## Linked lists as a class

- based on our earlier struct implementation
- data fields would be front, back
- some methods would be public-facing, e.g. insert, remove, print, lookup
- other methods may be private/internal, e.g. the search that returned a pointer
- nodes within the list could be represented either as another class or as a simple struct container (we'll take the latter approach, the actual node definition and access methods can be hidden within the class)

## class definition: data portion

```
class dllist {
   // will keep the nodes and admin info private
   private:
      // define the structure of the internal nodes
      struct node {
           int somedata;
           float moredata;
           node* next;
           node* prev;
      };

      // list's front/back will simply be pointers to the first/last nodes
      node* front;
      node* back;
```

## class def cont: methods

```
   private:
      node* create(int sd, float md); // create/initialize a new internal node
      node* search(int sd); // find node with given somedata field, return ptr to it

   public:
      dllist(); // constructor, initializes list as empty
      ~dllist(); // destructor, deletes any remaining nodes in list

      // a few example methods
      bool insertBack(int sd, float md); // create new node, insert at back
      void print(); // display current list content
      bool lookup(int sd, float &md); search for sd in somedata, get it's moredata
};
```

## methods, constructor

- method name specifies class & method
- constructors/destructors have no return type
- all fields are directly accessible inside the class methods
- if data type is declared *inside* class (e.g. node) then use the classname::typename format when referring to it from *outside*

```
// constructor, initialize list as empty
dllist::dllist() {
   front = NULL;
   back = NULL;
}
```

# methods: destructor

```
// destructor: delete any remaining nodes in the list
dllist::dllist()
{
    node* curr = front;
    while (curr != NULL) {
        node* target = curr;
        curr = curr->next;
        delete target;
    }
}
```

## create method

```
// create method's return type is node*, but since the return types appear “outside”
// the method but node is defined inside the class we use dllist::node*

dllist::node* dllist::create(int sd, float md)
{
   node* n = new node;
   if (n != NULL) {
      n->somedata = sd;
      n->moredata = md;
      n->next = NULL;
      n->prev = NULL;
   }
   return n;
}
```

## search method (internal)

```
dllist::node* search(int sd)
{
    node* n = front;
    while (n != NULL) {
        if (n->somedata == sd) {
            return n;
        }
        n = n->next;
    }
    return NULL;
}
```

## method to insert at back

```
bool dllist::insertBack(int sd, float md)
{
   node* n = create(sd, md);
   if (n == NULL) {
      return false;
   }
   if (front == NULL) {
      front = n;
      back = n;
   } else {
      back->next = n;
      n->prev = back;
      back = n;
   }
   return true;
}
```

## lookup method

```
bool dllist::lookup(int sd, float &md)
{
   // use the internal search routine we already created
   node* n = search(sd);
   if (n == NULL) {
      return false; // not found
   }
   md = n->moredata;
   return true;
}
```