

3.2

```

////////////////////////////////////
/*****
/* Klasse MinFindStack
/*
/* Unterstützung der Funktionen pop(),push() und minfind() in konstanter Zeit
/*
/* Idee: Für pop() und push() reicht es, die aktuelle Position des oben auf dem Stack
/* liegenden Elementes zu kennen, um in konstanter Zeit arbeiten zu können.
/* Beim Ausführen der Funktion push() muß dafür gesorgt werden, daß auch minfind() in
/* konstanter Zeit laufen kann. Das erreicht man, indem man jedesmal beim Einfügen eines
/* neuen Elementes dieses mit dem letzten vorhandenen vergleicht und das jeweils
/* kleinere auf einen zweiten Stack "MinList" legt. Somit ist dafür gesorgt, daß das
/* jeweils kleinste Element in konstanter Zeit abrufbar ist.
/*
/* Alle Operationen werden in konstanter Zeit unterstützt, was nicht weiter gezeigt
/* werden muß, da es weder Schleifen noch rekursive Aufrufe gibt. Alle Funktionen
/* bewegen sich im Stack-Array relativ zur Position CurrentSize, weshalb die Größe
/* des Stacks keine Auswirkung auf die Laufzeit der Funktionen hat.
/*****
////////////////////////////////////

#define ERROR 0xFFFF;

class MinFindStack
{
    private:
        int CurrentSize; // aktuelle Größe / Position
        int MaxSize; // maximale Größe des Arrays
        int *Elements; // Zeiger auf Elemente - Stack
        int *MinList; // Zeiger auf Minimum - Stack

    public:

        MinFindStack(int Size) // initialisieren mit Size als
        { // maximaler Stack - Größe
            CurrentSize = 0; // Größe zu Beginn 0
            MaxSize = Size; // maximale Größe aus dem Konstruktor
            Elements = new int[MaxSize]; // Stack-Array erzeugen
            MinList = new int[MaxSize]; // Minimum-Array erzeugen
        }

        int pop(void)
        {
            if (CurrentSize>0) // wenn Elemente vorhanden
            {
                --CurrentSize; // dekrementiere Stack-Größe
                return Elements[CurrentSize]; // gib Element zurück
            }
            else return ERROR; // Fehler bei leerem Stack
        }

        int push(int x)
        {
            if (CurrentSize>=MaxSize-1) return ERROR; // Fehler bei vollem Stack

            Elements[CurrentSize]=x; // Element oben auf Stack einfügen

            if (minfind(<x) MinList[CurrentSize]=minfind(); // wenn Element kleiner als letztes
            // dann lege es auf den MinList - Stack
            else MinList[CurrentSize]=x; // sonst lege letztes Element erneut auf
            // den MinList - Stack
            ++CurrentSize; // erhöhe Größe / Position

            return 1; // bei Erfolg gebe 1 zurück
        }

        int minfind(void)
        {
            if (CurrentSize>0) // wenn Elemente im Stack
                return MinList[CurrentSize-1]; // gebe oberstes des MinList-Stacks,
            // also kleinstes des Stacks, zurück
            else return ERROR; // ansonsten gebe größtmöglichen Wert
            // zurück
        }
};

```