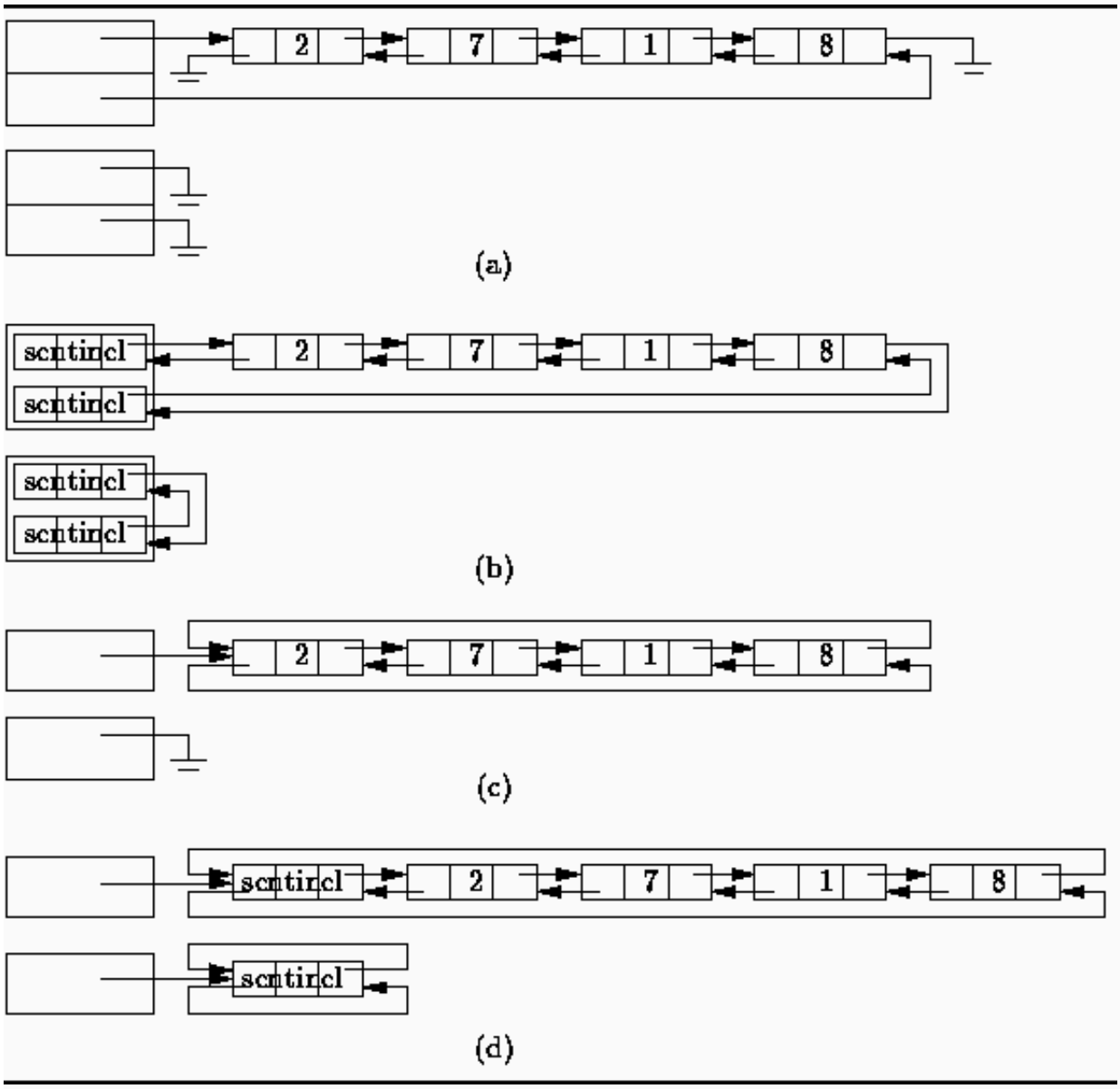
Doubly Linked Lists

Implement the simplest case as shown in Figure `(a)` below. Two pointers, say head and tail, are used to keep track of the list elements. One of them points to the first element of the list, the other points to the last. The first element of the list has no predecessor, therefore that pointer is null. Similarly, the last element has no successor and the corresponding pointer is also null. In effect, we have two overlapping singly-linked lists which go in opposite directions. Figure `(a)` also shows the representation of an empty list. In this case the head and tail pointers are both null.



A circular, doubly-linked list is shown in Figure gif (c). A circular list is formed by making use of pointers which would otherwise be null: The last element of the list is made the predecessor of the first element; the first element, the successor of the last.

Auxiliary Functions

Before attempting the main functions of this lab exercise, it is advisable that you tackle the auxiliary functions listed below.

```
int getValue(); // function to receive integer value to store in list
int getPosition(); // function to receive the position to delete/insert
Node* traverse (Node *np,int steps); // returns the pointer to the Node at `steps` posi
tions away
void connect (Node *node, Node *next);// connect two nodes so they are `doubly` linked
int isValid (Node *n); // basically, returns 0 if pointer value is NULL
Node* createNode (int val); // creates Node with `val` and links all initialized to NUL
L
```

Related CC Problems

- connect and traverse

Use the above functions to define the following functions and pass all the test cases presented herein.

Functions to define

```
int main();
void find(), ndisplay(), search(), ndelete();
void insert(), create(),
extern Node *ltail;
extern Node *lhead;
```