# Array Lists

## Lecture 15

Robb T. Koether

Hampden-Sydney College

Fri, Feb 16, 2018

1. Inlining Functions
2. List Implementations
3. Array Lists
4. Circular Array Lists

# Outline


1. Inlining Functions
2. List Implementations
3. Array Lists
4. Circular Array Lists

# Inlining Functions

## Definition (Inline Functions)

An inline function is copied and pasted, with appropriate modifications, into the calling function, thereby replacing a function call.

- A function may be inlined by using the **`inline`** keyword.

```
inline sqr(int n);
```

# Inlining Functions

## Inlining Functions

```
inline int sqr(int n)
{
    return n*n;
}
```

- For example,

  ```
  int s = sqr(2*a + b);
  ```

  is replaced with

  ```
  int s = (2*a + b)*(2*a + b);
  ```

# Inlining Functions

- What about

```
int s = sqr(sqr(2*a + b) + 5);
```

# Inlining Functions

- What about

```
int s = sqr(sqr(2*a + b) + 5);
```

- See inline depth.

# Outline


1. Inlining Functions
2. List Implementations
3. Array Lists
4. Circular Array Lists

# List Implementation

- We will implement lists in a number of ways.
- As an array.
  - Fixed head.
  - Circular.
- As a linked list.
  - Singly linked.
  - Doubly linked.
  - Circularly linked.
  - Recursively linked.

# Outline


1. Inlining Functions
2. List Implementations
3. Array Lists
4. Circular Array Lists

# The ArrayList Class

**Definition (Array List)**

An array list is an implementation of the List ADT that uses an array to store the list elements.

- The `ArrayList` class is very similar to the `Vectr` class.
- The difference is that the size is adjustable *after* the list has been constructed.

# Data Members

## `ArrayList` Data Members

- **`int`** `m_capacity` - The number of array positions allocated.
- **`int`** `m_size` - The number of elements in the list.
- `T* m_element` - A pointer to the allocated memory.

# The List Elements

- The list elements

$$a_0, \ldots, a_{\texttt{m\_size}-1}$$

  are stored in array positions `m_element[0]` through `m_element[m_size - 1]`.

# Validity Requirements

- The object is structurally valid provided
  - `m_capacity >= 0`.
  - `m_size >= 0` and `m_size <= capacity`.
  - If `m_capacity == 0`, then `m_element == NULL`.
  - If `m_capacity > 0`, then `m_element != NULL`.

# The ArrayList Class

## The `ArrayList` Class

- `arraylist.h`.
- `ListTest.cpp`.

# The `insert()` Function

- The `insert()` function must
  - Test that `pos` is valid.
  - Test that there is sufficient capacity to add one more element. If not, then call `setCapacity()` to double the capacity.
  - Shift the elements with indexes `pos` to `m_size - 1` to the right one position.
  - Then copy `value` to index `pos`.
  - Increment the size of the list.

# The `insert()` Function

## The `insert()` Function

```
template <class T>
void ArrayList<T>::insert(int pos, const T& value)
{
    assert(pos >= 0 && pos <= m_size);
    if (m_size == m_capacity)
        if (m_capacity == 0)
            setCapacity(1);
        else
            setCapacity(2 * m_capacity);
    for (int i = m_size - 1; i >= pos; i--)
        m_element[i + 1] = m_element[i];
    m_element[pos] = value;
    m_size++;
    return;
}
```

# The `remove()` Function

- What should the `remove()` function do?

# Outline


1. Inlining Functions
2. List Implementations
3. Array Lists
4. Circular Array Lists

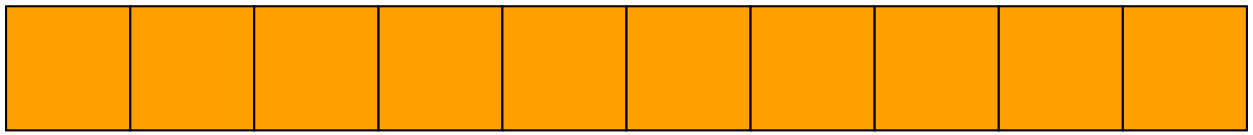
# Circular Array Lists

## Definition (Circular Array List)

A circular array list is like an ordinary array list, except that the elements may wrap around the ends of the array.

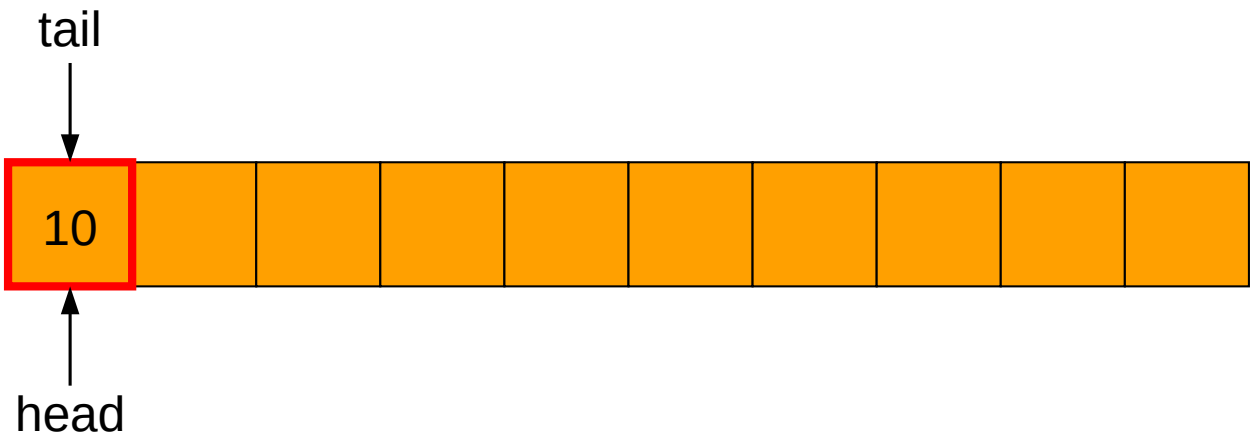
- This makes it much more efficient to add and remove elements from the front end of the list.

# Circular Array Lists Dynamics



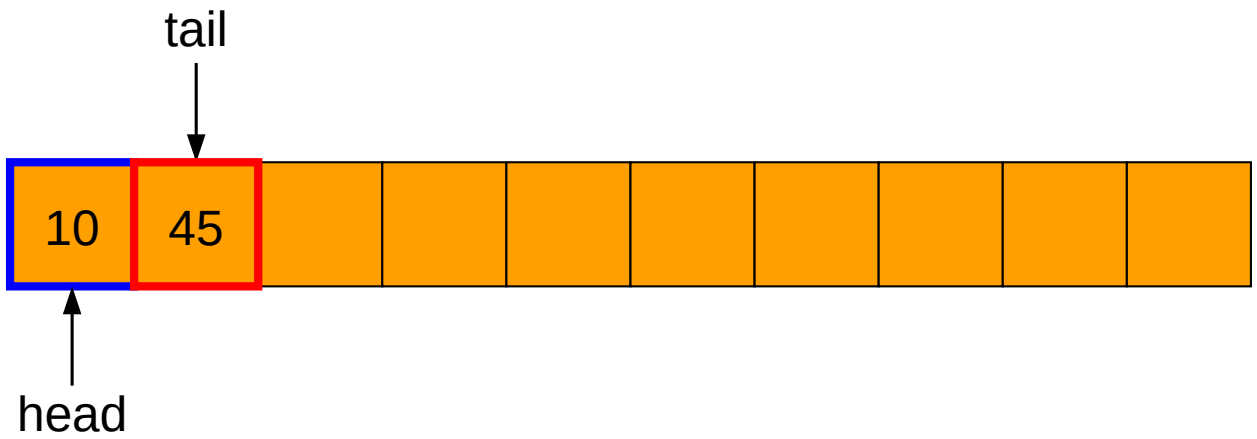
Begin with an empty list

# Circular Array Lists Dynamics



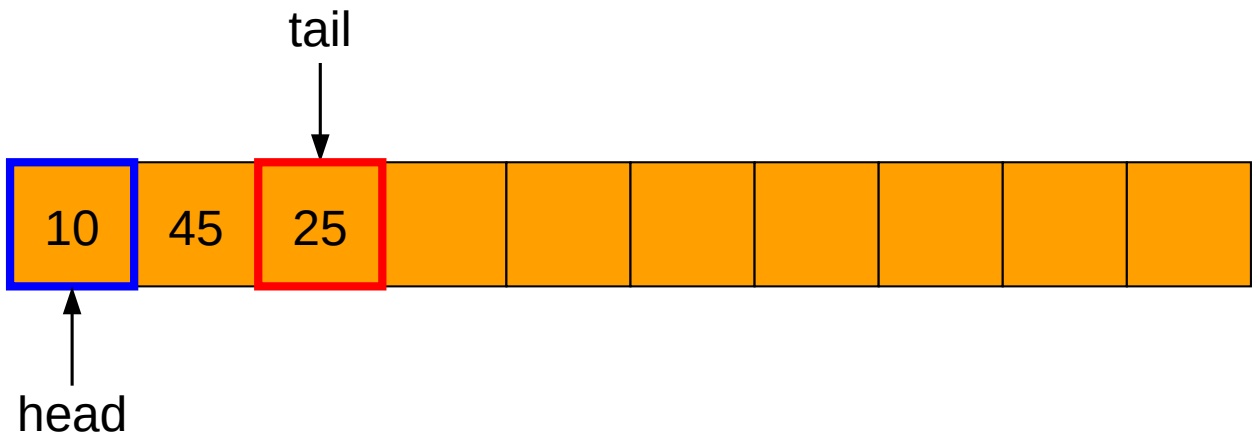

Add six elements at the tail(pushBack)

# Circular Array Lists Dynamics



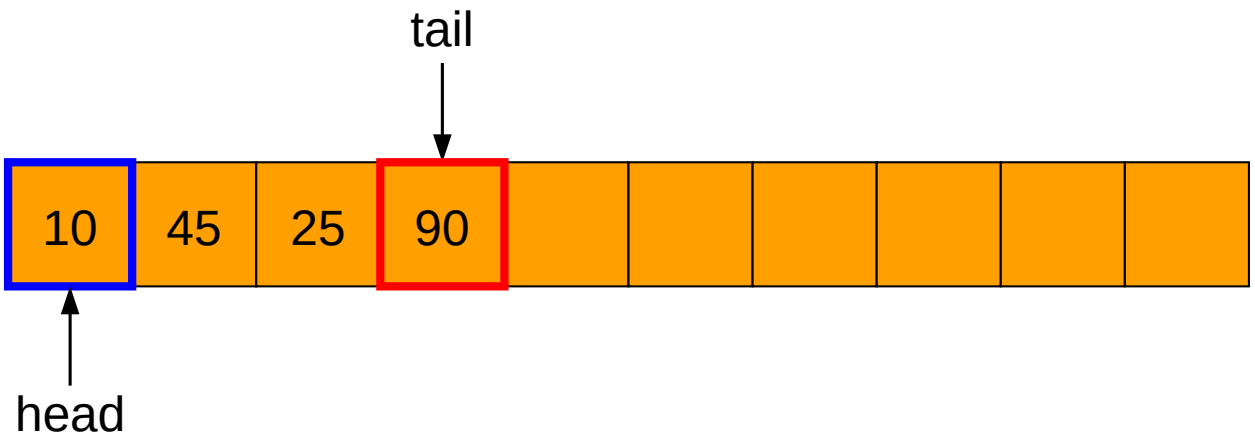

Add six elements at the tail(pushBack)

# Circular Array Lists Dynamics



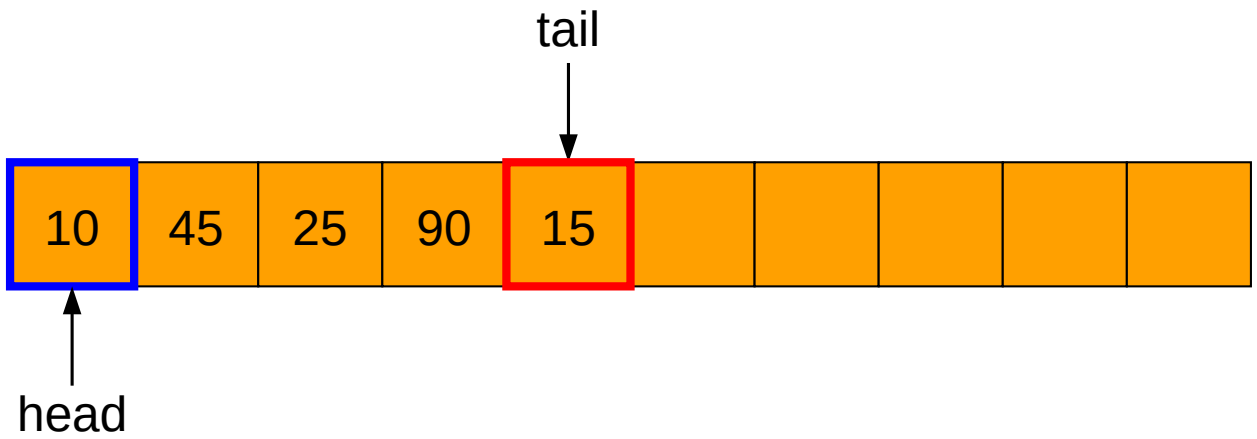

Add six elements at the tail(pushBack)

# Circular Array Lists Dynamics



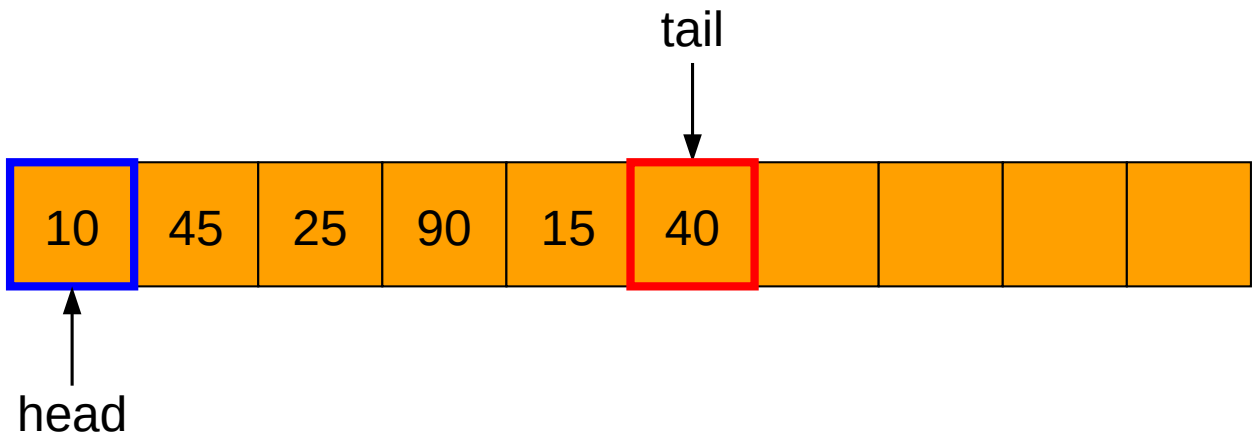

Add six elements at the tail(pushBack)

# Circular Array Lists Dynamics



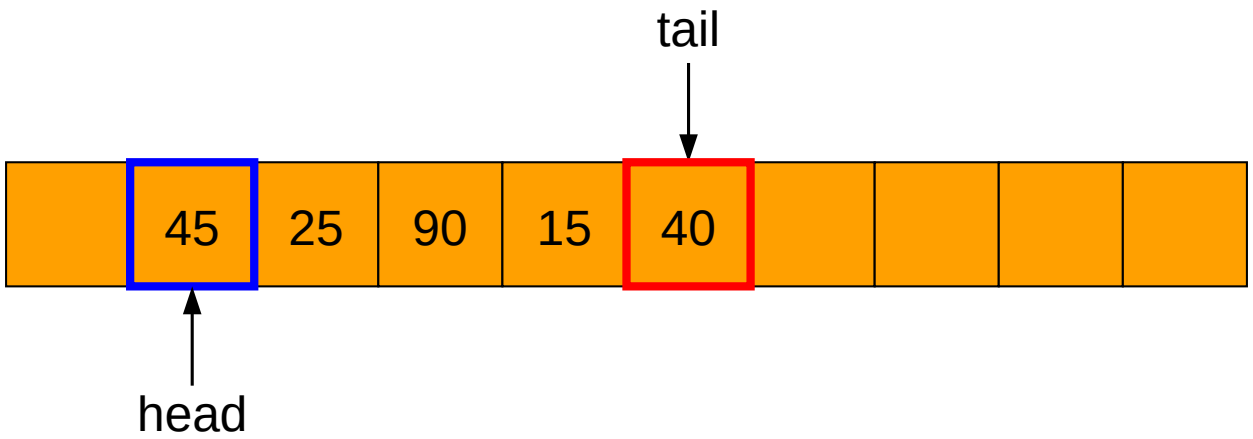

Add six elements at the tail(pushBack)

# Circular Array Lists Dynamics



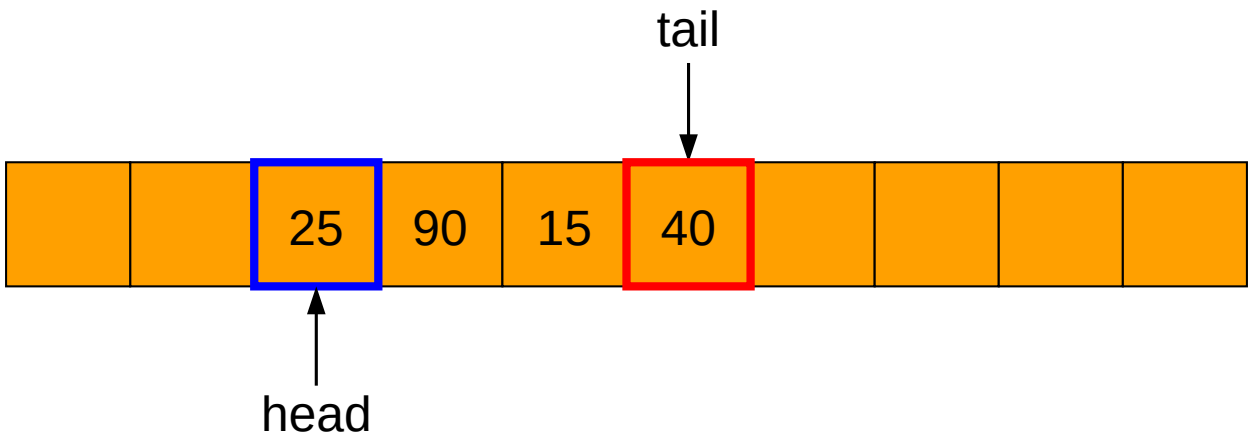

Add six elements at the tail(pushBack)

# Circular Array Lists Dynamics



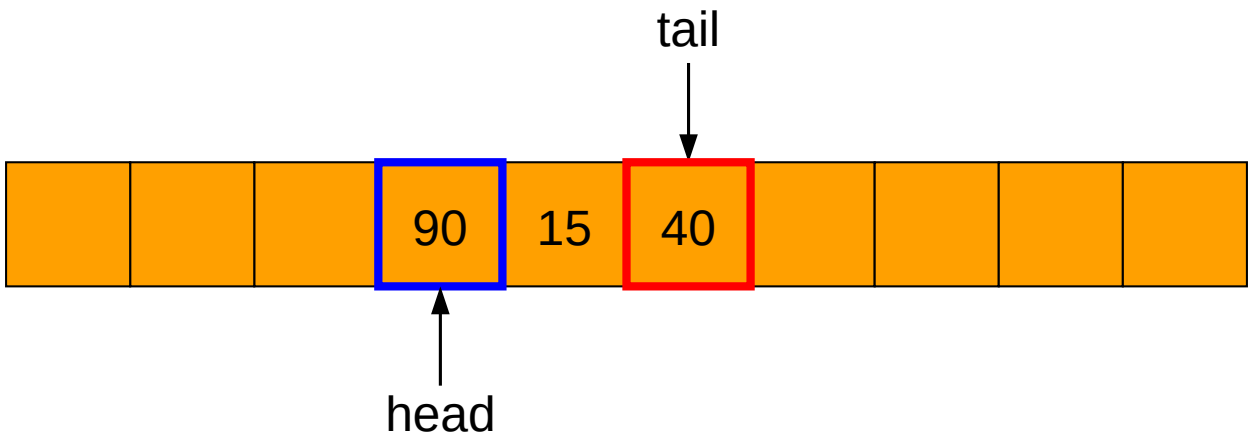

Delete three elements from the head

# Circular Array Lists Dynamics



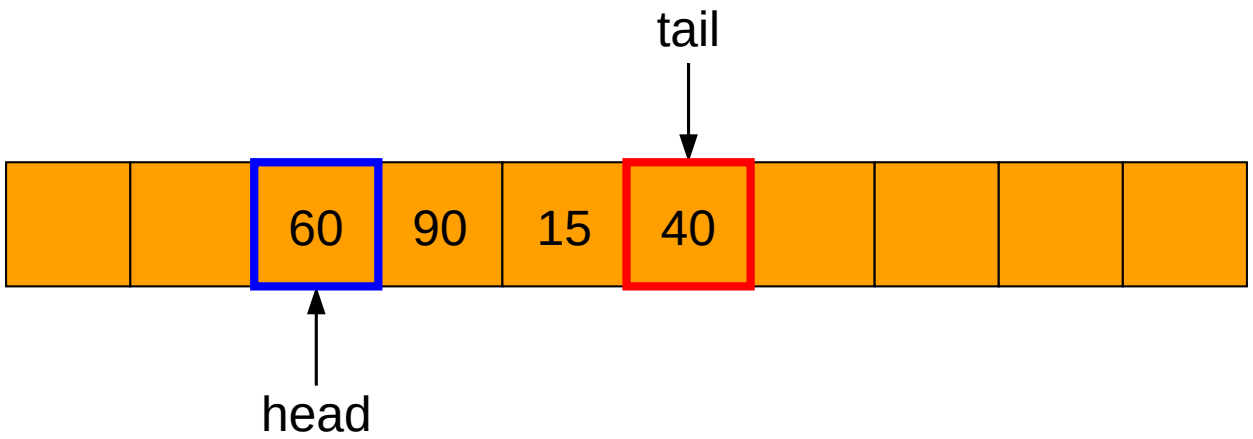

Delete three elements from the head (popFront)

# Circular Array Lists Dynamics



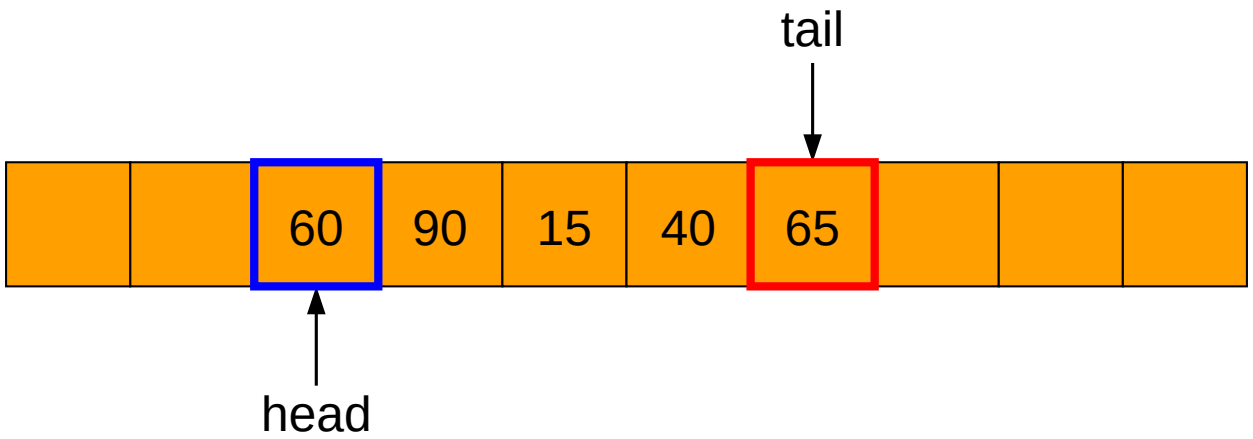

Delete three elements from the head (popFront)

# Circular Array Lists Dynamics



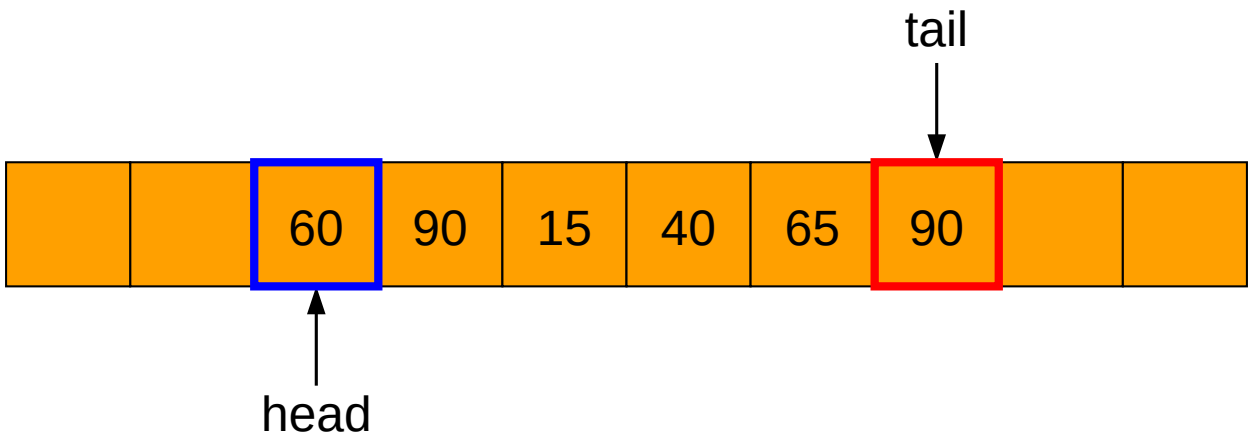

Add one element at the head (pushFront)

# Circular Array Lists Dynamics



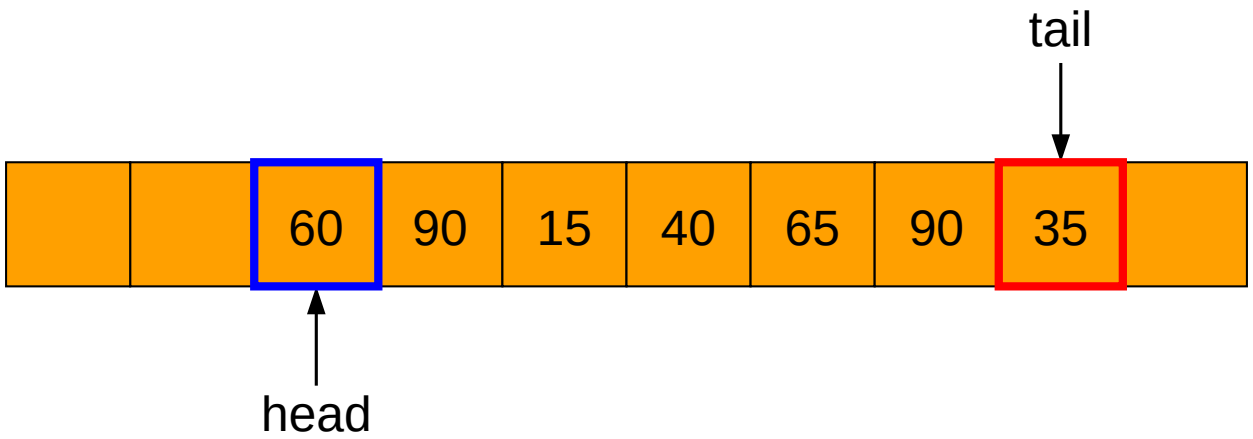

Add six elements at the tail (pushBack)

# Circular Array Lists Dynamics



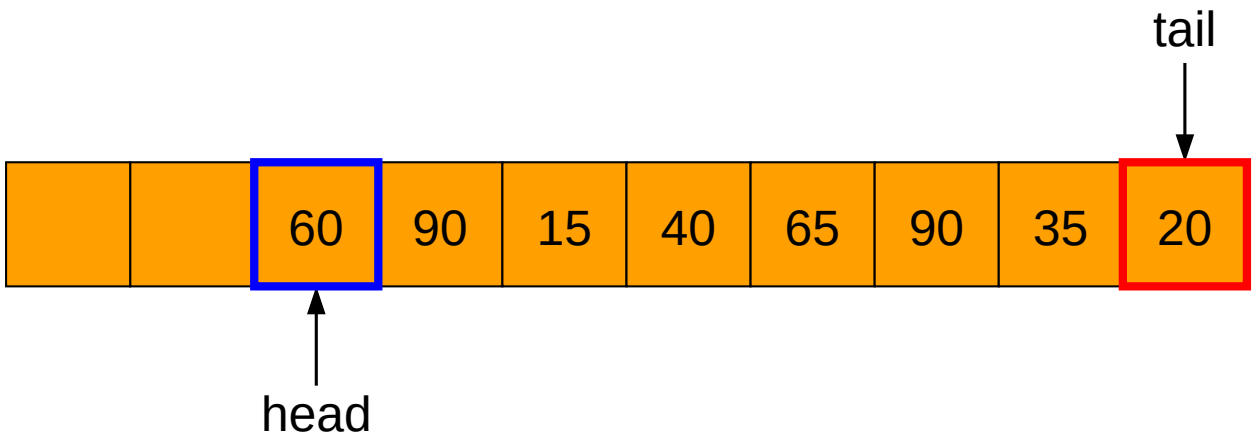

Add six elements at the tail (pushBack)

# Circular Array Lists Dynamics



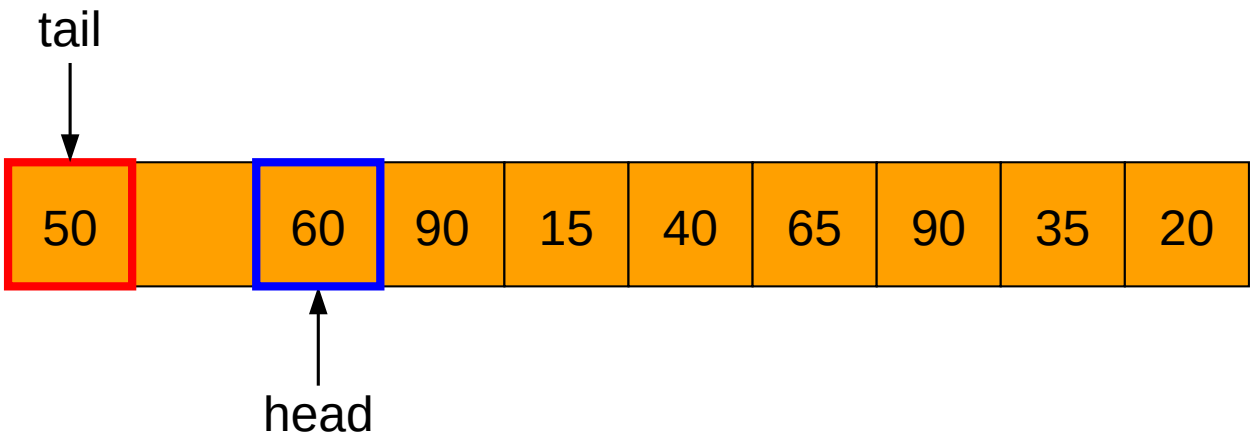

Add six elements at the tail (pushBack)

# Circular Array Lists Dynamics



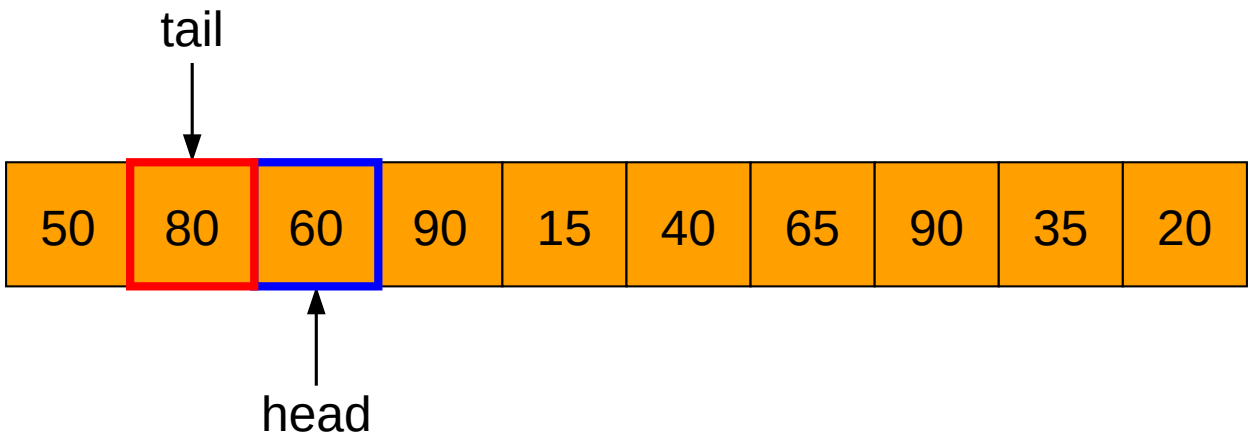

Add six elements at the tail (pushBack)

# Circular Array Lists Dynamics




Add six elements at the tail (pushBack)

# Circular Array Lists Dynamics




Add six elements at the tail (pushBack)

# The ArrayList Class

## The `ArrayList` Class

- `circarraylist.h`.
- `ListTest.cpp`.