Prof. G. Zachmann
Dipl.-Inf. C. Giesemann

TU Clausthal
Institut für Informatik
14. Januar 2006

Wintersemester 2005/2006

Übungen zu Grundlagen der Programmierung in C - Blatt IX

Abgabe vom 18.1.2006 bis 24.1.2006 in der angemeldeten Übung

Aufgabe 1 (Stirlingsche Zahlen, *6 Punkte*)

Hinweise:
Für die Berechnung der Stirlingschen Zahlen 1. Art S(n,r) gelten folgende Voreinbarungen:

- $S(n,r) = 0$ für $r < 0$ oder $r > n$
- $S(n,n) = 1\ \forall n$
- $S(n,0) = 0\ \forall n \geq 1$

Es gilt zudem folgende Gleichung:

- $S(n,r) = S(n-1,r-1) - (n-1) \cdot S(n-1,r)$

1. Schreiben Sie ein Programm, das die Stirlingschen Zahlen 1. Ordnung rekursiv berechnet. Fordern Sie den Benutzer auf **n** und **r** einzugeben und geben Sie die berechneten Zahlen in einer formatierten Tabelle wie im folgenden Beispiel für `r = 4` und `n = 4` aus:

```
n r|      0      1      2      3      4
-------------------------------------
 0 |      1      0      0      0      0
 1 |      0      1      0      0      0
 2 |      0     -1      1      0      0
 3 |      0      2     -3      1      0
 4 |      0     -6     11     -6      1
```

Aufgabe 2 (Modulare Exponentiation, *6 Punkte*)

Hinweise:
Überlegen Sie sich zuerst den Basisfall und dann den Rekursionschritt.
Für die Potenz von b^x gilt folgendes:

- $b^x = 1$, falls x = 0
- $b^x = b^{\frac{x}{2}} \cdot b^{\frac{x}{2}}$, falls x gerade
- $b^x = b \cdot b^{\frac{x-1}{2}} \cdot b^{\frac{x-1}{2}}$, falls x ungerade

1. Schreiben Sie ein Programm, das rekursiv die modulare Exponentiation durchführt: b^x mod m
   Fordern Sie den Benutzer auf b, x und m als `integer` Zahlen einzugeben und lesen Sie diese ein. Schreiben Sie eine rekursive Funktion zur Potenzberechnung und führen Sie darin die modulo-Berechnung bei jeder Rückgabe aus, damit die Werte nicht zu groß werden. Geben sie nach der Berechnung das Ergebnis aus. Testbeispiele:

- $b = 2, x = 3, m = 5$ mit $2^3 \bmod 5 = 3$
- $b = 3, x = 3, m = 6$ mit $3^3 \bmod 6 = 3$
- $b = 4, x = 2, m = 2$ mit $4^2 \bmod 2 = 0$
- $b = 5, x = 1, m = 3$ mit $5^1 \bmod 3 = 2$

Aufgabe 3 (Weg aus dem Labyrinth, *10 Punkte*)

Hinweise:
Ein Labyrinth kann als 2-dimensionales `array` dargestellt werden und besteht aus Mauern und Zwischenräumen. Die Mauern werden im Programm durch # und die Zwischenräume durch Leerzeichen beschrieben. Der Startpunkt ist mit **S** und der Ausgang mit > gekennzeichnet (Bild A). Der Spieler kann sich in 4 Richtungen bewegen (**rechts**, **unten**, **links**, **oben**), aber nur wenn der entsprechende Nachbarpunkt ein Leerzeichen enthält. Die Felder des zurückgelegten Weges werden mit **x** gekennzeichnet und sind damit nicht leer. Wenn der Spieler zurückgeht, wird die Spur gelöscht.
Die Wegesuche vom Startpunkt zum Ausgang lässt sich durch eine Rekursion mit Backtracking beschreiben. Startaufruf ist: `testeSchritt(1,1)`. Die Funktion `testeSchritt` testet zunächst, ob der Ausgang (>) erreicht ist und gibt in diesem Fall `true` zurück. Andernfalls wird getestet, ob der Punkt eine Wand ist oder schon besucht wurde und bei erfüllter Bedingung wird `false` zurückgegeben. Wenn das aktuelle Feld zulässig ist, wird der nächste Schritt versucht, d.h. die Funktion `testeSchritt` wiederum mit einem Nachbarpunkt aufgerufen. Die Funktion `testeSchritt` könnte z.B. folgende Struktur haben:

```
Funktion testeSchritt (x,y)
   integer x = x-Position
   integer y = y-Position
   bool-Rückgabewert = Ausgang gefunden

   falls Position(x,y) = Ausgang
       return true;
   falls Position(x,y) = besetzt(Wand oder x)
       return false;
   setze Markierung x an Position(x,y);
   falls testeSchritt(x+1,y) // nach rechts gehen
       return true;
   falls testeSchritt(x,y-1) // nach unten gehen
       return true;
   falls ... // nach links gehen
   ...
   falls ... // nach oben gehen
   ...
   entferne Markierung x an Position (x,y) (mit <Space> ersetzen)
   return false;
```

Die weiteren 3 Bilder stellen Stationen eines Beispieldurchlaufes dar. Im Bild B ist der Spieler zwei Schritte nach rechts gelaufen und befindet sich in einer Sackgasse. Nachdem alle 4 Nachbarpunkte `false` geliefert haben (3x Wand und 1x bereits besucht) erfolgt der Rücksprung mit `false`. Ein Schritt zurück liefert ebenfalls `false` (1x zurückgekehrt, 2x Wand, 1x bereits besucht), daher muß der Spieler einen weiteren Schritt zurückkehren (Startpunkt). Von dort gibt es eine noch nicht benutzte Wegalternative. In Bild C hat der Spieler 6 Schritte geschafft und in Bild D den Ausgang gefunden.

```
   Bild A          Bild B          Bild C          Bild D
############    ############    ############    ############
#S  #     ##    #xxx#     ##    #x  #     ##    #x  #xxxxx##
# ### ###  #    # ### ###  #    #x### ###  #    #x###x###x #
#      #  ##    #      #  ##    #xxxxx #  ##    #xxxxx # x##
#########  #    #########  #    #########  #    #########xx#
## #   ### #    ## #   ### #    ## #   ### #    ## #   ###x#
#  # #     #    #  # #     #    #  # #     #    #  # #xxxxx#
# ## # #####    # ## # #####    # ## # #####    # ## #x#####
#    #     >    #    #     >    #    #     >    #    #xxxxx>
############    ############    ############    ############

# = Wand, S = Startpunkt, > = Ausgang
```

1. Schreiben Sie ein Programm, das ein Labyrinth als `array` ([y_Dimension][x_Dimension]) nach Bild A enthält und eine Funktion `druckeLabyrinth` benutzt, um das Labyrinth auf dem Bildschirm auszugeben. In der Datei `CodeIX.txt` auf der Vorlesungsseite sind die `arrays` und die Funktion `druckeLabyrinth` enthalten, die Sie in Ihr Programm übernehmen können.
2. Implementieren Sie die rekursive Funktion `testeSchritt`.
   Hilfen zum Testen:
   - Am Anfang der Funktion eine Ausgabe des Labyrinths einbauen.
   - Einbauen einer Wartefunktion (z.B. getchar() – Warte auf Tastendruck) nach jeder Ausgabe zur besseren Betrachtung.
3. Testen Sie nun, ob Ihr Programm mit diesem weiteren Labyrinth (Bild E) funktioniert:

```
   Bild E
############
#S  #      #
# #### # # #
#      # ###
# ######   #
# ##   ### #
#  # #  #  #
# ## # ### #
#    #     >
############
```

4. Verändern Sie die Reihenfolge der Richtungen in Ihrer Funktion um zu testen, ob Sie in dem zweiten Labyrinth (Bild E) einen anderen Weg finden.