Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institut für theoretische Informatik
Dr. J. Giesen

17. November 2004

Informatik I für D-ITET — Serie 5 — WS 04/05

URL: `http://www.ti.inf.ethz.ch/ew/courses/inf1-ITET/`

Aufgabe 1

Holen Sie sich die Datei `libifet-1.17.tgz` von der Vorlesungswebseite, unt entpacken Sie sie mit dem folgenden Kommando.

```
gunzip -c libifet-1.17.tgz | tar xvf -
```

Als Folge erhalten Sie ein Unterverzeichnis `libifet-1.17` mit den Dateien dreier kleiner Bibliotheken: `libwindow`, `libturtle` und `libmath`. Wechseln Sie ins Verzeichnis `libifet-1.17` und erstellen Sie die Bibliotheken. Folgen Sie hierfür den Anweisungen in der Datei `README`.

Aufgabe 2 (6 Punkte)

Implementieren Sie in der Bibliothek `libmath`

- eine Funktion `double sqrt(double)` zur Berechnung von Quadratwurzeln mittels Newton-Iteration sowie
- eine Funktion `double random(double)` zur Berechnung von Pseudozufallszahlen aus dem Intervall $[0,1)$ nach der linearen Kongruenzmethode ($x_i = (a * x_{i-1} + c)\%m$).

Gehen Sie dabei wie folgt vor.

1. Die Deklarationen der Funktionen schreiben Sie in die Datei `libifet-1.17/include/IFET/math.h`.
2. Die entsprechenden Implementationen schreiben Sie in die Datei `libifet-1.17/src/math.C`.
3. Wählen Sie einen Generator aus, für den die notwendigen Berechnungen auf dem Datentyp `double` ausgeführt werden können. (Ggf. unter Zuhilfenahme von `int` beim Runden.) Auf der Webseite `http://random.mat.sbg.ac.at/~charly/server/node3.html` finden Sie eine Liste von Generatoren nach der linearen Kongruenzmethode. (Nicht alle dort aufgeführten Generatoren erfüllen jedoch die genannten Voraussetzungen.)

Erstellen Sie die Bibliothek `libmath` wie in Aufgabe 1. Schicken Sie die Dateien `math.h` und `math.C` an Ihre Übungsleiterin.

Aufgabe 3 (6 Punkte)

Verändern Sie das Programm `muenzwurf.C` aus der dritten Vorlesung, so dass es die in Aufgabe 2 implementierte Funktion `random()` zur Berechnung von Pseudozufallszahlen verwendet. Testen Sie den Generator! Liefert er "vernünftige" Ergebnisse?

Abgabe: Aufgaben 2 und 3: bis 28. November 2004, 12.00 Uhr, per Email.

Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institut für theoretische Informatik
Dr. J. Giesen

17. November 2004

Informatik I: Material aus der Vorlesung

Programm: math.h

```
// Programm: math.h
// Kleine Bibliothek mathematischer Funktionen.

namespace ifet {

  double sqrt(double n);
  // PRE: n >= 0.
  // POST: Gibt eine Approximation der Quadratwurzel
  //       von n zurueck.

} // namespace ifet
```

Programm: math.C

```
// Programm: math.C
// Kleine Bibliothek mathematischer Funktionen.

#include "math.h"
#include <cassert>

namespace ifet {

  double sqrt(double n)
  {
    // PRE: n >= 0.
    // POST: Gibt Approximation der Quadratwurzel
    //       von n zurueck.

    assert(n >= 0);
    if (n == 0) return 0;

    // Newton-Iteration: x_{i+1} = 1/2 (x_i + n/x_i)

    // Newton-Iteration: x_{i+1} = 1/2 (x_i + n/x_i)
    // Wir beginnen mit einem Startwert x_0 > sqrt(n).
    // Wie in der Theorie verlangen wir, dass der
    // Approximationswert in jedem Schritt kleiner
    // wird. Damit terminiert das Programm, und wir
    // koennen sogar beweisen, dass "ungefaehr"
    // das richtige herauskommt

    double curr = n + 1;  // aktueller Wert
    double prev;          // voriger Wert

    do {
      prev = curr;
      assert(curr > 0);
      curr = (curr + n / curr) / 2;
    } while (curr < prev);

    return curr;

  } // double sqrt(double)

} // namespace ifet
```

Programm: prime.C

```
// Programm: prime.C
// Testet, ob eine Zahl Primzahl ist.
// Wenn nicht, Ausgabe des kleinsten Primteilers.

#include <iostream>
#include "math.h"

int main ()
{
  // Einlesen der Eingabe
  std::cout << "n? (2 <= n <= "
            << std::numeric_limits<unsigned int>::max()
            << ") ";
  unsigned int n;
  std::cin >> n;

  // Alle Zahlen zwischen 2 und sqrt(n) sind
  // potentielle Teiler.
  unsigned int t = 2;
  unsigned int end = 1 + (unsigned int)(ifet::sqrt(n));
  while (t <= end && n % t != 0)
    ++t;

  // Ausgabe des Ergebnisses
  if (t > end || t >= n)
    std::cout << n << " ist Primzahl." << std::endl;
  else
    std::cout << n << " ist keine Primzahl und " << t
              << " ist ihr kleinster Primteiler."
              << std::endl;
  return 0;
}
```