Hochschule Deggendorf	***Platzziffer:***
Prüfungsfach: Algorithmen und Datenstrukturen	
Aufgabensteller: **Prof. Jüttner**	
Prüfungstermin: 12.7.12	
6 ***Aufgaben, Arbeitszeit: 90 Min.***	
zugelassene Hilfsmittel: **alle**	

☞ *Keinen Rotstift verwenden!* ☞ *nur an den vorgesehenen Stellen antworten!*	☞ *ggf. Rückseite verwenden!*

1. O-Notation

Gelten folgende Aussagen? Geben Sie eine Begründung für Ihre Antwort an. (6 Punkte)

a. $100n^2 \in 1/3 * n^3$

b. $(1/100^4)\ n^4 \in 10^7 * n^3$

c. $100n \in 3 * \log(n)$

2. Aufwand eines Sortieralgorithmus

Im Folgenden ist ein C-Algorithmus angegeben, der auf merkwürdige (saudumme) Art und Weise ein Feld von 3 int Zahlen aufsteigend sortiert. Das Feld wird als Parameter übergeben und sortiert.

```
void Saudumm_Sort(int feld[3])
{ int i;

  /* Bilde alle möglichen Kombinationen von feld auf lokalen Vektor Variablen */

  int f1[3] = {feld[0], feld[1], feld[2]};        int f2[3] = {feld[0], feld[2], feld[1]};
  int f3[3] = {feld[1], feld[0], feld[2]};        int f4[3] = {feld[1], feld[2], feld[0]};
  int f5[3] = {feld[2], feld[0], feld[1]};        int f6[3] = {feld[2], feld[1], feld[0]};

  /* Stelle fest, welcher Vektor sortiert ist  und kopiere Elemente wieder auf feld */

  if ((f1[0] <= f1[1]) && (f1[1] <= f1[2]))
    { for (i=0; i<=2; i++) feld[i] = f1[i]; return;}
  if ((f2[0] <= f2[1]) && (f2[1] <= f2[2]))
    { for (i=0; i<=2; i++) feld[i] = feld2[i]; return;}
  if ((f3[0] <= f3[1]) && (f3[1] <= f3[2]))
    { for (i=0; i<=2; i++) feld[i] = feld3[i]; return;}
  if ((f4[0] <= f4[1]) && (f4[1] <= f4[2]))
    { for (i=0; i<=2; i++) feld[i] = feld4[i]; return;}
  if ((f5[0] <= f5[1]) && (f5[1] <= f5[2]))
    { for (i=0; i<=2; i++) feldd[i] = feld5[i]; return;}
  if ((f6[0] <= f6[1]) && (f6[1] <= f6[2]))
    { for (i=0; i<=2; i++) feld[i] = feld6[i]; return;}
}
```

a. Geben Sie an, wie viele Bewegungen von Werten, d.h. Zuweisungen an Elemente von Vektoren im günstigsten und im ungünstigsten Fall beim Sortieren eines 3-elementigen Vektors mittels dieses Algorithmus anfallen. (5 Punkte)

b. Geben Sie an, wie viele Vergleiche von Werten im günstigsten und im ungünstigsten Fall beim Sortieren eines 3-elementigen Vektors mittels dieses Algorithmus anfallen. (5 Punkte)

c. Wie groß ist der Aufwand (O(?)), wenn auf diese saudumme Art und Weise Vektoren der Länge n (n>3) sortiert werden?
Hinweis: Überlegen Sie, wie viele Möglichkeiten (Permutationen) es gibt, n Elemente in einem Vektor anzuordnen. (Bei einer Anzahl von n Elementen gibt es n Möglichkeiten das erste Element anzuordnen, n-1 Möglichkeiten das zweite anzuordnen, n-2 für das dritte, n-3 für das vierte … 1 Möglichkeit für das letzte)
Für n=1 ist dies 1, für n=2 sind es 2, für n=3 sind es 6, für n=4 sind es 24, für n=5 gibt es 120 usw.)
(5 Punkte)

d. Warum heißt dieses Sortierverfahren wohl „Saudumm_Sort“? (2 Punkte)

3. Rekursion

a. Der Binomialkoeffizient $\binom{n}{k}$ ist definiert durch

$\binom{n}{k} := n! / (k!*(n-k)!)$ für natürliche Zahlen n und k mit $n \geq k$

zusätzlich gilt:
falls k>n ist der Wert 0
falls k=1 ist der Wert n
falls k=0 ist der Wert 1

Der Binomialkoeffizient lässt sich auch folgendermaßen rekursiv darstellen:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Schreiben Sie eine rekursive C-Funktion mit 2 Parametern vom Typ unsigned long und dem Ergebnistyp unsigned long, die den Binomialkoeffizient nach der obigen,

rekursiven Formel berechnet und das Ergebnis per return zurückgibt. Achten Sie auf die Terminierungsfälle. (16 Punkte)

4. Rekursion und Datentypen

Schreiben Sie eine rekursive C-Funktion verdopplung, die in einer Liste von Integerzahlen (Typ int) nach einem bestimmten Element sucht und jedes Vorkommen des Elements „verdoppelt“, d.h. das gesuchte Element kommt danach in der Liste doppelt so oft vor. Wo vorher ein Vorkommen des gesuchten Elements in der Liste war, sind es jetzt 2.

Beispiel: Aus der Liste 1->3->5->5->4->5->3->4->5 wird durch Verdopplung der Vorkommen des Elements 5 die Liste 1->3->5->5->5->5->4->5->5->3->4->5->5

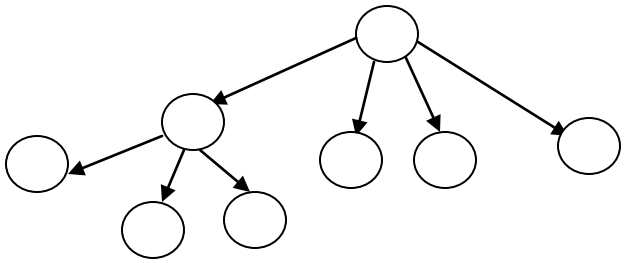
Die Liste soll per Parameter vom Typ Pointer auf Liste an die Funktion übergeben werden, das zu verdoppelnde Element ebenfalls per Parameter. Das Funktionsergebnis wird per return als Pointer auf eine Liste zurückgegeben. Zur Umsetzung der Anforderungen und Manipulation der Liste dürfen in der Funktion nur die aus der Vorlesung bekannten Listenfunktionen emptylist, append, head, tail, isempty verwendet werden. Die Schnittstellen dieser Listenfunktionen sind im Anhang dargestellt

(23 Punkte)

Lösungstipps:
- Überlegen Sie, was das Ergebnis bei einer leeren Liste ist (Terminierungsfall)
- Überlegen Sie, was Sie tun müssen, wenn das erste Element der Liste (head) das gesuchte Element ist?
- Überlegen Sie, was Sie tun müssen, wenn das erste Element der Liste (head) nicht das gesuchte Element ist?

5. Datentypen

Quartäre Bäume sind Bäume, die 4-fach verzweigen, d.h. ein quartärer Baum ist entweder leer oder besteht aus einem Knoten mit einem Datenelement, einem linken Ast, einem mittel-linken Ast, einem mittel-rechten Ast und einem rechten Ast, an denen jeweils ein Quartärer Baum hängt. Jeder Ast darf auch leer sein (s. Beispielskizze)



a. Definieren Sie mittels typedef einen C-Datentyp Quartärer_Baum, mit einem Integer Element im Knoten und Darstellung der Teilbäume über Pointer. (3 Punkte)

b. Geben Sie eine C-Funktion an, die einen Pointer auf den linken Teilbaum (Ast) eines Quartären Baums als Funktionsergebnis zurückliefert Der Baum, dessen Teilbaum zurückgegeben wird, soll per Pointer als Parameter an die Funktion übergeben werden. Kopieroperationen müssen nicht ausgeführt werden.
Falls von einem leeren Baum (d.h. Parameter hat Wert NULL) ein Teilbaum ermittelt werden soll, soll NULL zurückgegeben werden.
(6 Punkte)

c. Schreiben Sie eine C-Funktion, die die Anzahl der Knoten eines Quartären Baums zählt und als Funktionsergebnis zurückgibt. Der Baum soll per Pointer als Parameter übergeben werden. Ein leerer Baum wird durch den NULL-Pointer dargestellt und hat 0 Knoten. (Der Baum in der Skizze hat 8 Knoten)
(8 Punkte)

6. Hashfunktionen

a. Nennen Sie einen Vorteil und einen Nachteil eines offenen Hashverfahrens? (4 Punkte)

b. In einer öffentlichen Gemeindeverwaltung sollen die Daten der Bürger über eine Hashfunktion auf Basis des Geburtsdatums verwaltet werden. Das Geburtsdatum liegt als Zeichenkette der Länge 7 (char gebdat[7]) im Format TTMMJJ vor, d.h. die ersten beiden Zeichen stellen den Tag, die nächsten beiden den Monat und die letzten beiden das Jahr der Geburt dar. Das letzte Zeichen ist das Begrenzungszeichen ‘\0‘ für Zeichenketten. Beispiel: die Zeichenkette “151297“ steht für den 15.

Dezember 1997.
Geben Sie in C-Notation eine Hashfunktion an, die das Geburtsdatum in der angegebenen Form als Parameter besitzt und daraus einen Hashwert (Behälternummer) für 100 Behälter berechnet. Dabei sind die Behälter sind mit den Nummern 1 .. 100 durchnummeriert. (7 Punkte)

Anhang

Der Datentyp Liste von int hat folgende Signatur:

liste* Tail (liste *l) /* löschte erstes Element unter der Voraussetzung: Liste ist nicht leer! */

int Head (liste * l) /* liefert erstes Element, Liste bleibt unverändert, Voraussetzung: Liste ist nicht leer! */

liste* Append (liste *l, int e) /* fügt Element e vorne an */

int isempty (liste *l) /* Liste leer dann Ergebnis 1, sonst 0 , Liste bleibt unverändert */

liste* emptylist() /* Erzeugt eine leere Liste */