Lecture 9

This starts our journey to another data structure 'stack'. Dictionary meaning of stack is pile (쌓아 올린 더미, 퇴적, 산더미) of books or pile of CDs.

In computer science, Stack is a data structure based on the principle of LIFO (**L**ast **I**n **F**irst **O**ut). Stack is extensively(널리, 광범위하게) used in computer science.

Definition Of Stack

Stack is a data structure based on the principle of LIFO. LIFO means, that the data item that is inserted last is the first one to come out.

We define 2 operations on Stack, push and pop.
->PUSH means to insert data item inside stack. i.e. Increment top by 1 and add data value.
->POP means to delete data item from the stack. i.e. Get the value of topmost data and decrement top by1.

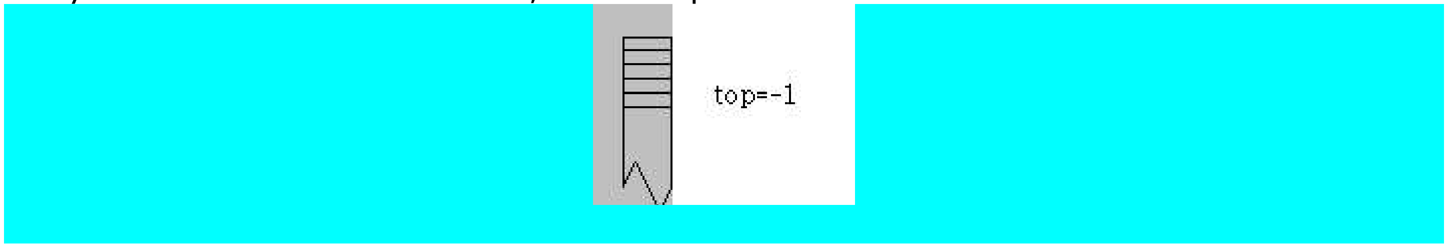
There is a variable '**top**', which stores the position of topmost(위치·지위 등이> 최고의, 최상(급)의) data element in the stack.

Now, try to understand this fully. To understand stack, we need to implement (도구, 용구) it. Stack can be implemented using an array and using a linked list. We will first implement stack using an array and then using a linked list.

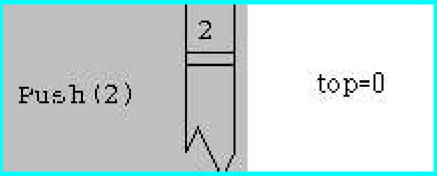
There is another operation, Peep. Peep means, To get the value of topmost item from the stack.
Difference between Pop and Peep is: Pop gets the value of topmost data item from the stack and decrement top by 1. Whereas, Peep only get the value of topmost data item from the stack but dont decrement top by 1.

Implementation of Stack using an Array

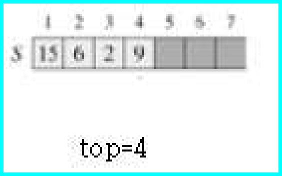
Play presentation to understand working of stack.
Imagine an empty array of size n, and a variable top which is initially set to -1. Why top is -1? As I told you, top stores the position of topmost data item in the stack, Since in an emty stack there is no data item, hence top=-1.



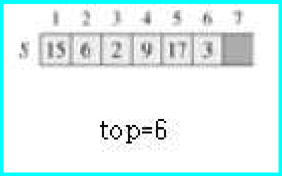
Data value 2 inserted now, will be inserted at position top+1 i.e top = 0.
Push(2)



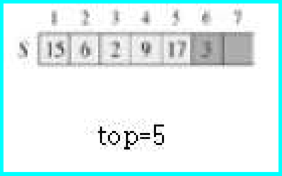
This is a list with 4 elements, hence top=4.
Push(15), Push(6), Push(2), Push(9)



If in the above list, we add 17 and then 3,
Push(17), Push(3)



If we perform delete operation i.e. pop on above list.
x= Pop(), x = 3.



Can I insert more than 7 elements in the above stack?NO, because the above stack with 7 element is full, it cannot accommodate (숙박시키다) any more elements.

Can I delete an element from an empty stack? No.

Summary:
A stack is a data structure based on the principle of LIFO. **All the insertions and deletion are done from only one end (from top end of stack)**.

Push operation means insert an element. Pop operation means delete an element. We cannot insert an element in a full stack. We cannot delete an element from an empty stack.

Now we will write a program to implement a stack.

Step1: Delcare an array, which will behave as a stack.

```
                int s[10]; // s is an array which we will implement as a stack.
                int top=-1;
```

Step2: We will make a menu, which will present user with 5 options: Push, Pop, Peep, Print, Exit.

```
        do
        {
                printf("1. Push an Element");
                printf("\n2. Pop an Element");
                printf("\n3.Peep an Element");
                printf("\n4. Print");
                printf("\n5. Exit");
                printf("\n Enter your choice");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: push(); break;
                        case 2: pop(); break
                        case 3: peep(); break;
                        case 4: print(); break;
                        case 5: break;
                        default: printf("This is none of the choices");
                }
        }while(choice!=5)
```

Step3: Next, we are only supposed to write functions for push, pop and print.
First I will start with push()
To push an element, I will check if stack is Full or not.

```
                void push()
                {
                        If (stackFull())
                                printf("Stack is full");
                        else
                        {
                                printf("Enter data value");
                                scanf("%d", &val);
                                top = top + 1;
                                s[top] = val;
                        }
                }

                boolean stackFull()
                {
                        if (top==9)
                                return 1;
                        else
                                return 0;
                }
```

Step4: Write the code for pop operation

```
        void pop()
        {
                if (stackEmpty())
                        printf("Cannot delete from an empty stack");
                else
                {
                        int x = s[top];
                        top = top – 1;
                        printf("element deleted is %d", x);
                }
        }

        boolean stackEmpty()
        {
                if (top==-1)
                        return 1;
                else
                        return 0;
        }
```

Step5: As I told you, Peep returns the value of topmost element and doesnot decrement top by 1.

```
void peep()
{
        if (stackEmpty())
                        printf("Cannot delete from an empty stack");
                else
                {
                        int x = s[top];
                        printf("element deleted is %d", x);
```

```
                    }
            }
```

Step6: Write the code for print operation

```
                    void print()
                    {
                        for(int i=0;i<=top;i++)
                            printf("\n%d",s[i]);
                    }
```

We are going to learn about applications of stack. There are many applications of stack. Recursion is implemented internally through stacks. But we will discuss another use of stack, which are

1. infix to prefix
2. infix to postfix
3. postfix to prefix
4. postfix to infix
5. evaluation of postfix.

We will start with evaluation of postfix,as this one is quiet easy to understand. But before starting with all this, let us understand what is infix, prefix and postfix.

An expression in which operator is included in between operands is called as infix expression. For example a+b or a+b * **c, a *** b+c /d - e * f / g.
An expression in which operator appear before operands is called as prefix expression. For example +ab, +a * bc.

An expression in which operator appear after operands is called as postfix expression. For example ab+ bc * a+

Evalution of Postifx expression

The best way to understand this is through flash presentation.

So, we saw in the flash presenataion that, we have with us a string which consisit of a postfix expression. Suppose 'str' is the string which contains postfix expression.

Step1: Our first step is to find the length of string and store it in a variable length.

```
            length = str.length();
```

Step2: Start a for loop from 0 until the length.

```
                    for(int i=0;i<length;i++) {
```

Step3: Scan through every character, if it is a operand, i.e. '1', then push 1 into the stack. If it is '5', then push 5 into the stack.

```
                    switch (str[i])
                    {

                         case '0': push(str[i] – '0'); break;
                         case '1': push(str[i] – '0'); break; // '1' – '0' is 1.
                         case '2': push(str[i] – '0'); break; // '2' – '0' is 2.
                         case '3': push(str[i] – '0'); break; // '3' – '0' is 3.
                         case '4': push(str[i] – '0'); break; // '4' – '0' is 4.
                         case '5': push(str[i] – '0'); break; // '5' – '0' is 5.
                         case '6': push(str[i] – '0'); break; // '6' – '0' is 6.
                         case '7': push(str[i] – '0'); break; // '7' – '0' is 7.
                         case '8': push(str[i] – '0'); break; // '8' – '0' is 8.
                         case '9': push(str[i] – '0'); break; // '9' – '0' is 9.
```

Step4: If it is a operator, perform 2 pop operations, store them in variables 'a' and 'b' . Perform the required operation on those popped operands. I am gonna put all this in function evaluate.

```
                         case '+': evaluate(str[i]);break;
                         case '-': evaluate(str[i]);break;
                         case '*': evaluate(str[i]);break;
                         case '/': evaluate(str[i]);break;
               }

void evaluate (char opr)
{
     int a = pop();
     int b = pop();
     int res;
     switch(opr)
     {
          case '+': res = b + a; break;
          case '-' : res = b – a; break;
          case '*': res = b * a; break;
          case '/': res = b / a; break;
     }
     push(res);
}
```

Step5: The result is contained inside the stack. Just pop it from the stack and display.

```
printf(“Result of evaluation is %d”, pop());
```

<u>Infix expression to Prefix expression</u>

To understand, how to convert infix to prefix, watch the presentation. Let me summarize what we have learnt from the presenataion. Each operator has a priority.

'(' has lowest priority of 0.
'+' and '-' has equal priority of 1.
'*' and '/' has equal priority of 2.
'$' has priority of 3.

Our first step is to reverse a string. Then find the length of string, and scan the string from 0 until the length. If character scanned is an

1 Operand, like '0', '1': then write it in the prefix string.

2 Operator,

  2.1 if stack is empty, then push the operator.

  2.2 if stack is not empty

    2.2.1 If it is a '(', push it into the stack. (Despite the fact that '(' has lowest priority, but '(' character is always pushed in.)

    2.2.2 Check if the priority of topmost operator <= priority of current operator, in that case, just push the current operator.

    2.2.3 Check if the priority of topmost operator > priority of current operator, in that case, Keep on popping until you find an operator of lower priority in the stack. Then push current operator.

  2.3 If it is a '(', pop all the operators from the stack until a ')' is found.