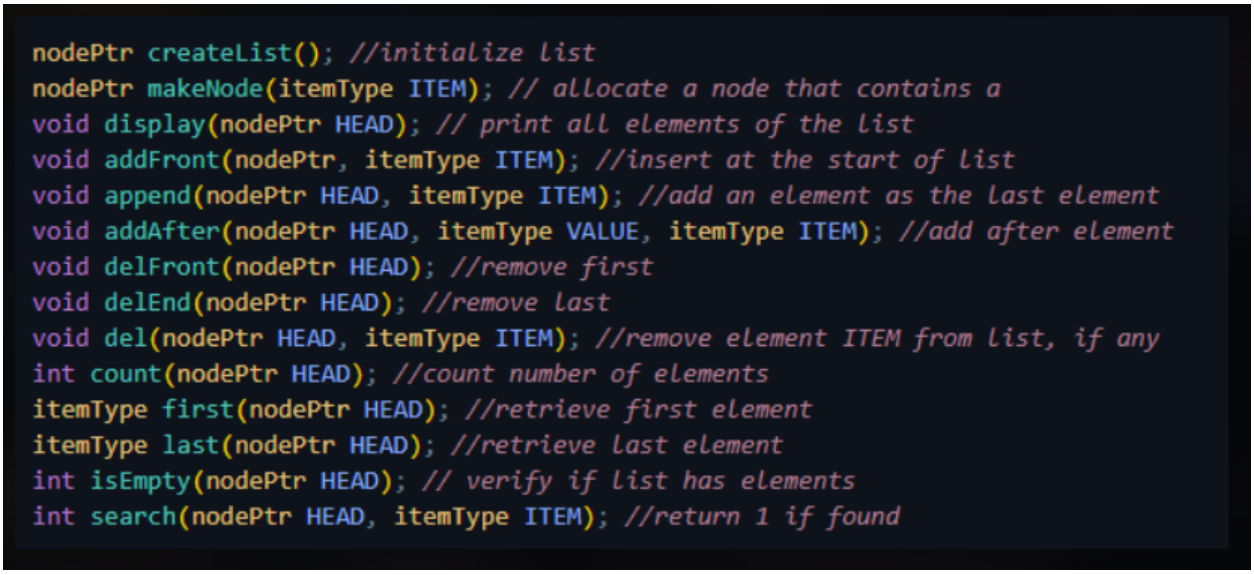
Data Struct: Stack and Queue

Created	@October 18, 2022 11:41 PM
Class	Data Struct
Type	Assignment
Materials	
Reviewed	☐



```
nodePtr createList(); //initialize list
nodePtr makeNode(itemType ITEM); // allocate a node that contains a
void display(nodePtr HEAD); // print all elements of the list
void addFront(nodePtr, itemType ITEM); //insert at the start of list
void append(nodePtr HEAD, itemType ITEM); //add an element as the last element
void addAfter(nodePtr HEAD, itemType VALUE, itemType ITEM); //add after element
void delFront(nodePtr HEAD); //remove first
void delEnd(nodePtr HEAD); //remove last
void del(nodePtr HEAD, itemType ITEM); //remove element ITEM from list, if any
int count(nodePtr HEAD); //count number of elements
itemType first(nodePtr HEAD); //retrieve first element
itemType last(nodePtr HEAD); //retrieve last element
int isEmpty(nodePtr HEAD); // verify if list has elements
int search(nodePtr HEAD, itemType ITEM); //return 1 if found
```

Stack

- **void addFront(nodePtr, itemType ITEM);** //insert at the start of list

Inserting a node at the front of linked list

Algorithm to add a new node at front of linked list

- Dynamically create a new node using malloc function.
- Set data field of new node.
- Set the next pointer of new node to head of the linked list.
- Set new node as new head of linked list. Update head pointer.

```
/*
Inserts a node in front of a singly linked list.
*/
void insertAtFront(int num) {
    /* Create a new Linked List node */
    struct node* newNode = (struct node*) malloc(sizeof(struct node));
    newNode->data  = num;
    /* Next pointer of new node will point to head node of linked list
    newNode->next = head;
    /* make new node as new head of linked list */
    head = newNode;
    printf("Inserted Element : %d\n", num);
}
```

- **void delFront(nodePtr HEAD)**; //remove first
- **itemType first (nodePtr HEAD){ //retrieve first element**

  int flag;

  flag = HEAD → next;

  return flag;

  }

Queue

- void append (nodePtr HEAD, itemType ITEM); //add an element as the last element

Inserting a node at the end of linked list

Algorithm to add a new node at front of linked list

- Dynamically create a new node using malloc function.
- Set data field of new node.
- Set next pointer of new node to NULL.
- Traverse from head node till tail node.
- Insert new after after tail node. Set next pointer of tail node to new node.

```
/*
Inserts a node after last node of linked list
*/
void insertAtEnd(struct node* head, int num){
    /* Input validation */
    if (head == NULL) {
       printf("Error : Invalid node pointer !!!\n");
       return;
    }
    /* creates a new node */
    struct node* newNode =(struct node*) malloc(sizeof(struct node));
    newNode->data  = num;
    newNode->next = NULL;
    /* Traverse from head to last node  */
    while(head->next != NULL)
       head = head->next;

    /* Insert newNode after Tail node */
    head->next = newNode;
}
```

- void delEnd (nodePtr HEAD); // remove last element

```c
// Deletes the last node of the linked list
void LastNodeDeletion()
{
    struct node *toDelLast, *preNode;
    if(stnode == NULL)
    {
        printf(" There is no element in the list.");
    }
    else
    {
        toDelLast = stnode;
        preNode = stnode;
        /* Traverse to the last node of the list*/
        while(toDelLast->nextptr != NULL)
        {
            preNode = toDelLast;
            toDelLast = toDelLast->nextptr;
        }
        if(toDelLast == stnode)
        {
            stnode = NULL;
        }
        else
        {

            /* Disconnects the link of second last node with last node */
            preNode->nextptr = NULL;
        }

        /* Delete the last node */
        free(toDelLast);
    }
}
```

- **itemType first (nodePtr HEAD){ //retrieve first element**

  int flag;

  flag = HEAD → next;

  return flag;

  }

- **itemType last (nodePtr HEAD){ //retrieve last element**

  int flag;

  while(HEAD! = NULL){

  if(HEAD==NULL){

  flag = HEAD → next;

  return flag;

  break;

  }

  HEAD=HEAD→next;

```
    }
}
```

Applications in Real Life

Stacks

The Stack is one of the most important data structures in computer science. Elements can be added to or removed from a stack at only one end. Since we view this end as the "top" of the stack, we use the term "push" for add, and the term "pop" for remove. Notice that the element popped from a stack is always the last one pushed onto it (among those still on it). Therefore, a stack is referred to as a Last-In-First-Out (LIFO) list.

Examples of stacks in "real life":

- The stack of trays in a cafeteria;
- A stack of plates in a cupboard;
- A driveway that is only one car wide. [Riley, p. 290]

Examples of stacks in computing:

- Back/Forward stacks on browsers;
- Undo/Redo stacks in Excel or Word;
- Activation records of method calls;

Queues

Elements can be added only at one end, the "rear", Elements can be removed only at the other end, the "front".
We call adding to a queue "enqueueing", and removing from a queue "dequeueing".
Since the element dequeued is always the first one enqueued (among those still on it), a queue is referred to as a First-In-First-Out (FIFO) list.

Examples of queues in "real life":

- A ticket line;
- An escalator;
- A car wash.

Examples of queues in computing:

- A printer queue;
- An input stream;