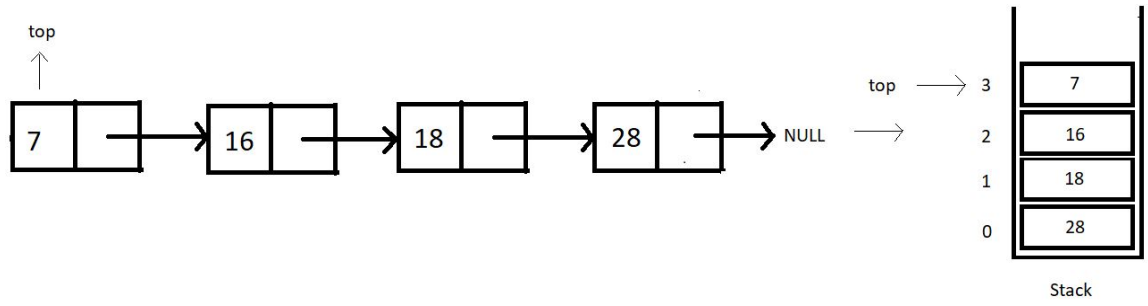
peek(), stackTop() and Other Operations on Stack Using Linked List (with C Code)

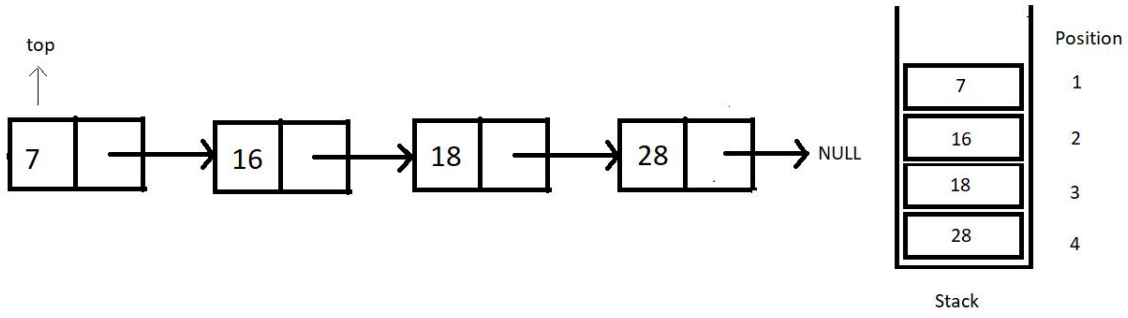
codewithharry.com/videos/data-structures-and-algorithms-in-hindi-31

In the last tutorial, we learned to implement stacks using linked lists. We saw how efficiently we can push and pop elements in a stack-linked list. We saw a few other operations, *isEmpty, isFull, traversal*. Today, we will cover the remaining operations. They are: peek, stackTop, etc.

Similar to what we did last time, we will first understand the algorithm behind the operations, followed by the coding section. Let's see them individually, but before that, let's have an example illustration of the stack we'll go into within today's tutorial.



1. peek: This operation is meant to return the element at a given position. Do mind that the position of an element is not the same as the index of an element. In fact, there is nothing as an index in a linked list. Refer to the illustration below.



Peeking in a stack linked list is not as efficient as when we worked with arrays. Peeking in a linked list takes O(n) because it first traverses to the position where we want to peek in. So, we'll just have to move to that node and return its data.

2. stackTop: This operation just returns the topmost value in the stack. That is, it just returns the data member of the *top* pointer.

3. stackBottom:

I will leave the last operation, stackBottom, for your homework. Try implementing this on your own, and let me know if you could. You should be able to code this since we have covered the concepts already in the stack arrays.

So, these were the only operations we had in mind to discuss with you all. You will come across several variations of these. Nevertheless, you are intelligent enough to be able to change your codes if necessary. We'll now move to our editors to code the operations we discussed today. I have attached the code snippet below. Refer to them while you code:

Understanding the code snippet below:

1. Copy everything we did in the last tutorial. This will save us some time. It will also prevent repetitions in the course. Our main focus for today is to discuss these three operations. So, creating the stack and other operations can be ignored since they have already been covered.

2. We'll start with the *peek* function.

3. peek():

- Create an integer function *peek*, and pass the position you want to peek in as a parameter.
- Since we have made the stack pointer global, we should not use that pointer to traverse; otherwise, we will lose the pointer to the top node. Rather create a new struct Node pointer *ptr* and give it the value of top.
- Run a loop from 0 to pos-1, since we are already at the first position.
- If our pointer reaches NULL at some point, we must have reached the last node, and the position asked was beyond the available positions, hence breaking the loop.
- If the current pointer found the position and it is not equal to NULL, return the data at that node, else -1.

```
int peek(int pos){
    struct Node* ptr = top;
    for (int i = 0; (i < pos-1 && ptr!=NULL); i++)
    {
        ptr = ptr->next;
    }
    if(ptr!=NULL){
        return ptr->data;
    }
    else{
        return -1;
    }
}
```

Code Snippet 1: Implementing peek function

4. stackTop():

- Create an integer function *stackTop*, and we are no longer passing any parameter since the top pointer is declared globally.
- Simply return the data member of the struct Node pointer *top,* and that's it.

```
int stackTop(){
    return top->data;
}
```

Code Snippet 2: Implementing stackTop function

Here is the whole source code:

```
#include<stdio.h>
#include<stdlib.h>

struct Node{
    int data;
    struct Node * next;
};

struct Node* top = NULL;

void linkedListTraversal(struct Node *ptr)
{
    while (ptr != NULL)
    {
        printf("Element: %d\n", ptr->data);
        ptr = ptr->next;
    }
}

int isEmpty(struct Node* top){
    if (top==NULL){
        return 1;
    }
    else{
        return 0;
    }
}

int isFull(struct Node* top){
    struct Node* p = (struct Node*)malloc(sizeof(struct Node));
    if(p==NULL){
        return 1;
    }
    else{
        return 0;
    }
}

struct Node* push(struct Node* top, int x){
    if(isFull(top)){
        printf("Stack Overflow\n");
    }
    else{
        struct Node* n = (struct Node*) malloc(sizeof(struct Node));
        n->data = x;
        n->next = top;
        top = n;
        return top;
    }
}

int pop(struct Node* tp){
    if(isEmpty(tp)){
        printf("Stack Underflow\n");
    }
    else{
        struct Node* n = tp;
        top = (tp)->next;
        int x = n->data;
```

```
        free(n);
        return x;
    }
}

int peek(int pos){
    struct Node* ptr = top;
    for (int i = 0; (i < pos-1 && ptr!=NULL); i++)
    {
        ptr = ptr->next;
    }
    if(ptr!=NULL){
        return ptr->data;
    }
    else{
        return -1;
    }
}

int main(){
    top = push(top, 28);
    top = push(top, 18);
    top = push(top, 15);
    top = push(top, 7);

    linkedListTraversal(top);
    for (int i = 1; i <= 4; i++)
    {
        printf("Value at position %d is : %d\n", i, peek(i));
    }
    return 0;
}
```

Code Snippet 3: Using peek function

Let's now push some elements into the stack and see if the operations work all good.

```
    top = push(top, 28);
    top = push(top, 18);
    top = push(top, 15);
    top = push(top, 7);
```

Code Snippet 4: Using push function to put some elements inside the stack

Since we have pushed the elements, we can call our peek function in a loop, printing the whole array.

```
    for (int i = 1; i <= 4; i++)
    {
        printf("Value at position %d is : %d\n", i, peek(i));
    }
```

Code Snippet 5: Using peek function to print the whole stack

The output we received was:

```
Value at position 1 is : 7
Value at position 2 is : 15
Value at position 3 is : 18
Value at position 4 is : 28
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 1: Output of the above program

Our peek operation worked accurately. I have left stackBottom up to you all. Don't mind sending me your doubts regarding the course. We are now planning to move to an application of these stacks called parenthesis matching.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial, where we'll see an interesting topic, **parenthesis matching** in stacks. Till then, keep coding.