10. The Pecking Order (List and Array Data Structures)

Blot out, correct, insert, refine,
Enlarge, diminish, interline.
J. Swift, *On Poetry*, (1733) l.85.

Lists
One of the most common structures, both in Computer Science and in real life is the list. Write down a list of your all time favourite records. The list I thought of was (no doubt you disagree):

1. *Bohemian Rhapsody* - Queen
2. *Stairway to Heaven* – Led Zeppelin
3. *Your Song* – Elton John
4. *Who made who* – AC/DC
5. *Sweet Child of Mine* – Guns 'n Roses

A list is just a sequence of things in some order.

After writing this list I could not think of any more, but later in the evening I realised I had missed out *Every Breath You Take* by the Police, and Bruce Springsteen's *The Ghost of Tom Joad*. My new list is

1. *Bohemian Rhapsody* – Queen
2. *Every Breath You Take* - Police
3. *Stairway to Heaven* – Led Zeppelin
4. *Your Song* – Elton John
5. *Who made who* – AC/DC
6. *Sweet Child of Mine* – Guns 'n Roses
7. *The Ghost of Tom Joad* – Bruce Springsteen

Lists can thus have new entries added anywhere, including in the middle, and their length can increase as needed.

In fact on second thoughts, Guns 'n Roses are not that good, so they should not be in the list at all:

1. *Bohemian Rhapsody* – Queen
2. *Every Breath You Take* - Police
3. *Stairway to Heaven* – Led Zeppelin
4. *Your Song* – Elton John
5. *Who made who* – AC/DC
6. *The Ghost of Tom Joad* – Bruce Springsteen

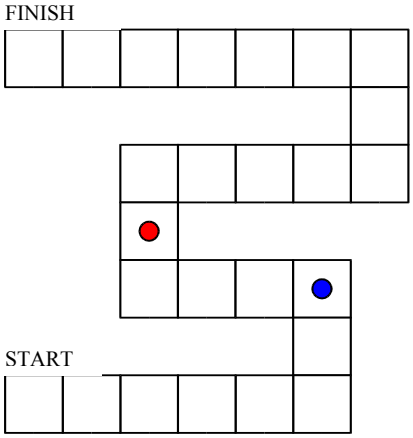
Entries can be removed from lists.

We use lists all the time. Whenever I go shopping, I take a list of things to buy, as I have a terrible memory and would otherwise be bound to come home without the

main thing I went for. I do not go round the shop finding the items in the order they appear on the list as I am also lazy and that would involve walking round the shop several more times than I need to. As I walk past an item I take it and remove it from the list by crossing it out. Again things are being removed from the middle of the list.

The "To-Do" list is one of the most common ways that people use to organise their work or studies. It simply involves keeping a list of the things that need to be done. When a new task arrives, due perhaps to an email or phone call, it is added to the To-do list. When a task is finished, it is rather satisfyingly crossed off the list. This way nothing is forgotten. Most people keep their To-do lists on a piece of paper. Periodically the list gets in such a mess that they copy out the items remaining on to a fresh piece of paper. This reflects the fact that a list is a dynamically changing thing, but that paper is static. One way devised to get over this problem is to use post-it notes (small slightly sticky squares of paper) instead of paper (Norman, 1993). Norman describes post-it notes as "*the decade's most important artefact*". This is in part due to their ability to implement **dynamic data structures**: ever changing structures. They can be arranged along the edge of a shelf, on the desk, on the monitor, etc. Each new task is placed on a post-it note. The medium used to store the list is now as dynamic as the list itself. Items in the list can be shuffled about. Deleting an item involves simply removing and throwing away that post-it note. Items no longer need to be repeatedly copied onto a new sheet.

Lists can also be fixed, never changing in length, though with the values of the entries changing. Many simple children's board games are based on list-like structures. Remember all those games you played which involved racing round a snake-like board where how far you moved depended on the throw of a dice. The board is just a list of squares, the list entries being either empty or holding pieces.

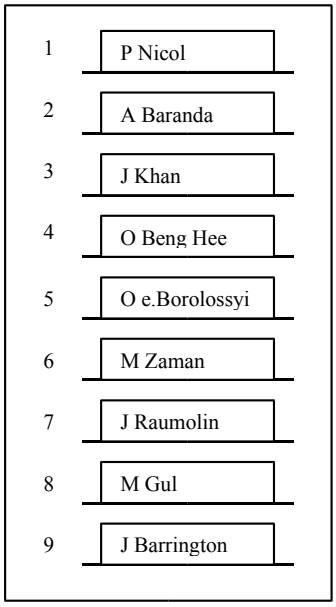


Lists can, in general, increase or decrease in length with the extra ones being added at the start, end or anywhere in between. Elements in a list have a position, but when something new is added the positions change. By making restrictions or generalisations on the idea of a list, we get a range of different data structures with different uses as we shall see.

Arrays

A common variation on list-like structures is that of an **array**: a fixed sized list whose elements are accessed by labels or **subscripts**. Most squash clubs operate a squash

ladder (similarly chess clubs). This is physically a card with numbered slits in. Each member of the ladder writes their name on a tabbed card that fits into the slots. If you play someone higher than you on the ladder and beat them, you swap places, the aim being to be the person at the top of the ladder. The labels on the slits thus give each player their current rank in the club.



The squash ladder is an array structure. All the entries are of the same kind: they are all names of squash players. Each position is numerically labelled, so you can easily ask questions of it such as who is the current No 1 or who is three positions higher than me? The ladder also has a predetermined number of positions. If there are more people wishing to be on the ladder than there are slots, a waiting list may be set up, or a second (division) ladder created, but the original physical ladder cannot have new slots added as there is no space. When a new person arrives who is known to be good, a new slot cannot be created in the middle of the ladder to start them in a position close to their correct one. The best that could be done if this were required is to shuffle each player down one position to create a gap. This would be time-consuming.

At sports events such as ice-skating (gymnastics and boxing are similar), marks are awarded by a panel of judges. The judges sit in a line of seats, with the seat labelled by their country. Before technology took over, the judges would hold up cards with their scores. On the television screen each skater's scores are given in a row, again with the marks. This is another example of an array: a fixed size list of things of interest (such as scores) each with a label to identify them (such as the country).

6.0	5.9	5.4	5.9	5.8	6.0
GB	FR	DE	US	GR	AU

In ice skating scores, the "things" are thus numbers and the labels are letter pairs that indicate the countries of the judges.

A row of houses down one side of a street is also organised like an array. Each house is labelled by its postal address: for example 2, 4, 6, 8, etc. assuming it is on the even

side of the street in Britain. When a postman has a letter to deliver, they know where it goes from the number. The number is just a label; the actual house is the thing it is labelling. In normal circumstances, new houses are not added in the middle – once your house is given a number, it is not changed.

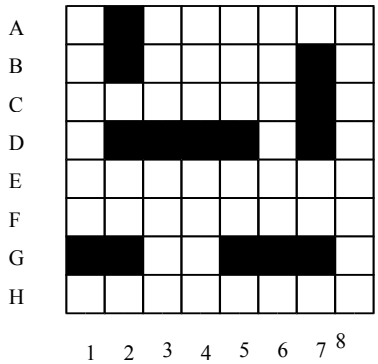
My photo albums are similar to an array. They are “flip albums”, with leaves for holding each photo. I sort through my photos when they come back from the developers and put them in albums in the order they were taken. Putting the films into the individual slots is very time consuming taking time proportional to the number of photos taken. When I go on holiday however I often take dozens of films. Sometimes one of the films gets left in a coat pocket and is not developed with the others. It of course turns up just after I have put the other photos into the flip album. I then have the problem of putting the new photos in their correct position somewhere in the middle. I must find the correct position for the new photos and shuffle many of those already placed up to make space. This takes as much time per photo moved as originally – the original time was wasted. Arrays have the same problem, to put new things into the middle without just overwriting the existing ones (putting new photos so they cover the old ones for example) takes a lot of time. We will look at linked list data structures that overcome this problem of arrays in a later chapter.

A one-dimensional array such as the examples above is in many ways similar to a list. The difference is that arrays are of fixed sizes, with each cell "named" by a subscript. An array consists of a fixed number of slots, that things can be put in or taken out of. New slots cannot however be created.

Multidimensional Arrays

Arrays do not need to be just a single row of things. They can also come in grids or tables – a whole series of rows like a chessboard. To allow for games to be recorded, each square on the board has a label, but here the labels are pairs. One part of the label gives the row of the square and the other the column. In chess the rows are given numbers from 1 to 8. The columns are labelled by letters a to h. Thus a common first move is for white to move their pawn from e2 to e4.

The game of Battleships is similar. In this game, you each position battleships of various sizes on a grid (a 2-dimensional array), then try to find the other person's ships. You do this by calling out the grid reference (subscripts) of squares such as "B3" where you think the ship might be. If part of a ship is there you are told you have a hit, otherwise a miss.



Cinemas and theatres use a similar system to label seats, with a letter indicating the row, and a number giving the seat with in the row. To find your seat you find the row labelled with the letter that is indicated on your ticket. You then look at the numbers on the street backs to find your seat. Cinemas are thus just arrays of people.

Arrays have a fixed size. In general you cannot add extra seats in a cinema when it gets full. You have to turn the extra people away. If you are in a group of three, and the cinema is so full that there are only pairs of seats left, you cannot add an extra seat in the middle so you can all sit together. You could try asking everyone in a row to move up one, but that would make you unpopular, especially if the film had started as it takes time and is disruptive. Such are the problems of arrays.

Maps also often label positions on the map by super-imposing a 2-dimensional grid over the map. The index in a London A-Z map for example tells you where each street is on a given page. This is done by giving a column letter from A to K and a position number from 1 to 7. This narrows down the street to a single square. In fact an A-Z book is actually a 3-dimensional array as it is a whole series of pages stacked on top of one another. Any particular square in the book is accessed by giving three labels, rather than just two. The third label is the page number. Each page is labelled in exactly the same way using the further two labels.

Arrays do not need to stop at 3-dimensions. For each extra dimension we need an extra set of labels. The company that produce the London A-Z book also produce A-Zs of many other cities. The whole collection of A-Zs sat on a shelf is organised like a 4-dimensional array. The fourth set of labels is the name of the city. For example, the position of Nelson's Column is given by the four labels (London, 77, 1, H). The Crucible Theatre in Sheffield where the World Snooker Championships are held each year is in the square (Sheffield, 4, 3, F).

A final example of a 2-dimensional array is the multiplication table. It is a constant array (its values never change) that is considered so important that we are made to learn it off-by-heart in childhood. The subscripts to this array are the 2 numbers we wish to multiply. The value in the cell is the pre-computed answer to each multiplication.

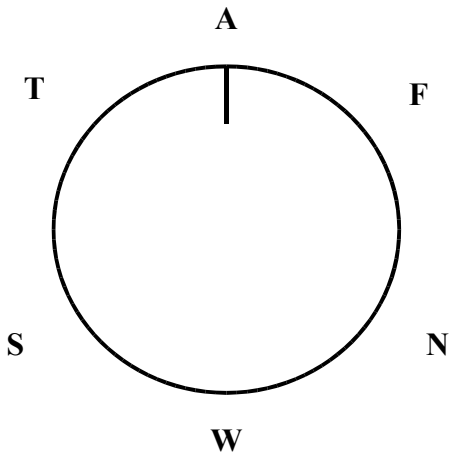
Look-up Tables

Lookup tables (in some circumstances called **bucket arrays**) are really just arrays, where the subscripts correspond to known information, and the array contains further information that might be looked up in the table.

The children's book, "The Pop-Up Mice of Mr Brice" (Dr Seuss, 1998) contains a lookup table. It is about a house full of 26 mice. Their names each start with a different letter of the alphabet. At the beginning is a pop-up picture showing twenty six doors, each with the name of a mouse and corresponding letter of the alphabet. The doors can be opened to reveal a message. Open Ursula's door to find the message "Ursula is out". Open Xavier's door to find the message "Xavier is in". The sequence of doors are being used as a lookup table. The subscripts are the letters of the alphabet, representing the names of the different mice. The contents of the lookup table are the messages. If you know the name of a mouse, you can use the table to find out whether they are in or out.

There is a lookup table on the front of my washing machine at home. The washing machine has many different settings depending on whether I wish to wash whites or coloureds, hot or cold, etc. Each programme has a separate position on a dial. Somehow I need to easily work out the correct setting for the current wash. Each position of the dial is labelled with a letter. On the front of the machine is a table describing the programme that each letter stands for. It is a look-up table. To do a wash, I find the entry in the table for the programme I want, and it tells me the letter I should set the dial to.

Whites	95	**A**
Fast Coloured	60	**F**
Non-fast Coloured	40	**N**
Woolens	40	**W**
Spin Dry		**S**
Tumble Dry		**T**



A multiplication table is similarly also a lookup table. It allows you to look up the answers to multiplication problems that have been pre-computed. Most multiplication tables stop at the 12 times table. Why do you rarely see a 20 times table or a 100 times table? This is because lookup tables use up a lot of space, either on paper on in your memory if you rote-learnt it as a child. You therefore only want to use it as a look up method if you will be using its contents many times. This is likely for the entries in 10 or 12 times tables but larger values are likely to be used less, so you find a different method when this is needed (long multiplication for example).

Maps also contain lookup tables, usually called *Legends*, which tell you what each symbol used on the map means. A symbol such as a black dashed line can be looked up in the table. There its meaning, a footpath, is found.

Adjacency Matrices

A special form of 2-dimensional array /lookup table is known as an **adjacency matrix.** These are used to indicate when pairs of things are connected in some way, possibly giving some information about the connection. The two sets of labels used are the same in this case. This kind of structure is often used in road atlases to give distances between cities. Both the rows and columns of the table are labelled by the same set of cities. To find the distance between two cities, you look up the row of the place you are starting from and the column of the place you are going to. The number in the table where they meet gives the distance between them. This is an adjacency matrix, where every pair has an entry, since a distance is known for every pair of cities.

Several newspapers give the season's football scores in an adjacency matrix that is updated each week. The rows and columns are labelled by team names. The rows represent the home team and the column the away team. Blank entries indicate that the teams have not met. If the teams have met then the score is given as the entry. Here two teams being "connected" means that the corresponding match has been played. The same information could just be given as lists of the scores from each week. This would make it easy to ask questions such as "who played who in the 4th week of the season?" or "what were this week's scores?" However, it would take more space, and would make it harder to find the score of any particular team. It would also be harder to ask the question as to whether two teams have played yet. The organisation of data chosen thus depends on what question is being asked. Papers that include an adjacency matrix also give the current week's scores as a normal list as most readers are interested mainly in the current week's scores. Different readers at different times want different information, so need different data structures.

Records

A **record** is a very common list-like structure that is used to package lots of different information together to describe an object or entity. A record differs from an array or list in that the order of entries has no significance. Entries are usually accessed by named labels. Furthermore, each entry in an array holds the same kind of information as every other entry: all entries are names of football teams or all entries are numbers. A record differs in that the different entries can be different: one a name, another a number. Each entry in a record is called a **field** and the labels are **field identifiers**.

Record-like structures abound in everyday life. Dates are one example. A date might consists of a day field, a day of the month field, a month field and a year field, such as:

Saturday 28th August 1999

The order we write the entries does not matter:

Saturday August 28th 1999

is still the same date, for example.

Times are also given as a record such as:

12:03 PM

Here we have given an hour, a number of minutes and an indication of morning or afternoon.

Tracks on CDs are often described by records (of the computer science rather than the vinyl kind) on the cover. For example the 7th track on the album *Rumours* by Fleetwood Mac (which happens to be the old BBC Grand Prix Theme Music) is described by a record consisting of the track's name and its running time:

The Chain 4:28

Since a time can be considered as a record, this is an example of a record where one of the fields is itself a different kind of record. The inside sleeve of the Album *Harvest* by Neil Young uses a longer record with an extra field to give additional notes about other contributors to the track. For example, the track *Heart of Gold,* is described by the record:

Heart of Gold
3:05
with the Stray Gators / additional vocals by James Taylor & Linda Ronstadt

Whenever you fill out a form, you are filling in a record. Each entry on the form is a record field. Here the fact that fields are labelled is made explicit since the labels such as "Name" are written next to the boxes. I recently filled in a Birth Certificate for my daughter or rather I supplied information to a Registrar who filled it out in front of me. It had fields such as name, date of birth, place of birth, mother's name, mother's occupation etc. A Registrar is someone whose job is thus to fill out records about the major events in people's lives: births, deaths and marriages. These records have been filled out for hundreds of years for every person in the country. It is these records that my father searched through to build up my family's family tree back to the 15th Century. Unfortunately he still has not shown that I am related to the former Viceroy of India, George Nathanial Curzon. Nor can I be sure I am part of the Norman family who came to Britain with William the Conqueror and whose stately home is Keddleston Hall. To date I just know I am descended from a family of Lead Miners.

Records are often grouped together in other structures. We have already seen records inside records. We also often have an array of records. The Football League tables given each week on Grandstand and in the Sunday Newspapers are an example of this. Each line in the League table is a record, giving for example, the name of a team, the number of games played and points scored. Each line describes an entity: a football team. These records are then grouped together in an array. Notice that each entry in the array is of a single kind: a football team record, even though each record is combining several different pieces of information. Take for example the fantasy Premiership table below. An example record is:

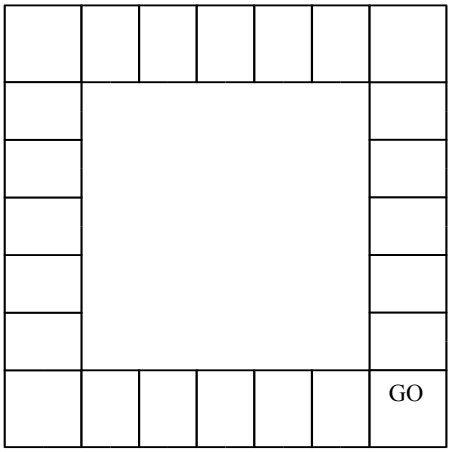
Newcastle United: Played 2, Points 4.

This record is the 3rd entry in the array. Each record has fields: Team, Played and Points. The array has subscripts from 1 to 7.

	Team	Played	Points
1	Leeds United	3	9
2	Sheffield Wednesday	3	5
3	Newcastle United	2	4
4	Chelsea	2	3
5	Liverpool	2	2
6	Arsenal	3	1
7	Manchester United	3	0

Circular Lists

When you think of a list, you probably think of something with a start and an end, but it is possible to have never-ending lists. An example is seen in the board game Monopoly. It has a board that is just a list of squares representing the streets of London. However, the last square, *Mayfair*, is connected back to the start square, *Go,* in a loop, so do not get to the end of the game by crossing a finishing line. There is no end point – you could travel round and round forever. The game finishes when someone is bankrupt, not when a finish line is reached.



Summary

Choosing a good way of organising things can make a big difference to the task of manipulating those things. Lists and their variations are some of the simplest and most common ways of organising things.

A **list** is just a sequence of things in a given order. In general a list's length can increase or decrease. New elements can be added to or removed from the start, end or in the middle. However, a list does not have to change. The elements of a lists are usually accessed in order.

An **array** is a list of a fixed size where all entries are the same kind of thing. It can neither grow nor shrink though its elements can be changed. Its elements are accessed using position labels (know as **subscripts**). Given a position the element at that position can be accessed. In a **multidimensional array** each position is identified by more than 1 label. For example, in a 2-dimensional array, 2 labels are needed to identify each element – a row label and a column label.

A **lookup table** is an array used to look up information about something. The things we want to find information about are used as subscripts. The array contains the information that is to be looked up.

An **adjacency matrix** is a 2-dimensional lookup table that contains information about the connectivity of a set of things. The same labels are used for both sets of subscripts. Entries in the table indicate whether the pair given by its position labels are "connected" or not.

A **record** is a collection of items that give information about a single entity. The separate pieces of information are called **fields**, and they may be of different types. The positions of the fields are not relevant as they are accessed solely by **field identifiers**.

A **circular list** is just a list, whose last element is followed by the first element. It is thus a never-ending list.