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

Allgemeine 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 **7 + 8 + 14 + 14 + 7 = 50 Punkte** erzielt werden.

1. **Aufgabe (7 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.

   Implementiere eine Funktion

   ```
   int fibo_rec(int n)
   ```

   , die rekursiv durch Aufruf von sich selbst die Fibonacci-Zahl f_n berechnet und zurückgibt. Prüfe in dieser Funktion auch, ob die Variable `n` einen zulässigen Wert ≥ 1 hat. Falls dies nicht der Fall ist, gib eine Fehlermeldung aus und beende das Programm.

   Lösung

   Siehe Datei `aufgabe_1.c`.

2. **Aufgabe (8 Punkte)**

(a)
Welchen Wert hat der Ausdruck

```
64 / 8 / 2
```

in der Programmiersprache C? Begründe Deine Antwort und erläutere in diesem Zusammenhang das Konzept der Linksassoziativität.

Welchen Wert hätte der genannte Ausdruck, wenn der Operator `/` rechtsassoziativ ausgewertet werden würde?

(b)
Welche Bildschirmausgabe erzeugt das folgende Programm?

```
#include<stdio.h>

int main(void)
{
  int i = 1;
  int j = 0;
  int k = 0;
  double a[4] = {8.0, 2.0, 3.0, 4.0};

  printf("%d\n", 17 - 5 % 3);
  printf("%+d\n", i || j && k);
  printf("%+.3f\n", *(a+2) + a[3] - (double)(++j));
  printf("%.3f\n", *a+2 + a[3] - (double)(k++));
  printf("%d\n", j + k + !j * 3);
  printf("%.3f\n", i == j ? -a[2] : +a[2]);
}
```

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

Lösung

Teilaufgabe (a): Der Wert von `64 / 8 / 2` ist `4`. Bei Rechtsassoziativität von `/` wäre der Wert `16`. Erläuterung von Linksassoziativität: Siehe Vorlesungsfolien.

Teilaufgabe (b): `15`
`+1`
`+6.000`
`14.000`
`2`
`-3.000`

3. **Aufgabe (14 Punkte)**

In der Vorlesung wurden numerische Verfahren zum Finden von Nullstellen einer Funktion $f(x)$ diskutiert, die Bisektion und das Newton-Raphson-Verfahren. Bei beiden Verfahren handelt es sich um iterative Verfahren, bei denen schrittweise die gesuchte Nullstelle immer besser approximiert wird.

(a)
Erläutere präzise und vollständig einen Iterationsschritt der Bisektion. Gehe dabei davon aus, dass zu Beginn des Iterationsschrittes ein Intervall durch die beiden Werte $x_L < x_R$ gegeben ist, in dem mindestens eine Nullstelle vorliegt.

(b)
Erläutere präzise und vollständig einen Iterationsschritt des Newton-Raphson-Verfahrens. x_n soll dabei den aktuellen Schätzwert für die gesuchte Nullstelle beschreiben. Beschreibe zunächst die geometrische Grundidee eines Iterationschritts in Worten (eventuell ergänzt durch eine Skizze). Gib dann eine entsprechende Formel an, die den neuen Schätzwert für die gesuchte Nullstelle, x_{n+1}, mit x_n in Beziehung setzt.

(c)
Implementiere eine Funktion

```
void NR(double *p_x, double eps)
```

, in der das Newton-Raphson-Verfahren ausgeführt wird. Der Parameter `p_x` enthält die Adresse einer `double`-Variable, die beim Funktionsaufruf den Startwert von x für das Newton-Raphson-Verfahren enthält und in der beim Verlassen der Funktion der iterativ berechnete Schätzwert für eine Nullstelle von $f(x)$ gespeichert wird. Die Iteration soll abgebrochen werden, sobald $|x_{n+1} - x_n| <$ `eps` gilt. Die Funktion `NR` kann die beiden Funktionen

```
double f(double x)
double df_dx(double x)
```

verwenden, die $f(x)$ beziehungsweise $f'(x)$ berechnen. Mögliche Konvergenzprobleme und andere Fehlerabfragen können bei dieser Aufgabe ignoriert werden.

(d)
Nenne einen Vorteil der Bisektion gegenüber dem Newton-Raphson-Verfahren.
Nenne einen Vorteil des Newton-Raphson-Verfahrens gegenüber der Bisektion.

Lösung

Teilaufgabe (a): Falls $f(x_L)f((x_L + x_R)/2) < 0$, ersetze das aktuelle x_R durch das neue $x_R = (x_L + x_R)/2$. Andernfalls ersetze das aktuelle x_L durch das neue $x_L = (x_L + x_R)/2$.

Teilaufgabe (b): Approximiere die Funktion $f(x)$ bei x_n durch ihre Tangente. Der Schnittpunkt der Tangente mit der x-Achse entspricht dem neuen Schätzwert x_{n+1}. Die mathematische Beziehung zwischen x_{n+1} und x_n lautet damit

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{2}$$

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

Teilaufgabe (d): Bei der Bisektion ist das Finden einer Nullstelle garantiert, beim Newton-Raphson-Verfahren nicht. Das Newton-Raphson-Verfahren ist schneller als die Bisektion.

4. **Aufgabe (14 Punkte)**

Vervollständige das folgenden C++-Programm, indem Du an der mit dem Kommentar gekennzeichneten Stelle die vier Methoden der Klasse `Vector` implementierst.

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

class Vector
{
public:
  Vector(int n_, double v_[]);
  ~Vector();
  void Print() const;
  double Norm() const;

private:
  int n;
  double *v;
};

// ***** Hier die Methoden der Klasse Vector implementieren. *****

int main(void)
{
  double tmp2[2] = {1.0, 1.0};
  Vector a(2, tmp2);
  tmp2[0] = -3.0;
  tmp2[1] = 4.0;
  Vector b(2, tmp2);
  a.Print();
  printf("norm␣=␣%f\n", a.Norm());
  b.Print();
  printf("norm␣=␣%f\n", b.Norm());
  double tmp4[4] = {1.0, 1.0, 1.0, 1.0};
  const Vector c(4, tmp4);
  c.Print();
  printf("norm␣=␣%f\n", c.Norm());
}
```

Die Klasse `Vector` repräsentiert einen Vektor mit einer vom Benutzer wählbaren Anzahl von Komponenten. Die Methoden sollen Folgendes leisten:

- `Vector(int n_, double v_[])`
  Ein Konstruktor, der einen neuen Vektor mit `n_` Elementen anlegt und mit den Einträgen des Arrays `v_` initialisiert.
  - Der Parameter `n_` muss eine positive Zahl sein. Falls dies nicht der Fall ist, soll eine Fehlermeldung ausgegeben und das Programm beendet werden.

  - Der Wert von `n_` soll in der Variable `n` gespeichert werden.
  - Mit Hilfe von `malloc` soll dynamisch ein `double`-Array mit `n_` Elementen angefordert (Fehlerabfrage beim Speicheranfordern nicht vergessen) und dessen Adresse in `v` gespeichert werden.
  - Die ersten `n_` Elemente des Arrays `v_` sollen in das Array `v` kopiert werden.

- `~Vector()`
  Der Destruktor soll den dynamisch reservierten Speicher wieder freigeben.
- `void Print() const`
  Diese Funktion soll den gespeicherten Vektor in speziell formatierter Weise (siehe unten) auf `stdout` ausgeben.
- `double Norm() const`
  Diese Funktion soll die Euklidische Norm des gespeicherten Vektors berechnen und zurückgeben.

Nach Implementierung dieser Methoden soll das Programm die folgende Bildschirmausgabe liefern.

```
(+1.000 , +1.000)
norm = 1.414214
(-3.000 , +4.000)
norm = 5.000000
(+1.000 , +1.000 , +1.000 , +1.000)
norm = 2.000000
```

Lösung

Siehe Datei `aufgabe_4.C`.

5. **Aufgabe (7 Punkte)**

Was gibt das folgende C-Programm auf `stdout` aus? Eine präzise und vollständige verbale Beschreibung der Ausgabe ist ausreichend. Es ist nicht notwendig jede ausgegebenen Zeile explizit hinzuschreiben.

```
#include<stdio.h>

int main(void)
{
  int i1, i2;
  for(i1 = 1; i1 <= 100; i1++)
  {
    for(i2 = 2; i2 <= i1/2; i2++)
    {
      if(i1%i2 == 0)
        break;
    }
    if(i2 == i1/2+1)
      printf("%d\n", i1);
  }
}
```

Lösung

Alle Primzahlen ≤ 100 in aufsteigender Reihung, eine Zahl pro Zeile.