Basic Abstract Data Types

- List: a list is a sequence of 0 or more objects of a given type

We have "ideas" of: length, first element, last element, previous element, next element, empty list, ith element.

Hence we can propose operations

Insert (x, p, L); (insert x in L at position p)

Locate (x,L)	empty (L)
Retrieve (p,L)	create (L)
delete (p,L)	first (L)
Next (p,L)	printlist (L)
previous (p,L)	...

Implementation 1.

- Use array.

```
type List = record
              elements: array [1 .. maxlength]
                        of element type;
              last: integer
            end;
     position: integer;
  var L: list;
```

Some operations: To Insert

L. last $\geq$ maxlength $\Rightarrow$ error
$P > L.\ last\ + 1 \Rightarrow$ error
else "squeeze in"

Try Delete, Locate, etc on your own.

Implementation 2.

• Use pointers.

```
type
      celltype = record element: elementtype;
                        next:     ↑ celltype
                 end;
      LIST = ↑ celltype;
      position = ↑ celltype;
       var p : LIST; x : elementtype
Insert (x,L)       * insert at ist place *
(var L: LIST)
      new (p);
      p ↑ . element := x;
      p ↑ . next := L;
      L := p;
```

Do DELETE and LOCATE on your own.

Basic Abstract Data Types

- Queue:

1. Queue is a sequence of elements that allows new elements to be inserted at end, deleted at front.

2. Operations
   Procedure Empty (var Q: QUEUE);
   create an empty queue;
   function IsEmptyy (Q: QUEUE): boolean;
   test emptiness;
   Procedure Insert (e: elementtype; var Q: QUEUE);
   Procedure Delete (var Q: QUEUE);
   Procedure Readfirst (var e: element-type; Q: QUEUE);

- Implementation

```
Use an array:
  type Queue = record (may be circular)
                 first,last: integer
                 item: array[1..max] of elementtype
       end
Use pointers:
  type link = ^cell;
       cell = record
                 item = elementtype;
                 next = link
               end;

        Queue = record
           first: link
           last: link
        end;
```

Basic Abstract Data Types

- Stack:

1. A <u>Stack</u> is a sequence of elements that allows insert/ delete at only one end.

2. Operations:

   (a) Create empty stack

   (b) Test if stack empty

   (c) Insert e into S

   (d) Delete top element of S

We can also specify in Pascal, as for Queue.

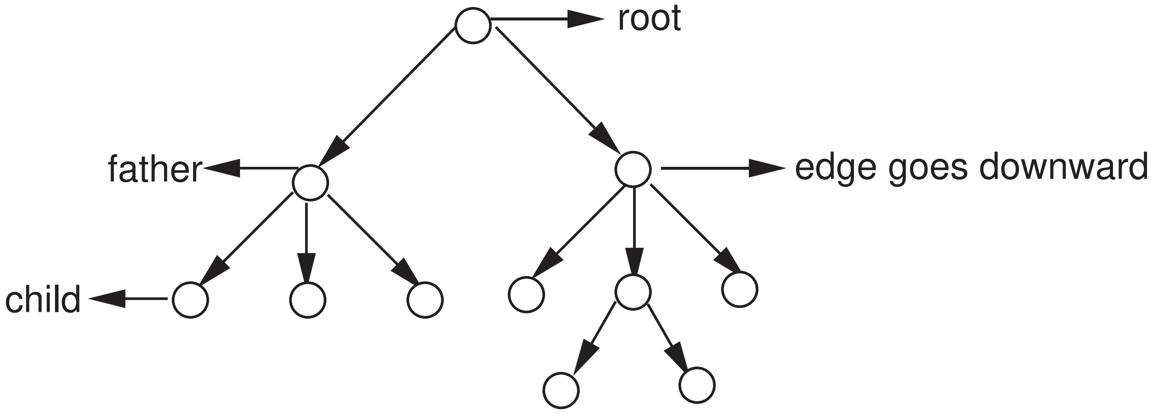
• Stack Implementation

```
Use an array:
type Stack = record
       top: 0..max;
       item: array[1..max] of elementtype;
     end;

Use Pointers:

  type link = ^ cell;
       cell = record
               item: element type;
               next: link
              end
       stack = link;
```

Trees

In Graph Theory: a tree is a connected acyclic graph.

In Computer Science: we usually mean <u>rooted, labeled, and ordered tree</u>



But in some applications, edges pointing upward is needed.

Binary trees

- rooted
- a node has ≤ 2 children

```
                    0 root
                   / \
                  /   \
                 /     \
                /       \
               /\       /\
              /  \     /  \
left subtree /    \   /    \ right subtree
             ------   ------
```

- Number of Binary trees of n nodes:

$$
\begin{aligned}
B_n &= \begin{cases} 1 & \text{if } n = 0 \\ \displaystyle\sum_{i=0}^{n-1} B_i \quad B_{n-i-1} & \text{otherwise} \end{cases} \\
&= \frac{1}{n+1}\binom{2n}{n}
\end{aligned}
$$

This is the n-th Catalan number.

Examples:

```
emtpy tree                 o
                          / \
                         o   o

   o

   o                       o
  /                       / \
 /                       /   \
o                       o     o
                             / \
                            /   \
                           o     o

  o                       o
   \                     / \
    \                   /   \
     o                 o     o
                            /
                           /
                          o
```

ADT Tree:

1. rooted, labeled, ordered trees
2. with the following operations:

- Parent(n, T)
- leftchiled(n, T)
- rightsibling(n, T)
- label(n, T)
- create(T)