahres aber das aktuelle Jahr eingesetzt. Da der einzig existierende Konstruktor Parameter verlangt, kann für die Klasse **tDatum** kein Objekt erzeugt werden, ohne es zu initialisieren.

**Konvertierungskonstruktor**

Wenn Sie einer float-Variablen eine Integer-Variablen zuweisen, wird diese automatisch konvertiert. Beim Erstellen einer Klasse können Sie festlegen, welche Typen auf ähnliche Weise automatisch konvertiert werden sollen. Dazu legen Sie einen Konverter mit nur einem Parameter an, der den gewünschten Konvertierungstyp haben soll.

**Typkonvertierung**

Ein Konstruktor mit nur einem Parameter führt dazu, dass der Compiler diesen Konstruktor verwendet, um den Parametertyp zu konvertieren.

class tBruch

{

public:

tBruch(char \*);

Addiere(tBruch&);

};

...

char Eingabe[MAXSTR];

getline(cin, Eingabe, MAXSTR);

tBruch b1(Eingabe);

getline(cin, Eingabe, MAXSTR);

b1.Addiere(Eingabe);

**Automatischer Aufruf**

In der Klasse **tBruch** gibt es einen Konstruktor, der als Parameter einen Zeiger auf den Typ char und damit einen C-String akzeptiert. Die Funktion Addiere()akzeptiert lediglich  den Typ **tBruch**. Der Compiler akzeptiert dennoch den Aufruf von Addiere() mit einem C-String als Parameter, weil er ihn mit Hilfe des Konstruktors in **tBruch** überführen kann.

**explicit**

Der Konvertierungskonstruktor wird immer automatisch aufgerufen, wenn eine Konvertierung gebraucht wird. Wenn Sie das nicht wünschen, können Sie dem Konvertierungskonstruktor das Schlüsselwort explicit voranstellen. Dann muss die Konvertierung durch die Funktionsschreibweise explizit angefordert werden.

class tBruch

{

public:

explicit tBruch(long);

...

};

tBruch bruch=12; // das läuft nicht durch den Compiler

tBruch bruch(12); // so funktioniert's

**Standardkonstruktor**

**Ohne Parameter**

Als Standardkonstruktor wird derjenige Konstruktor bezeichnet, der ohne Parameter aufgerufen werden kann. Das bedeutet nicht, dass der Konstruktor keine Parameter haben darf. Auch ein Konstruktor mit Parametern, die vollständig mit Vorgabewerten besetzt sind, ist ein Standardkonstruktor, da er ebenfalls ohne Parameter aufgerufen werden kann. Definiert die Klasse gar keinen eigenen Konstruktor, so erstellt der Compiler einen eigenen, leeren Standardkonstruktor. Sobald Sie selbst einen Konstruktor definieren, entfällt der automatisch generierte Konstruktor. Das ist auch dann der Fall, wenn keiner Ihrer Konstruktoren ohne Parameter auskommt. In diesem Fall wird das Anlegen eines Objekts ohne Parameter fehlschlagen. Im obigen Beispiel würde das einfache Anlegen eines Objekts vom Typ **tDatum** oder auch das Anlegen eines Arrays zu einem Compiler-Fehler führen, da kein Konstruktor existiert, der ohne Parameter auskommt.

[1]

Auch für den Aufruf von Konstruktoren von Basisklassen ist diese Form der Initialisierung wichtig. An der entsprechenden Stelle wird darauf noch einmal eingegangen.

http://www.willemer.de/informatik/cpp/construct.htm