# Lists

## Lecture 14

Robb T. Koether

Hampden-Sydney College

Wed, Feb 14, 2018

# Outline

# ADTs

## Definition (Abstract Data Type)

An abstract data type (ADT) is a data type that is described by its interface (how it's used), not by its implementation.

- For example, we know how to work with **`floats`** even though we do not know how **`floats`** are stored or how the operations on them are performed.
- The same is true of **`ints`**, **`bools`**, `strings`, and so on.

# Outline

# The List ADT

- A list is a collection of elements arranged in a physical order, but not necessarily in a logical order.

$$\{a_0, \ldots, a_{n-1}\}.$$

- $a_0$ is at the head of the list.
- $a_{n-1}$ is at the tail of the list.
- The size of the list is the number of elements in the list.
- The elements $a_i$ may be of any type, but the elements in the list must all be of the same type.
- That is, the structure is homogeneous.
- A list is a generalization of an array.

# Outline

# List Constructors

## List Constructors

```
List();
List(int sz);
List(int sz, const T& value);
List(const List& lst);
```

- `List()` – Construct an empty list.
- `List(int)` – Construct a list of the given size, initialized to default value of type.
- `List(int, T&)` – Construct a list of the give size, initialized to the given value.
- `List(List&)` – Construct a copy of the given list.

# Outline

# The List Destructor

## The List Destructor

```
~List();
```

- `~List()` – Destroy the list.

# Outline

# List Inspectors

## List Inspectors

```
int size() const;
bool isEmpty() const;
```

- `size()` – Return the number of elements in the list.
- `isEmpty()` – Return **`true`** if the list is empty.

# Outline

# List Mutators

## List Mutators

```
void element(int pos, const T& value);
void insert(int pos, const T& value);
void remove(int pos);
void makeEmpty();
```

- `element()` – Assign to the element in the given position the given value.
- `insert()` – Insert the given value into the given position.
- `remove()` – Remove the value from the given position.
- `makeEmpty()` – Remove all the elements.

# List Mutators

## List Mutators

```
void pushFront(const T& value);
void pushBack(const T& value);
T popFront();
T popBack();
```

- `pushFront()` – Insert the value at the “head” of the list.
- `pushBack()` – Insert the value at the “tail” of the list.
- `popFront()` – Remove and return the element at the “head” of the list.
- `popBack()` – Remove and return the element at the “tail” of the list.

# Outline

# List Facilitators

## List Facilitators

```
void input(istream& in);
void output(ostream& out) const;
bool isEqual(const List& lst) const;
```

- `input()` – Read a list from the given input stream.
- `output()` – Write a list to the given output stream.
- `isEqual()` – Determine whether two lists are equal.

# Input and Output

- The input and output format of a list is

$$\{\, a_0,\ \ a_1,\ \ \ldots,\ \ a_{n-1} \,\}$$

- That is, the list is delimited by curly braces $\{\ \}$ and the elements are separated by commas.
- This works for any data type for which a comma is not part of the value.
- For which data type(s) will this not work?

# Outline

# List Operators

## List Operators

```
List& operator=(const List& lst);
T operator[](int pos) const; // r-value
T& operator[](int pos);      // l-value
```

- **operator**=() – Assign one list to another list.
- T **operator**[]() – Return a copy of the element in the given position (*r*-value).
- T& **operator**[]() – Return a reference to the element in the given position (*l*-value).

# Outline

# Other Member Functions

## Other Member Functions

```
void swap(List& lst);
int search(const T& value) const;
void sort();
bool isValid() const;
```

- `swap()` – Swap the values of two lists.
- `search()` – Search the list for the given value. Return the position where it was found, or return $-1$ if it was not found.
- `sort()` – Sort the elements into ascending order.
- `isValid()` – Check the structural integrity of the list.

# Outline

# Non-member Operators

## Non-member Operators

```
istream& operator>>(istream& in, List& lst);
ostream& operator<<(ostream& out, const List& lst);
bool operator==(const List& lst1, const List& lst2);
bool operator!=(const List& lst1, const List& lst2);
```

- `operator>>()` – Read a list from the given input stream.
- `operator<<()` – Write a list to the given input stream.
- `operator==()` – Determine whether two lists are equal.
- `operator!=()` – Determine whether two lists are not equal.

# Outline

# Example

## Example (List ADT Example)

- `ListDemo.cpp`