

Klausur Grundgebiete der Informatik 1

WS 07/08

(BPO07/DPO04)

Datum: 19.02.2008

Keine offizielle Lösung!

Name:
<i>(in Druckschrift)</i>	
Matr.Nr.:
Unterschrift:

Keinerlei Gewähr auf Richtigkeit!

Aufgabe	Max. Punkte	Korrektur		Einsicht	
		Punkte	Kürzel	Punkte	Kürzel
1	15				
2	7				
3	11				
4	6				
5	16				
6	6				
7	14				
8	25				
Σ	100				

Aufgabe 1 – Wissensfragen

Bitte beachten Sie, dass bei den folgenden Teilaufgaben - sofern in der Aufgabenstellung nicht anders verlangt - ein oder mehrere Antworten richtig sein können.

- (a) Wie würde der C-Compiler die folgenden Konstanten interpretieren? Kreuzen Sie für die folgenden C-Konstanten den impliziten C-Datentyp in der Tabelle an.

	int	double	char	unsigned int
'a'			<input checked="" type="checkbox"/>	
-328	<input checked="" type="checkbox"/>			
.11e2		<input checked="" type="checkbox"/>		
5u				<input checked="" type="checkbox"/>
'\0'			<input checked="" type="checkbox"/>	

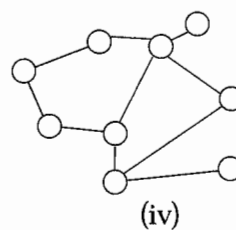
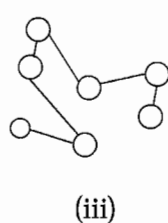
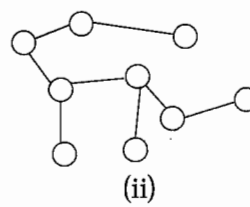
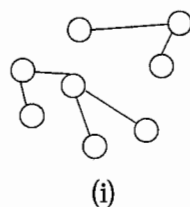
- (b) In welcher Reihenfolge wird ein binärer Suchbaum beim Postorder-Durchlauf durchlaufen?

- Wurzel, linker Teilbaum, rechter Teilbaum
 Linker Teilbaum, Wurzel, rechter Teilbaum
 Linker Teilbaum, rechter Teilbaum, Wurzel

- (c) Welche Laufzeitkomplexität besitzt die binäre Suche in einem sortierten Feld der Größe n ?

- $O(n^2)$
 $O(n \log_2 n)$
 $O(\log_2(n^2))$
 $O(\log_2 n)$

- (d) Gegeben sind die folgenden Graphen. Kreuzen Sie für jeden Graphen alle zutreffenden charakteristischen Eigenschaften an.



- (i) zusammenhängend
 zyklensfrei
 Baum
- (ii) zusammenhängend
 zyklensfrei
 Baum
- (iii) zusammenhängend
 zyklensfrei
 Baum
- (iv) zusammenhängend
 zyklensfrei
 Baum

Aufgabe 2 – Fehlersuche

Eine C-Funktion soll das Skalarprodukt zweier `integer`-Vektoren berechnen und einen `integer`-Wert als Ergebnis zurückliefern. Mathematisch ausgedrückt ergibt sich das Skalarprodukt zweier Vektoren vec_a und vec_b der Länge len als:

$$scalarproduct(vec_a, vec_b) = \sum_{i=0}^{len-1} vec_a[i] * vec_b[i]$$

Der folgende C-Quellcode zur Berechnung des Skalarprodukts enthält jedoch sowohl Syntax- als auch Semantikfehler. Gehen Sie davon aus, dass der hier verwendete C-Compiler keine impliziten Variableninitialisierungen vornimmt.

Finden Sie die Fehler und markieren Sie diese im Quelltext. Geben Sie eine korrigierte Version des Quellcodes an.

```

1  int* scalarproduct( int* vec_a, int *vec_b, int len ){
2
3     int i, res ;
4
5     for( i = 0; I < len; i++ )
6     {
7         res += vec_a * vec_b
8     }
9
10    return res;
11
12 }
```

```

int scalarproduct (int*vec_a, int vec_b, int len) {
    int i, res = 0;
    for (i=0; i < len ; i++)
    {
        res += *(vec_a+i) * *(vec_b+i);
    }
    return res;
}
```

Aufgabe 3 – Zeiger

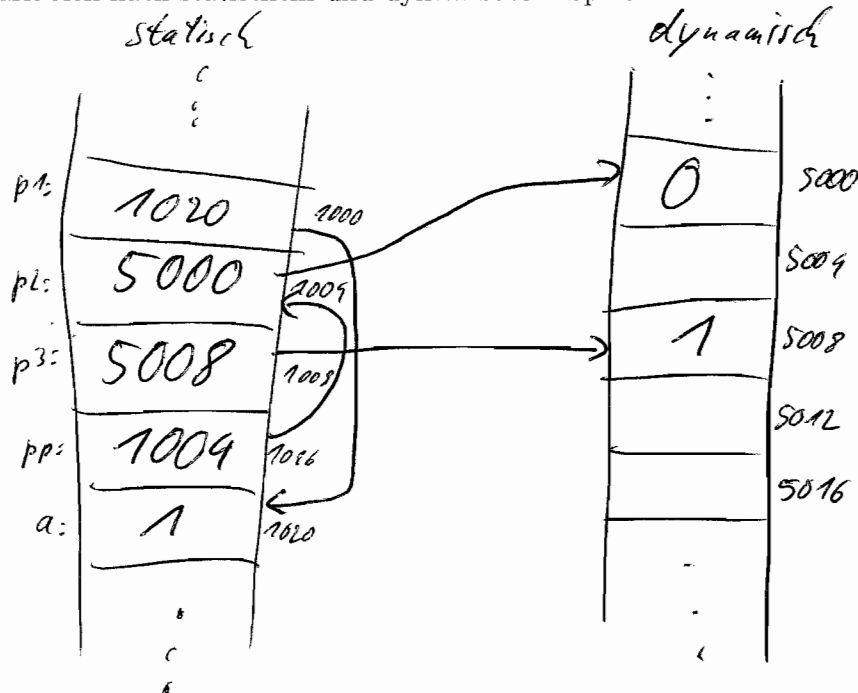
Gegeben ist der folgende C-Programmcode. Gehen Sie davon aus, dass die Bitbreiten der int-Variablen sowie der Zeigervariablen 32 Bit betragen.

```

1 #include <stdio.h>
2 #include <malloc.h>
3
4 int main(){
5     int *p1, *p2, *p3, **pp;
6     int a = 1;
7
8     p1 = &a;
9     p2 = (int*) malloc(sizeof(int));
10    p3 = (int*) malloc(sizeof(int));
11
12    *p2 = 0;
13    *p3 = a;
14    pp = &p2;
15
16    printf("%d, %d, %d, %d\n", *p1, *p2, *p3, **pp);
17    return 0;
18 }

```

- (a) Welche Ausgabe erzeugt das Programm? *1, 0, 1, 0*
- (b) Zeichnen Sie eine Skizze der im Speicher abgelegten Variablen mit deren Inhalten. Verdeutlichen Sie dabei die Zeigerstruktur durch Pfeile und unterscheiden Sie die Variablen nach statischem und dynamischem Speicherbereich.



Aufgabe 4 – Call by value / Call by reference

Gegeben ist der folgende C-Programmcode:

```
1 void f3( int* x, int* y ){
2     *x = 5;
3     y = 6;
4 }
5 void f2( int x, int y ){
6     x = 5;
7     y = 6;
8 }
9 void f1( int* x, int* y ){
10    *x = 5;
11    *y = 6;
12 }
13 int main(){
14     int a = 0, b = 0, c = 0, d = 0, e = 0, f = 1;
15
16     f1( &a, &b );
17     f2( c, d );
18     f3( &e, &f );
19
20     return a + b + c + d + e + f;
21 }
```

Geben Sie jeweils den Wert der Variablen a, b, c, d, e und f in Zeile 20 sowie den Rückgabewert der main-Funktion an.

$a=5, b=6, c=0, d=0, e=5, f=1$

~~Wert~~ Rückgabewert: 77

Aufgabe 5 – O-Notation

- (a) Zeigen Sie mit Hilfe der Definition der O-Notation die folgende Aussage. Geben Sie ein $c \in \mathbb{R}$ und ein $n_0 \in \mathbb{N}$ an, für die die Aussage gilt.

$$n\left(1 + \frac{1}{n^2}\right) \in O(n)$$

$$c = 2, n_0 = 1$$

$$\text{damit gilt } \forall n \geq n_0: n\left(1 + \frac{1}{n^2}\right) \leq c \cdot n$$

$$\Rightarrow O\left(n\left(1 + \frac{1}{n^2}\right)\right) = O(n)$$

- (b) Berechnen Sie mit Hilfe der Rechenregeln der O-Notation das Laufzeitverhalten der folgenden Funktion.

$$\sum_{i=1}^{n-1} 2(i+1) = \sum_{i=1}^n 2(i+1) - 2(n+1)$$

$$= \frac{2(n+1)(2(n+1)+1)}{2} = \frac{2(n+1)(2n+3)}{2} - 2(n+1)$$

$$= 2n^2 + 3n + 2n + 3 = \overset{(-2(n+1))}{\cancel{2n^2 + 3n + 2n + 3}} 2n^2 + 3n + 1$$

$$O(2n^2 + \overset{3n}{\cancel{3n}} + \overset{1}{\cancel{3}}) = \overset{3n}{\cancel{O(2n^2 + 3n + 3)}} O(n^2)$$

- (c) Die Algorithmen A, B und C besitzen folgende Laufzeitfunktionen T_A , T_B und T_C :

$$T_A(n) = n^2 + n - 2$$

$$T_B(n) = n^2 + n$$

$$T_C(n) = n + 14$$

Vergleichen Sie die drei Laufzeitfunktionen in Abhängigkeit von n . Zeigen Sie, für welche n welcher Algorithmus der schnellste ist. Für welches n sind die Algorithmen T_A und T_C gleich schnell?

$$T_A = T_C \Leftrightarrow n^2 + n - 2 = n + 14 \Leftrightarrow n^2 = 16 \Leftrightarrow n = 4$$

\Rightarrow für $n=4$ sind beide Algorithmen gleich schnell

- für $n < 4$ ist T_A schneller
- für $n > 4$ ist T_C schneller
- T_B ist für kein n der Schnellste

Aufgabe 6 – Stacks

Die Funktion

$$f(n) = n! = \prod_{i=1}^n i = 1 * 2 * 3 * \dots * n, n > 0$$

kann folgendermaßen als rekursive C-Funktion implementiert werden:

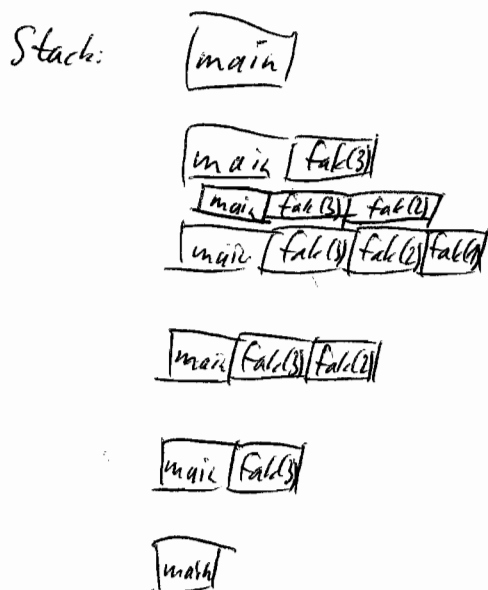
```
int fak(int n)
{
    if(n==1) return n;
    return n * fak(n-1);
}
```

(a) Zeichnen Sie den Call-Stack für die Ausführung des folgenden C-Programms:

```
int main()
{
    int n=3;
    int f;
    f = fak(n);
    return f;
}
```

Geben Sie hierzu zuerst die Aufrufreihenfolge der Funktionen als Liste an. Zeichnen Sie anschließend den Laufzeitstack für die C-Funktionsaufrufe nach jedem der einzelnen Aufrufe.

Liste: $main \rightarrow fak(3) \rightarrow fak(2) \rightarrow fak(1)$



- (b) Welche Laufzeitkomplexität besitzt die Funktion `fak` in Abhängigkeit von `n`? Geben Sie die Komplexität in der O-Notation an. Geben Sie eine kurze Begründung an.

$O(n)$

Die Funktion `fak(n)` wird `n`-mal aufgerufen, die Funktion selber hat die Komplexität $O(1)$

- (c) Geben Sie nun eine iterative Variante von `int fak(int n)` in C-Syntax an.

```
int fak(int n) {  
    int res = 1, i;  
    for (i = 0; i <= n; i++) {  
        res *= i;  
    }  
    return res;  
}
```

Aufgabe 7 – Selection Sort

Gegeben sei das Array A der Größe $N = 8$ mit folgenden Schlüsselwerten:

$A[] = \{ 2, 8, 8, 34, 0, 15, 9, 23 \}$

- (a) Sortieren Sie das Array A unter Verwendung des *Selection Sort* Algorithmus. Stellen Sie dabei jeden Zwischenschritt dar und markieren Sie das aktuelle Element, mit dem verglichen wird, sowie das minimale Element.

1) 2 8 8 34 0 15 9 23
 2) 0 8 8 34 2 15 9 23
 3) 0 2 8 34 8 15 9 23
 4) 0 2 8 34 8 15 9 23
 5) 0 2 8 8 34 15 9 23
 6) 0 2 8 8 9 15 34 23
 7) 0 2 8 8 9 15 34 23
 8) 0 2 8 8 9 15 23 34

- (b) Untersuchen Sie die Stabilität von Selection Sort. Begründen Sie Ihre Antwort anhand der Sortierung des Arrays A.

nicht stabil

in Zeile 3 ist die erste '8' an die Stelle 4 gerätet. Von dort geht sie vor Zeile 5 an die Stelle 3, wo sie bis zum Ende bleibt. Somit steht sie ~~hinter~~ hinter der anderen 8, die vorher an 2. Stelle stand.

- (c) Geben Sie eine möglichst kleine obere Schranke für die Laufzeit von Selection Sort an und begründen Sie Ihre Antwort.

äußere Schleife: $N-1$

innere Schleife: zwischen $N-1$ und 1 Vergleich, also im Mittel $\frac{N-1}{2}$

$$\Rightarrow \frac{(N-1)^2}{2}$$

Aufgabe 8 – Datenstruktur: Verkettete Listen

Gegeben seien die folgenden globalen C-Definitionen für die Datenstruktur `List` einer einfach verketteten Liste mit ihren Knotenelementen vom Typ `Node`:

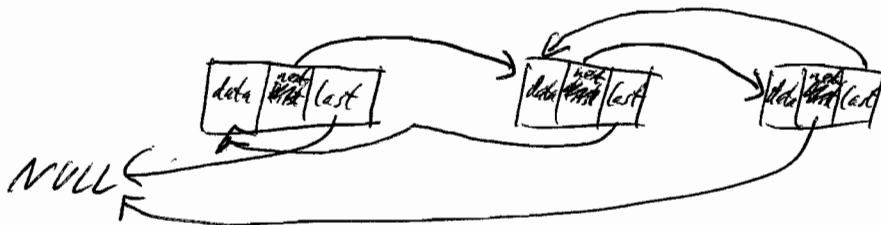
```
typedef struct node {
    int* data;
    struct node* next; } Node, *nodeptr;

typedef struct list {
    nodeptr first, last; } List, *listptr;
```

- (a) Erweitern Sie nun die Definition der einfach verketteten Liste zur doppelt verketteten Liste. Diese enthält zusätzlich zum Verweis auf den Nachfolger auch einen Verweis auf den Vorgänger. Aktualisieren Sie die oben gegebenen C-Definitionen, um die Datenstruktur `DList` zu definieren.

```
typedef struct node {
    int* data;
    struct node *next, *last; } Node, *nodeptr;
```

- (b) Zeichnen Sie eine beispielhafte Skizze für die Datenstruktur `DList` bestehend aus drei `Node`-Objekten. Gehen Sie davon aus, dass die Liste nicht zyklisch ist, d.h. Anfang und Ende mit `NULL` abgeschlossen werden.



(c) Implementieren Sie die C-Funktion

`void insert_before(nodeptr item, nodeptr cursor, listptr L)`,
 welche ein Element `item` an der Stelle *vor* `cursor` in die *einfach verkettete* Liste
 einfügt. Gehen Sie davon aus, dass die Liste mindestens ein Element enthält.
 Gehen Sie weiterhin davon aus, dass das Element `cursor` in der Liste enthalten
 ist.

```

void insert_before (nodeptr item, nodeptr cursor, listptr L) {
    nodeptr np;
    if (L->first == cursor) {
        item->next =
        L->first = item;
    }
    else {
        np = L->first;
        while (np->next != cursor) np = np->next;
        np->next = item;
    }
    item->next = cursor;
}
  
```

- (d) Implementieren Sie diese C-Funktion nun möglichst effizient für die *doppelt verkettete* Liste. Gehen Sie auch hier davon aus, dass die Liste mindestens ein Element enthält. Gehen Sie weiterhin davon aus, dass das Element `cursor` in der Liste enthalten ist.

```

void insert_before (nodeptr item, nodeptr cursor, (istptr L) {
    if (L->first == cursor) {
        L->first = item;
        item->next = cursor;
        cursor->last = item;
    }
    else {
        item->next = cursor;
        item->last = cursor->last;
        cursor->last->next = item;
        cursor->last = item;
    }
}

```

- (e) Untersuchen Sie das Laufzeitverhalten der Operation `insert_before` für beide Implementierungen aus den Unterpunkten (c) und (d).

einfach verkettete: while-Schleife hat im worstcase n Durchläufe
 $\Rightarrow O(n)$

doppelt verkettete: keinerlei Schleifen oder Rekursionen, alles nur Einzelschritte

$\Rightarrow O(1)$