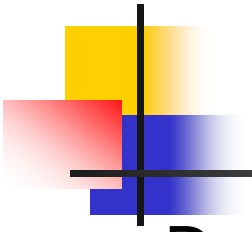
Linked Lists



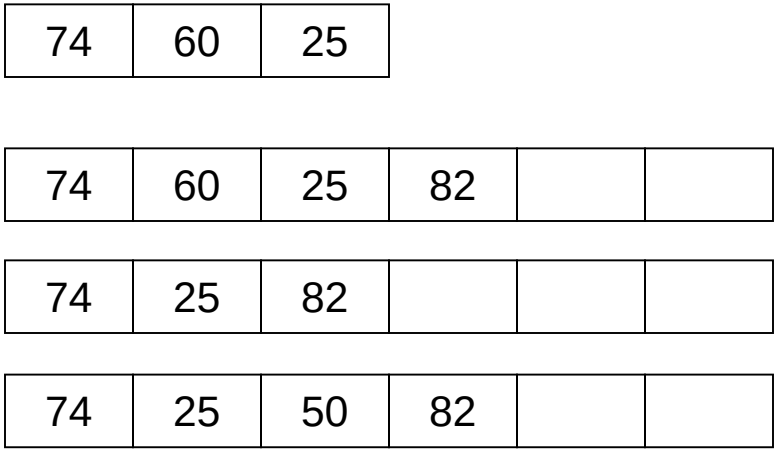
- Dynamic Arrays vs Linked Lists
- Node Class for Linked List
- Singly Linked List
  - Linked List Toolkit
- Circular Linked List
- Node Class for Doubly Linked List
- Doubly Linked List
- BagList Class
- Dynamic Arrays vs Linked Lists vs Doubly Linked Lists
- STL Vector vs STL List

Dynamic Arrays

- Arrays have contiguous storage of a fixed size that doesn't respond well to changes such as:
  - additions/deletions
    - requires shifting of element values
    - may require array resizing
- Example:

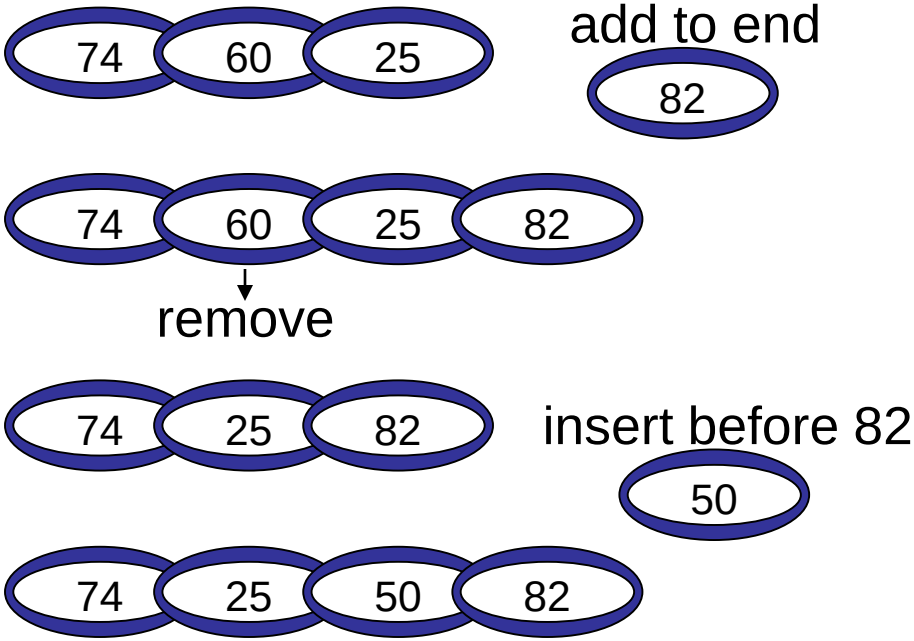
```
int *myArray;
myArray = new int [3];
myArray[0] = 74;
myArray[1] = 60;
myArray[2] = 25;
```

  - add 82 to end of array
    - resize array
  - remove 60
    - shift elements
  - insert 50 before 82
    - shift elements



Linked Lists

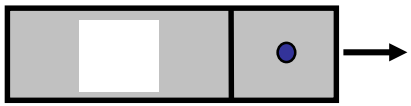
- Linked List is a dynamic data structure that can grow and shrink as program executes
  - adding/inserting elements
  - deleting elements



Node Class for Linked List

- Nodes are the independent items in a linked list
  - data field
    - typedef/alias allows item of ANY defined type
  - pointer
    - connects to adjacent item in list
    - also called a link

```
class node {
public:
  using value_type = double;
private:
  value_type data_field;
  node *link_field;
};
```



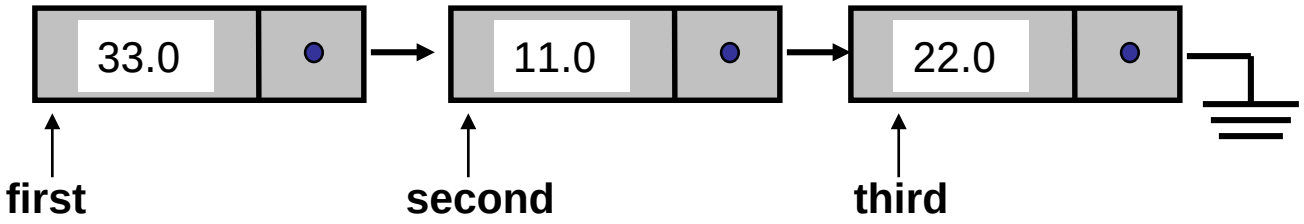
Node Class for Linked List

- Node class methods
  - Parameter constructor with defaults
  - Field mutators (i.e. *setters*)
  - Field accessors (i.e. *getters*)
    - Constant and non-constant link accessors

```
// constant pointer cannot change referenced node
const node* link() const { return link_field; }
// pointer can change referenced node
node* link() { return link_field; }
```

Using node Objects

- Creating node objects for linked list

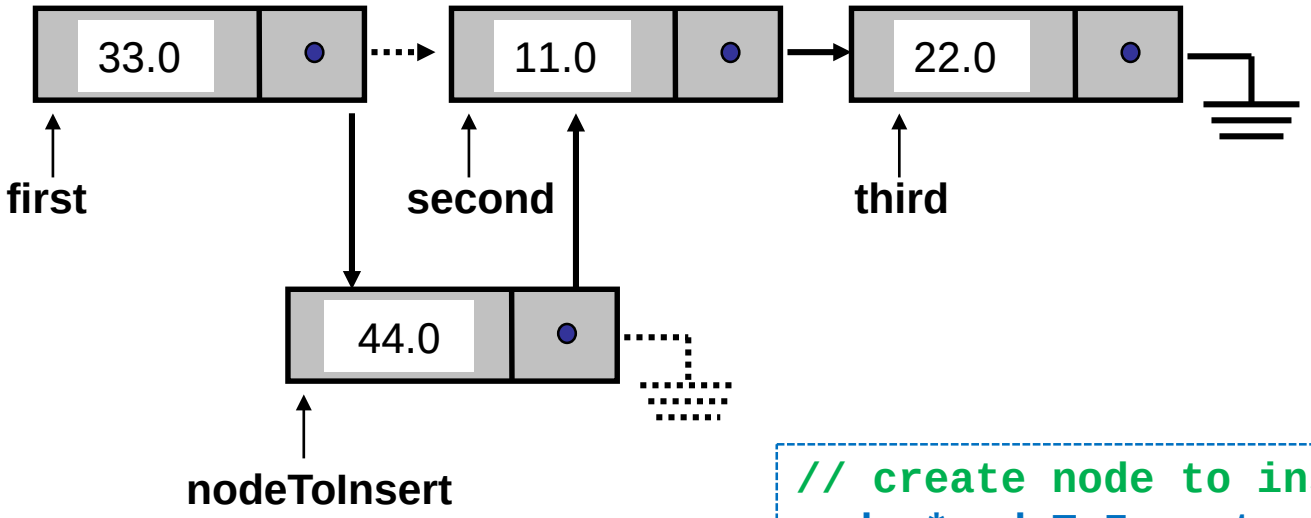



```
// create default node
node *empty = new node();
// create node with data
node *third = new node(22.0);
// create node with data and link
node *second = new node(11.0, third);
node *first = new node(33.0, second);
```

don‘t forget to release memory!

Using node Objects

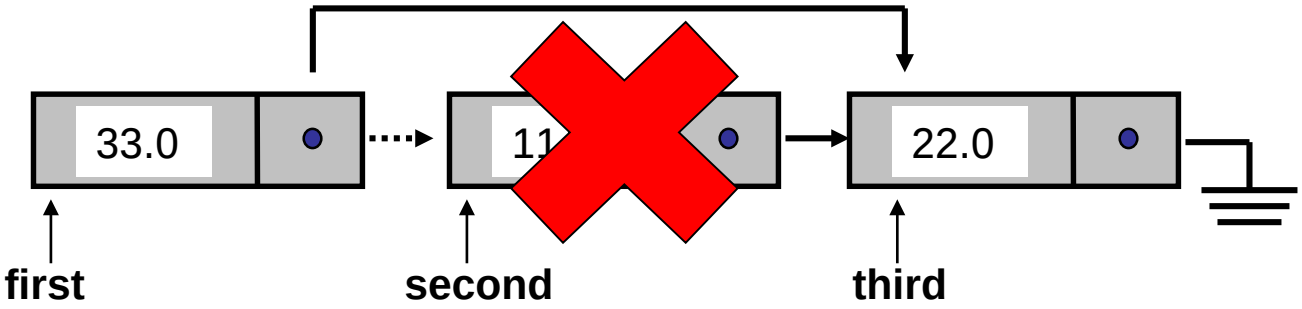
- Inserting node object in linked list




```
// create node to insert
node *nodeToInsert = new node(44.0);
// update pointers
nodeToInsert->set_link(second);
first->set_link(nodeToInsert);
```

Using node Objects

- Removing node object from linked list

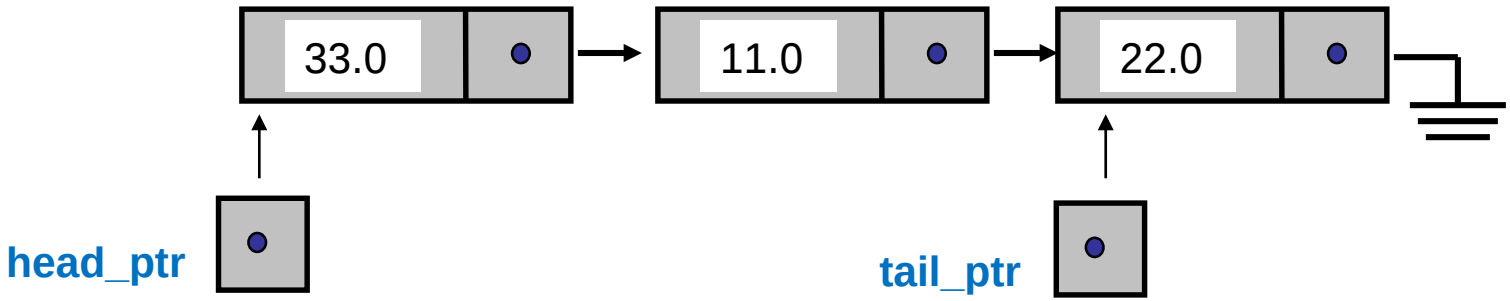



```
// update pointers
first->set_link(third);
// release memory
delete second;
```

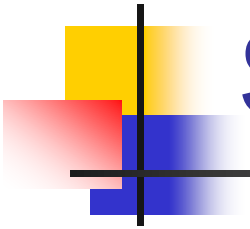
Singly Linked List

- Linked list consists of node pointers
  - Head → points to first node in list
    - aka head pointer
  - Tail → points to last node in list
    - aka tail pointer
  - nullptr → null pointer constant
    - implicitly convertible and comparable to any pointer type

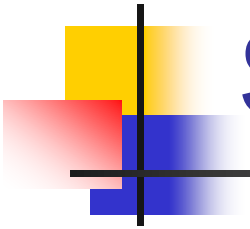
```
node *head_ptr;
node *tail_ptr;
```



Singly Linked List Toolkit



- Extra functions needed with linked list over array for accessing items
- Included functions:
  - list_length → number of nodes in list
  - list_head_insert → insert data at beginning
  - list_insert → insert data before given node
  - list_search → search list for given data and return first node pointer to found data or nullptr
    - return constant node pointer
    - return non constant node pointer

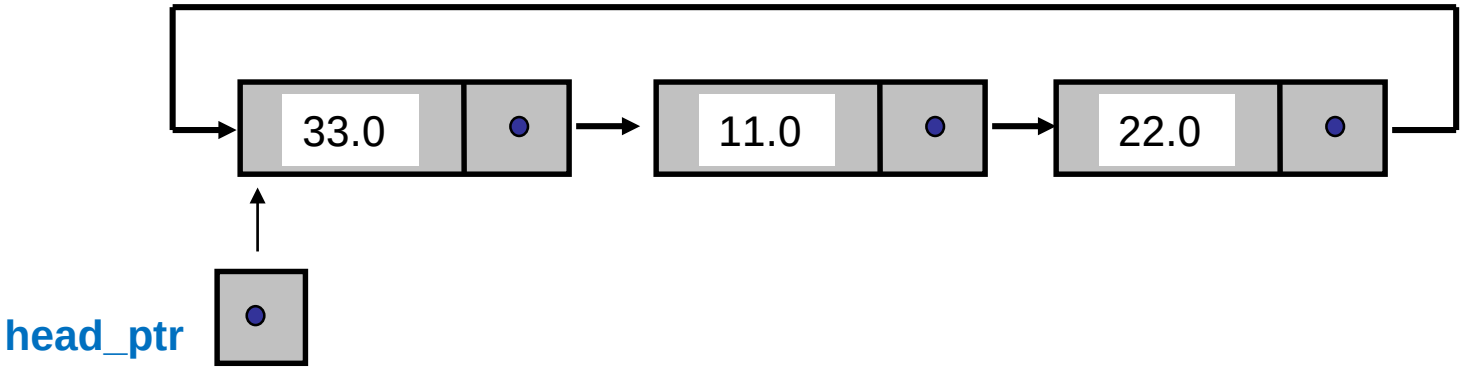


Singly Linked List Toolkit

- Included functions:
  - list_locate → locate node at given position (1, 2, ...) or nullptr
    - return constant node pointer
    - return non constant node pointer
  - list_head_remove → remove node at beginning
  - list_remove → remove node linked to given node pointer
  - list_clear → release memory for all nodes in list
  - list_copy → copy all nodes from source list

Circular Linked List

- All nodes in linked list are connected to form a circle



Node Class for Doubly Linked List

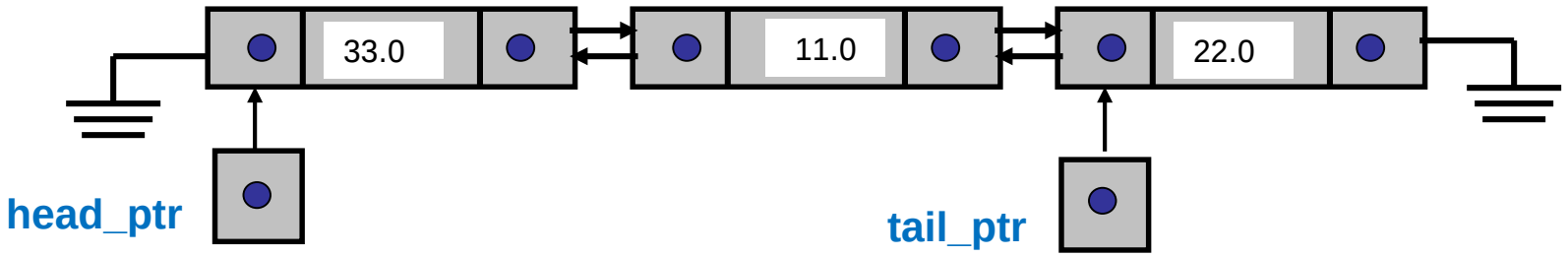
- Node for doubly linked list contains two pointers
  - Next node
  - Previous node

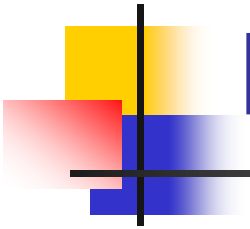
```
class dnode {
public:
  using value_type = double;
private:
  value_type data_field;
  dnode *link_fore;
  dnode *link_back;
};
```

Doubly Linked List

- Cursor/Pointer can move forward and backward through list

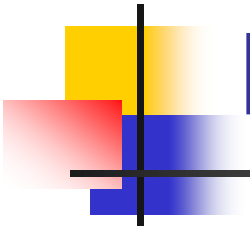
```
dnode *head_ptr;
dnode *tail_ptr;
```





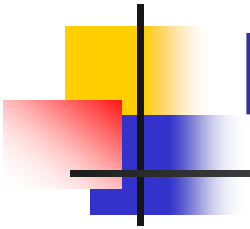
BagList Class

- Rules for implementation
  - Items stored in linked list
  - First node in list is stored in member variable **`head_ptr`**
  - Total number of items stored in list stored in member variable **`many_nodes`**
- Ensure that node value_type matches bagList value_type
  - **`using value_type = node::value_type;`**



BagList Class

- Member functions
  - default constructor → create empty bag
  - copy constructor → create bag copied from source
  - destructor → clear list of all nodes
  - size() → return number of items in bag
  - count → count number of item occurrences
  - insert → insert an item



BagList Class

- erase_one → remove an item, if found
- erase → remove all items, return count of items erased
- grab → return a random item from bag
- += → copy items from source bag to current
- = → reset current bag to source

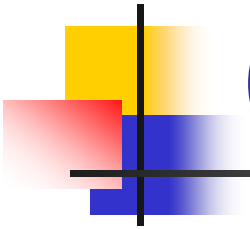
- Non-member function
  - + → create new **`bagList`** object from two added **`bagList`** objects

Dynamic Arrays vs Linked Lists vs Doubly Linked Lists

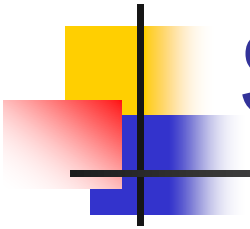
- Guidelines for Choosing Between Dynamic Array and Linked Lists

Operation	Recommendation
Frequent random access operations	Dynamic array
Operations occur at a single direction cursor/pointer	Linked list
Operations occur at a two way direction cursor/pointer	Doubly linked list
Frequent container resizing	Linked list

Standard Template Library (STL)



- Software library of common C++ classes which began as a generic programming initiative first released by HP in 1994
- STL includes
  - Containers
  - Iterators
  - Algorithms
  - Functions



STL Vector vs STL List

- STL Vector class
  - Sequence container that changes in size
  - Internally uses dynamically allocated array
  - Consumes *more* memory than array due to resizing ability
- STL List class
  - Sequence container that changes in size
  - Internally uses doubly linked list
    - Constant time insert and delete operations anywhere within sequence
    - Iteration possible in both directions