Klausur zur "Einführung in die Programmierung für Physiker" (SoSe 2024)
27. September 2024, 10:15 bis 11:45

Hinweise:

- Falls nicht explizit anders vermerkt beziehen sich alle folgenden Fragen und sämtlicher Programmcode auf die in der Vorlesung ausgiebig diskutierte Programmiersprache C.
- Beim Implementieren einzelner Funktionen müssen auch die für deren fehlerfreie Kompilierung benötigten `#include`-Anweisungen angegeben werden.
- Es können insgesamt **50 Punkte** erzielt werden.

1. **Aufgabe (13 Punkte)**

   Die Fibonacci-Folge $f_1, f_2, f_3, \ldots$ ist durch das rekursive Bildungsgesetz

   $$f_n = f_{n-1} + f_{n-2} \tag{1}$$

   für $n \geq 3$ mit den Anfangswerten $f_1 = f_2 = 1$ definiert.

   (a)
   Implementiere die Funktion

   `void fibo_compute(int n_max, int **p_f)`

   . Die Funktion soll zunächst ein `int`-Array mit `n_max` Elementen dynamisch anfordern (Fehlerabfrage beim Speicheranfordern nicht vergessen). Die Adresse dieses Arrays soll in dem Zeiger vom Typ `int *` gespeichert werden, dessen Adresse mit der Variable `p_f` an die Funktion übergeben wird. Das Array soll dann mit den Fibonacci-Zahlen gefüllt werden: 0-tes Array-Element mit f_1, 1-tes Array-Element mit f_2, ..., `n_max`-1-tes Array-Element mit $f_{\texttt{n\_max}}$. Verwende dazu eine Schleife, die einmalig durch die Elemente des Arrays mit Indizes `2`, `3`, ..., `n_max - 1`, läuft und die entsprechenden Fibonacci-Zahlen berechnet und dort speichert. Die Berechnung soll mit Hilfe von Gleichung (1) und den bereits berechneten und gespeicherten Fibonacci-Zahlen mit kleineren Indizes stattfinden.

   (b)
   Implementiere die Funktion

   `void fibo_free(int **p_f)`

   , die den von `fibo_compute` dynamisch reservierten Speicher wieder freigibt und den Zeiger, dessen Adresse mit der Variable `p_f` an die Funktion übergeben wird, auf `NULL` setzt.

   (c)
   Schreibe ein vollständiges C-Programm, das die Fibonacci Zahlen $f_1, f_2, \ldots, f_{10}$ berechnet und untereinander auf dem Bildschirm ausgibt. Das Programm soll dabei die in Teilaufgabe **(a)** und Teilaufgabe **(b)** implementierten Funktionen verwenden. Nimm dafür an, dass die beiden Funktionen in der include-Datei `fibo.h` deklariert und in `fibo.c` implementiert sind. Dein Programm kann diese Dateien verwenden, soll aber nicht Teil von `fibo.c` sein, sondern in einer separaten `.c`-Datei verfasst werden. Gib dynamisch angeforderten Speicher vor Programmende ordnungsgemäß wieder frei.

   Lösung

   Siehe Dateien `aufgabe_1.c`, `fibo.h`, `fibo.c`.

2. **Aufgabe (8 Punkte)**

   Schreibe ein C-Programm, das beim Aufruf zwei ganze Zahlen als Kommandozeilenargumente erwartet. Das Programm soll diese Zahlen multiplizieren und das Ergebnis in eine neu angelegte Textdatei mit Namen `result.txt` im aktuellen Verzeichnis schreiben. Prüfe dabei, ob tatsächlich zwei Kommandozeilenargumente angegeben wurden. Falls nicht, soll das Programm mit einer Fehlermeldung beendet werden. Prüfe außerdem, ob das Anlegen der Textdatei mit `fopen` funktioniert hat. Falls nicht, soll das Programm ebenfalls mit einer Fehlermeldung beendet werden.

   Lösung

   Siehe Datei `aufgabe_2.c`.

3. **Aufgabe (13 Punkte)**

In der Vorlesung wurde das Runge-Kutta-Verfahren 2-ter Ordung diskutiert, mit dem gewöhnliche Differentialgleichungen der Form

$$\dot{\mathbf{y}}(t) \quad = \quad \mathbf{f}(\mathbf{y}(t), t) \tag{2}$$

für gegebene Anfangsbedingungen

$$\mathbf{y}(t=0) \quad = \quad \mathbf{y}_0 \tag{3}$$

numerisch gelöst werden können. Dabei handelt es sich um ein iteratives Verfahren, bei dem der Runge-Kutta-Schritt

$$\mathbf{k}_1 \quad = \quad \mathbf{f}(\mathbf{y}(t), t)\tau \tag{4}$$
$$\mathbf{k}_2 \quad = \quad \mathbf{f}\Big(\mathbf{y}(t) + (1/2)\mathbf{k}_1, t + (1/2)\tau\Big)\tau \tag{5}$$
$$\mathbf{y}(t+\tau) \quad = \quad \mathbf{y}(t) + \mathbf{k}_2 + \mathcal{O}(\tau^3) \tag{6}$$

mit kleiner Schrittweite τ hinreichend oft wiederholt wird.

(a)
Betrachte im Folgenden das Anfangswertproblem

$$m\ddot{z}(t) \quad = \quad -kz - \beta\dot{z} \quad , \quad z(t=0) \quad = \quad z_0 \quad , \quad \dot{z}(t=0) \quad = \quad 0. \tag{7}$$

Schreibe dieses auf die Standardform (2) und (3) um, d.h. drücke $\mathbf{y}(t)$ und $\mathbf{y}_0$ durch die in Gleichung (7) vorkommenden Größen aus und gib $\mathbf{f}(\mathbf{y}(t), t)$ so an, dass (2) und (3) äquivalent zu (7) sind.

(b)
Von welcher Ordnung in τ ist der numerische Fehler, wenn die Zeitentwicklung von $t = 0$ bis zu fest vorgegebener Zeit $t = T > 0$ mit dem angegebenen Runge-Kutta-Verfahren 2-ter Ordnung berechnet wird? Begründe Deine Antwort, gegebenenfalls mit einer kurzen Rechnung.

(c)
Vervollständige ein C-Programm, dass das Anfangswertproblem (7) mit dem angegebenen Runge-Kutta-Verfahren 2-ter Ordnung löst, d.h. N Runge-Kutta-Schritte mit Schrittweite τ ausführt. Dabei soll die Trajektorie am Bildschirm ausgegeben werden (eine Zeile pro Runge-Kutta-Schritt; in jeder Zeile zuerst t, dann $z(t)$). Verwende als Startpunkt das folgende Programmskelett und ergänze dieses an den durch die drei Kommentare gekennzeichneten Stellen.

```
#include<stdio.h>
#include<stdlib.h>

double m = 1.0;
double k = 1.0;
double beta = 0.1;
double z_0 = 1.0;

double tau = 0.1; // Runge-Kutta-Schrittweite
double N = 500; // Anzahl der Runge-Kutta-Schritte

// berechnet f(y(t),t) * tau für gegebenes y und t
// speichert das Ergebnis in f_times_tau
void compute_f_times_tau(double y[2], double t, double f_times_tau[2])
{

  // !!!!! Hier Code ergänzen (1) !!!!!

}

int main(void)
{
  int i1;


  double t = 0.0;
  double y[2];
  // initialisiere y gemäß den Anfangsbedingungen

  // !!!!! Hier Code ergänzen (2) !!!!!

  printf("%+.5e %+.5e\n", t, y[0]);

  // führe N Runge-Kutta-Schritte 2-ter Ordnung aus
  for(i1 = 0; i1 < N; i1++)
  {

    // !!!!! Hier Code ergänzen (3) !!!!!

    printf("%+.5e %+.5e\n", t, y[0]);
  }
}
```

Hinweis: Eine allgemeine und elegante Programmstruktur, wie im Programmierprojekt umgesetzt (z.B. Verwendung von Funktionszeigern, von `struct`*, etc.), ist hier nicht erforderlich. Das Hinzufügen einiger elementarer Rechenoperationen an den markierten Stellen ist ausreichend, um die Aufgabe zu lösen.*

Lösung

Teilaufgabe (a):
$\mathbf{y}(t) = (z(t), v(t))$
$\mathbf{y}_0 = (z_0, 0)$.
$\mathbf{f}(\mathbf{y}(t), t) = (v(t), -(k/m)z(t) - (\beta/m)v(t))$
(alternativ kann statt $z(t)$ und $v(t)$ auch `y[0]` und `y[1]` geschrieben werden)

Teilaufgabe (b):
Für eine Zeitentwicklung von $t = 0$ nach $t = T$ werden T/τ Runge-Kutta-Schritte (6) benötigt. Da ein Schritt einen Fehler der Ordnung τ^3 hat, wie in Gleichung (6) angegeben, ergibt sich ein Gesamtfehler der Ordung $(T/\tau) \times \tau^3 = \mathcal{O}(\tau^2)$.

Teilaufgabe (c):
Siehe Datei `aufgabe_3.c`.

4. **Aufgabe (8 Punkte)**

(a)

Was gibt das folgende C++-Programm auf `stdout` aus? Eine verbale Beschreibung der Ausgabe ist nicht ausreichend. Gib jede ausgegebene Zeile explizit an.

```
#include<stdio.h>
#include<stdlib.h>

class SQ
{
public:
  SQ(int i);
  ~SQ();
  void Print() const;
private:
  int sq;
};

SQ::SQ(int i)
{
  sq = i*i;
}

SQ::~SQ()
{
  printf("+++\n");
}

void SQ::Print() const
{
  printf("%d\n", sq);
}

int f(int i)
{
  static int m = 0;
  m++;
  SQ sq(i);
  sq.Print();
  return m;
}

int main(void)
{
  int i1;
  int k;
  for(int i1 = 1; i1 <= 3; i1++)
```

```
    k = f(i1);
  printf("k = %d\n", k);
}
```

(b)
Betrachte das gleiche Programm, wie in Teilaufgabe **(a)**, wobei die Funktion `f` wie folgt verändert wird:

```
int f(int i)
{
  int m = 0;
  m++;
  SQ sq(i);
  sq.Print();
  return m;
}
```

Handelt es sich nach wie vor um ein Programm, das ein C++-Compiler ohne Fehlermeldung übersetzen würde? Falls nein, erläutere den Fehler. Falls ja, gib wie in Teilaufgabe **(a)** die Bildschirmausgabe an.

(c)
Betrachte das gleiche Programm, wie in Teilaufgabe **(a)**, wobei die Klassendefinition von `SQ` wie folgt verändert wird:

```
class SQ
{
public:
  SQ(int i);
  ~SQ();
private:
  void Print() const;
  int sq;
};
```

Handelt es sich nach wie vor um ein Programm, das ein C++-Compiler ohne Fehlermeldung übersetzen würde? Falls nein, erläutere den Fehler. Falls ja, gib wie in Teilaufgabe **(a)** die Bildschirmausgabe an.

Lösung

Teilaufgabe (a):

```
1
+++
4
+++
9
+++
```

```
k = 3
```

Teilaufgabe (b):

```
1
+++
4
+++
9
+++
k = 1
```

Teilaufgabe (c):

Außerhalb der Klasse `SQ`, d.h. in der Funktion `f`, wird die private-Methode `Print` aufgerufen. Dies ist nicht gestattet und liefert einen Fehler beim Kompilieren.

5. **Aufgabe (8 Punkte)**

(a)

Betrachte die folgende Funktion, die `b` von `a` subtrahiert.

```
int a_minus_b(int a, int b)
{
  if(b == 0)
    return a;
  if(b > 0)
    return a_minus_b(a-1, b-1);
  return a_minus_b(a+1, b+1);
}
```

Wie nennt man die verwendete Technik, bei der eine Funktion sich selbst aufruft?

Gib das Laufzeitverhalten der Funktion `a_minus_b` an.

(b)

Welche Bildschirmausgabe erzeugt das folgende Programm?

```
#include<stdio.h>

int main(void)
{
  int i = 3;
  double a[3][3] = {
    { 1.0, 2.0, 3.0 },
    { 4.0, 5.0, 6.0 },
    { 7.0, 8.0, 9.0 }
    };
  printf("%d\n", 15 / 3 * 5);
  printf("%d\n", 3 == i + 1);
  printf("%d\n", 7 & 1 << 2);
  printf("%+.3f\n", (*(a+2))[1]);
  printf("%+.3f\n", *(&a[2][2] - 2));
  printf("%.1f\n", i + 1 == (int)(a[1][0]) ? 4.0 : 7.0);
}
```

Hinweis: Verwende gegebenenfalls die der Klausurangabe beiliegende Tabelle zum Vorrang von Operatoren.

Lösung

Teilaufgabe (a):
Rekursion.
Laufzeitverhalten $\mathcal{O}(\texttt{b})$.

Teilaufgabe (b):

```
25
0
```

```
4
+8.000
+7.000
4.0
```