# Chapter 2

# Data structures

## 2.1 General principles

In this chapter an overview of the data structures is given, as well as indicating how memory management is undertaken. For more information about how to use and develop data structures, you should see chapter 8 on designing data structures.

One of the main thrusts of Meschach is to use C's data structuring ability to "package" the objects so that they are self-contained and can be dealt with as single entities. This is combined with C's memory allocation and de-allocation techniques to make basic mathematical objects (vectors, matrices, permutations etc) work more like their mathematical counterparts. So, a vector structure contains not only the array of its components, but also the dimension of the vector, and the amount of allocated memory (which may be larger than the dimension). This vector can be used for ordinary vector operations, computing matrix–vector products, solving systems of linear equations, or just for storing data. If there is a mismatch in, say, the size of the vector and the vectors or matrices that it operates with, then an error is raised to indicate this. The vector can also be created when needed, and destroyed when not. It can be re-sized when desired to be larger or smaller.

The type of floating point number is `Real`, which is one of the floating point types. The default floating point type is `double`.

The integer vector and permutation data structures are very similar to the vector data structure, and contain not only the array of values, but also the current dimension or size of the integer vector or permutation and the amount of allocated memory in this array. Permutations are really restricted integer vectors; they are initialised differently (to the identity permutation, instead of all zeros) and the permutation routines preserve the property of being a permutation.

Matrices are represented by a more complex data structures, and are essentially a two-level data structure. To have variable size 2-dimensional arrays in C, pointer-to-pointer structures are needed, such as

```
Real **Aentries;
  ......
```