

11 Übung zu Informatik zum 28.1.2010 Blatt 11

11.1

```
program Klammerung;
type Keller=^Klammer; {Keller-Datentyp: einfach verkettete Liste}
Klammer=Record Zeichen:char; next:Keller end;

function Pruf(s:String):String; {prüft auf korrekte Klammerung und
    gibt entsprechenden Text zurück: 'Ist (nicht) korrekt '}
var Klammerterm, neu: Keller; i: integer; korrekt:Boolean=true;
begin
    new(Klammerterm); {verkettete Liste initialisieren}
    Klammerterm^.Zeichen:=s[1]; {da Programm bei leerem String beendet
        ist s mindestens 1 Zeichen lang. Dieses wird das erste Element
        der Liste}
    Klammerterm^.next:=nil;
    For i:=2 to length(s) do {Alle anderen Klammern durchgehen}
        begin
            if (ord(s[i])>=65) and (ord(s[i])<=90) then {Bei Großbuchstaben (
                Klammer auf), Buchstaben an den Anfang der verketteten Liste
                setzen}
                begin
                    new(neu);
                    neu^.Zeichen:=s[i];
                    neu^.next:=Klammerterm; {Neue Klammer auf->Eintrag am Anfang der
                        Liste, Rest wird weitergeschoben}
                    Klammerterm:=neu;
                end
            else if (ord(s[i])>=97) and (ord(s[i])<=122) then {Bei
                Kleinbuchstaben (Klammer zu) prüfen, ob diese Klammer auch
                zuletzt geöffnet wurde}
                begin
                    if Klammerterm=nil then korrekt:=False else {Wenn keine geöffnete
                        Klammer vorhanden, kann auch keine geschlossen werden -> nicht
                        korrekter Term}
                    begin
                        if ord(Klammerterm^.Zeichen)<>(ord(s[i])-32) then korrekt:=False
                        else {In ASCII-Tabelle sind Klein- und Großbuchstaben 32
                            entfernt. Hier wird geprüft,
                            ob der zu dem aktuellen Kleinbuchstabe passende Großbuchstabe
                            auch der nächsten zu schließenden Klammer entspricht, also am
                            Anfang der verketteten
                            Liste steht}
                            begin
                                Klammerterm:=Klammerterm^.next {Wenn passende schließende
                                    Klammer eingegeben wird, wird die öffnende aus dem Anfang der
                                    Liste entfernt}
                            end
                        end
                    end
                end
            end;
            if Klammerterm<math>\Diamond</math>nil then korrekt:=false; {Sind noch offene Klammern
                vorhanden, ist der Term nicht korrekt}
```

```

if korrekt=true then pruf:= 'Dies ist ein korrekt geklammerter Term!'
    else Pruf:= 'Dieser Term ist nicht korrekt geklammert!' {Rückgabe
        des Ergebnisses}
end;

procedure eingabe;
var Ausdruck: String;
begin
    readln(Ausdruck); {Eingabe des Klammer-Terms}
    if Ausdruck<>' ' then {Abbruchbedingung, hier auch Vorkehrung gegen
        leere Eingaben}
        begin
            writeln(Pruf(Ausdruck)); {Prüfung und Ausgabe}
            eingabe {erneute Eingabe eines Terms möglich}
        end
    end;

begin
    writeln( 'Bitte einen zu prüfenden Klammerterm aus Grossbuchstaben (
        oeffnend) und Kleinbuchstaben (schliessend) eingeben! Leere
        Eingabe zum Beenden! ');
    {Begrüßung, Hinweis und Anweisung}
    eingabe;
end.

```

11.2

1) $\frac{n^4}{3} \in \theta[n^3] \Rightarrow c_1 n^3 \leq \frac{n^4}{3} \leq c_2 n^3 \Rightarrow \frac{n}{3} \leq c_2$

Offensichtlich falsch, da jede Konstante c_2 irgendwann von x überschritten wird

2) $n \cdot \log[n] \in O[n] \Rightarrow n \cdot \log[n] \leq n \cdot c_1 \Rightarrow \log[n] \leq c_1$

Auch falsch, da der Logarithmus gegen unendlich geht, wenn auch langsam.

3) $[\log[n]]! \in O[n^{\log[\log[n]]}] \Rightarrow [\log[n]]! \leq n^{\log[\log[n]]} * c_1$

Kann nicht stimmen, da Fakultät sehr schnell wächst und der Exponent von n durch $\log(\log(n))$ äußerst langsam wächst.

4) $f \in \log[n^{o[1]}] \Rightarrow f \in o[\log[n]]$

Es gilt also zu beweisen, dass eine Funktion $\log(n^k)$ mit $k > 1$ für alle k und n größer ist, als eine Funktion $c \log(n)$ mit $c > 1$.

Durch das Logarithmus-Gesetz können wir $c \log(n)$ auch also $\log(n^c)$ schreiben. Somit wäre die Aussage bewiesen.

11.3

Die Prozedur ermitteln führt folgende Befehle nach Start aus:

1. *var currentmax : Integer;*
2. *var t : Integer;*
3. *currentmax := -1000;*
 1. *Schleife t := 0 bis N* {setze t=0..N, insgesamt N+1 Durchläufe}
 2. *currentmax := arr[t];*
 3. *append(L.arr[t]);*
{Prozeduraufruf zeigen}
 4. *var h : Liste;*
 5. *if L <> nil then;* (worst Case: immer True)
 6. *h := L;*
 1. *writeln(h.info);* {Schleife, worst case: N mal}
 2. *h := h.next;*
 3. *Pruefe h.next = nil;*

Damit ergibt sich eine Laufzeit von ungefähr $3 + (N + 1)(6 + 3N) = 3 + 6N + 3N + 3N^2 = 3N^2 + 9N + 3$

Dies entspricht $O(N^2)$

11.4

Der 2. und der 3. können nicht gemeinsam gewinnen!

Der 3. setzt voraus, dass alle 3 beim ersten Schuss treffen.

Der 2. sagt aus, dass wenn der 1. beim ersten Schuss trifft, dieser seine Wette gewinnt, also der zweite und/oder der dritte nicht trifft.

Trifft der 1. beim ersten Schuss, so gilt nach 2., dass der 2. oder/und der 3. nicht trifft. Also hat der 3. seine Wette verloren.

Trifft der 1. beim ersten Schuss nicht, so macht 2. keine Aussage. Trotzdem verliert der 3. seine Wette dann, da der 1. nicht getroffen hat.

Treffen alle drei beim 1. Schuss, so hat 1. die Wette verloren, da der 2. und der 3. getroffen haben, wodurch auch der 2. seine Wette verliert, da der 1. getroffen und verloren hat.