Tumpalan, John Karl B.
2018 – 02385 CMSC 123 – U5L

Both stack and queue are types of linear data structures or abstract data types. Both are subsets of the List ADT (in array implementation) and these two have only some modifications in terms of data manipulation or input and output.

Stack follows the Last In, First Out (LIFO) structure order wherein the data that you last added to the data structure will be the first one to be deleted or popped when we try to manipulate the data inside it. We can only delete/insert data through the Top of Stack (TOS) which is commonly viewed as the length of the elements inside the data structure. Stack's most common errors are Stack Underflow (trying to access data below the first index) and Stack Overflow (trying to access, insert, or remove data above the Stack's max length).

On the other hand, Queue follows the First In, First Out (FIFO) structure order. Data manipulation includes Enqueuing and Dequeuing. The first elements / nodes to be enqueued are also the first nodes to be dequeued. We can only remove an element on one side and insert on the other side. Same common error for Queues are the Underflow and Overflow errors which means the index to be accessed are already out of range (higher than max length, lower than the head)

Tumpalan, John Karl B.
2018 – 02385 CMSC 123 – U5L

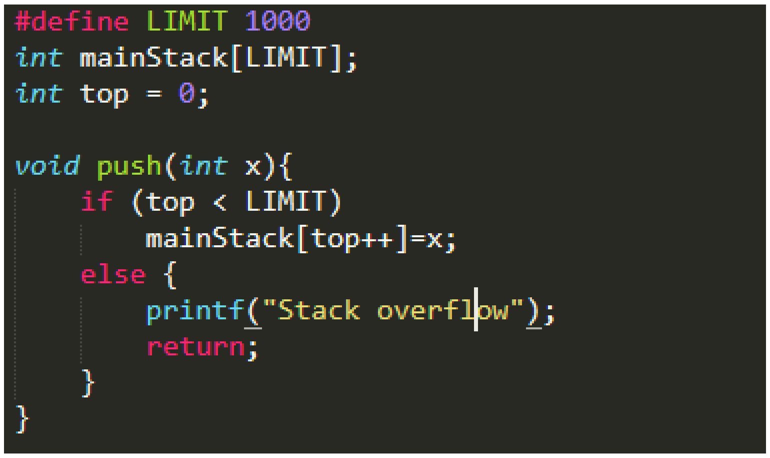
Stacks

Array implementation is simpler and quicker but is limited in size when declared statically.

				TOP OF STACK

Explanation 1:

Implementation for stack's array implementation

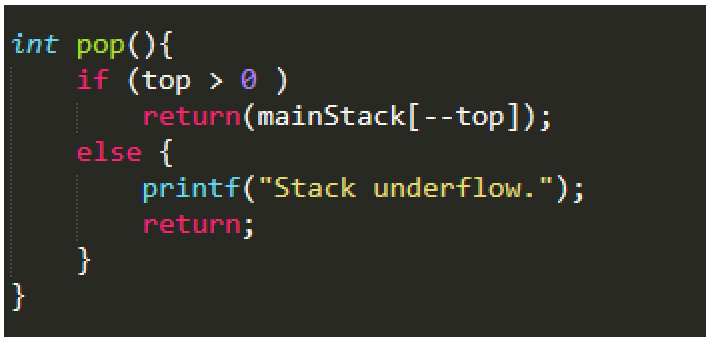


```
#define LIMIT 1000
int mainStack[LIMIT];
int top = 0;

void push(int x){
    if (top < LIMIT)
        mainStack[top++]=x;
    else {
        printf("Stack overflow");
        return;
    }
}
```

PUSH

					NEW TOP OF STACK

Explanation 2:



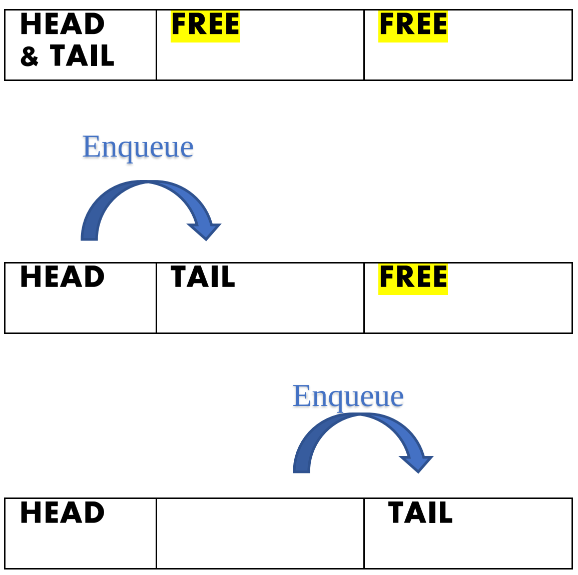
```
int pop(){
    if (top > 0 )
        return(mainStack[--top]);
    else {
        printf("Stack underflow.");
        return;
    }
}
```

POP

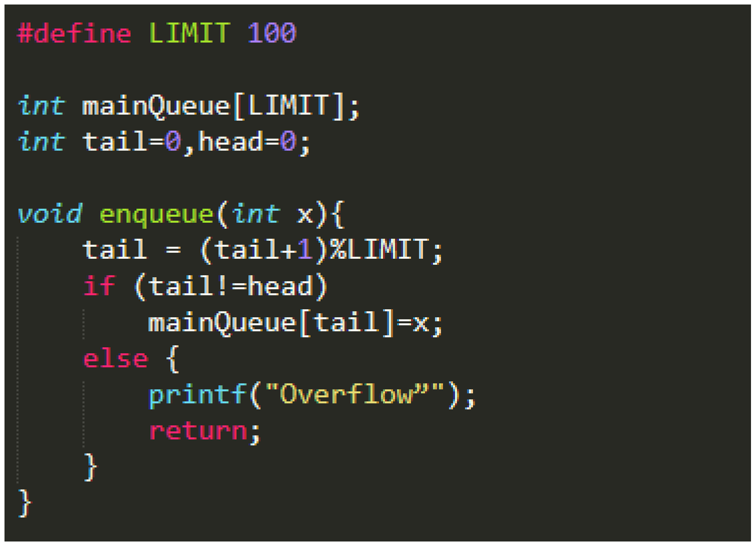
				NEW TOP OF STACK

Tumpalan, John Karl B.
2018 – 02385 CMSC 123 – U5L

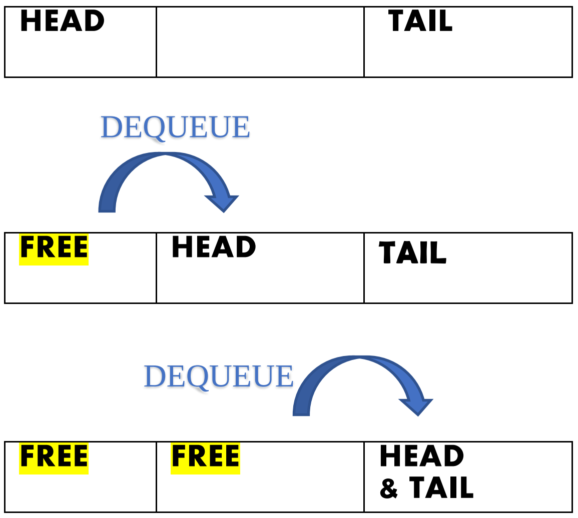
Queues



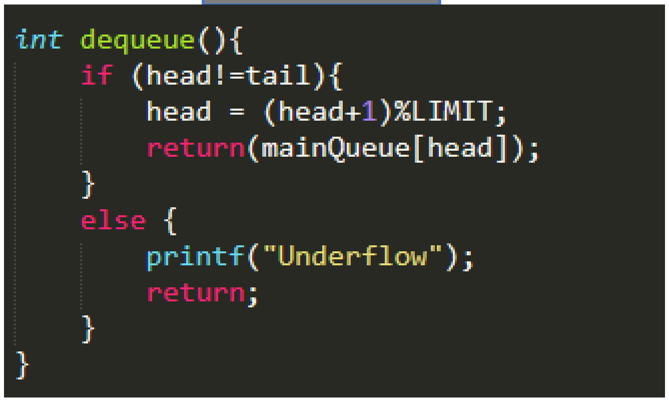
ENQUEUING



```
#define LIMIT 100

int mainQueue[LIMIT];
int tail=0,head=0;

void enqueue(int x){
    tail = (tail+1)%LIMIT;
    if (tail!=head)
        mainQueue[tail]=x;
    else {
        printf("Overflow”");
        return;
    }
}
```



DEQUEUING



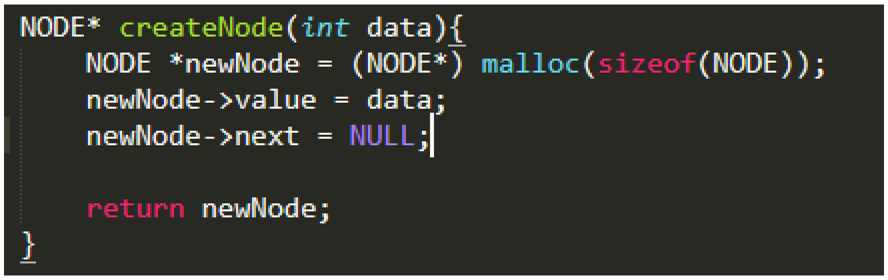
```
int dequeue(){
    if (head!=tail){
        head = (head+1)%LIMIT;
        return(mainQueue[head]);
    }
    else {
        printf("Underflow");
        return;
    }
}
```

Tumpalan, John Karl B.
2018 – 02385 CMSC 123 – U5L

Linked list implementation:

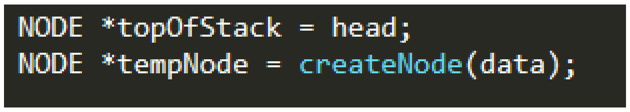
Stacks

Linked list implementation is more complex as you need to allocate memory for each node, but you have more control on memory allocation — removing the limitation in size.



```
NODE* createNode(int data){
    NODE *newNode = (NODE*) malloc(sizeof(NODE));
    newNode->value = data;
    newNode->next = NULL;

    return newNode;
}
```

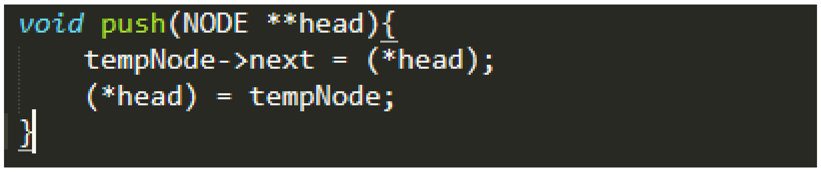
First, we declare the top of the stack and create new data for pushing.



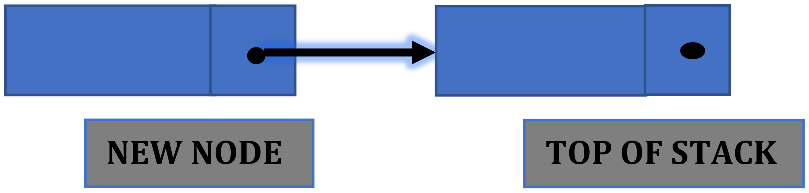
```
NODE *topOfStack = head;
NODE *tempNode = createNode(data);
```

PUSH

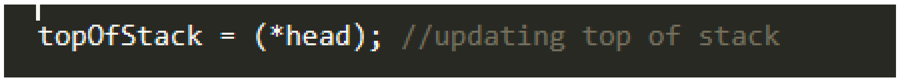
We already created a node, so we will insert the created node at the head of the linked list every time we wish to push new data to the stack.



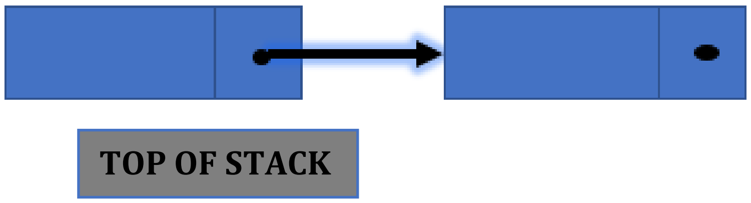
```
void push(NODE **head){
    tempNode->next = (*head);
    (*head) = tempNode;
}
```



and then we update the top of stack



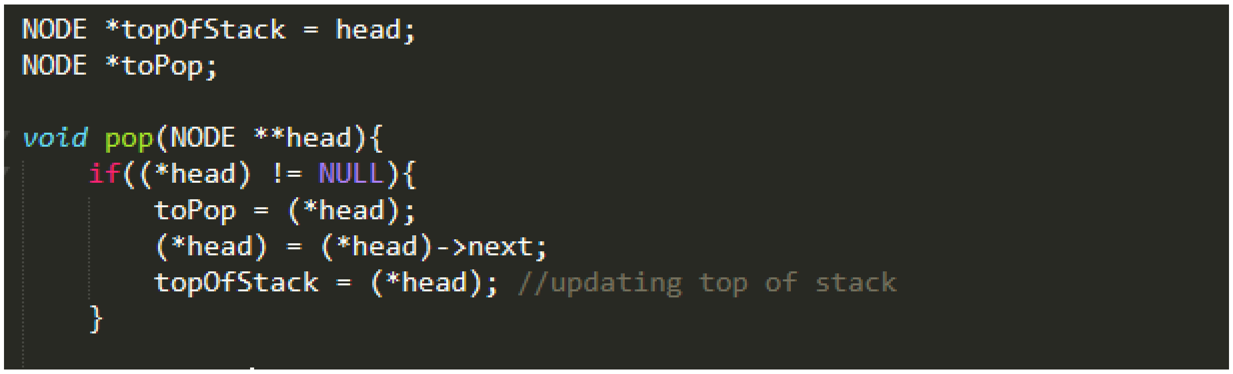
```
topOfStack = (*head); //updating top of stack
```



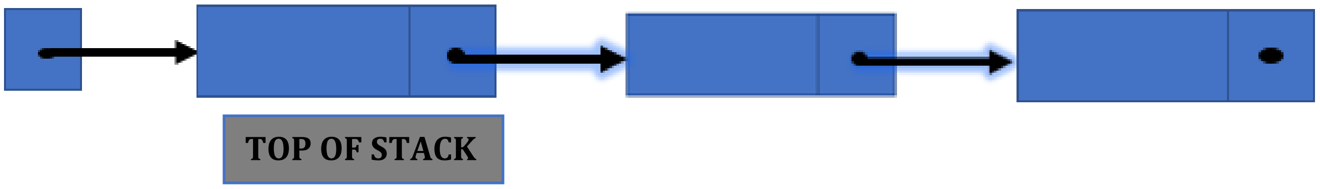
Tumpalan, John Karl B.
2018 – 02385 CMSC 123 – U5L

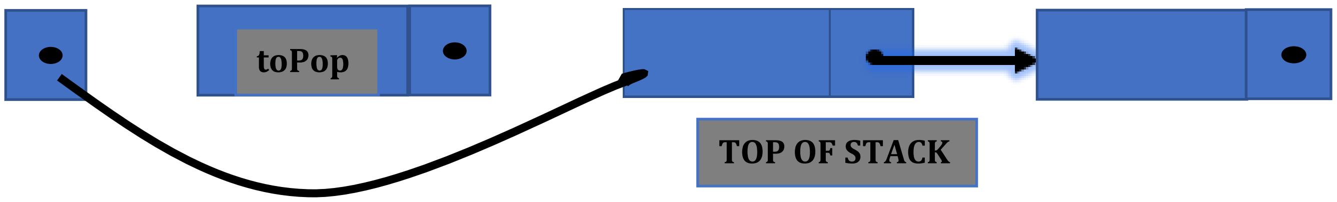
POP

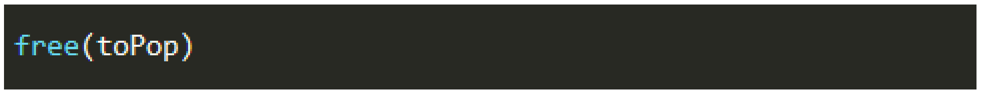
For popping, we just need to delete at head because we only add and delete nodes on one side if we are working on a stack. It is much easier to pop at head as we won't need another pointer for maintaining the linkage of the list. Instead, we can use the head pointer.



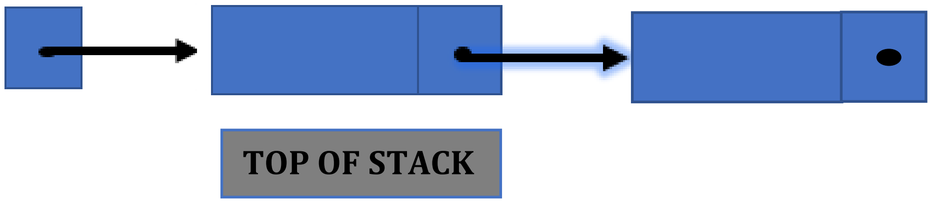
```
NODE *topOfStack = head;
NODE *toPop;

void pop(NODE **head){
    if((*head) != NULL){
        toPop = (*head);
        (*head) = (*head)->next;
        topOfStack = (*head); //updating top of stack
    }
```







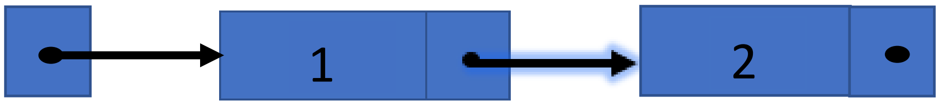
```
free(toPop)
```

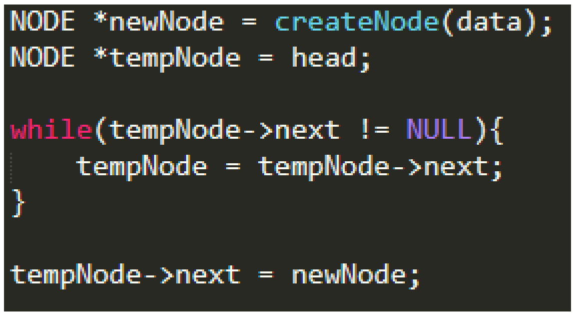


Tumpalan, John Karl B.
2018 – 02385 CMSC 123 – U5L

QUEUES

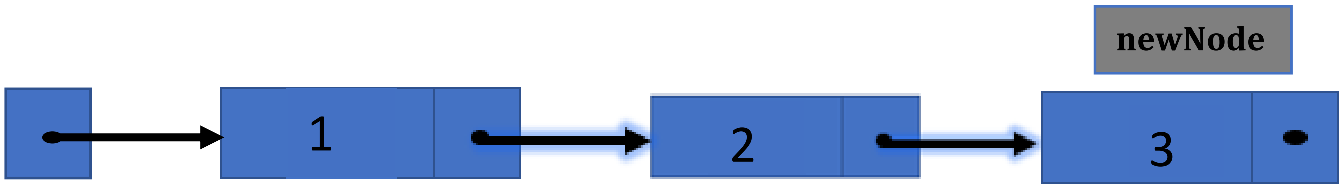
Enqueuing



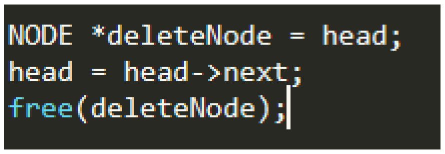


```
NODE *newNode = createNode(data);
NODE *tempNode = head;

while(tempNode->next != NULL){
    tempNode = tempNode->next;
}

tempNode->next = newNode;
```

Enqueuing only happens on one side of the linked list. It is more efficient to enqueue at the tail than on the head.

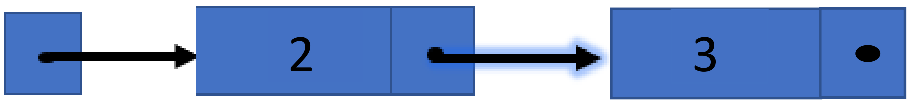


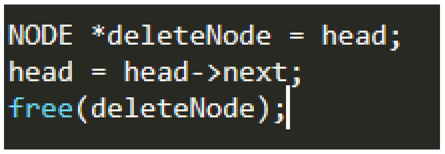
Dequeuing



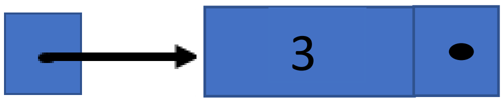
```
NODE *deleteNode = head;
head = head->next;
free(deleteNode);
```

Dequeuing is more efficient at head because if it is done at the tail, we will need additional pointer to keep track on the new tail if the current tail has been deleted or dequeued.





```
NODE *deleteNode = head;
head = head->next;
free(deleteNode);
```



Tumpalan, John Karl B.
2018 – 02385 CMSC 123 – U5L

Common applications:

Queue:

As an ADT that uses FIFO data structure, it can be used to create programs with similar input-output process like bank queueing systems to organize waiting clients or for Call Center phone systems to hold calls until a Customer Service Representative is free.

According to Birch, it can also be used on Breadth First Search (BFS) wherein a queue can be used to store nodes that are to be processed. The adjacent node is added to a queue during processing to allow processing in the same order that they are viewed.

Additionally, the LRU cache implementation is essentially a queue structure, whilst a queue is not efficient enough on its own, one can be used to keep the order of cache items.

Stack:

The simplest application of a stack is to reverse a word. You push a given word to the stack and then create a new word by popping letter by letter from the stack. It also works in way like how “Undo” features work, or like the feature in a doubly linked list wherein we just change the current pointer to the ‘prev’ pointer to revert the changes that has been made.

It can also be used on Expression Conversion like Infix to Postfix, Postfix to Prefix, etc.

According to Sravani (2019), more advanced application of stacks can be used in Backtracking (game playing, finding paths, exhaustive searching), Memory management, run-time environment for nested language features, language processing and Parsing.

Tumpalan, John Karl B.
2018 – 02385 CMSC 123 – U5L

References:

Sravani, Btech Computer Science Engineering, National Institute of Technology, Andhra Pradesh,Tadepalligudem (2019) retrieved from https://www.quora.com/What-are-the-real-applications-of-stacks-and-queues

Birch, J. (2019), Data Structures: Stacks & Queues. Retrieved from https://medium.com/@hitherejoe/data-structures-stacks-queues-a3b3591c8cb0

https://www.studytonight.com/data-structures/queue-data-structure

Code snippets on Array Implementation are guided by Sir Arian Jacildo's Lecture Slides on Array Implementation (CMSC 123)