## *Stack Mechanism Discussion with Time Complexity*

# 6. *Peek Operation*

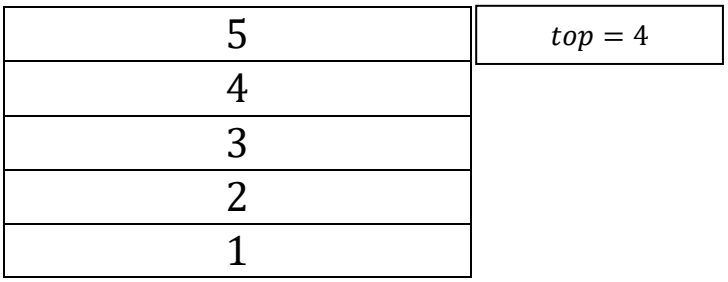
```
int peek(Stack st)
{
    if (!isEmpty(st))
    {
        return st.s[st.top];
    }
    return -1;
}
……
case 7:

 cout << "The element at the top is " << peek(stck) << endl;
break;
```

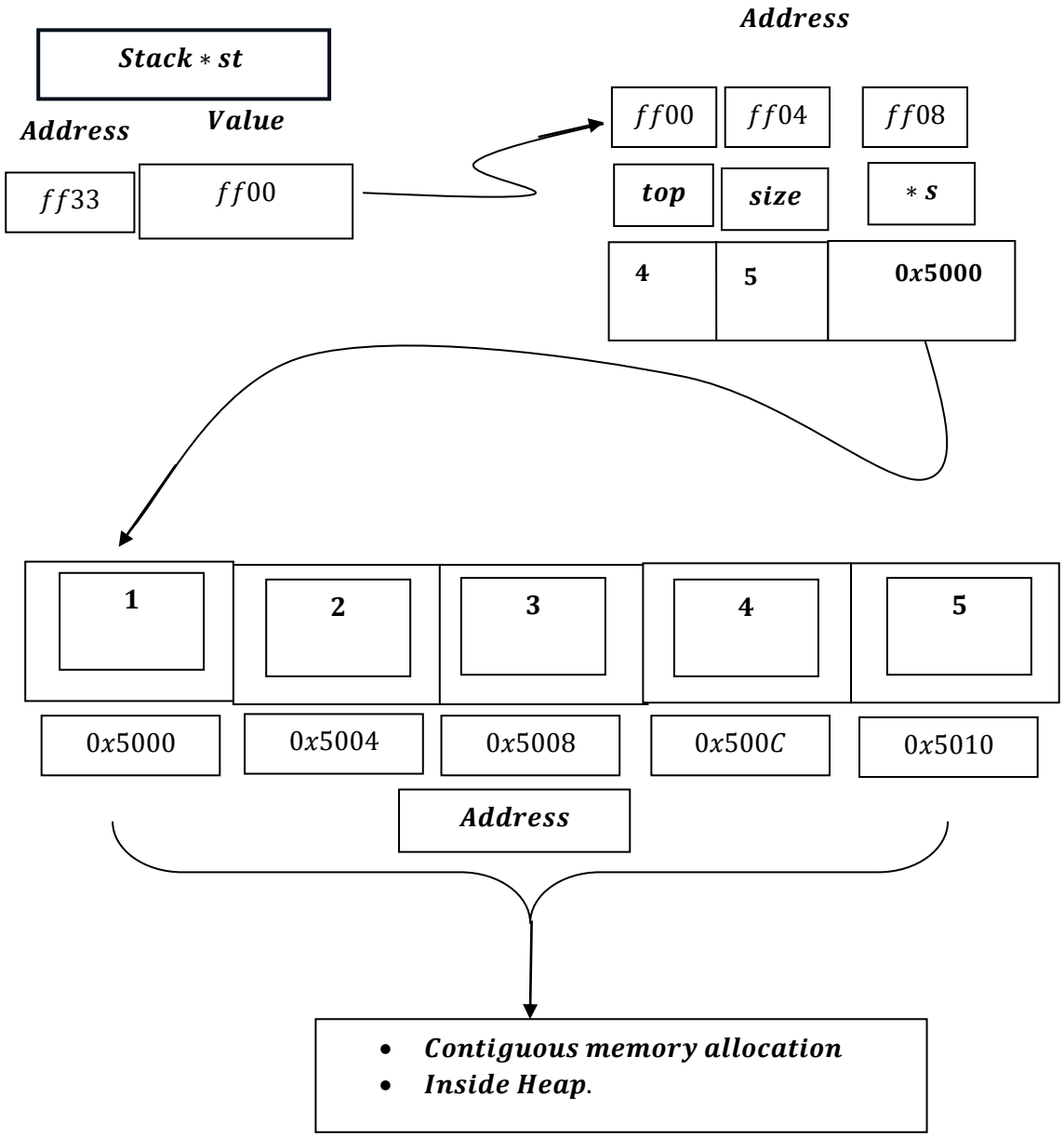
***If stack is not empty , then:***

***Return st. s[st. Top];***

***i. e. return the value stored in the Top of the Stack. This is the peek operation.***

***Else return − 1 , which means stack is empty.***


5
4
3
2
1
$top = 4$


Stack * st
Address
Value
ff33
ff00
Address
ff00
ff04
ff08
top
size
* s
4
5
0x5000
1
2
3
4
5
0x5000
0x5004
0x5008
0x500C
0x5010
Address
Contiguous memory allocation
Inside Heap.

$$st \rightarrow s[st \rightarrow top] \Longrightarrow s[4]$$

$$return\ value\ stored\ in\ BaseAddress + (index\ \times size\ of\ int)$$

$$i.e. return\ value\ stored\ in\ 0x5000 + (4 \times 4\ bytes)$$

$$i.e. return\ value\ stored\ in\ 0x5000 + 16$$

$$i.e. return\ value\ stored\ in\ 0x5000 + 10$$

$$i.e. return\ value\ stored\ in\ 0x5010$$

$$\Longrightarrow return\ 5$$

$$Hence\ value\ of\ Top\ is\ : 5.$$

$$Similarly,$$

$$\rightarrow When\ top = 3\ , element\ returned : 4$$

$$\rightarrow When\ top = 2\ , element\ returned : 3$$

$$\rightarrow When\ top = 1\ , element\ returned : 2$$

$$\rightarrow When\ top = 0\ , element\ returned : 1$$

$$\rightarrow When\ top = -1, it\ returns - 1\ means\ stack\ is\ empty.$$

# *Time Complexity*

```
     int peek(Stack st)
{
    if (!isEmpty(st))
    {
        return st.s[st.top];
    }
    return -1;
}
```

**1.** ***Function overhead due to function call which includes creation of stack frame for the function peek() takes constant amount of time : O(1).***

**2.** ***isEmpty() function return 1 or 0 , when return 0 is false , when return 1 its true , here ! isEmpty() generally its considered when isEmpty() returns 0 i.e. false then for if condition its become true i.e. stack is not empty. if condition check takes constant time : O(1).***

***Then inner statement runs:***

$$\rightarrow \textbf{Runs return statement: } st.s[st.top] \begin{bmatrix} \textbf{Retuns value stored at Top takes constant time } O(1). \end{bmatrix}$$

***Else , when isEmpty() return 1 i.e. List is empty then ! isEmpty() makes the condition false and returns*** $-1$.

***This takes constant time i.e.*** $O(1)$.

***Hence , Time Complexity is :*** $O(1) + \big(O(1) + O(1)\big) + O(1)$

$= O(1)$

***i.e. constant `c` time complexity.***

---

***When*** `peek()` ***is called:***

1. ***A stack frame (activation record) is created.***
2. ***The return address is stored.***
3. ***Parameters are handled.***
4. ***Control jumps to the function.***
5. ***After execution, the stack frame is removed.***

***These steps:***

- ***Do not depend on input size*** $(n)$.
- ***Take a fixed number of CPU instructions.***

$$\textbf{Hence, Function Call Overhead} = O(1).$$

***The function call overhead, including stack frame creation and destruction for*** `peek()`***, takes constant time*** $O(1)$***, since it does not depend on the input size.***

***Stack Data structure basically shows how call stack frame works i.e. in LIFO method , but in Stack Data Structure we manually do push() and pop() , where in call stack frame, it creates stack frame for entire function containing local variables , parameters , return address and saved registers and adds or push another stack frame if called again and pops whole stack frame automatically thus different from Stack datastructure , yet conceptually similar.***

---