Aufgabe zur Selbstevaluierung

Die folgende Aufgabe sollten Sie ohne die Verwendung von Hilfsmitteln in höchstens 30 Minuten lösen können.

Schließen Sie alle Fenster außer dem Editor und dem Terminal, insbesonders dürfen Sie keinen Webbrowser verwenden.

Schalten Sie elektronische Geräte (z.B.: Handy) aus und bewahren Sie sie möglichst schwer zugänglich auf.

Starten Sie mit einer Datei, die nur das übliche Gerüst enthält:

```
#include<iostream>
using namespace std;

int main() {
  return 0;
}
```

Haben Sie Ihr Programm fertiggestellt, melden Sie dies der Übungsleitung. Ihr Programm wird kurz überprüft und falls alles in Ordnung ist, können Sie den Saal für eine kurze Pause verlassen, oder an einer der Zusatzaufgaben weiter arbeiten.

Sie sollten jedenfalls nicht anderen Studierenden helfen.

Aufgabenstellung

Erstellen Sie ein Programm, das das folgende Muster erzeugt (die Zeilen- und Spaltennummern dienen nur der einfacheren Orientierung, sie müssen **nicht** ausgegeben werden)**:**

	0	1	2	3	4	5	6	7	8
0	*			*			*		
1		*			*			*	
2			*			*			*
3	*			*			*		
4		*			*			*	
5			*			*			*

Zeilen#	Spalten mit *		
0	0,3,6		
1	1,4,7		
2	2,5,8		
3	0,3,6		
4	1,4,7		
5	2,5,8		

Zusatzaufgaben:
/home/Xchange/ue3/zusatz1.pdf

Verwenden Sie nur jeweils ein Statement `cout<<'*';`, `cout <<' ';` und `cout<<'\n';` und sonst keine weiteren Ausgabestatements. (Verwendung von Funktionen, um die Ausgabestatements zu “maskieren” ist nicht erlaubt.)

Anmerkung: Es empfiehlt sich die Anwendung des Moduloperators

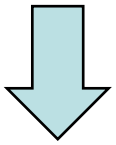
Erweiterungen

1) Programmieren Sie eine Funktion, die als Parameter die Anzahl der Zeilen, die Anzahl der Spalten und den Modul (rechter Operand des Modulo-Operators) erhält und das entsprechende Muster ausgibt. Schreiben Sie ein Programm, um die Funktion einfach testen zu können.
2) Erweitern Sie Ihre Funktion aus Beispiel 1 mit einem zusätzlichen booleschen Parameter **`invertieren`**. Falls dieser Parameter **`true`** ist, so soll das inverse Muster erzeugt werden (Sterne und Leerzeichen vertauscht).
3) Erweitern Sie Ihre Funktion aus Beispiel 2 mit zwei zusätzlichen char Parametern, mit denen die auszugebenden Zeichen (statt `'*'` und `' '`) festgelegt werden können.
4) Erweitern Sie die Funktion aus Beispiel 1 mit einem zusätzlichen string Parameter. Statt der Sterne sollen der Reihe nach die Zeichen aus dem String ausgegeben werden. Falls der String zu kurz ist, soll er als zyklisch betrachtet werden (also wieder beim Zeichen 0 begonnen werden). Zugriff auf ein Zeichen eines Strings **`x`** ist mit **`x.at(pos)`** möglich. Die Anzahl der Zeichen in einem String **`x`** liefern **`x.length()`** und **`x.size()`** (die beiden Methoden sind Synonyme). Das erste Zeichen hat die Position 0.

Rekursion vs. Iteration (1)

Jede Schleife (Iteration) kann als Rekursion formuliert werden.

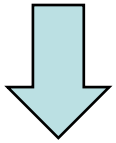
```
//Ergebnis 0 berechnen
while (bedingung) {//bedingung abhängig von Variablen b1,...,bi
  //Ergebnisk aus Ergebnisk-1 berechnen
  //Variablen b1,...,bi der Bedingung manipulieren
}
//Ergebnisn nach n Schleifendurchgängen
```



```
void f(b1,...bi, ergebnisk-1) {
  if (!bedingung) //Rekursionsende
    return; //Ergebnisn nach n Schleifendurchgängen
  //Ergebnisk aus Ergebnisk-1 berechnen
  //Variablen b1,...bi der Bedingung manipulieren
  f(b1',...,bi',ergebnisk);
}
```

Beispiel Iteration in Rekursion umwandeln

```
int sum {0};
int n;
cin >> n;
while (n)
  sum += n--;
cout << sum << '\n';
```

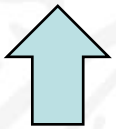


```
void f(int n, int sum) {
  if (!n) { //Rekursionsende
    cout << sum << '\n';
    return;
  }
  f(n - 1, sum + n); //nicht --n!
}

int n;
cin >> n;
f(n,0);
```

besser

```
int f(int n) {
  return n ? f(n - 1) + n : 0;
}
```



```
int f(int n) {
  if (!n) {
    return 0;
  }
  return f(n - 1) + n;
}
```

Verzicht auf tail-recursion

```
int f(int n, int sum) {
  if (!n) {
    return sum;
  }
  return f(n - 1, sum + n);
}

int n;
cin >> n;
cout << f(n,0) << '\n';
```

universität wien

Rekursion vs. Iteration (2)

Jede Rekursion kann in eine Iteration umgewandelt werden.

Im Allgemeinen nicht immer einfach. Es können komplexe, verschachtelte Schleifenstrukturen notwendig sein.

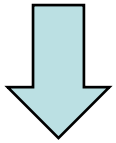
Im schlimmsten Fall implementiert man einen eigenen Stack und verwendet ihn analog zum Aufrufmechanismus der rekursiven Version.

Für einfache Rekursion (Kopf- bzw. End-Rekursion; head- bzw. tail-recursion) ist die Umwandlung allerdings wieder schematisch möglich (optimierende Compiler führen sie teilweise automatisch durch).

Head Recursion

```
int f(int n, int sum) {
  if (n) {
    sum = f(n - 1, sum); //rekursiver Aufruf (nicht ganz) am Beginn
  }
  return n + sum;
}

int n;
cin >> n;
cout << f(n,0) << '\n';
```

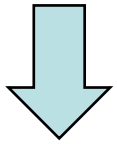


```
int sum {0};
int n;
cin >> n;
int i {0};
while (i <= n) //Schleife wird umgekehrt durchlaufen
  sum += i++;
cout << sum << '\n';
```

Tail Recursion

```
int f(int n, int sum) {
  if (!n) { //Rekursionsende
    return sum;
  }
  return f(n - 1, sum + n);  //rekursiver Aufruf am Ende
}


int n;
cin >> n;
cout << f(n,0) << '\n';
```



```
int sum {0};
int n;
cin >> n;
while (n)
  sum += n--;
cout << sum << '\n';
```

Beliebt, da der Compiler tail-call-optimization (TCO, im rekursiven Fall auch als tail-recursion-optimization bezeichnt) durchführen kann. Die Funktion kann aufgerufen werden, ohne einen neuen Stack-Frame zu bekommen. Der Stack-Frame der aufrufenden Funktion kann wiederverwendet werden.
Einige Programmiersprachen (vor allem funktionale) bieten nur tail-recursion und keine Schleifen an.

Rekursion vs. Iteration (3)

In der Regel ist die Iteration effizienter

Laufzeit: Stackverwaltung und Funktionsaufrufe sind verglichen mit einer einfachen Schleifensteuerung verhältnismäßig teuer

Speicherplatz: Es wird zusätzlicher Platz am Stack für Funktionsaufrufe und Parameter benötigt.

Gut optimierende Compiler können aber auch Rekursionen effizient umsetzen.

Warum sollten wir also Rekursion überhaupt verwenden?

Einfacher lesbare und verständliche Programme

Einfachere Korrektheitsbeweise (in funktionalen Programmiersprachen)

Beispiel

Programmieren Sie eine rekursive Funktion mit einem Parameter n, die sich n mal rekursiv aufruft. Verwenden Sie keine Schleife.

Also das Äquivalent zu

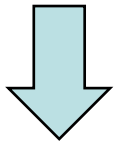
```
int n;
cin >> n;
while (n--);
```

Lösung

Programmieren Sie eine rekursive Funktion mit einem Parameter n, die sich n mal rekursiv aufruft. Verwenden Sie keine Schleife.

Also das Äquivalent zu

```
int n;
cin >> n;
while (n--);
```

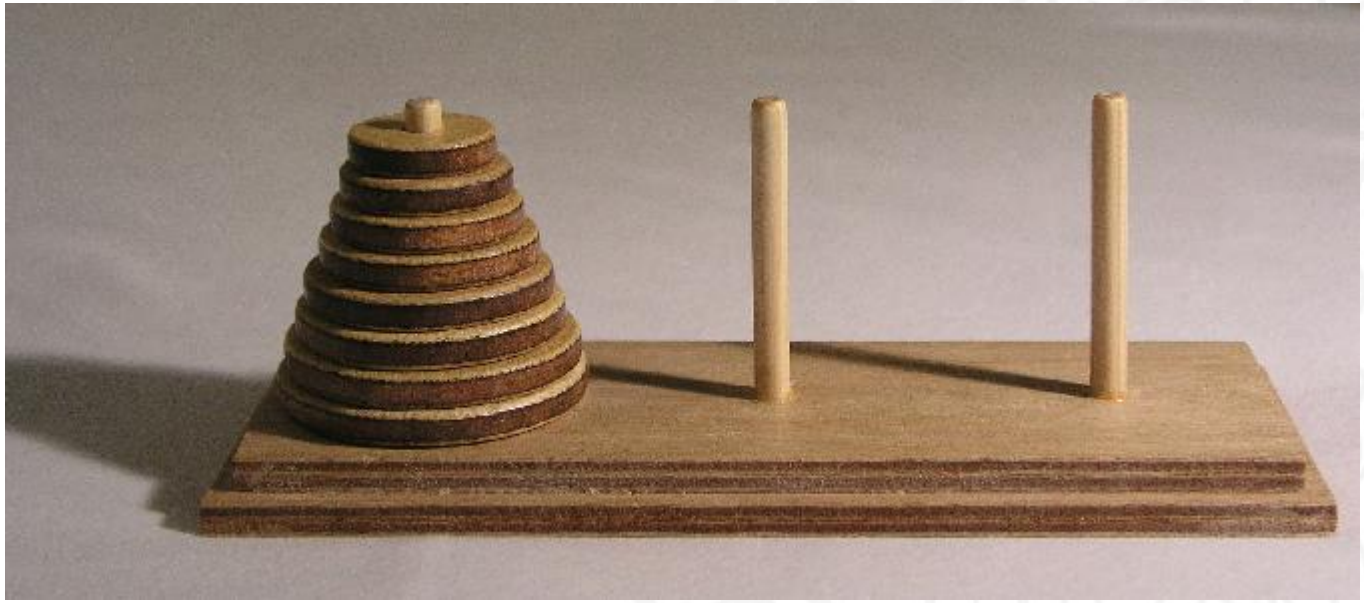


```
void f(int n) {
  if (!n) return;
  f(n-1);
}
```

Geben Sie jeweils vor und nach dem rekursiven Aufruf den Wert des Parameters n aus und rufen Sie die Funktion für den Wert 5 auf.
Überlegen Sie, wie die Ausgabe genau zustande kommt.

Türme von Hanoi

Ein Beispiel, dass es für manche Probleme einfacher ist, eine rekursive Lösung zu finden, als eine iterative.



Quelle: https://en.wikipedia.org/wiki/Tower_of_Hanoi

Bewege alle Scheiben ganz nach rechts. Es darf immer nur eine Scheibe bewegt werden und niemals eine größere auf einer kleineren Scheibe liegen.

universität wien

Rekursiver Ansatz

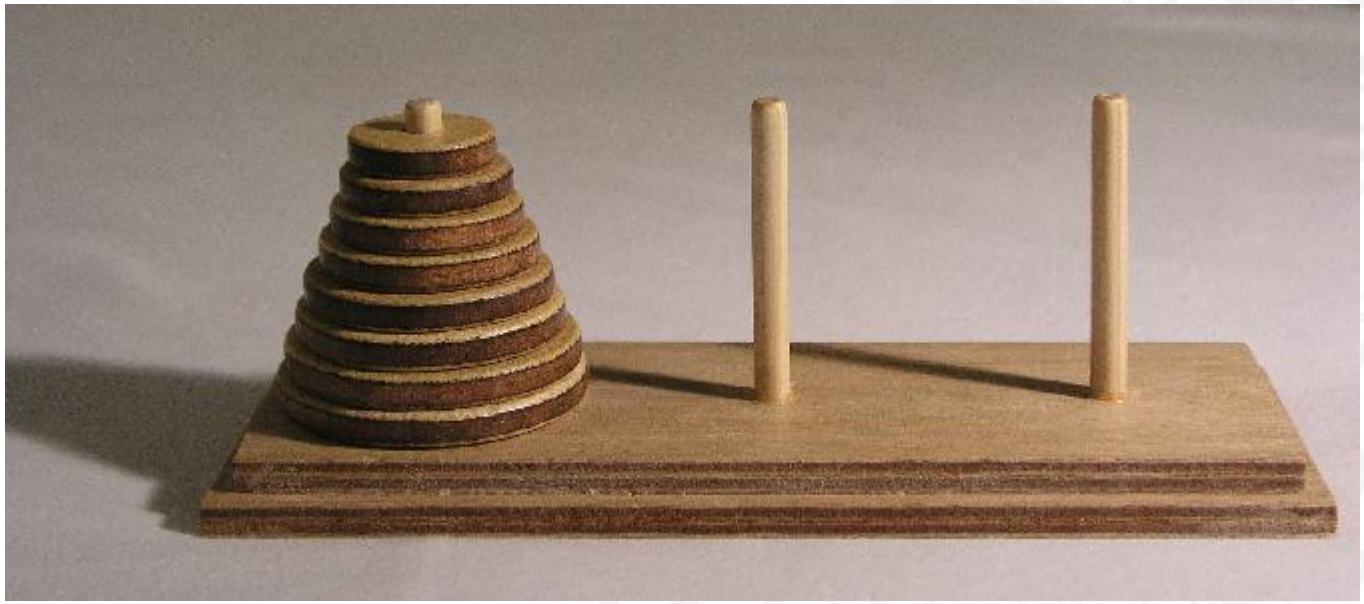
Teile das Problem so, dass ein Teil wieder wie das ursprüngliche Problem (nur in irgend einer Art "kleiner") aussieht.

Nimm an, das kleinere Problem wäre schon gelöst und konstruiere aus dieser hypothetischen Lösung eine Gesamtlösung.

Gib eine Lösung für ein genügend kleines Problem an (Rekursionsabbruch)

Vgl. Mathematik: Vollständige Induktion

Rekursiver Algorithmus



Wir nehmen an, wir wüssten schon, wie wir n-1 Scheiben optimal verschieben können. Da es egal ist, auf welchen Stab wir verschieben, können wir dann n-1 Scheiben in die Mitte schieben, dann die größte Platte ganz nach rechts und schließlich n-1 Scheiben von der Mitte nach rechts. Fertig.

universität wien

Rekursionsabbruch

Darf man nicht vergessen, sonst bricht das ganze Kartenhaus in sich zusammen (analog zu Startschritt bei vollständiger Induktion).

Im konkreten Fall kann man n=1 (oder als echter Informatiker auch n=0) wählen. Wenn man eine Scheibe hat, so legt man diese einfach nach rechts (wenn man keine Scheibe hat, ist man sofort fertig).

Beobachtung

Aus dieser Rekursion kann man nun sofort die notwendige Anzahl von Zügen berechnen.

Es gilt: $Z_n = Z_{n-1} + 1 + Z_{n-1} = 2Z_{n-1} + 1$

größte Scheiben nach rechts

n-1 Scheiben in die Mitte

n-1 Scheiben nach rechts

Die Mathematik liefert uns Techniken, solche Rekurrenzgleichungen aufzulösen. Im konkreten Fall ist dies besonders einfach:

$S_n = Z_n + 1 = 2Z_{n-1} + 2 = 2(Z_{n-1} + 1) = 2S_{n-1}$ fortgesetzt einsetzen

$S_n = 2^n$

$Z_n = 2^n - 1$

Die Anzahl der nötigen Züge steigt exponentiell.

Rekursives Programm

Lässt sich nun direkt schreiben:

```
Funktion (string stab_start, string stab_hilfe, string stab_ziel, int n)
  //Rekursionsabbruch
  falls n 1 ist, gib aus "Zug von " stab_start " nach " stab_ziel und stoppe
  //n-1 in die Mitte
  Funktion(stab_start, stab_ziel, stab_hilfe, n-1)
  //1 nach rechts
  gib aus "Zug von " stab_start " nach " stab_ziel
  //n-1 von Mitte nach rechts
  Funktion(stab_hilfe, stab_start, stab_ziel, n-1)
```

Schreiben Sie die Funktion und ein Programm um sie zu testen.

Zusatzaufgaben: /home/Xchange/ue3/zusatz2.pdf

universität wien

Anwendung von Rekursion

Wir konnten ein Programm schreiben, ohne dass wir noch wirklich wissen, wie wir das Problem für ein bestimmtes n wirklich praktisch lösen.

Einfach durch eine passende Zerlegung des Problems.

Den aufwendigen Teil (bis zum einfachsten Fall mit nur einer Scheibe zu rekursieren und dann die Gesamtlösung Schritt für Schritt daraus zu konstruieren) übernimmt der Rechner.

Das Auffinden einer iterativen Lösung ist wesentlich schwieriger.

Zusatzaufgaben:
/home/Xchange/ue3/zusatz2.pdf

universität wien

Erweiterungen zu Türmen von Hanoi (1)

Instrumentieren Sie Ihr Programm, um folgende Fragen beantworten zu können:

Wie viele Züge werden für die Lösung benötigt?

Wie oft wird die Funktion aufgerufen?

Wie tief ist die Rekursion maximal (wie viele rekursive Instanzen der Funktionen sind maximal gleichzeitig am Stack repräsentiert)?

(Sie dürfen globale Variablen zum Akkumulieren der gesuchten Größen verwenden, da wir die Konzepte für eine einfache Lösung ohne globale Variablen noch nicht kennengelernt haben. Eine etwas komplexere Lösung ohne Verwendung globaler Variablen ist aber auch mit den bisher bekannten Mitteln möglich.)

Erweiterungen zu Türmen von Hanoi (2)

Im Pseudocode (und wahrscheinlich auch in Ihrer Implementierung) wurde als Rekursionsabbruch der Fall n==1 gewählt. Man könnte aber auch n==0 wählen.

- Wie ändert sich das Programm in diesem Fall?
- Ändert sich die Anzahl der Aufrufe der rekursiven Funktion?
- Ändert sich die maximale Rekursionstiefe?
- Was sind die Vor- und Nachteile der jeweiligen Implementierungen?
- Für welche Version sollte man sich entscheiden?

universität wien

Erweiterungen zu Türmen von Hanoi (3)

Finden Sie eine iterative Lösung für das Problem

Wie können Sie sicher sein, dass Ihre Lösung optimal ist (nur die minimal notwendige Anzahl von Zügen braucht)?

universität wien

Erweiterungen zu Türmen von Hanoi (4)

Versuchen Sie eine "grafische" Ausgabe analog zu /home/Xchange/ue3/hanoi_print_pr1 zu realisieren

Das ist, wenn Sie ausschließlich die bis jetzt behandelten Konstrukte verwenden, sehr umständlich und trickreich.

Versuchen Sie es trotzdem. Auf welche Hindernisse stoßen Sie?

Falls Sie schon Arrays oder die Klasse vector kennen, sollte die Implementierung nicht allzu schwierig sein.

universität wien

Erweiterungen zu Türmen von Hanoi (5) schwierig

Stellen Sie sich die Stäbe zyklisch angeordnet vor und die Scheiben dürfen nur um jeweils einen Stab im Uhrzeigersinn bewegt werden.

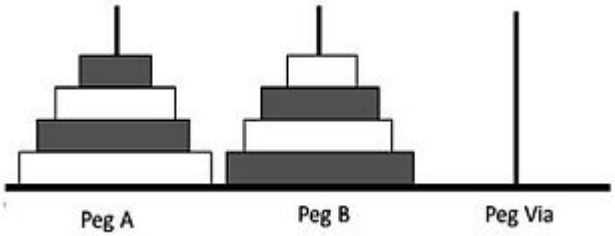
Wie sieht die Rekursion aus? (es ist hilfreich, zwei sich gegenseitig aufrufende Rekursionen zu betrachten).

Schreiben Sie ein Programm, das die Züge ausgibt

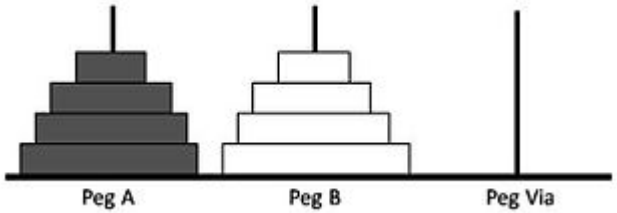
Erweiterungen zu Türmen von Hanoi (6) schwierig

Versuchen Sie ein Programm für die zweifärbige Version zu erstellen:

Startkonfiguration



Zielkonfiguration



Beachten Sie, dass die untersten Scheiben ausgetauscht werden (verzichtet man darauf, so erhält man eine etwas einfachere Version).

universität wien

Erweiterungen zu Türmen von Hanoi (7) schwierig

Wie könnte die Lösung aussehen, wenn 4 Stäbe vorhanden sind

- Man könnte offenbar einen Stab ignorieren und wie gehabt lösen.
- Aber kann die Anzahl der Züge vermindert werden, wenn der zusätzliche Stab genutzt wird?
- Schreiben Sie ein Programm für diese Variante
- Was ist die minimale Anzahl an notwendigen Zügen? (sehr bis extrem schwierig)