Thema, Ziele: C++ and ROMability, Placement `new`, Dynamic Memory Management

Aufgabe 1: C++ and ROMability

In der Vorlesung haben wir gesehen, dass unterschiedliche Konstrukte ROMable sind, jedoch nicht jeder Compiler diese Konstrukte auch im ROM plaziert. In dieser Aufgabe untersuchen Sie die ROMability und die Optimierung von ROMable Konstrukten des g++-Compilers. Für die Untersuchung müssen Sie unter Umständen Optimierungsstufen setzen.

Die folgenden Optionen des GNU-Compilers können für diese Aufgabe nützlich sein:

`-E`	Precompile only, der Output wird auf stdout geschrieben
`-S`	Assembleroutput, ohne Objectfile erzeugen
`-c`	nur compilieren
`-O0`	keine Optimierung
`-O1`	Optimierungsstufe 1 (siehe g++ - Help für Details)
`-O2`	Optimierungsstufe 2
`-O3`	Optimierungsstufe 3
`-Os`	Optimierung auf Codegrösse

Achten Sie bei allen Untersuchungen darauf, dass ihre kleinen Testprogramme nicht vollständig "wegoptimiert" werden, da die definierten Variablen nicht weiter verwendet werden. Damit das nicht passiert, können Sie den Inhalt der Variablen in die Konsole schreiben.

a) Umsetzung von Strings: Untersuchen Sie, wie die untenstehenden Stringdefinitionen umgesetzt werden. Wird `"World"` mit `"Hello World"` gemeinsam verwendet? Gibt es allenfalls Unterschiede in Abhängigkeit der Optimierungsstufen?

```
const char* pc1 = "Hello World";
const char* const pc2 = "World";
```

b) Wie werden Tabellen umgesetzt? Gibt es einen Unterschied in der Umsetzung der folgenden beiden Definitionen? Was passiert wenn Sie beiden Definitionen das Schlüsselwort `static` voranstellen?

```
const int table1[] = {1, 2, 3};
int table2[] = {1, 2, 3};
```

c) Integerkonstanten: Für die Definition von Integerkonstanten stehen Ihnen drei Möglichkeiten offen: `const int`, `enum` und `#define`. Wie werden diese Varianten umgesetzt? Geben Sie in Ihrem Testprogramm dem Compiler die Chance, eine Immediate-Adressierung zu verwenden. Um herauszufinden, ob eine Konstante mehrfach im Speicher angelegt wird (ohne Immediate-Adressierung), müssen Sie die definierte Konstante mehrfach im Programm verwenden.

d) Floating Point-Konstanten: Für die Definition von Floating Point-Konstanten stehen Ihnen `const double` und `#define` zur Verfügung. Wie werden diese Varianten umgesetzt? Eine Immediate-Adressierung ist mit `double`s kaum möglich. Um herauszufinden, ob eine Konstante mehrfach im Speicher angelegt wird, müssen Sie die definierte Konstante mehrfach im Programm verwenden.

e) Können Sie herausfinden, ob vtbl's im ROM abgelegt werden?

Aufgabe 2: Placement `new`

Bei Embedded Systems kann es sinnvoll sein, dass ein bestimmter RAM-Buffer für diverse Aufgaben zur Verfügung steht (z.B. als selbst verwalteter Heap). Speicherbereiche aus diesem Buffer können im Programm frei zugeteilt werden.

- Sie haben einen Buffer von 128 Bytes zur Verfügung.
- Ab Byte `0` sollen Sie einen Array von 3 `uint32` plazieren, welche die Werte `0x55`, `0xa354f60b` und `0x45da12` haben.
- Ab Position `64` steht ein `double` (Wert: `3.14159`).
- Auf Position `80` soll ein Array von zwei Elementen der folgenden Struktur zu liegen kommen.

```
struct FooStruct
{
  char str[5];
  int key;
};
```

  Schreiben Sie in diesen Array geeignete Werte.

a) Implementieren Sie die vorgegebene Konfiguration mit Hilfe des Placement `new`. Dazu müssen Sie den Header `<new>` inkludieren.

b) Zeichnen Sie die Memory Map des Buffers.

c) Geben Sie den Inhalt des Buffers Byte um Byte auf die Konsole aus, d.h. erstellen Sie die Memory Map mit Hilfe eines Programms.

d) Untersuchen und diskutieren Sie den erhaltenen Output.

Aufgabe 3: Fixed-size Pool

Ihre Aufgabe ist es, einen Fixed-size Pool in Anlehnung an die in der Vorlesung gezeigte Variante zu implementieren. Sie müssen die Klassen `PoolAllocator`, `HeapException`, `HeapSizeMismatch` und `OutOfHeap` realisieren, wobei die letzten beiden Klassen Unterklassen von `HeapException` sind. Die Exceptionklassen müssen möglichst schlank implementiert werden (was heisst das?).

Packen Sie alle Klassendeklarationen in das File `PoolAllocator.h` und implementieren Sie alle Elementfunktionen implizit inline. Alle Klassen müssen in den Namespace `dynamicMemory` gelegt werden. Nachfolgend finden Sei einen Ausschnitt aus der Deklaration der Klasse `dynamicMemory::PoolAllocator`.

```
template<std::size_t heapSize, std::size_t elemSize>
class PoolAllocator
{
  public:
    PoolAllocator(/* TODO: params */)
    {
      // TODO: implement this
    }
    void* allocate(std::size_t bytes) throw(HeapException)
    {
      // TODO: implement this
      // throw a HeapSizeMismatch Exception if 'bytes' doesn't match elemSize
      // throw a OutOfHeap Exception if requested 'bytes' aren't available
    }
    void deallocate(/* TODO: params */) throw()
    {
      // TODO: implement this
      // add this element to the freelist
      // set the pointer to 0
    }
```

```
  private:
    union Node
    {
        uint8_t data[elemSize];
        Node* next;
    };
    Node* freeList;
};
```

Das Testprogramm finden Sie im Verzeichnis `./Vorgabe`. Der Output des Testprogramms soll wie folgt aussehen (Die Anfangsadresse kann unterschiedlich sein):

```
p[0] = 0x28cbe0
p[1] = 0x28cbe4
p[2] = 0x28cbe8
p[1] = 0
p[3] = 0x28cbe4
Heapsize mismatch exception occurred
```

f) Implementieren und testen Sie die Klassen vollständig. Erweitern Sie das Testprogramm so, dass auch eine `OutOfHeap`-Exception geprüft wird.

g) Erweitern Sie die Elementfunktion `dynamicMemory::PoolAllocator::allocate()` so, dass die Anzahl der angeforderten Bytes nicht mehr genau der Elementgrösse entsprechen muss, sondern kleiner oder gleich der Elementgrösse sein kann.

Aufgabe 4: Block Allocator

Implementieren Sie nun einen Block Allocator. Die Klasse `BlockAllocator` soll aus 4 Fixed-size Pools gemäss Aufgabe 3 b) bestehen. Die einzelnen Elementgrössen können der Klasse `BlockAllocator` mittels Templateparameter mitgegeben werden. Der Block Allocator nimmt jeweils den kleinstmöglichen Pool, um die Anforderungen zu erfüllen. Das bedeutet auch, dass auf den nächstgrösseren Pool zugegriffen wird, falls der kleinere Pool voll ist.