A Doubly-Linked List Class

Version 1b

The *DLL* class

For your doubly-linked list class, the header file, *dll.h*, should look like:

```
#ifndef __DLL_INCLUDED__
#define __DLL_INCLUDED__

#include <stdio.h>

typedef struct dll DLL;

extern DLL *newDLL(void (*d)(void *,FILE *),void (*f)(void *));
extern void insertDLL(DLL *items,int index,void *value);
extern void *removeDLL(DLL *items,int index);
extern void unionDLL(DLL *recipient,DLL *donor);
extern void *getDLL(DLL *items,int index);
extern void *setDLL(DLL *items,int index,void *value);
extern int sizeDLL(DLL *items);
extern void displayDLL(DLL *items,FILE *);
extern void displayDLLdebug(DLL *items,FILE *);
extern void freeDLL(DLL *items);

#endif
```

The header file contains the function signatures of your public methods while the code module, *dll.c*, contains their implementations.

The only local includes that *dll.c* should have is *dll.h*.

Method behavior

Here are some of the behaviors your methods should have. This listing is not exhaustive; you are expected, as a computer scientist, to complete the implementation in the best possible manner.

- *newDLL* - The constructor is passed a function that knows how to display the generic value stored in a linked list node. That function is stored in a *display* field of the *DLL* object:

  ```
  //d is the display function
  //f is the freeing function
  DLL *newDLL(void (*d)(void *,FILE *),void (*f)(void *))
      {
      DLL *items = malloc(sizeof(DLL));
      assert(items != 0);
      items->head = 0;
      items->tail = 0;
      items->size = 0;
      items->display = d;
      items->free = f;
      return items;
      }
  ```

  By the same token, the constructor is passed in a function that knows how to free the generic value.
- *insertDLL* - It runs in constant time for insertions at a constant distance from the front and from the back. The doubly-linked list uses zero-based indexing.
- *removeDLL* - It runs in constant time for removals at a constant distance from the front and from the back.
- *unionDLL* - The union method takes two lists and moves all the items in the donor list to the recipient list. If the recipient list has the items 3,4,5 and the donor list has the items 1,2, then, after the union, the donor list will be empty and recipient list will have the items 3,4,5,1,2. The union method runs in constant time.
- *getDLL* - The method returns the value at the given index. It runs in constant time for retrievals at a constant distance from the front and from the back.

- *setDLL* - The method updates the value at the given index. If the given index is a valid index for the list, the replaced value is returned. If the given index is equal to the size, the value is appended to the list and a null pointer is returned. It runs in constant time for updates at a constant distance from the front and from the back.
- *sizeDLL* - The size method returns the number of items stored in the list.
- *displayDLL* - The display method prints the items stored in the list. If the list holds the integers 5, 6, 2, 9, and 1, with 5 at the front and 1 at the back, the method would generate this output:

  5,6,2,9,1

  with no preceding or following whitespace. An empty list displays as .
- *freeDLL* - This method walks through the list, freeing the generic values (using the passed-in freeing function) and the nodes that hold them. If the freeing function is null, the generic value is not freed.

Assertions

Include the following assertions in your methods:

- *newDLL* - The memory allocated shall not be zero.
- *insertDLL* - The index shall be greater than or equal to zero and less than or equal to the size.
- *removeDLL* - The size shall be greater than zero. The index shall be greater than or equal to zero and less than the size.
- *getDLL* - The index shall be greater than or equal to zero and less than the size.
- *setDLL* - The index shall be greater than or equal to zero and less than or equal to the size.

Testing your DLL class

Modify the testing program found in the *sll class description* to work with doubly-linked lists. Make sure you add additional testing to make sure the time constraints of all methods are met.