ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatik I - D-ITET

Wintersemester 2003/2004

Prof. Markus Gross, Bruno Heidelberger, Richard Keiser, Nicky Kern, Edouard Lamboray, Christoph Niederberger, Tim Weyrich, Felix Eberhard, Manuel Graber, Nathalie Kellenberger, Marcel Kessler, Lior Wehrli

Uebung 9 - Funktionen II

Ausgabe: 12. Januar 2004
Abgabe: 19. Januar 2004
Autor: Richard Keiser

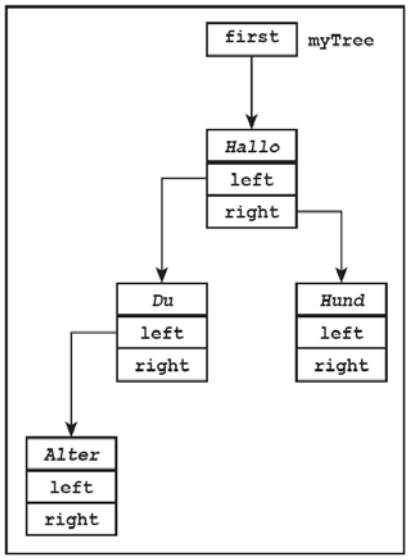
1. Fakultät (3 Punkte)

Die mathematische Funktion der Fakultät ist folgendermassen definiert:

$$n! = \prod_{i=1}^{n} i = n \prod_{i=1}^{n-1} i = n \cdot (n-1)! \text{ und } 0! = 1$$

Die Funktion eignet sich sehr gut für eine rekursive Implementierung, weil die Fakultät von n gleich n mal der Fakultät von (n-1) ist. Schreibe ein Programm, welches eine Zahl einliest, die Fakultät davon rekursiv berechnet und dann ausgibt.

2. Suchbaum (7 Punkte)



In dieser Aufgabe wird ein einfacher Suchbaum für Strings behandelt (sie wurde in ähnlicher Form an der Prüfung im Sommer 2003 gestellt). Der Suchbaum besteht aus Knoten, in denen jeweils ein String gespeichert ist. Jeder Knoten hat zwei Nachfolger, einen *Linken* und einen *Rechten*. Es gilt die Regel, dass die Knoten lexikalisch, also wie in einem Wörterbuch, geordnet sind. Alle Knoten, die kleiner sind als der Aktuelle, werden "unterhalb" des linken Nachfolgers gespeichert, solche die grösser sind, "unterhalb" des Rechten.
Das nebenstehende Bild zeigt einen solchen Suchbaum. Die Felder `links` und `rechts` sind Zeiger auf die entsprechenden Nachfolger. `links/rechts`-Felder ohne entsprechende Pfeile sind `NULL`-Zeiger.
Das Beispiel zeigt einen entsprechenden Baum, nachdem die Worte *Hallo Du Alter Hund* (in dieser Reihenfolge) eingefügt wurden. *Hallo* ist als erstes Element die Wurzel des Baumes. *Du* ist nun kleiner und ist dementsprechend der linke Nachfolger, etc.
Der Baum, den wir für diese Aufgabe verwenden, soll mit Hilfe der folgenden zwei `structs` implementiert werden:

```
// Dieser struct beschreibt die Knoten des Baumes.
// Im Beispiel also die Knoten 'Hallo', 'Du', 'Alter' und 'Hund'
struct treeNode {
            char* name;
            treeNode *left, *right;
};

// Dieser struct enthält nur den Zeiger auf die Wurzel des Baumes
// Der Baum ist leer, wenn first auf keinen Knoten zeigt (NULL ist).
struct treeRoot {
            treeNode *first;
};
```

a) Anlegen eines neuen Knotens ***(2 Punkte)***

Schreibe eine Funktion

```
treeNode* newNode(char* s);
```

die einen neuen Knoten dynamisch alloziert, diesen sinnvoll initialisiert und zurückgibt. Das Feld `name` des Knotens soll den in `s` übergebenen String erhalten. Verwende dazu die Funktionen `strcpy()` und `strlen()`. Beachte, dass Du nicht davon ausgehen kannst, dass `s` nach dem Aufruf von `newNode()` noch weiter existieren wird.

b) Löschen eines Knotens ***(1 Punkt)***

Schreibe eine Funktion

```
void deleteNode(treeNode* n);
```

die den gesamten Speicher freigibt, welcher von dem Knoten `n` belegt wird.

Einfügen eines neuen Knotens

Knoten können in Suchbäumen immer nur am Ende angefügt werden, d.h. an Knoten, die noch keinen entsprechenden Nachfolger haben. Es muss, beginnend bei der Wurzel, entschieden werden, ob der Knoten links oder rechts anzufügen ist. Hat der aktuelle Knoten bereits einen entsprechenden Nachfolger, muss dieselbe Frage für den Nachfolger entschieden werden. Erreicht man das Ende des Baumes, d.h. einen Knoten ohne entsprechenden Nachfolger, so kann man den neuen Knoten einfügen. Die Wurzel braucht eine Sonderbehandlung, falls der Baum noch komplett leer ist.

c) Anfügen an einen Knoten ***(2 Punkte)***

Schreibe eine Funktion

```
void addToNode(treeNode* node, char* name);
```

die einen neuen Knoten erstellt (mit Hilfe von `newNode()`) und unter einem gegebenen Knoten `node` anfügt. Die Funktion soll entscheiden, ob der neue Knoten an den rechten oder linken Nachfolgezeiger angehängt werden soll. Falls ein Knoten `name` schon existiert, soll nichts geschehen. Hat der Knoten `node` schon einen entsprechenden Nachfolger, so soll die Funktion sich selber rekursiv aufrufen, bis der neue Knoten eingefügt werden kann.

Hinweis:
Verwende für den Vergleich die Funktion `int strcmp(char* a, char* b).`
Die Funktion liefert eine Zahl < 0 wenn a < b, == 0 wenn a == b und > 0 wenn a > b.

d) Einfügen in einen Baum *(1 Punkt)*

Schreibe eine Funktion, welche einen neuen Knoten in einen Baum einfügt.

e) Anwendung *(1 Punkt)*

```
// zeigt auf den Baum in der obigen Abbildung
treeRoot* myTree;

void printNode(treeNode* n, int level=1)
{
      if (n->left) printNode(n->left, level + 1);
      cout << level << ": " << n->name << endl;
      if (n->right) printNode(n->right, level + 1);
}
```

Ueberlege was printNode(myTree->first) macht. Schreibe nun eine Anwendung, welche einen Baum erzeugt und anschliessend die Worte *Hallo Du Alter Hund* einzeln und in dieser Reihenfolge einfügt. Gib dann den Baum mit der Funktion printNode(myTree->first) aus. Vergiss nicht auch den Output abzugeben.

3. Unix: Pipes (fakultativ)

Die Unix-Shell ist ein sehr mächtiges Werkzeug. So kann der Input eines Programmes anstelle von der Tastatur ganz einfach aus einem File gelesen werden. Auch der Output kann anstatt auf die Kommandozeile in ein File umgelenkt werden:
Wenn in `in.file` der gesamte Input für ein Programm steht und wir rufen dieses per
`> prg <in.file >out.file`

auf, so seht ihr keinen Output und müsst (hoffentlich) nichts eingeben. Wenn ihr nun aber das File `out.file` per `cat out.file` ausgebt, so seht ihr, dass der gesamte Output in dieses File geschrieben wurde.

Ein sehr nützliches Tool ist `grep`. Es durchsucht ein File nach einem bestimmten Muster und gibt alle Zeilen aus, in denen das Muster (ein beliebiger String) gefunden wurde:
`> grep muster file`

Ein anderes nützliches Tool ist die Pipe. Sie wird mit dem Symbol | dargestellt und lenkt den Output eines Programmes direkt an den Input eines anderen:
`> ps -ef | grep username` (ersetze username mit deinem eigenen!)

Hier wird die gesamte Prozessliste (Ausgabe von `ps -ef`) dem Programm `grep` übergeben, welches nur die Zeilen durchlässt, welche Deinen Username enthalten. Vergleiche dafür die Ausgabe von `ps -ef` mit der obigen.

Weitere nützliche Tools sind `head`, `tail` und `wc`:

`> head -n file`	(gibt die ersten `n` Zeilen von `file` aus)
`> tail -n file`	(gibt die letzten `n` Zeilen von `file` aus)
`> wc file`	(gibt die Anzahl Zeilen, Wörter und Zeichen in `file` aus)