3.2 DESIGN PATTERNS FOR LIMITED MEMORY

When composing designs for devices with a limited amount of memory, the most important principle is not to waste memory, as pointed out by Noble andWeir (2001). This means that the design should be based on the most adequate data structure, which offers the right operations.

LINEAR DATA STRUCTURES

- In contrast to data structures where a *separate memory area is reserved foreach item, linear data structures are those where different elements are located next to each other in the memory.*
- Examples of non-*linear data structures include common implementations of lists and tree-like data structures, whereas linear data structures can be lists and tables, for instance.*
- The difference in the allocation in the memory also plays a part in the quality properties of data structures.

Linear data structures are generally better for memory management than non-linear ones forseveral reasons, as listed in the following:

- **Less fragmentation**. Linear data structures **occupy memory place from one location**, whereas non-linear ones can be located in different places. Obviously, the former results in less possibility for fragmentation.
- **Less searching overhead**. Reserving a linear block of memory for **several items onlytakes one search for a suitable memory element** in the run-time environment, whereas non-linear structures require one request for memory per allocated element. Combined with adesign where one object allocates a number of child objects, this may also lead to a serious performance problem.
- **Design-time management**. Linear blocks **are easier to manage at design time**, asfewer reservations are made. This usually leads to cleaner designs.
- **Monitoring**. Addressing can be performed in a monitored fashion, because it is possible tocheck that the used index refers to a legal object.
- **Cache improvement**. When using linear data structures, it is more likely that the next data element is already in cache, as cache works internally with blocks of memory. A related issueis that most caches expect that data structures are used in increasing order of used memory locations. Therefore, it is beneficial to reflect this in designs where applicable.
- **Index uses less memory**. An absolute reference to an object usually consumes 32 bits, whereas by allocating objects to a vector of 256 objects, assuming that this is the upper limit ofobjects, an index of only 8 bits can be used. Furthermore, it is possible to check that there will be no invalid indexing.

BASIC DESIGN DECISIONS

1. Allocate all memory at the **beginning of a program**. This ensures that the application always has all the memory it needs, and memory allocation can only fail at the beginning of the program.
2. **Allocate memory for several items, even if you only need one**. Then, one can build a policy where a number of objects is reserved with one allocation request. These objects can then be used later when needed
3. Use standard allocation sizes

4. Reuse objects
5. Release early, allocate late
6. **<u>Use permanent storage or ROM when applicable</u>**. In many situations, it is not even desirable to keep all the data structures in the program memory due to physical restrictions.
7. **<u>Avoid recursion</u>**. Invoking methods obviously causes stack frames to be generated. While the size of an individual stack frame can be small – for instance, in Kilo Virtual Machine (KVM), which is a mobile Java virtual machine commonly used in early Java enabled mobile phones, the size of a single stack frame is at least 28 bytes (7 × 4 bytes) –functions calling themselves recursively can end up using a lot of stack, if the depth of the recursion is not considered beforehand.

<u>Data Packing</u>

Data packing is probably the most obvious way to reduce memory consumption.There are several sides to data packing.

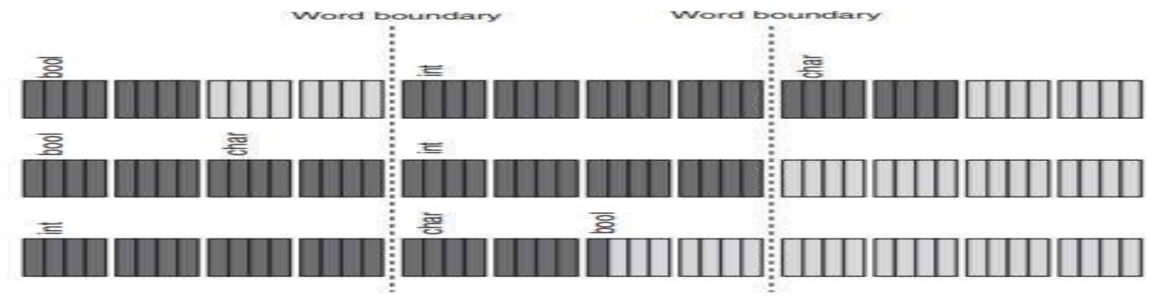

Figure 2.3 Saving memory by considering data structure alignment

Use compression with care. In addition to considering the data layout in memory, there are severalcompression techniques for decreasing the size of a file.

- **Table compression, also referred to as nibble coding or Huffmancoding, is about encoding each element of data in a variable number of bits so that the more common elements require fewer bits.**

- **Difference coding is based on representing sequences of data according to the differences between them.** This typically results in improved memory reduction than table compression, but also sometimes leads to more complexity, as not only absolute values but also differences are to be managed.

- **Adaptive compression is based on algorithms that analyze the data to be compressed and then adapt their behavior accordingly**. Again, **further complexity is introduced, as it is the compression algorithm that is evolving,not only data.**

-