# CS302 - Data Structures

## *using C++*

Topic: Using the ADT List

Kostas Alexis

# Using the ADT List

- Items are referenced by
  - Position in list
- Each item has
  - A unique predecessor
  - A unique successor
  - Operations
    - Head (or front) does not have a predecessor
    - Tail (or end) does not have a successor

# Using the ADT List

- Items are referenced by
  - Position in list
- Each item has
  - A unique predecessor
  - A unique successor
  - Operations
    - Head (or front) does not have a predecessor
    - Tail (or end) does not have a successor

- Specifications of ADT operations
  - Define operation contract for the ADT List
  - Do not specify
    - How to store the list
    - How to perform the operations

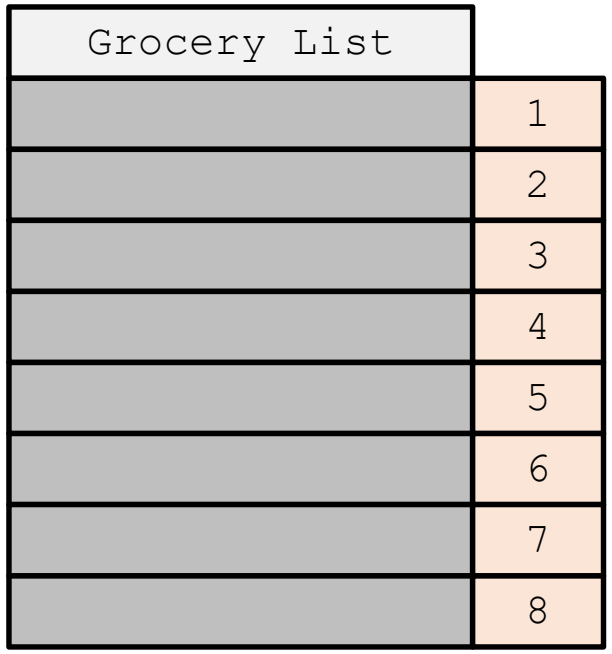
# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

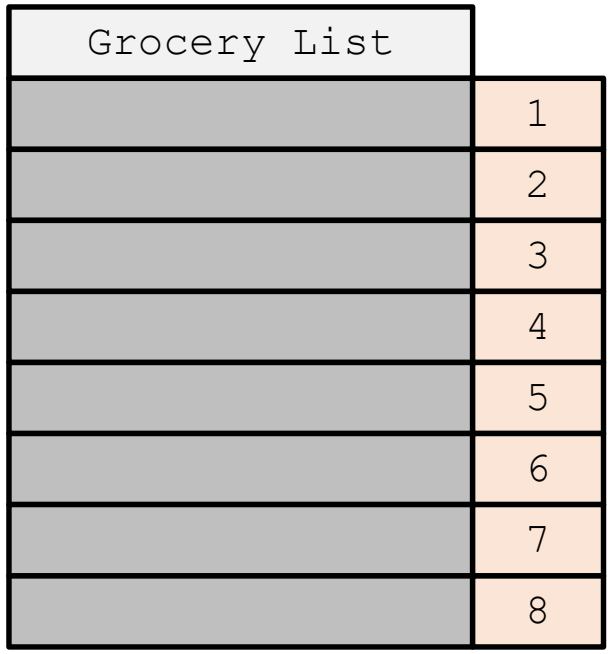
# Using the ADT List




```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```




```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

| Grocery List | |
|---|---|
| Apples | 1 |
| | 2 |
| | 3 |
| | 4 |
| | 5 |
| | 6 |
| | 7 |
| | 8 |

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

| Grocery List | |
|---|---|
| Apples | 1 |
| Oranges | 2 |
| | 3 |
| | 4 |
| | 5 |
| | 6 |
| | 7 |
| | 8 |

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
```

# Using the ADT List

| Grocery List | |
|---|---|
| Apples | 1 |
| Oranges | 2 |
| Cheese | 3 |
| | 4 |
| | 5 |
| | 6 |
| | 7 |
| | 8 |

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
groceryList->insert(3,"Cheese");
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

| Grocery List | |
|---|---|
| Apples | 1 |
| Oranges | 2 |
| Cheese | 3 |
| Tomatoes | 4 |
| | 5 |
| | 6 |
| | 7 |
| | 8 |

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
groceryList->insert(3,"Cheese");
groceryList->insert(4,"Tomatoes");
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

| Grocery List | |
|---|---|
| Bread | 1 |
| Apples | 2 |
| Oranges | 3 |
| Cheese | 4 |
| Tomatoes | 5 |
| | 6 |
| | 7 |
| | 8 |

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
groceryList->insert(3,"Cheese");
groceryList->insert(1,"Bread");
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

| Grocery List | |
|---|---|
| Bread | 1 |
| Apples | 2 |
| Oranges | 3 |
| Nachoes | 4 |
| Cheese | 5 |
| Tomatoes | 6 |
| | 7 |
| | 8 |

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
groceryList->insert(3,"Cheese");
groceryList->insert(1,"Bread");
groceryList->insert(4,"Nachos");
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

| Grocery List | |
|---|---|
| Bread | 1 |
| Apples | 2 |
| Nachos | 3 |
| Cheese | 4 |
| Tomatoes | 5 |
| | 6 |
| | 7 |
| | 8 |

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
groceryList->insert(3,"Cheese");
groceryList->insert(1,"Bread");
groceryList->insert(4,"Nachos");


groceryList->remove(3);
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

| Grocery List | |
|---|---|
| Bread | 1 |
| Apples | 2 |
| Nachos | 3 |
| Low-Fat Cheese | 4 |
| Tomatoes | 5 |
| | 6 |
| | 7 |
| | 8 |

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
groceryList->insert(3,"Cheese");
groceryList->insert(1,"Bread");
groceryList->insert(4,"Nachos");


groceryList->remove(3);
groceryList->setEntry(4, "Low-Fat Cheese");
```

# Using the ADT List

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

| Grocery List | |
|---|---|
| Bread | 1 |
| Apples | 2 |
| Nachos | 3 |
| Low-Fat Cheese | 4 |
| Tomatoes | 5 |
| | 6 |
| | 7 |
| | 8 |

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
groceryList->insert(3,"Cheese");
groceryList->insert(1,"Bread");
groceryList->insert(4,"Nachos");


groceryList->remove(3);
groceryList->setEntry(4, "Low-Fat Cheese");

int numberOfEntries = groceryList->getLength()
std::cout << "The list contains " << numberOfEntries;
std::cout << "entries, as follows:" << std::endl;
```

# Using the ADT List

| Grocery List | |
|---|---|
| Bread | 1 |
| Apples | 2 |
| Nachos | 3 |
| Low-Fat Cheese | 4 |
| Tomatoes | 5 |
| | 6 |
| | 7 |
| | 8 |

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
groceryList->insert(3,"Cheese");
groceryList->insert(1,"Bread");
groceryList->insert(4,"Nachos");


groceryList->remove(3);
groceryList->setEntry(4, "Low-Fat Cheese");

int numberOfEntries = groceryList->getLength()
std::cout << "The list contains " << numberOfEntries;
std::cout << "entries, as follows:" << std::endl;
for (int position = 1; position <= numberOfEntries; position++)
          std::cout << list->getEntry(position) << " is entry " << position <<std::endl;
```

# Using the ADT List

```
The list contains 5 entries, as follows:

Bread is entry 1
Apples is entry 2
Nachos is Entry 3
Low-Fat Cheese is entry 4
Tomatoes is entry 5
```

| | |
|---|---|
| Bread | 1 |
| Apples | 2 |
| Nachos | 3 |
| Low-Fat Cheese | 4 |
| Tomatoes | 5 |
| | 6 |
| | 7 |
| | 8 |

```
/** ADT list: Link-based implementation
    @file ListInterface.h */
#ifndef LIST_INTERFACE_
#define LIST_INTERFACE_
template<class ItemType>
class ListInterface
{
public:
      virtual bool isEmpty() const = 0;
      virtual int getLength() const = 0;
      virtual bool insert(int newPosition, const ItemType& newEntry)
      = 0;
      virtual bool remove(int position) = 0;
      virtual void clear() = 0;
      virtual ItemType replace(int position, const ItemType&
      newEntry) = 0;

      virtual ~ListInterface() { }
};    // end ListInterface
#endif
```

```
ListInterface<std::string>* groceryList = new SomeList<std::string> ();
groceryList->insert(1,"Apples");
groceryList->insert(2,"Oranges");
groceryList->insert(3,"Cheese");
groceryList->insert(1,"Bread");
groceryList->insert(4,"Nachos");


groceryList->remove(3);
groceryList->setEntry(4, "Low-Fat Cheese");

int numberOfEntries = groceryList->getLength()
std::cout << "The list contains " << numberOfEntries;
std::cout << "entries, as follows:" << std::endl;
for (int position = 1; position <= numberOfEntries; position++)
          std::cout << list->getEntry(position) << " is entry " << position <<std::endl;
```

# Thank you