Documentation for List.h and List.c

Steven Andrews, © 2015

Description

This library supports basic list management for unsorted lists. These lists are designed to be very simple to use, somewhat like the Python list data type. They use dynamic memory allocation, so that they expand themselves as needed.

The code that calls these lists should call the functions here for adding elements to lists and manipulating lists. However, it is fine for the code to read the list directly rather than through functions.

Currently supported list types are as follows

abbreviation	list
li	long integers
v	void*

Dependencies

```
List.h
string2.h
```

History

10/7/15	Started. Wrote support for long integers.
8/22/16	Updated. Added void* list type.

Data structures

```
typedef struct liststructli{
  int max;
  int n;
  long int *xs;
  } *listptrli;

typedef struct liststructv{
  int max;
  int n;
  void **xs;
  } *listptrv;
```

In the data structure, `max` gives the allocated size of the list. It is allowed to equal 0, in which case `xs` is equal to `NULL`. `n` is the number of elements currently in the list. `xs` is the actual list, which is allocated to size `max` and filled from element 0 to element `n-1`. If their is a dual list, then its element is `xd`; this list has the same allocation size and number of elements in the list, and these elements correspond to the ones in the first list.

Code documentation

Internal functions

`int List_ExpandLI(listptrli list,int spaces);`

Expands memory allocated for existing list `list` by `spaces` spaces, without changing list contents. `spaces` is allowed to be negative for list shrinking. The list can be shrunk sufficiently that some contents are lost. Returns 0 for success or 1 for unable to allocate memory.

`int List_ExpandV(listptrv list,int spaces);`

Expands memory allocated for existing list `list` by `spaces` spaces, without changing list contents. `spaces` is allowed to be negative for list shrinking. The list can be shrunk sufficiently that some contents are lost. Returns 0 for success or 1 for unable to allocate memory.

Memory management

`listptrli List_AllocLI(int max);`

Allocates a new empty list, set up for `max` spaces. `max` is allowed to equal 0. Returns the list or `NULL` if unable to allocate memory.

`listptrv List_AllocV(int max);`

Allocates a new empty list, set up for `max` spaces. `max` is allowed to equal 0. Returns the list or `NULL` if unable to allocate memory.

`void List_FreeLI(listptrli list);`

Frees all memory allocated for list `list`. `list` is allowed to be `NULL`, in which case this does nothing.

`void List_FreeV(listptrv list);`

Frees all memory allocated for list `list`. `list` is allowed to be `NULL`, in which case this does nothing.

Reading lists

`int List_MemberLI(const listptrli list,long int x);`

Tests to see if `x` is a member of list `list`, returning 1 if so and 0 if not.

Adding elements to lists

`listptrli List_ReadStringLI(char *string);`

Reads a string of elements of the given type (e.g. long integers for the LI suffix) from `string` and returns them in a newly created list. Returns the list for success or `NULL` for failure, where this failure could arise from a memory allocation error (unlikely) or a string reading error (likely).

```
listptrv ListAppendItemV(listptrv list,void *newitem);
```

Appends item `newitem` to list `list`, returning `list` on success or `NULL` on failure to allocate memory. If `list` is entered as `NULL`, then a new list is created and is returned.

Combining lists

```
int List_AppendListLI(listptrli list,const listptrli newstuff);
```

Appends the contents of the list `newstuff` to the end of the existing list `list`, expanding `list` as needed. Returns 0 for success or 1 for memory allocation error.

```
int List_RemoveListLI(listptrli list,const listptrli remove);
```

Removes items that are in the list `remove` from the list `list`. For each item that is in `remove`, if it is in `list` in multiple copies, then only the last copy is removed. If an item that is in `remove` is not found in list, then this simply continues on to the next item. Returns the number of items that were removed from list `list`.