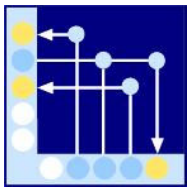
Hochschule Aalen
Fakultät Elektronik und Informatik
Studiengang Informatik



# Programmieren 2

Vorlesung im Wintersemester 2014/2015
Prof. Dr. habil. Christian Heinlein

## 1. Übungsblatt (14. Oktober 2014)

## Aufgabe 1: Gebührenpflichtige Konten

Erweitern Sie das Konto-Beispiel der Vorlesung (Skript, § 2.1 und § 2.2) wie folgt um gebührenpflichtige Konten:

a) Definieren Sie eine Datenstruktur `struct ChargedAccount`, die alle Datenfelder von `struct Account` „erbt“ und zusätzlich ein Datenfeld `count` besitzt, in dem die Anzahl der ausgeführten Buchungen pro Konto gespeichert wird (Anfangswert 0)!

b) Implementieren Sie Funktionen `newChargedAccount` und `initChargedAccount` zur Erzeugung und Initialisierung gebührenpflichtiger Konten! `newChargedAccount` soll als Parameter den Kontoinhaber erhalten.

c) „Überschreiben“ Sie die Funktionen `deposit` und `withdraw` für gebührenpflichtige Konten so, dass bei jeder solchen Buchung das Datenfeld `count` um eins erhöht wird! Zur Ausführung der eigentlichen Buchung soll die entsprechende Funktion für gewöhnliche Konten aufgerufen werden.
(Die Funktion `transfer` muss nicht überschrieben werden. Warum?)

d) Implementieren Sie eine Funktion `charge`, die als Parameter ein gebührenpflichtiges Konto sowie einen Centbetrag erhält und für jede ausgeführte Buchung des Kontos diesen Betrag als Buchungsgebühr vom aktuellen Kontostand abzieht! Anschließend soll der Buchungszähler `count` auf 0 zurückgesetzt werden.

e) Schreiben Sie, analog zu § 2.1.6 und § 2.2.7, ein Hauptprogramm, in dem auf gewöhnlichen, limitierten und gebührenpflichtigen Konten diverse Buchungen ausgeführt werden, u. a. Überweisungen zwischen unterschiedlichen Arten von Konten! Verfolgen Sie durch Ausgabeanweisungen jeweils den genauen Programmablauf!
Rufen Sie am Schluss für jedes gebührenpflichtige Konto die Funktion `charge` mit Parameterwert 10 Cent auf und geben Sie dann alle Kontostände aus!

Orientieren Sie sich bei der Lösung dieser Aufgabe an der Implementierung limitierter Konten (Skript, § 2.2)!

Der Code zur Implementierung gewöhnlicher und limitierter Konten aus dem Vorlesungsskript steht auf der Vorlesungs-Webseite zur Verfügung und kann übernommen werden.

## Lösung

### Code für gewöhnliche und limitierte Konten (vgl. Skript)

```
#include <stdlib.h>             /* malloc, exit, NULL */
#include <stdio.h>              /* printf */

/* Zeichenkette variabler Länge. */
typedef const char* String;

/* Dynamisches Objekt der Größe n Byte erzeugen. */
void* newObject (int n) {
    void* this = malloc(n);
    if (this == NULL) {
        printf("newObject: out of memory\n");
        exit(1);
    }
    return this;
}

/* Funktionszeigersatz für Konten. */
struct AccountOps {
    void (*deposit) ();         /* Funktion für Einzahlung. */
    void (*withdraw) ();        /* Funktion für Abhebung. */
    void (*transfer) ();        /* Funktion für Überweisung. */
};

/* Konto. */
struct Account {
    struct AccountOps* ops;     /* Zeiger auf Funktionszeigersatz. */
    long number;                /* Kontonummer. */
    String holder;              /* Kontoinhaber. */
    long balance;               /* Kontostand in Cent. */
};

/* Zeiger auf Konto. */
typedef void* Account;          /* Achtung: void*! */

/*
 * Funktionen für alle Arten von Konten.
 */

/* Nummer von Konto this liefern. */
long number (struct Account* this) {
    return this->number;
}

/* Inhaber von Konto this liefern. */
String holder (struct Account* this) {
    return this->holder;
}
```

```
/* Kontostand von Konto this liefern. */
long balance (struct Account* this) {
    return this->balance;
}

/* Betrag amount auf Konto this einzahlen. */
void deposit (struct Account* this, long amount) {
    this->ops->deposit(this, amount);
}

/* Betrag amount von Konto this abheben. */
void withdraw (struct Account* this, long amount) {
    this->ops->withdraw(this, amount);
}

/* Betrag amount von Konto this auf Konto that überweisen. */
void transfer (struct Account* this, int amount, Account that) {
    this->ops->transfer(this, amount, that);
}

/*
 * Funktionen für gewöhnliche Konten.
 */

/* Implementierung von deposit für Account-Objekte. */
void depositAccount (struct Account* this, int amount) {
    this->balance += amount;
}

/* Implementierung von withdraw für Account-Objekte. */
void withdrawAccount (struct Account* this, int amount) {
    this->balance -= amount;
}

/* Implementierung von transfer für Account-Objekte. */
void transferAccount (struct Account* this, int amount,
                                        struct Account* that) {
    withdraw(this, amount);
    deposit(that, amount);
}

/* Funktionszeigersatz für gewöhnliche Konten. */
struct AccountOps opsAccount = {
    depositAccount, withdrawAccount, transferAccount
};

/* Nächste zu vergebende Kontonummer. */
long nextNumber = 1;
```

```
/* Gewöhnliches Konto this mit Inhaber h initialisieren. */
void initAccount (struct Account* this, String h) {
    this->number = nextNumber++;
    this->holder = h;
    this->balance = 0;
}

/* Gewöhnliches Konto mit Inhaber h erzeugen. */
Account newAccount (String h) {
    struct Account* this = newObject(sizeof(struct Account));
    this->ops = &opsAccount;
    initAccount(this, h);
    return this;
}

/*
 * Funktionen für limitierte Konten.
 */

/* Limitiertes Konto. */
struct LimitedAccount {
    struct Account super;       /* Account-Daten. */
    long limit;                 /* Kreditlinie in Cent (positiv). */
};

/* Zeiger auf limitiertes Konto. */
typedef void* LimitedAccount;   /* Achtung: void*! */

/* Kreditlinie des limitierten Kontos this liefern. */
long limit (struct LimitedAccount* this) {
    return this->limit;
}

/* Überprüfen, ob Betrag amount von limitiertem Konto this */
/* abgezogen werden kann, ohne die Kreditlinie zu überschreiten. */
int check (struct LimitedAccount* this, long amount) {
    if (this->super.balance - amount >= -this->limit) return 1;
    printf("Unzulässige Kontoüberziehung!\n");
    return 0;
}

/* Implementierung von withdraw für LimitedAccount-Objekte. */
void withdrawLimitedAccount (struct LimitedAccount* this, long amount) {
    if (check(this, amount)) {
        withdrawAccount((struct Account*)this, amount);
    }
}
```

```
/* Implementierung von transfer für LimitedAccount-Objekte. */
void transferLimitedAccount (struct LimitedAccount* this,
                                    long amount, struct Account* that) {
    if (check(this, amount)) {
        transferAccount((struct Account*)this, amount, that);
    }
}

/* Funktionszeigersatz für limitierte Konten. */
struct AccountOps opsLimitedAccount = {
    depositAccount, withdrawLimitedAccount, transferLimitedAccount
};

/* Limitiertes Konto this mit Inhaber h */
/* und Kreditlinie l initialisieren. */
void initLimitedAccount (struct LimitedAccount* this, String h, long l) {
    initAccount((struct Account*)this, h);
    this->limit = l;
}

/* Limitiertes Konto mit Inhaber h und Kreditlinie l erzeugen. */
LimitedAccount newLimitedAccount (String h, long l) {
    struct LimitedAccount* this =
                            newObject(sizeof(struct LimitedAccount));
    this->super.ops = &opsLimitedAccount;
    initLimitedAccount(this, h, l);
    return this;
}
```

**Code für gebührenpflichtige Konten**

```
/* Gebührenpflichtiges Konto. */
struct ChargedAccount {
    struct Account super;       /* Account-Daten. */
    int count;                  /* Buchungszähler. */
};

/* Zeiger auf gebührenpflichtiges Konto. */
typedef void* ChargedAccount;   /* Achtung: void*! */

/* Kontoführungsgebühren für Konto this abbuchen. */
void charge (struct ChargedAccount* this, int amount) {
    this->super.balance -= this->count * amount;
    this->count = 0;
}

/* Implementierung von deposit für ChargedAccount-Objekte. */
void depositChargedAccount (struct ChargedAccount* this, int amount) {
    depositAccount((struct Account*)this, amount);
    this->count++;
}
```

```
/* Implementierung von withdraw für ChargedAccount-Objekte. */
void withdrawChargedAccount (struct ChargedAccount* this, int amount) {
    withdrawAccount((struct Account*)this, amount);
    this->count++;
}

/* Funktionszeigersatz für gebührenpflichtige Konten. */
struct AccountOps opsChargedAccount = {
    depositChargedAccount, withdrawChargedAccount, transferAccount
};

/* Gebührenpflichtiges Konto this mit Kontoinhaber h initialisieren. */
void initChargedAccount (struct ChargedAccount* this, String h) {
    initAccount((struct Account*)this, h);
    this->count = 0;
}

/* Gebührenpflichtiges Konto mit Kontoinhaber h erzeugen. */
struct ChargedAccount* newChargedAccount (String h) {
    struct ChargedAccount* this =
                          newObject(sizeof(struct ChargedAccount));
    this->super.ops = &opsChargedAccount;
    initChargedAccount(this, h);
    return this;
}
```

**Hauptprogramm**

```
/* Test. */
int main () {
    /* Verschiedene Konten erzeugen. */
    Account a = newAccount("Max Mustermann");
    LimitedAccount b = newLimitedAccount("Monika Musterfrau", 1000);
    ChargedAccount c = newChargedAccount("Moritz Mustermann");

    /* Verschiedene Buchungen ausführen. */
    deposit(a, 1000);
    transfer(a, 500, b);
    transfer(a, 500, c);
    transfer(b, 2000, c);       /* Unzulässige Kontoüberziehung. */
    transfer(c, 1000, b);

    /* Kontoführungsgebühren abbuchen. */
    charge(c, 10);

    /* Kontodaten ausgeben. */
    printf("Konto a: %ld, %s, %ld\n", number(a), holder(a), balance(a));
    printf("Konto b: %ld, %s, %ld\n", number(b), holder(b), balance(b));
    printf("Konto c: %ld, %s, %ld\n", number(c), holder(c), balance(c));

    return 0;
}
```