A Singly-Linked List Class

Version 1a

The *SLL* class

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

```
#ifndef __SLL_INCLUDED__
#define __SLL_INCLUDED__

#include <stdio.h>

typedef struct sll SLL;

extern SLL *newSLL(void (*d)(void *,FILE *),void (*f)(void *));
extern void insertSLL(SLL *items,int index,void *value);
extern void *removeSLL(SLL *items,int index);
extern void unionSLL(SLL *recipient,SLL *donor);
extern void *getSLL(SLL *items,int index);
extern void *setSLL(SLL *items,int index,void *value);
extern int sizeSLL(SLL *items);
extern void displaySLL(SLL *items,FILE *);
extern void displaySLLdebug(SLL *items,FILE *);
extern void freeSLL(SLL *items);

#endif
```

The header file contains the function signatures of your public methods while the code module, *sll.c*, contains their implementations.
The only local includes that *sll.c* should have is *sll.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, minimizing asymptotic run times and space usage (this is true for all your data structures).

- *newSLL* - 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 *SLL* object:

  ```
  //d is the display function
  //f is the freeing function
  SLL *newSLL(void (*d)(void *,FILE *),void (*f)(void *))
      {
      SLL *items = malloc(sizeof(SLL));
      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.
- *insertSLL* - It runs in constant time for insertions at the very end of the list and runs in constant time for insertions that are a constant distance from the front. The singly-linked list uses zero-based indexing.
- *removeSLL* - It runs in constant time for removals at a constant distance from the front.
- *unionSLL* - 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.
- *getSLL* - The method returns the value at the given index. It runs in constant time for retrievals at the very back of the list and at a constant distance from the front.

- *setSLL* - 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 the very back of the list and at a constant distance from the front.
- *sizeSLL* - The size method returns the number of items stored in the list.
- *displaySLL* - 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 `{}`.
- *freeSLL* - 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:

- *newSLL* - The memory allocated shall not be zero.
- *insertSLL* - The index shall be greater than or equal to zero and less than or equal to the size.
- *removeSLL* - The size shall be greater than zero. The index shall be greater than or equal to zero and less than the size.
- *getSLL* - The index shall be greater than or equal to zero and less than the size.
- *setSLL* - The index shall be greater than or equal to zero and less than or equal to the size.

Testing your SLL class

Here is a sample testing program that uses the *displayINTEGER* function found in the *integer* class. Note how it is passed to the *newSLL* constructor.

```
#include <stdio.h>
#include <stdlib.h>
#include "sll.h"
#include "integer.h"

static void showItems(SLL *items)
    {
    printf("The items are ");
    displaySLL(items,stdout);
    printf(".\n");
    printf("The items (debugged) are ");
    displaySLLdebug(items,stdout);
    printf(".\n");
    }

int main(int argc,char **argv)
    {
    SLL *items = newSLL(displayINTEGER,freeINTEGER);
    showItems(items);
    insertSLL(items,0,newINTEGER(3));                   //insert at front
    insertSLL(items,sizeSLL(items),newINTEGER(2));      //insert at back
    insertSLL(items,1,newINTEGER(1));                   //insert at middle
    showItems(items);
    printf("The value ");
    INTEGER *i = removeSLL(items,0));                   //remove from front
    displayINTEGER(i,stdout);
    printf(" was removed.\n");
    freeINTEGER(i);
    showItems(items);
    int x = getINTEGER((INTEGER *) getSLL(items,0));    //get the first item
    printf("The first item is %d.\n",x);
    printf("Freeing the list.\n");
    freeSLL(items);
    return 0;
    }
```

The output of this program should be:

```
The items are {}.
The items (debugged) are head->{},tail->{}.
The items are {3,1,2}.
The items (debugged) are head->{3,1,2},tail->{2}.
The value 3 was removed.
The items are {1,2}.
The items (debugged) are head->{1,2},tail->{2}.
The first item is 1.
```

You may download the integer class with these commands:

```
wget beastie.cs.ua.edu/cs201/testing/integer.c
wget beastie.cs.ua.edu/cs201/testing/integer.h
wget beastie.cs.ua.edu/cs201/testing/test-integer.c
wget beastie.cs.ua.edu/cs201/testing/makefile
```

The last command retrieves a program for testing the integer class. Compile and run this program with the commands:

```
make test
```

You should model all your classes and makefiles after the the integer class.

The superior student will test his or her singly-linked list class with other data types besides *INTEGER*s by implementing such classes as *REAL* and *STRING*.