Data types and structures

Primitive types

Boolean (true or false, 1 or 0)

Character

Integer, integral or fixed-precision values

Floating-point, single-precision real number values

Double, a wider floating-point size

Enumerated type, a small set of uniquely named values

Composite types

Array

Record (also called tuple or struct)

Union

List

Stream

Set

Multiset

Stack

Queue

Double-ended queue

Tree

Graph

We will see that all data structures are named sets or are built of named sets.

A *character* is a unit of data that roughly corresponds to a symbol, such as a letter in the written form from an alphabet of a natural language.

Examples of characters include letters (a, b, c), numerical digits (1, 2, 3), common punctuation marks (such as "." or "-"), and the whitespace.

A character is related to its meaning. As a result, a character, its meaning and the connection between them form a named set.

An *enumerated type* (also called *enumeration*, *enum,* or *factor*) is a data structure that consists of a set of named values called elements, members or enumerators of the type.

For example, an enumerated type called color may be defined to consist of the enumerators Red, Green, Black, While, Missing, and Blue. In some languages, the declaration of an enumerated type also intentionally defines an ordering of its members while in others, the enumerators are unordered.

Utilization of named values shows that an enumerated type is a named set. An ordering determines one more named set.

An *array* describes a collection of elements (values or variables) labeled by one or more indices (identifying keys) that can be computed at run time by the program. It is analogous to the mathematical concepts of vector and matrix. Array types with one and two indices are often called vector type and matrix type, respectively.

Utilization of labels, which are specific names, shows that an array is a named set.

An *associative array* (also called a *map*, *symbol table*, or *dictionary*) is a data structure composed of a collection of (key, value) pairs in such a way that each possible key appears at most once in the collection.

Systems of pairs are binary relations, which, in turn, are named sets. This means that an associative array is a named set.

A *multimap* (sometimes also *multihash*) is a generalization of a map or associative array in which more than one value may be associated with and returned for a given key.

By the same reason as before, a multimap is a named set.

A *record* (also called *struct* or *compound data*) is a collection of fields, possibly of different data types, typically in fixed number and sequence. The fields of a record may also be called members or elements.

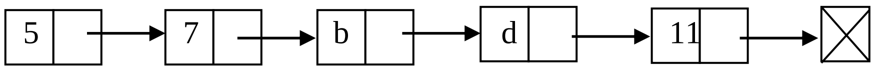
Records are distinguished from arrays by the fact that their number of fields is typically fixed, each field has a name, and that each field may have a different type.

As fields in a record are associated with types, a record is a named set.

A *list* or *sequence* represents an ordered sequence of records (values), in which the same value may occur more than one time. Order is constructed by a link of each element (node) to the next element (node). The last value has the link to the empty element (null) to facilitate traversal of the list. As a result, each element (or *node*) of a list has two fields – one for the value and the other for the link.

An example of a list is a computer representation of the mathematical concept of a finite sequence. The (potentially) infinite analog of a list is a *stream*.

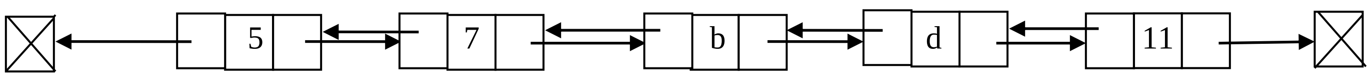
Lists are used for various applications.



Example. A list with 3 integer and 2 character elements.

A list consists of pairs (value, link). Systems of pairs are binary relations, which, in turn, are named sets, that is, a list is a named set. At the same time, pairs' links to the next element form one more named set. As a result, a list is a named set of the second order.

A *doubly linked list* is similar to a list but its elements (nodes) have two links - one to the previous node and the other to the next node in the sequence of nodes. As a result, each element (or *node*) of a doubly linked list has three fields – one for the value, one for the forward link and one for the backward link.



Example. A doubly linked list with 3 integer and 2 character elements.

In a similar way to a list, a doubly linked list is a named set of the second order.

A *stream* is a sequence of data elements that come (made available) over time. A stream can be thought of as items on a conveyor belt being processed one at a time but not in large batches.

As elements of a stream are ordered, it is a named set.

A *set* is a data structure that stores certain values, without any particular order, and no repeated values. It is a computer implementation of the mathematical concept of a finite set. Unlike most other data structures, rather than retrieving a specific element from a set, one typically tests a value for membership in a set.

A set *X* is a single named set because in it all elements have the same name – "an element from the set *X*."

A generalization of the data structure set is the data structure *multiset* (or multiple membership set) or *bag*, which is similar to a set but allows repeated (equal) values (duplicates of elements).

A multiset *M* is a named set, the support of which consists of the elements of *M* and the set of names consists of the types of these elements.

Multisets were rediscovered many times and those who did this gave to them other names:

aggregate, *bag*, *heap*, *bunch*, *sample*, *weighted set*, *occurrence set*, and *fireset* (finitely repeated element set)

According to Knuth, the Indian mathematician Bhascara Acharya (circa 1150) was the first researcher who used multisets.

A *stack* is an ordered collection of elements with two principal operations: **push**, which adds an element to the collection, and **pop**, which removes the most recently added element that was not yet removed. The order in which elements come off a stack gives rise to its alternative name, **LIFO** (Last in, First out). An additional operation may give access to the top without modifying the stack.

Any ordered collection of elements is a named set. Consequently, a stack is a named set.

A *queue* is an ordered collection of elements with two principal operations: **addition** of entities to the rear terminal position, known as *enqueue*, and **removal** of entities from the front terminal position, known as *dequeue*. It has the alternative name FIFO (First-In-First-Out)

Any ordered collection of elements is a named set. Consequently, a *queue* is a named set.

A *double-ended queue* (*dequeue*, often abbreviated to *deque*, pronounced *deck*) is a data structure, which generalizes a queue, in which elements can be added to or removed from either the front (head) or back (tail).

Any ordered collection of elements is a named set. Consequently, a double-ended queue is a named set.

A *priority queue* is a data structure similar to a queue or stack but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

Any ordered collection of elements is a named set. Consequently, a stack is a named set. At the same time, elements with priorities form one more named set. As a result, a priority queue is a named set of the second order.

An *undirected graph G* consists of a finite set *V* of vertices or nodes or points, together with a set *E* of unordered pairs of these vertices called edges, arcs or lines.

An undirected graph *G* is a named set *G* = (*V*, *E*, *V*).

A *directed graph G* consists of a finite set *V* of vertices or nodes or points, together with a set *A* of ordered pairs of these vertices called arrows, directed edges, directed arcs or directed lines.

A directed graph *G* is a named set *G* = (*V*, *A*, *V*).

A *labeled graph* is a graph in which some value (label), such as a symbolic label or a numeric attribute (cost, capacity, length, etc.) is associated to each edge.

A graph is a named set. At the same time, edges with labels form one more named set. As a result, a labeled graph is a named set of the second order.

A *tree* is defined recursively (locally) as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reference is duplicated, and none points to the root.

A tree is an undirected graph without cycles. Consequently, it is a named set.

Selection of Data Structures

The choice of the particular data structure depends on two criteria:

- ◊ It must be sufficiently rich to represent the relevant relationships between data elements
- ◊ It should be sufficiently simple allowing one to effectively process the data when it's necessary
- ◊ It has to be compatible with the utilized technical means, e.g., programming language