Übungen Ing.-Inf. – Rekursion

Der C-Coding Styleguide ist einzuhalten!

Folgende Einstellungen sind für Debug und Release (All Configurations) vorzunehmen:

Einstellung	**Wert**
Solution Platform	x86
Properties->Conf. Properties->C/C++->General->Warning Level	Level4 (/W4)
Properties->Conf. Properties->C/C++->General->Treat Warnings As Errors	Yes (/WX)
Properties->Conf. Properties->C/C++->General->SDL checks	Yes (/sdl)
Properties->Conf. Properties->C/C++->Code Generation->Basic Runtime Checks	Default
Properties->Conf. Properties->C/C++->Code Generation->Security Checks	Enable Security Checks (/GS)

Solution muss in Debug und Release fehlerfrei kompilierbar sein.

Packen Sie das gesamte Verzeichnis der Solution in eine Zip-Datei und laden Sie diese in Moodle pünktlich hoch.

Oft können Problemstellungen mittels rekursiver Programmierung effizient gelöst werden. Dabei sind drei Verfahren zur rekursiven Lösung je nach Problemstellung zu unterscheiden. Welches Verfahren zielführend ist, hängt von der Problemstellung ab.

1. Die Problemstellung kann jeweils um 1 reduziert werden.
2. Die Problemstellung kann in zwei (gleichgroße) Teile aufgeteilt werden
3. Sonstige Verfahren (Teilweise Überschneidungen zu Verfahren 1 und 2)

Falls das Abbruchkriterium nicht greift, führt dies zu einem Stackoverflow und einer Exception!

Aufgabe 1: (Verfahren 1)

Implementieren Sie eine rekursive Funktion in C, welche y wie folgt berechnet:

$$y = a^b \qquad a, b \in N_0$$

Aufgabe 2: (Verfahren 1)

Implementieren Sie eine rekursive Variante von Bubblesort. In dieser Variante ist nur eine for-Schleife enthalten, die das größte Elemente nach rechts sortiert. Wann und wie muss sich DoBubbleSortRecursiv rekursiv aufrufen? Wie lautet das Abbruchkriterium?

```
void DoBubbleSortRecursiv(int* paiA, unsigned int uiSize)
{
   unsigned uiX;
   int iDummy;




   for (uiX = 0U; uiX < (uiSize - 1U); uiX++)
   {
      if (paiA[uiX] > paiA[uiX + 1])
         {
            iDummy = paiA[uiX];
            paiA[uiX] = paiA[uiX + 1];
            paiA[uiX + 1] = iDummy;
         }
      }
```

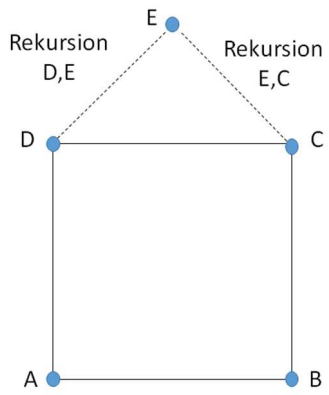
```
    }


}
```

Aufgabe 3: (Verfahren 2)

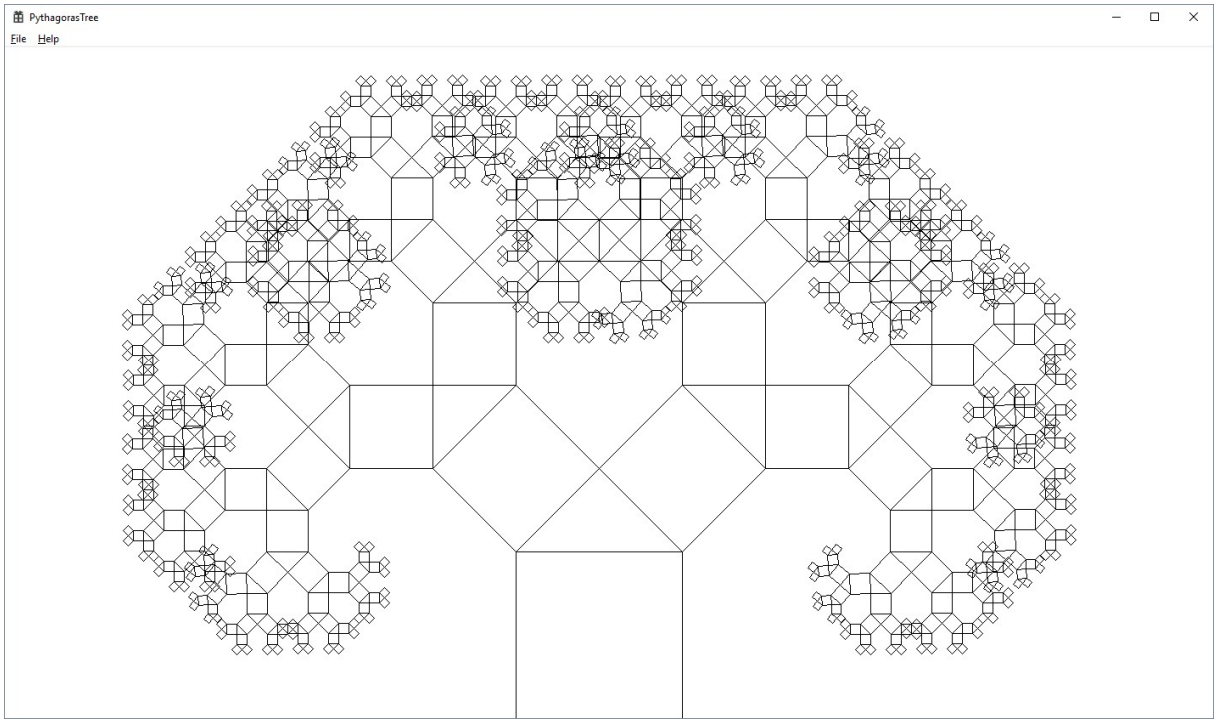
Es soll das Programm PythagorasTree fertig implementiert werden. Sie finden in Moodle eine Solution RecursivProgramming mit einem entsprechenden WINAPI-Projekt darin. Sie müssen nur noch die Funktion

```
void DrawPythagorasTree(HDC hdc, int icount, POINT Pa, POINT Pb)
```

ergänzen. In Zeile 167 befindet sich der erste Aufruf von DrawPythagoras.



Die Funktion erhält einen Handle auf den Device Kontext (zum Zeichnen), die Anzahl rekursiv durchzuführender Rekursionsschritte sowie die Punkte A und B. Aus diesen werden die Punkt D und C berechnet und ein Quadrat gezeichnet. Aus D und C wird E berechnet und zweimal wird mit E dann DrawPythagorasTree rekursiv aufgerufen.



Aufgabe 4: (Verfahren 2)

Es soll das Programm DrawLine fertig implementiert werden. Sie finden in Moodle eine Solution RecursivProgramming mit einem entsprechenden WINAPI-Projekt darin. Sie müssen nur noch die Funktion

```
void DrawLine(HDC hdc, int iX1, int iY1, int iX2, int iY2);
```

ergänzen. In Zeile 153 befindet sich der erste Aufruf von DrawLine.
Liegen der Start- und der Endpunkt einer Linie „nah“ nebeneinander, so müssen nur zwei Punkte gesetzt werden. Überlegen Sie sich eine Bedingung (Hilfe bietet der Namensgeber der Aufgabe 3). Ansonsten ist die Linie zu halbieren und die beiden gleich langen neuen Linien sind mit der Funktion DrawLine zu rekursiv zu zeichnen.
Eine Halbierung einer Koordinate erfolgt durch X3 = (X1 + X2) / 2;

Aufgabe 5: (Verfahren 3)

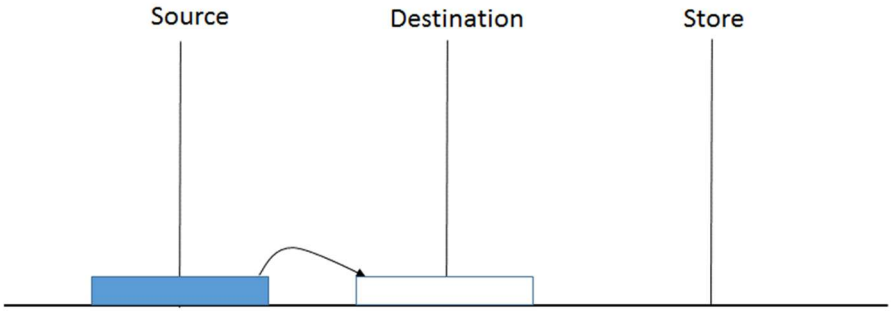
Die (neue) Fibonacci-Folge ist eine unendliche Folge von Zahlen f_n und ist wie folgt definiert.

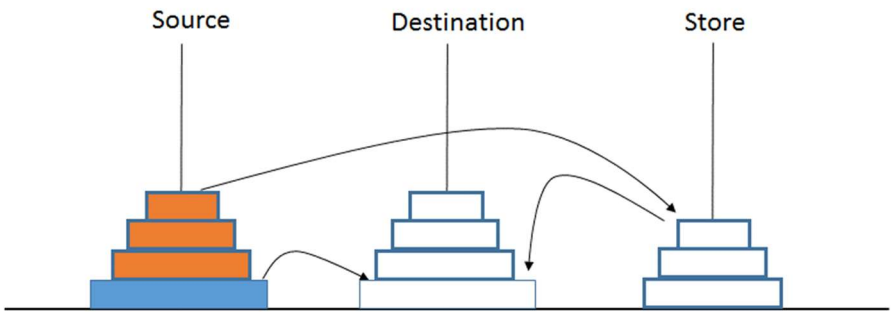
$$f_n = \begin{cases} 0 & ;n = 0 \\ 1 & ;n = 1 \\ f_{n-2} + f_{n-1} & ;n > 1 \end{cases} \qquad n \epsilon N_0$$

Implementieren Sie eine rekursive Funktion in C, welche die n-te Fibonacci-Zahl berechnet.

Aufgabe 6: (Verfahren 3)

Es soll das Towers of Hanoi Problem mit einer möglichst einfachen Programmlösung fertig implementiert werden. Es soll der Turm, der aus n Ringen besteht, von Source nach Destination unter Nutzung eines Zwischenspeichers (Store) verschoben werden. Es darf immer nur ein Ring von einer Säule auf eine andere bewegt werden. Es darf immer nur ein kleinerer Ring auf einem größeren Ring liegen. Alle Ringe haben unterschiedliche Größen.
Liegt nur ein Ring auf Source, so ist das Problem trivial lösbar. Liegen mehr als ein Ring auf Source, so kann das Problem rekursiv gelöst werden.





```
#include <stdio.h>
#include <stdlib.h>

void TowersOfHanoi(int,char,char,char);

int main(int argc, char *argv[])
{
  TowersOfHanoi(3,'A','B','C');

  return 0;
}


void TowersOfHanoi(int iRings, char cSource, char cDestination, char cStore)
{

if (iRings == 1)
   {
   printf("Move from %c to %c\n",cSource, cDestination);
   }
else
   {









   }
}
```