Objekte und Felder unter Ausnahmezustand ☺

Untersuchen Sie das Verhalten von Objekten sowie Feldern von Objekten beim Auftreten von Programmausnahmen (Exceptions)

Benutzen Sie dabei die folgenden Programmfragmente:

```
try { // sequenz 1
      X x;
}
catch(...) { cout<<"something ugly happened"<<endl; }
...
try { // sequenz 2
      X* p = new X;
      delete p;
}
catch(...) { cout<<"something ugly happened"<<endl; }
...

try { // sequenz 3
      X x[10];
}
catch(...) { cout<<"something ugly happened"<<endl; }
...

try { // sequenz 4
      X* p = new X[10];
      delete[] p;
}
catch(...) { cout<<"something ugly happened"<<endl; }
```

mit einer (von Ihnen geeignet zu entwerfenden) Klasse **X**, bei der sich steuern lässt, dass der n-te Aufruf des Konstruktors fehlschlägt (eine Ausnahme wirft).

Vorschlag:

```
class X { ...
      public: static void setXlimit() { ... }
      ...
};

... X::setXlimit(5); // der 6. Konstruktorruf scheitert
```

Beobachten Sie die Effekte auf die Ausführung der Sequenzen 1 - 4 hinsichtlich folgender Fragen:

1. Welche Garantien gibt C++ beim Erzeugen von lokalen und dynamischen Objekten und Feldern von Objekten, wenn dabei keine Exceptions auftreten? Welche Konstruktor-/Destruktoraufrufe sind zu beobachten?
2. Welche Garantien gibt C++ beim Erzeugen von lokalen und dynamischen Objekten und Feldern von Objekten, wenn dabei Exceptions auftreten? Welche Konstruktor-/ Destruktoraufrufe sind zu beobachten?
3. Treten bei 1. und 2. evtl. memory leaks auf?
4. Treten bei 1. und 2. evtl. memory leaks auf, falls im X-Konstruktor weitere dynamische Daten angefordert werden?

```
class X { ...
      int* p;
public:
      X(): p(new int) { ... }
      ~X() {delete p; ... }
};
```

Zur Beobachtung der Konstruktor-/Destruktoraufrufe können Sie in den jeweiligen Operationen Ausschriften à la

```
X() {
      ...
      std::cout<<"X() at "<<this<<std::endl;
      ...
}
```

erzeugen lassen, wobei die Ausgabe eines Zeigers stets die Ausgabe des Adresswertes des Zeigers in hexadezimaler Form nach sich zieht.

Für die Suche nach memory leaks sollten Sie ein Werkzeug wie **valgrind** verwenden (auf (x86 oder AMD) Linux-Maschinen problemlos aus Quellen zu erstellen, bzw. im Linux-Pool installiert).

Senden Sie als Lösung die Antworten auf die Fragen in möglichst knapper und verallgemeinerter Form (d.h. frei von Details der Testsequenzen) ein.