CSCI203

Week 1 – Lecture B

Data Structures

- In this lecture we will examine a few basic data structures:
  - The Array.
  - The List.
  - The Stack.
  - The Queue.
  - The Record.

Arrays

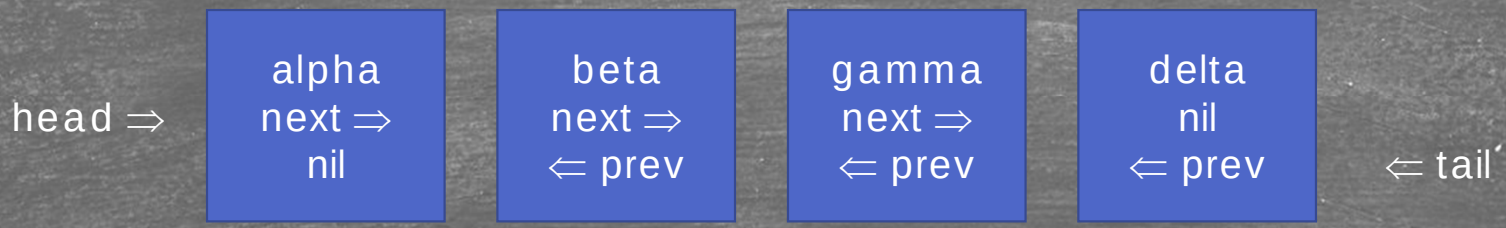
- An array is a data structure consisting of a fixed number of data items of the same type
  - E.g. table: array[1..50] of integer, letters: array[1..26] of character
- Any array element is directly accessible via an index value
  - E.g. x = array[27], initial = letters[7]
- Arrays can have more than one index, multidimensional arrays
  - E.g. heights: array[1..20,1..20] of real is an array with 400 elements
- Initializing an array takes n operations for an array of n elements

Lists

- A list is a collection of items arranged in some order
- Unlike an array, elements of a list cannot be directly accessed via an index
- List items (nodes) are records containing data and a pointer to the next node in the list
  - E.g. type node = record
    contents: stuff
    next: ^node
- A list may also have a pointer back to the previous node in the list
  - E.g. type node = record
    contents: stuff
    next: ^node; prev: ^node

Lists

- Special pointers head (and tail for doubly linked lists) are maintained to point to the first (and last) elements of the list.
  - E.g.
    head: ^node
    tail: ^node



Lists

- Insert an item onto a list start

```
item: ^node
procedure listaddstart(item)
      item^.next = head
      head = item
```

- Insert an item onto a list end

```
procedure listaddend(item)
      tail^.next = item
      item^.prev = tail
      item^.next = nil
      tail = item
```

Lists

- Insert an item into a list after a specific node

```
item: ^node
procedure listaddmid(item, match)
        ptr: ^node
        ptr = head
        while ptr^.contents ≠ match & ptr^.next ≠ nil do
                ptr = ptr.next
        item^.prev = ptr
        item^.next = ptr^.next
        ptr^.next = item
        ptr = item^.next
        if ptr = nil then
        tail = item
        else
        ptr^.prev = item
```

Stacks

- A stack is a data structure which holds multiple elements of a single type
- Elements can be removed from a stack only in the reverse order to that in which they were inserted (LIFO, Last In First Out)
- A stack can be implemented with an array and an integer counter to indicate the current number of elements in the stack

Stacks

- E.g.

```
stack: array[1..50] of integer
 ctr: integer
 ctr = 0
```

Stacks

- To put an element on the stack

```
procedure push(elt)
   ctr = ctr + 1
   stack[ctr] = elt
```

- To remove an element from the stack

```
procedure pop(elt)
   if ctr = 0 then
         elt = nil
   else
         elt=stack[ctr]
         ctr = ctr - 1
   fi
```

Queues

- A queue is a data structure which holds multiple elements of a single type
- Elements can be removed from a queue only in the order in which they were inserted (FIFO, First In First Out)
- A queue can be implemented with an array and two integer counter to indicate the current start and next insertion positions

Queues

- E.g.

```
queue: array[1..50] of integer
  start: integer
  next: integer
  start = 1; next = 1
```

Queues

- To put an element in the queue

```
procedure enqueue(elt)
   queue[next] = elt
   next = next+1
   if next > 50 then next = 1
```

- To take an element out of the queue

```
procedure dequeue(elt)
   if start = next then
         elt = nil
   else
         elt = queue[start]
   fi
   start = start +1
   if start > 50 then start = 1
```

Records (Structures)

- A record is a data structure consisting of a fixed number of items
- Unlike an array, the elements in a record may be of differing types and are named.
- E.g.

```
type person = record
   name: string
   age: integer
   height: real
   female: Boolean
   children: array[1:10] of string
```

Records

- An array may appear as a field in a record
- Records may appear as elements of an array
- E.g.

```
staff: array[1..50] of person
```

- Records are typically addressed by a pointer
- E.g. type boss = ^person declares boss to be a pointer to records of type person
- Fields of a record are accessible via the field name
- E.g.

```
staff[5].age, boss^.name
```