Stack Mechanism Discussion with Time Complexity

7. Traverse Operation

```
void traverseStack(int stack[])
{

    if (top == -1)
    {

        cout << "Stack is empty" << endl;
        return;
    }

    for (int i = top; i >= 0; i--)
    {

        cout << stack[i] << " ";
    }
    cout << endl;
}
…..
case 4:
 traverseStack(stack);
 break;
```

$\boldsymbol{If\ (top = -1)\ then}$:

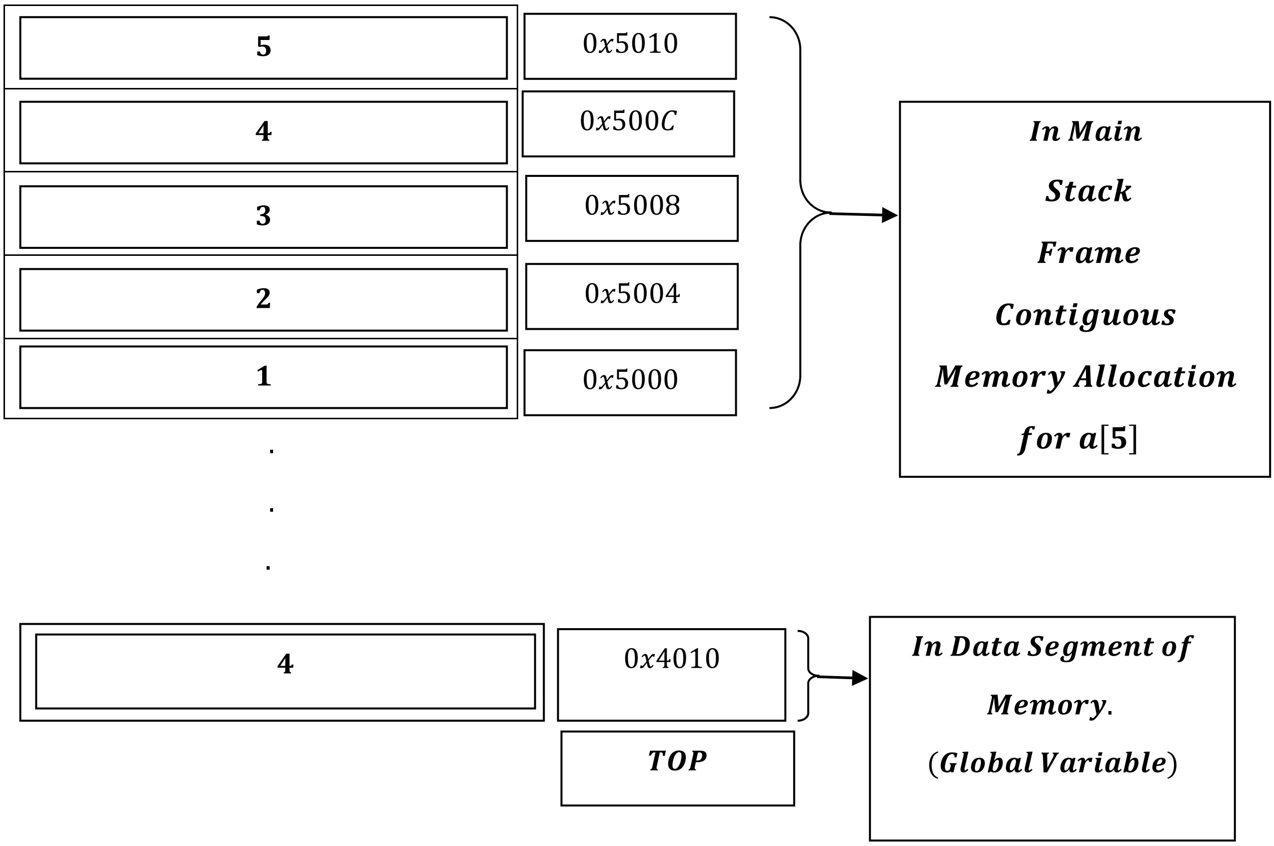
$\boldsymbol{print\ output}$: ``$\boldsymbol{Stack\ is\ empty}$``.

$\boldsymbol{return\ void\ and\ exit}$.

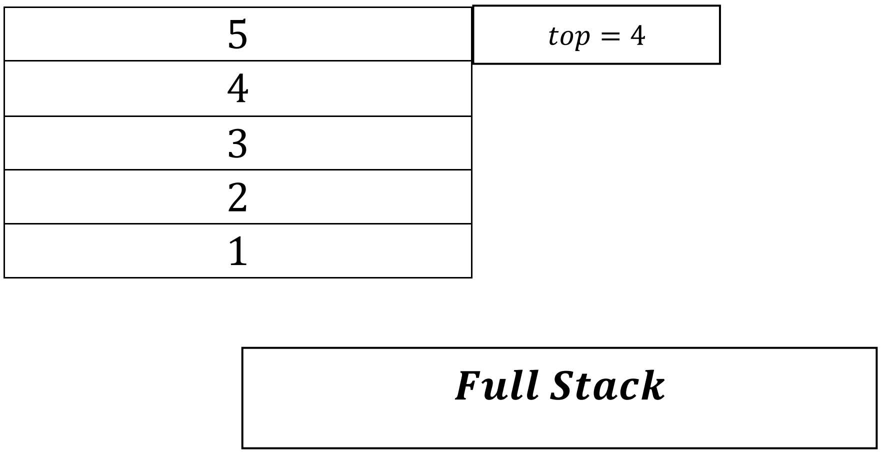
$\boldsymbol{Else}$

[$\boldsymbol{Stack\ Traversal}$]

Full Stack



This is Physical Demonstration



This is Logical Demonstration

$\underline{\textit{Stack Traversal}}$

$As\ now, i = Top = 4\ \ and\ i = 4 \geq 0$

$Stack[i = 4].$

$\Rightarrow Stack + 4.$ $[Stack + 4, represents\ contiguous\ memory\ allocation]$

$\Rightarrow Base\ Address + 4[index] \times 4bytes.$

$\Rightarrow 0x5000 + 16.$

$\Rightarrow 0x5000 + 10. [16_{10} \approx 10_{16}]$

$\Rightarrow 0x5010$

$Print\ ``value\ stored\ at\ address : 0x5010 \rightarrow 5``.$

$Now, i -- \Rightarrow i = i - 1 \Rightarrow i = 4 - 1 = 3. [Post\ Decrement]$

$i = 3\ \ and\ i = 3 \geq 0$

$Stack[i = 3].$

$\Rightarrow Stack + 3.$ $[Stack + 3, represents\ contiguous\ memory\ allocation]$

$\Rightarrow Base\ Address + 3[index] \times 4bytes.$

$\Rightarrow 0x5000 + 12.$

$\Rightarrow 0x5000 + C. [12_{10} \approx C_{16}]$

$\Rightarrow 0x500C$

$Print\ ``value\ stored\ at\ address : 0x500C \rightarrow 4``.$

$Now, i -- \Rightarrow i = i - 1 \Rightarrow i = 3 - 1 = 2. [Post\ Decrement]$

$i = 2\ \ and\ i = 2 \geq 0$

$Stack[i = 2].$

$\Rightarrow Stack + 2.$ $[Stack + 2, represents\ contiguous\ memory\ allocation]$

$\Rightarrow Base\ Address + 2[index] \times 4bytes.$

$\Rightarrow 0x5000 + 8.$

$\Rightarrow 0x5008.$

$Print$ ``$value\ stored\ at\ address : 0x5008 \rightarrow 3$``.

$Now, i -- \Rightarrow i = i - 1 \Rightarrow i = 2 - 1 = 1. [Post\ Decrement]$

$i = 1\ \ and\ i = 1 \geq 0$

$Stack[i = 1].$

$\Rightarrow Stack + 1.$ $[Stack + 1, represents\ contiguous\ memory\ allocation]$

$\Rightarrow Base\ Address + 1[index] \times 4bytes.$

$\Rightarrow 0x5000 + 4.$

$\Rightarrow 0x5004.$

$Print$ ``$value\ stored\ at\ address : 0x5004 \rightarrow 2$``.

$Now, i -- \Rightarrow i = i - 1 \Rightarrow i = 1 - 1 = 0. [Post\ Decrement]$

$i = 0 \ \ and\ i = 0 \geq 0$

$Stack[i = 0].$

$\Longrightarrow Stack + 0.$ $[\,Stack + 0, represents\ contiguous\ memory\ allocation]$

$\Longrightarrow Base\ Address + 0[index] \times 4bytes.$

$\Longrightarrow 0x5000 + 0.$

$\Longrightarrow 0x5000.$

$Print\ ``value\ stored\ at\ address : 0x5000 \rightarrow 1``.$

$Now, i - - \Longrightarrow i = i - 1 \Longrightarrow i = 0 - 1 = -1. [Post\ Decrement]$

$Now, i = -1\ and\ i = -1 \geq 0$, $loop\ condition\ becomes\ false$

$and\ loop\ exists.$

$$Time\ Complexity$$

```
void traverseStack(int stack[])
{

    if (top == -1)
    {

        cout << "Stack is empty" << endl;
        return;

    }

    for (int i = top; i >= 0; i--)
    {

        cout << stack[i] << " ";
    }
    cout << endl;
}
```

$\mathbf{1.}\, \boldsymbol{Function\ overhead\ due\ to\ function\ call\ which\ includes\ creation\ of\ stack\ frame\ for\ the\ function\ traverseStack()\ takes\ constant\ amount\ of\ time:O(1).}$

$\mathbf{2.}\, \boldsymbol{if\ (top = -1) True\ [Takes\ constant\ `c`\ time:O(1)] then}:$

$$\rightarrow \boldsymbol{Print\ ``Stack\ is\ empty."}\ \begin{bmatrix} \boldsymbol{Takes\ constant\ `c`\ time}: \\ \boldsymbol{O(1)} \end{bmatrix}$$

$$\rightarrow \boldsymbol{return\ void\ and\ exit.}\ \begin{bmatrix} \boldsymbol{Takes\ constant\ `c`\ time}: \\ \boldsymbol{O(1)} \end{bmatrix}$$

$\mathbf{3.}\,\boldsymbol{if\ (top = -1) False\ then}$:

$\mathbf{3.}\,\boldsymbol{a.\ for\ loop\ runs\ n-1\ to\ 0\ times}$:

$\boldsymbol{inner\ statement\ `stack[i]`\ runs\ `n`\ times i.e.}$

$\boldsymbol{[\,0+1+2+\cdots+n-1\ \Rightarrow 1+2+\cdots+n = n\ times = O(n).]}$

$\mathbf{3.}\,\boldsymbol{b.\ Print\ `endl`\ i.e. newline\ takes\ O(1) time.}$

$\rightarrow \boldsymbol{When\ if\ condition\ is\ true}: [\boldsymbol{Best\ Case}\,]$

$$\boldsymbol{Time\ Complexity}: \boldsymbol{O(1) + \big(O(1) + \big(O(1) + O(1)\big)\,\big) = \Omega(1)}$$

$\rightarrow \boldsymbol{When\ if\ condition\ is\ false}: [\boldsymbol{Worst\ Case}]$

$$\boldsymbol{Time\ Complexity}: \boldsymbol{O(1) + \big(O(1) + \big(O(n) + O(1)\big)\,\big) = O(n)}$$
