# Data Structures: Stacks and Queues (Day 17)

Before introducing new material, here are the recursive versions of the algorithms from the linked list notes (Day 15).

Practice: Create a recursive version of this function to create a linked list from an array of values. Answer will appear in the next set of notes.

```
//Copies the contents of an array into a dynamically
//growing list.

struct node *array_to_list (int a[ ], int j, int n)
{
     struct node *head;

     if ( j >= n)  //base case
          return NULL;
     else
     {    head = malloc(sizeof(struct node));
          head -> data = a[j];
          head ->next = array_to_list(a, j+1, n);
```

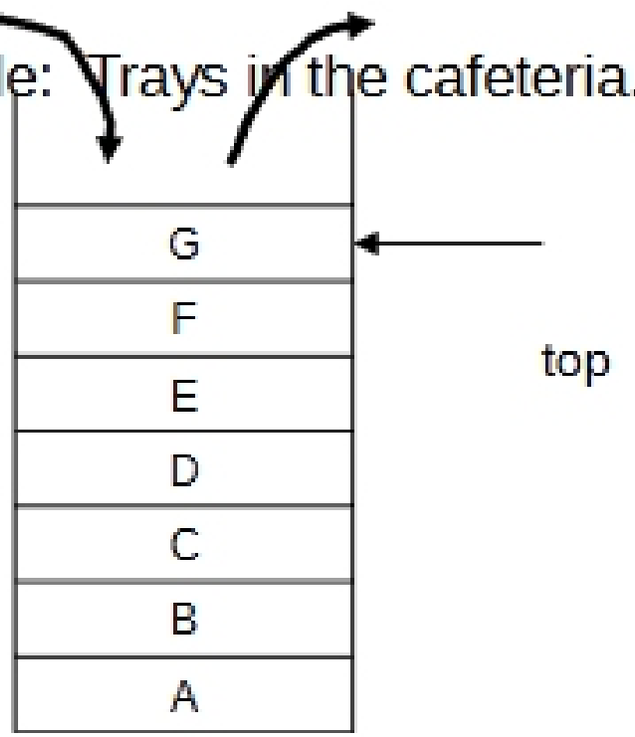
Prac … s in a link…

```
//Count the number of nodes in a list

 int count(struct node *head)
{
     if (head == NULL)  //base case
          return 0;
     else
          return (1 + count(head->next));
```

## Stacks

- A *stack* is a collection of items into which new items are inserted and from which items are deleted from only one end, called the *top* of the stack.
- Different implementations are possible; although the concept of a stack is unique.
- Example: Trays in the cafeteria.

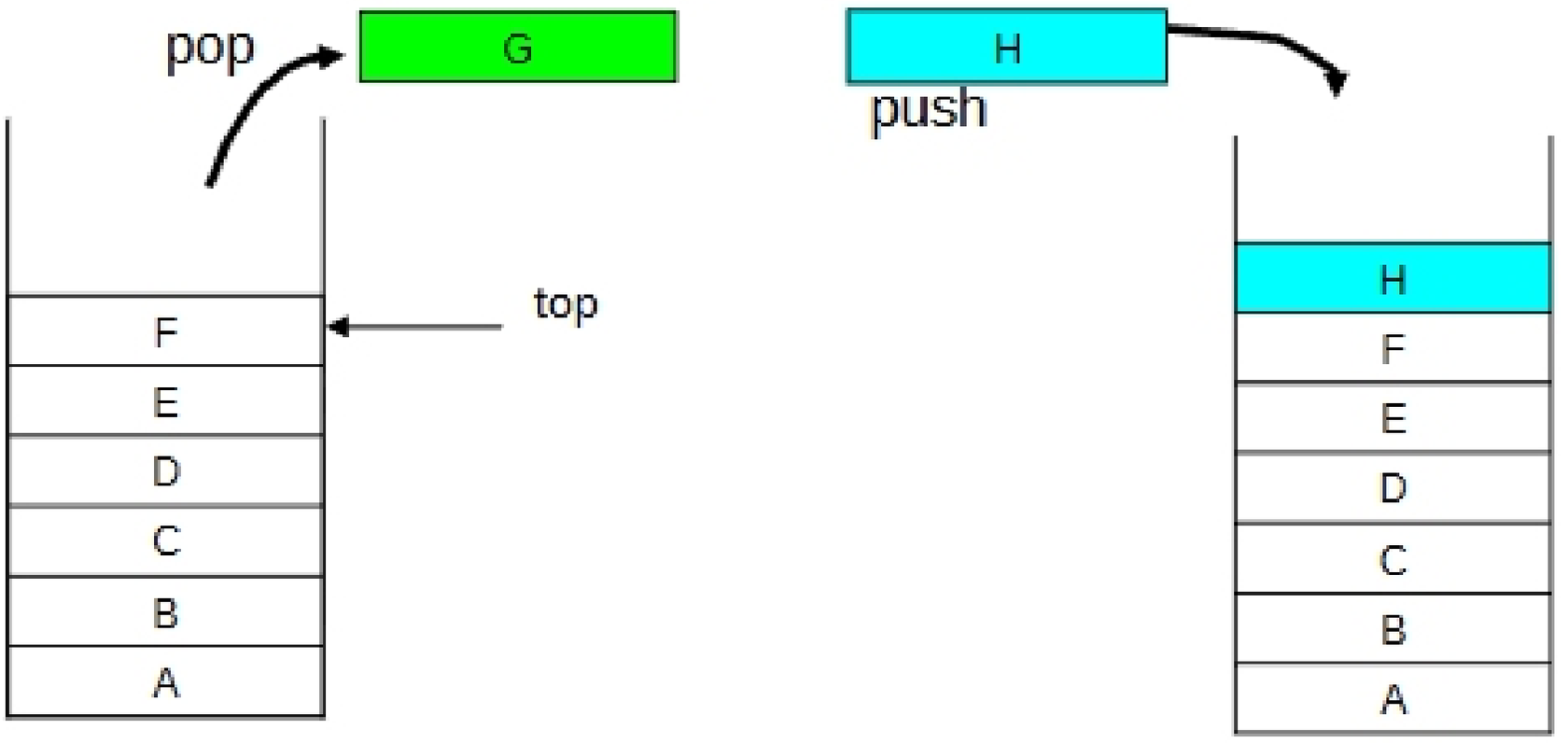


- There are two primary operations defined on a stack:
  - **push**: add a new item to the top of the stack.
  - **pop**: removes the item from the top of the stack.

- A stack is also known as a push-down list.

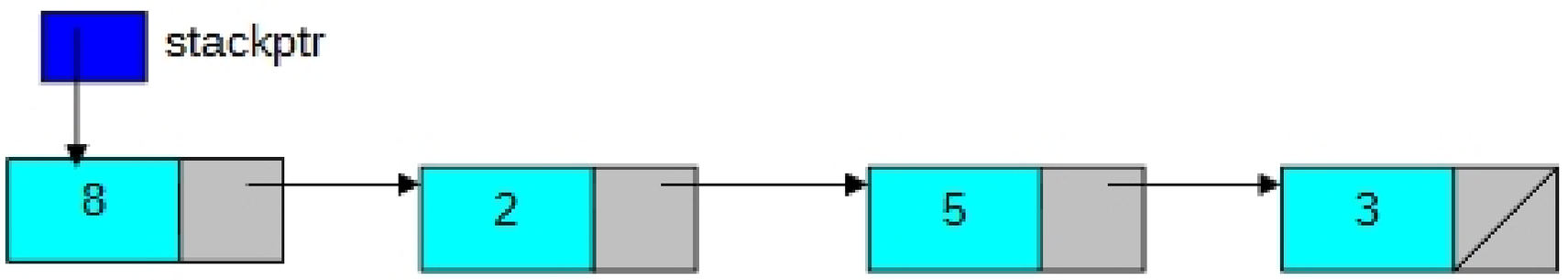
- The "access policy" of a stack is LIFO (**L**ast **I**n **F**irst **O**ut).

- A stack is a dynamic structure. It changes as elements are added to and removed from it.




- A stack can be implemented as a constrained version of a linked list. A stack is referenced via a pointer to the top element of the stack. The link member in the last node of the stack is set to NULL to indicate the bottom of the stack.

- Example:




where *stackptr* points to the top of the stack.

- Note that stacks and linked lists are represented identically. The difference is that insertions and deletions occur anywhere in a linked list but are constrained to only the top of a stack.

- Function **push** creates a new node and places it on the top of the stack.

- Function **pop** removes a node from the top of the stack and frees the memory that was allocated to the popped node, and returns the popped node.

## Implementation of a simple stack of integers

```
struct stackNode{
      int data;
      struct stackNode  *nextptr;
};

//inserts a node at the top of the stack
void push(struct stackNode **topptr, int info)
{
      struct  stackNode *newptr;

      newptr = (struct stackNode *)
                  malloc (sizeof (struct stackNode));
      if (newptr != NULL)
      {     newptr->data = info;
            newptr->nextptr = *topptr;
            *topptr = newptr;
      }
      else
         printf("%d not inserted. No memory available.\n", info);
}

//remove a node from the top of the stack
int pop(struct stackNode **topptr)
{
      struct stackNode *tempptr;
```

```
      topptr = (topptr)->nextptr;
      free(tempptr);
      return popvalue;
}

//check for an empty stack
int isEmpty(struct stackNode *topptr)
{
      return topptr ==NULL;
`
```