Hintergrundinformationen zur Vorlesung
GRUNDLAGEN DER INFORMATIK I
Studiengang Elektrotechnik
WS 02/03

AG Betriebssysteme FB3
Kirsten Berkenkötter

1 Vom Problem zum Programm

- Aufgabenstellung analysieren
  - Gibt es Unklarheiten? Wenn ja, zuerst klären.
  - Verstehe ich die Aufgabenstellung? Wenn nicht, nachfragen!
  - Was sind meine Voraussetzungen?
  - Was ist mein Ziel?
- Sinnvolle Teilaufgaben einteilen
  - Ist es möglich, die Aufgabe in Teilaufgaben zu zerlegen?
  - Wenn ja, werden diese später Funktionen im Programm.
- Was geht herein, was kommt heraus?
  - Für jede Teilaufgabe/Funktion:
    - Was bekommt sie als Eingabe, also als Parameter übergeben?
    - Gibt sie ein Ergebnis zurück?
    - Wenn ja, als Rückgabewert oder indem ein (Zeiger-) Parameter überschrieben wird?
    - Welche Typen haben die Parameter und die Rückgabwerte?
- Wenn Funktionen, ihre Parameter und ihre Rückgabewerte festehen, können die Funktionsköpfe sowie die geschweiften Klammern für den Funktionskörper bereits in C programmiert werden:
  - *rückgabetyp funktionsname(parametertyp1 parameter1, parametertyp2 parameter2)*
    {
    }
  - Beispiel:
    unsigned long fakultaet(int zahl)
    {
    }
- Zu jeder Funktion einen Kommentar schreiben, der angibt, was die Funktion tut (kurze Beschreibung), welche Parameter sie erwartet (Bedeutung der Parameter) und was sie zurückgibt.
- Außerdem können der Kopf des Hauptprogramms sowie die geschweiften Klammern in das C-Programm geschrieben werden:
  - *int main()*
    {
    }

- Was soll das Hauptprogramm tun?
  - Welche Variablen werden benötigt?
  - Welchen Typ haben die Variablen?
  - Haben die Variablen einen Anfangswert?
  - Müssen Werte mit *scanf()* vom Benutzer erfragt werden?
  - Müssen Werte ausgegeben werden?
  - In beiden Fällen: <*stdio.h*> einbinden, d.h. *#include* <*stdio.h*> an den Anfang des C-Programms schreiben.
  - Werden Zufallszahlen oder andere Hilfsroutinen wie *qsort()* oder *bsearch()* benötigt? Dann die entsprechende Bibliothek, z.B. <*stdlib.h*> einbinden.
  - In welcher Reihenfolge laufen die einzelnen Programmschritte ab, d.h., in welcher Reihenfolge werden die Funktionen abgearbeitet?
- So weit wie möglich C-Code für das Hauptprogramm auffüllen, d.h. Variablendeklarationen, Variableninitialisierungen, Einlesen von Werten, Funktionsaufrufe, etc. Wenn man sich noch nicht im klaren darüber ist, wie einzelne Teile der Aufgabenstellung zu realisieren sind, schreibt man an diese Stelle im Programm zunächst einen Kommentar, der beschreibt, was dort ablaufen soll. Generell kann man nie zu viele Kommentare schreiben!!!
- Nachdem der Grobablauf im Hauptprogramm klar ist, werden die Funktionen ausprogrammiert, d.h. sie werden mit C-Code gefüllt, der ihrer Aufgabe entspricht. Dabei muss genau wie für das Hauptprogramm überlegt werden,was die Funktion tun soll. Werden Hilfsroutinen aus Bibliotheken benötigt, die noch nicht im Programm eingebunden sind, die jeweilige Bibliothek hinzufügen!
- Kompilieren und Fehler korrigieren, Verbesserungen, etc.

Anmerkungen:

- wenn möglich, bereits nach jeder Funktion testen, ob das Programm funktioniert, d.h. das Programm schrittweise aufbauen
- nie versuchen, alles auf einmal zu erschlagen
- immer erst versuchen, eine einfache Lösung zu programmieren und das Programm dann zu erweitern

2 Beispiele

2.1 Fakultaet

Aufgabenstellung: Schreibt ein Programm, das die Fakultaet einer Zahl n berechnet. Die Zahl soll vom Benutzer erfragt werden.

- Aufgabenstellung: klar
  Voraussetzungen: Zahl n
  Ergebnis: Fakultaet von n
- Sinnvolle Teilaufgabe:
  Berechnung der Fakultaet in einer Funktion (wiederverwendbar!)
- Parameter: Zahl n, natürliche Zahl, also int
  Rückgabewert: Fakultaet von n, große natürliche Zahl, am Besten unsigned long
- Aussehen der Funktion:

  ```
  unsigned long fakultaet(int n)
  {
  }
  ```

- Kommentar hinzufügen:

  ```
  /* fakultaet berechnet die Fakultaet einer Zahl
  /* Parameter: die Zahl */
  /* Rückgabewert: die Fakultaet */
  ```

- Hauptprogramm zufügen:

  ```
  int main()
  {
  }
  ```

- Ablauf des Hauptprogramm:
  eine Variable für die Zahl, die eingelesen werden soll, Typ int
  eine Variable für das Ergebnis, Typ unsigned long
  Zahl wird mit *scanf()* eingelesen und nach der Berechnung ausgelesen
- Hauptprogramm programmieren:

  ```
  #include <stdio.h>
  int main()
  {
    int zahl;
    unsigned long ergebnis;

    printf("Geben Sie eine Zahl ein: ");
    scanf("%d", &zahl);
    ergebnis = fakultaet(zahl);
   printf("Fakultaet von %d ist %u!\n", zahl, ergebnis);
  }
  ```

- Funktion ausprogrammieren:

```
unsigned long fakultaet(int zahl)
{
  unsigned long ergebnis;
  if (zahl > 1)
    ergebnis = zahl * fakultaet(n-1);
  else
    ergebnis = 1;
  return ergebnis;
}
```

2.2 Aufgabe 5 Übungszettel 4

- Aufgabenstellung: klar
  Voraussetzung: zwei Arrays und zwei Zeiger auf diese Arrays
  Ziel: Vergleichen und Einlesen, bis der Benutzer nicht mehr möchte

- Sinnvolle Teilaufgaben:
  - Einlesen des neuen Vergleichsarrays
  - Vergleichen der beiden Arrays
  - Abfrage, ob noch eine Runde durchgeführt werden soll oder nicht, Zeiger vertauschen

- 1. Teilaufgabe: Einlesen
  Parameter: Zeiger auf das Array, Typ Zeiger auf int-Array, Laenge des Arrays, Typ int
  Rückgabewert: keiner

- 2. Teilaufgabe: Vergleichen
  Parameter: Zeiger auf die zu vergleichenden Arrays, Typ Zeiger auf int-Arrays, Laenge der Arrays, Typ int
  Rückgabewert: keiner, da die Änderungen nur auf dem Bildschirm dargestellt werden

- 3. Teilaufgabe: Weitere Runden
  Parameter: Zeiger auf die zu vergleichenden Arrays, Typ Zeiger auf die int-Array Zeiger
  Rückgabewert: 0 wenn ja, -1 wenn nein, also Typ int

- Funktionen:

```
void eingeben(int *array, int laenge)
{
}
void vergleichen(int *array1, int *array2, int laenge)
{
}
int neueRunde(int **array1, int **array2)
{
}
```

- Kommentare an jede Funktion schreiben:

/ eingeben dient dazu, Werte in ein Array einzulesen*
/ die Werte werden vom Benutzer eingegeben und mit scanf() eingelesen*/*
/ Parameter: das Array, kein Rückgabewert */*
...

- Hauptprogramm zufügen:

  int main()
  {
  }

- Ablauf des Hauptprogramms:
  zwei Arrays vom Typ int anlegen, die Größe des Arrays durch ein *#define MAX 10* festlegen (bessere Wiederverwendbarkeit)
  zwei Zeiger auf Arrays vom Typ int festlegen und mit jeweils einem der Arrays initialisieren
  das Array, das zuerst zum Vergleichen dient, mit Nullen initialisieren
  in einer do-while-Schleife zunächst die Funktion *eingeben()* (Einlesen der Vergleichswerte) und dann die Funktion *vergleichen()* (Vergleichen der Arrays) aufrufen
  Schleife bricht ab, wenn *neueRunde()* -1 zurückgibt

- Hauptprogramm programmieren: s. Lösung zu Übungszettel 5

- Ablauf der Funktion *eingeben()*:
  Zählervariable deklarieren, Typ int
  in einer for-Schleife die Werte für das Array einlesen

- Ablauf der Funktion *vergleichen()*:
  Zählervariable deklarieren, Typ int
  in einer for-Schleife die Werte für das Array vergleichen und bei Unterschieden die Stelle (d.h. den Index im Array) ausgeben

- Ablauf der Funktion *neueRunde()*:
  eine Variable vom Typ char für die Antwort des Benutzers ('j' für neue Runde, sonst nicht)
  eine Hilfvarible vom Typ Zeiger auf int-Array, falls die Zeiger getauscht werden müssen
  den Benutzter mit *printf()* fragen, ob er noch eine Runde machen möchte
  die Antwort mit *scanf()* einlesen
  wenn der Benutzer 'j' antwortet, die Zeiger auf die int-Arrays tauschen und 0 zurückgeben
  wenn der Benutzer etwas anderes antwortet, aufhören, d.h. -1 zurückgeben

- Funktionen ausprogrammieren: s. Lösung zu Übungszettel 5