NOTE: The first command-line argument passed (after the program name) should be the directory of the images

The file "SourceCode" contains all the source files and a makefile.txt textfile. The code files compile to an executable named "LinkedList."

To compile: `make -f makefile.txt`

To run: `./LinkedList directoryName`

General

- The function `printList` in main.cpp is a function template that has two parameters: a constant reference to LinkList object of type T, and a string to store the list name that has a default value of "Linked List." It prints out a list of all the items in the LinkList. If the list is empty, it prints out "Empty list."
- `main` function: first gives an example of LinkList operations on an integer list. Then reades bitmap files from directory and creates a char* list. It creates a copy of the list, and then performs operations on the original list. The copied list retains all the filenames read from the file, which are then displayed on the screen.

`class node NodeClass.cpp`

- Class template for holding a linked list node
- `val` is a private variable of type T
- `*next` is a node pointer to the next node
- Has a default constructor with no arguments that creates an empty node with a `NULL` pointer
- Has a constructor that takes an item of type T, which creates a node, stores the item in `val`, and sets the pointer to `NULL`

`class LinkList`

- Class template for holding a linked list
- `*head` points to first node in list and `*tail` points to last node
- `numElements` to store number of items in the list
- Default constructor creates an empty list and sets head and tail to `NULL`
- the `operator=(const LinkList &rhs)` function moves through the rhs list, makes a copy of each node, and adds it to the new list.
- `push_back(const T&)` appends a new node to the end of the list and `push_front(const T&)` appends a new node to the front of the list
- `pop_front` removes the first node from the list and deletes it from memory. `pop_back` removees the last node and deletes it from memory
- `remove` moves to the i^{th} node in the list. It uses two `node` pointers to traverse the list, with one pointing to the current node, and the other pointing to the one before it.
- `clear` deletes every node in the linked list, and sets the head and tail pointers to `NULL.`
- `get(i)` returns a copy of the i^{th} node in the list.
  - throws a `listException` error if the index is out of bounds (less than 1 or greater than numElements)
- `find(const T&)` moves through the list and compares each node value to the value passed as an argument until it finds a match, and then it returns the index of that node. If it can't find that value, it returns "-1".
- `size` returns an int = the number of items in the linked list

`class LinkedListCstring`

- specialized template class to hold c-string type linked lists
- Note: all of the member functions have the same functionality as the ones in the LinkList class, so descriptions below just have to do how `LinkedListCstring` is different than `LinkList`
- the `operator=(const LinkList &rhs)` function moves through the rhs list, makes a copy of each node, and adds it to the new list. Each time it copies a node, it makes a copy of the val contained in the node to avoid shallow copying.
- `push_back` and `push_front` both have parameter types of `char*`
- `get` returns a `char*`

- `find(char*)` moves through the list and used strcmp() to compare each node value to the value passed as an argument until it finds a match, and then it returns the index of that node. If it can't find that value, it returns "-1".

`listException class`

- For catching errors that occur when user tries to access an out of bound index in the list
- `msg` is a string to store an error message
- `what` is a function that returns the value stored in string