CRICOS PROVIDER 00123M

School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Stack – Queue - Linked List

adelaide.edu.au

seek LIGHT

Stacks- review

- LIFO
- Different implementations
  - Linked list implementation
  - Array implementation
- Both implementations can guarantee O(1) complexity for the basic operations.
  - Push
  - Pop
  - IsEmpty

Applications of Stacks

- Tracking function calls
- Postfix Expressions – find the result
  - Put input numbers in stack as you reach them
  - When you reach an operation in input string,
    - If there are less than the number of operands of that operation in stack, then error
    - Else, apply the operation on them (top two for example) and push the result in stack
  - When you reach the end of input string,
    - If there are more than one elements in the stack then error
    - Else return the top of the stack

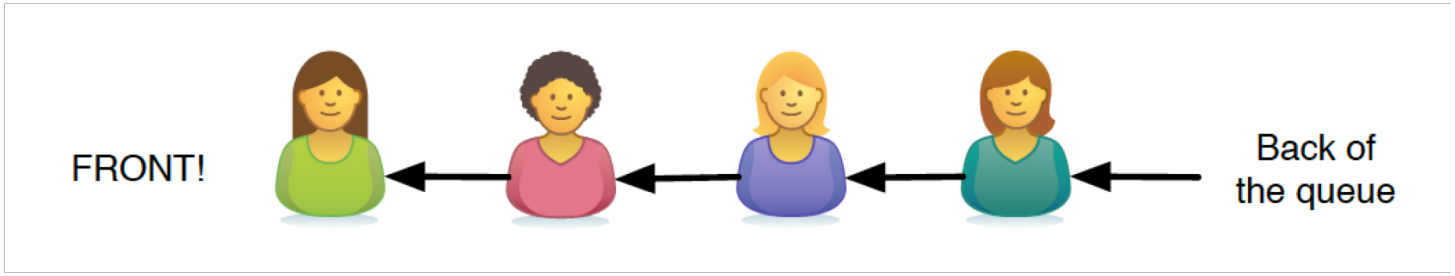
  Optional practical assignment for pre-fix expressions
- Balancing Symbols

Balancing brackets pseudo-code

```
Create an empty stack
While nextchar!= null
        If nextChar is left bracket
                Push nextChar on stack
        If nextChar is right bracket
                if stack is empty
                    error
                else
                    char= pop stack
                    If nextChar is the right counterpart of char
                            Nothing
                    Else
                            Error
If stack has element(s) left in it
        Error
```

Queue

- A queue is a data structure that retrieves data in the same order in which it was stored.
- You have access to the front of the queue, to remove things, and the back of the queue, to add things.
- This is called First-In/First-Out (FIFO).
- Think about the lineup at a coffee shop. You join the queue from the end of the queue and are served from the front of the queue.



Queue operations

- The operations associated with a queue are:
  - add - we add an element to the back of the queue
  - remove - we take the front element from the queue and return its value
  - empty - we return true if the queue is empty, false otherwise
- What is the precondition for remove?

Queue Implementations

- Queues are very easy to implement in linked lists
  - Add adds a node to the end of the queue. This means that you will need a **end** pointer!
    - If the list is initially empty, then both end and front need an update otherwise, just the end pointer should be updated
  - Remove removes the node at the front, returns the value and destroys the old node, updating **front** to point to the new head.
    - **Check first if it is empty. Moreover, if the list becomes empty, end pointer should also be updated.**
  - Empty checks to see if **end** or **front** point to NULL.
- Queues can also be implemented using an array
  - A circular array implementation
  - **We need to keep size, as well as start and end pointers**
- Basic operations take O(1) for both linked list and array

Notes for queues

- Black box
  - Inside, we may have a linked list or an array
  - While you have access to the whole chain, the functions that you use in this data structure restrict you to only accessing certain elements.
- This enforces the FIFO semantics of the data structure and this allows you to write your code knowing that this will be enforced.

But why?

- An obvious question is why we go to such great lengths to produce so many different types of lists.

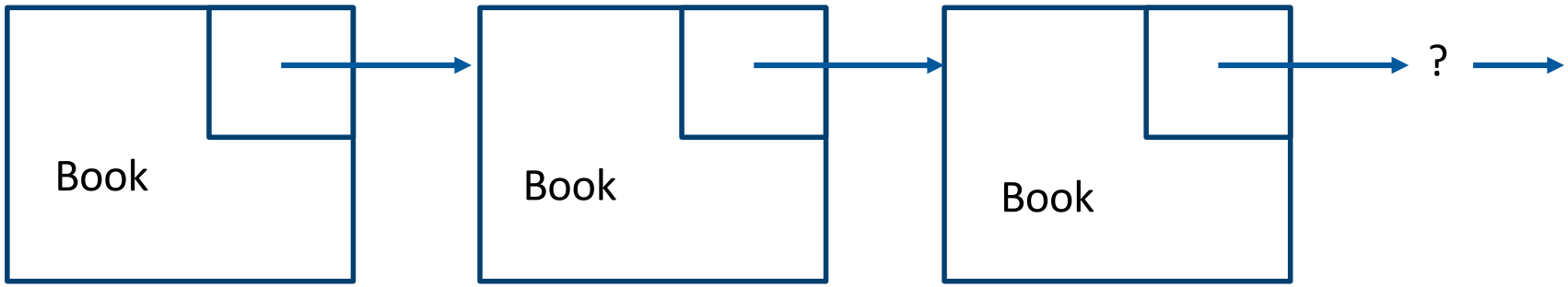
Suitable Abstractions Assist Programming

- Using the right ADT at the right time enforces correct behavior.
  - It's important that things don't jump the queue.
  - It's important that activation records don't get out of order in the stack.

- Applying an existing, well-understood solution to a problem:
  - Saves you time
  - Lowers the risk of incorrect code
  - Reduces the programmer's burden

Review on Linked List

- A linked list is a data structure that is composed of objects that have:
  - values
  - a pointer to another object of the linked list type
- Not in consecutive places in the memory

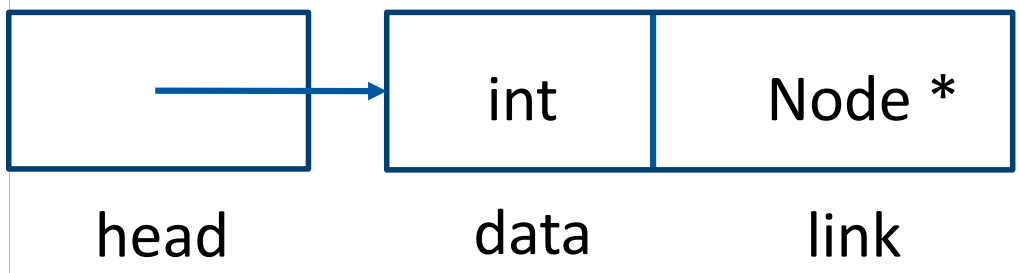


Nodes

```
struct Node {
   int data;
   Node *link;
}
```

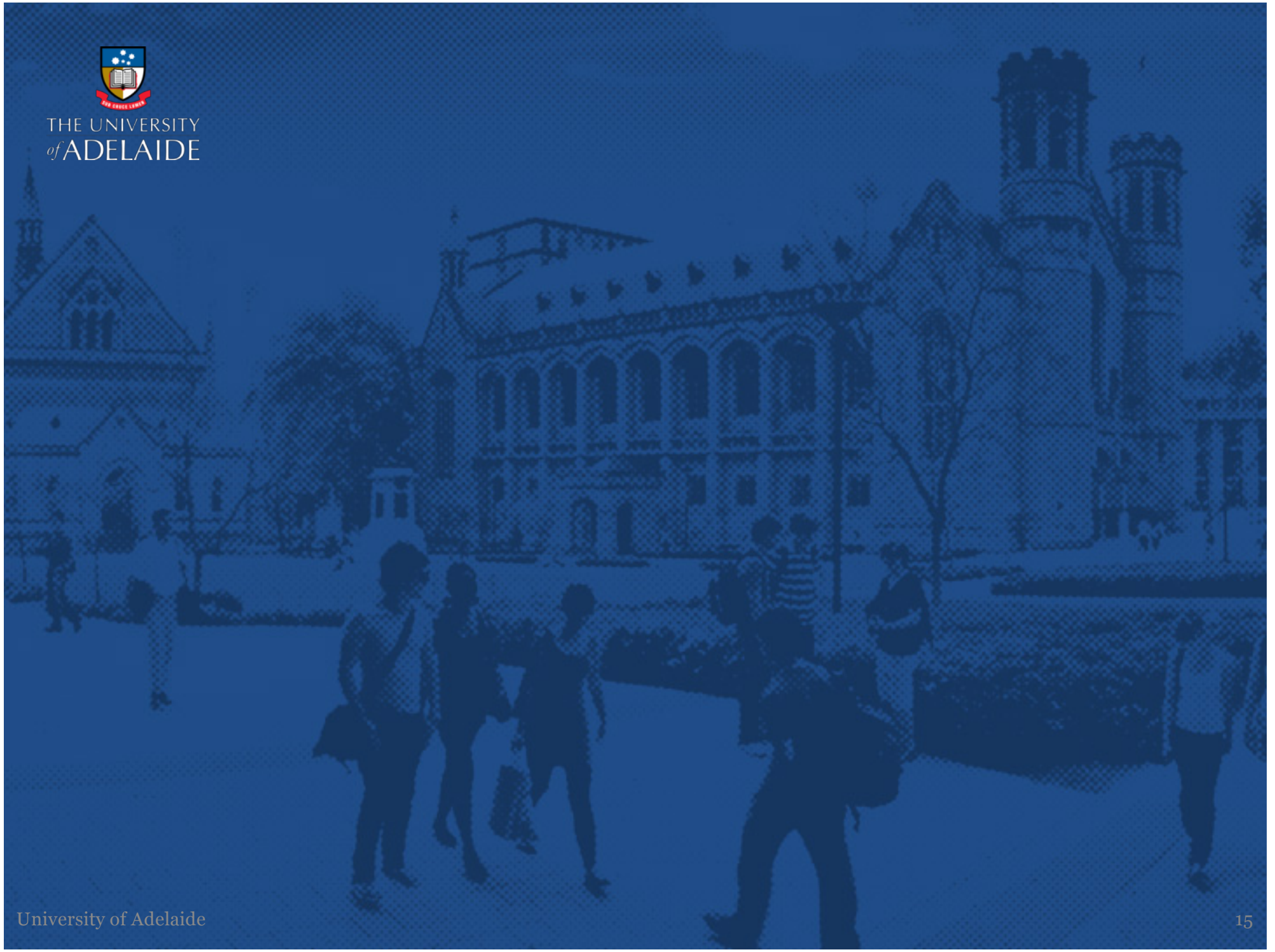
Keeping Track of List

```
Node *head;
head = new Node;
head->data = 3;
head->link = NULL;
```



Linked Lists as ADTs

- Lower level ADT than List, with many applications
- What functions should be provided?
  - InsertFront
  - InsertMiddle
  - RemoveFront
  - RemoveMiddle
  - Search
  - GetItem
  - Traverse the whole list (like print)
- Nodes are taken from the heap
- Don’t forget to delete them when you are done with them
  - destructor
- Compare to arrays and dynamic arrays


THE UNIVERSITY
of ADELAIDE