Data Structures

Ahsan Ijaz

Data Structures

A data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways.

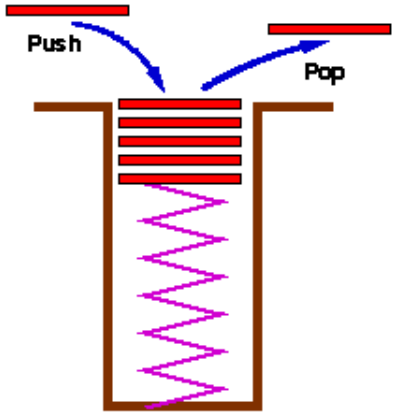
- **Fixed-size** data structures have fixed size assigned to them at compile time one-dimensional arrays,two-dimensional arrays and structs.
- **Dynamic data structures** grow and shrink during execution.

Dynamic Data Structures

- Linked Lists
- Stacks
- Queues
- Trees

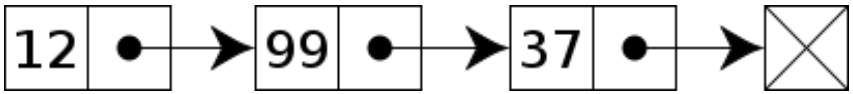
Stacks

- Stacks are important in compilers and operating systems: Insertions and removals are made only at one end of a stack, its top.



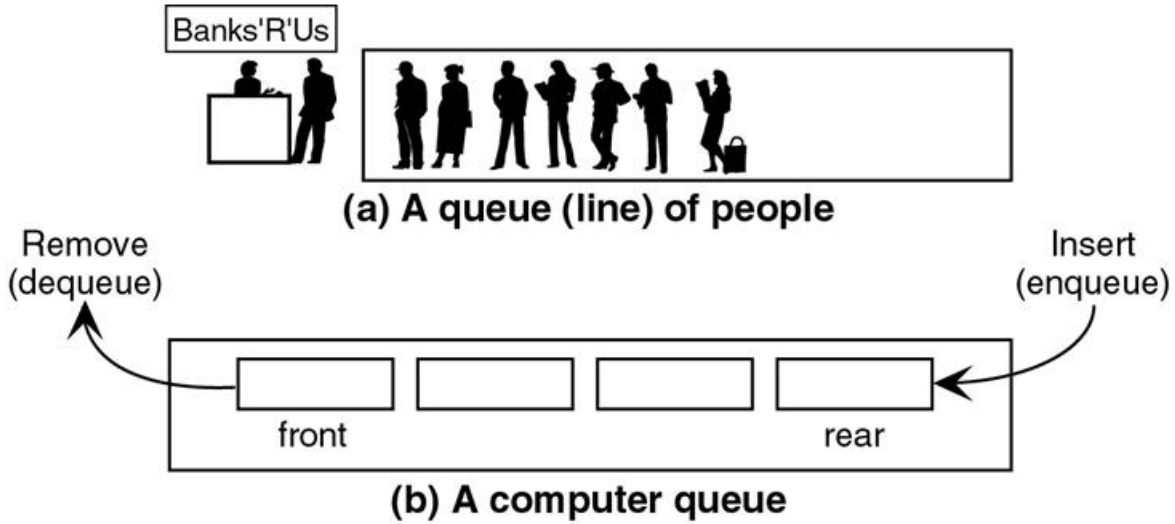
Linked Lists

- Linked lists are collections of data items lined up in a row. Insertions and removals can be made anywhere in a linked list.



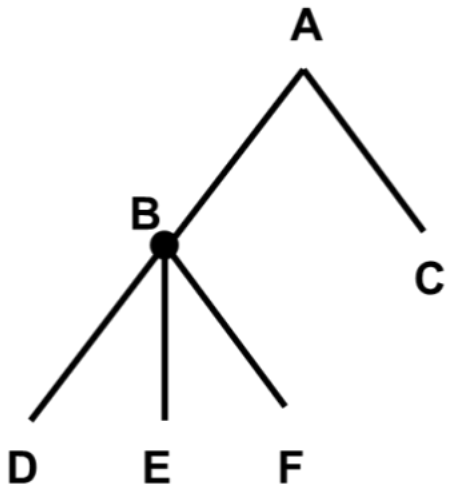
Queues

- Queues represent waiting lines; insertions are made at the back (also referred to as the tail) of a queue and removals are made from the front (also referred to as the head) of a queue. This makes the queue *aFirst-In-First-Out (FIFO)* data structure.



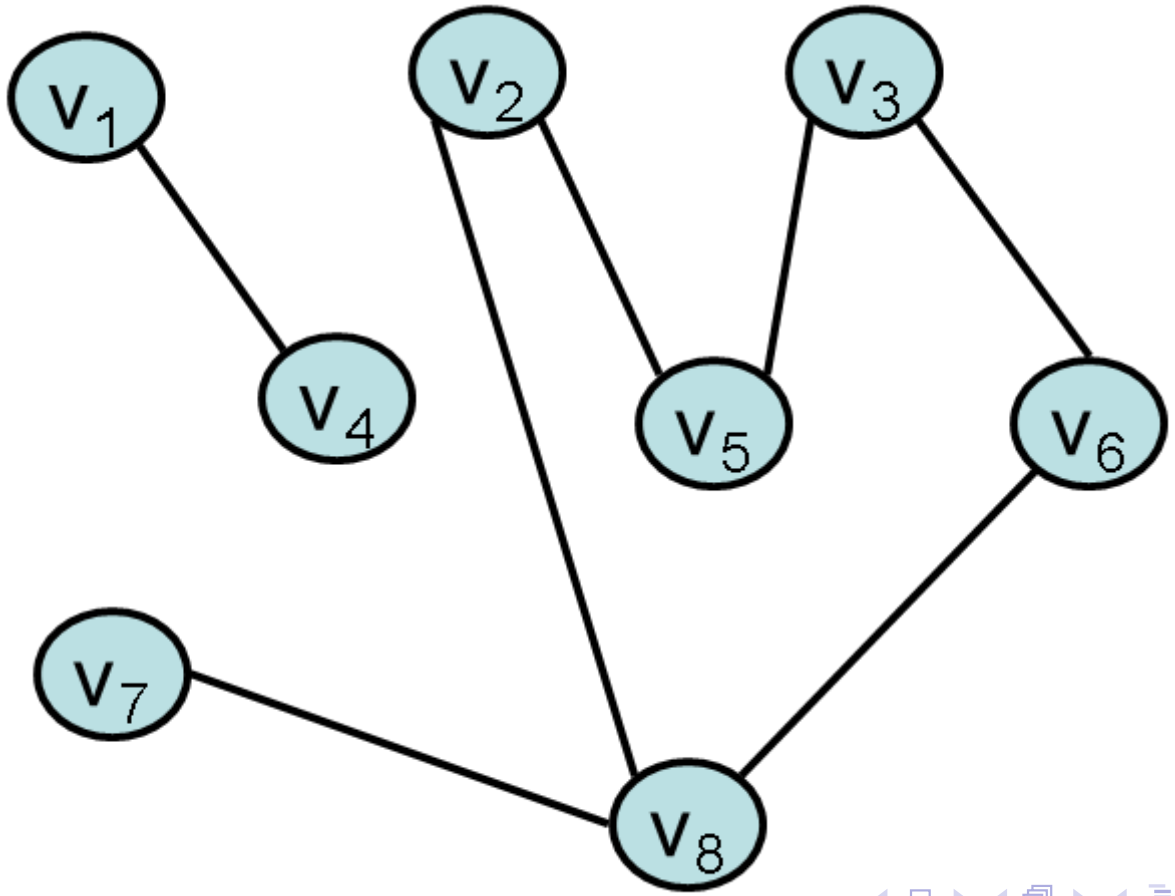
Trees

- Trees facilitate high-speed searching and sorting of data and efficient elimination of duplicate data items. It imitates a hierarchical tree structure set of linked nodes which are ordered and directed and each node has zero or more children.



Graph

- A graph data structure consists of a finite set of ordered pairs, called edges, of certain entities called nodes or vertices.



Self Referential Classes

- A self-referential class contains a pointer member that points to a class object of the same class type.

Class Declaration

```
1  class Node
2  {
3  public:
4
5      Node( int ); // constructor
6      void setData( int ); // set data member
7      int getData() const; // get data member
8      void setNextPtr( Node * ); // set pointer to next Node
9      Node *getNextPtr() const; // get pointer to next Node
10
11  private:
12
13     int data; // data stored in this Node
14     Node *nextPtr; // pointer to another object of same type
15
16 }; // end class Node
```

Self Referential Classes

- A class Node has two private data members; **integer member data** and **pointer member nextPtr**.
- Member nextPtr points to an object of type **Node**.
  - It is another object of the same type as the one being declared here, hence the term "self-referential class.
- Member **nextPtr** is referred to as a **link**.
  - i.e. **nextPtr** can "tie" an object of type Node to another object of the same type.

Self Referential Classes

- For a linked data structure, the last node in the link must be set to null.
- Self-referential class objects can be linked together to form useful data structures such as lists, queues, stacks and trees.

Self Referential Classes

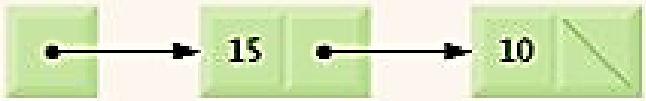


Figure: two self-referential class objects linked together to form a list.

- The slash represents a null (0) pointer placed in the link member of the second self-referential class object to indicate that the link does not point to another object.
- A null pointer indicates the end of a data structure.

Dynamic Memory Allocation and Data Structures

- Creating and maintaining dynamic data structures requires *dynamic memory allocation*.
- When that memory is no longer needed by the program, it can be released so that it can be reused to allocate other objects in the future.