- **الگوریتم** In order tree walk **را به صورت یک الگوریتم غیر بازگشتی بیان کنید.**

در روابط بازگشتی ما از حافظه استک استفاده می کردیم. به این صورت که برای هر نود ابتدا تابع را برای فرزند چپ صدا می زدیم و سپس مقدار خود نود را چاپ می کردیم و بعد از آن برای فرزند راست تابع را صدا می زدیم و دوباره به سراغ پدر آن نود می رفتیم و .....

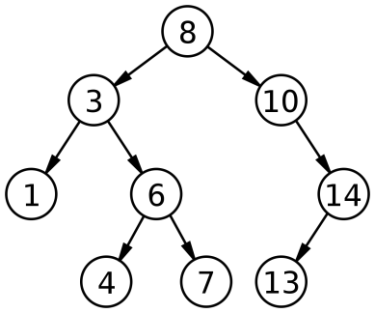
در اینجا نیز هدف ما همین است. برای پیاده سازی in order tree walk **به صورت غیر بازگشتی ما از یک پشته (** Stack **) استفاده می کنیم. به این صورت که اگر نودی دارای فرزند چپ بود به داخل پشته اضافه می شود و نشانگر** current **به سراغ فرزند چپ آن می رود. بعد از اینکه این روند تا رسیدن به یک برگ ادامه یافت ، شروع می کنیم و از پشته داده ها را میگیریم و در صورت داشتن فرزند راست نشانگر** current **را به سراغ فرزند راست برده و فرآیند را تکرار می کنیم و اگر فرزند راست نداشت تا زمانی که پشته خالی نشده از آن عضو خارج کرده و داشتن فرزند راست را بررسی می کنیم.**

شبه کد آن به این صورت است:

```
Stack buffer[n]; // Create a stack for keeping the nodes addresses

inorder_tree_walk(Tree A)
{
    current = root[A]; // Start from head

    if (current == nullptr)
    {
        return; // Empty tree
    }

    while (true)
    {

        if (left[current] != nullptr)
        {
            buffer.push(current); // Keep in stack
            current = left[current]; // Switch to left child
            continue;
        }

        print(key[current]); // Print current

        if (right[current] != nullptr)
        {
            current = right[current]; // Switch to right child
            continue;
        }
```

```
        if (buffer == empty)
        {
            break; // Exit condition
        }

        while (buffer != empty)
        {
            current = buffer.pop(); // Buffer clearing
            print(key[current]);

            if (right[current] != nullptr)
            {
                current = right[current];
                break;
            }
        }
    }
}
```

مثال : درخت زیر را در نظر بگیرید.




```
>> stack = empty
>> current = 8
Left[current] != nullptr :
       Stack.push(current)
       Current = left[current]
       Continue
>> stack = 8
>> current = 3
Left[current] != nullptr :
       Stack.push(current)
       Current = left[current]
       Continue
>> stack = 3 | 8
>> current = 1
```

```
left[current] == nullptr
** Print(1)
right[current] == nullptr
stack != empty :
	current = stack.pop() => current = 3
	** Print(3)
	Rignt[current] != nullptr :
		Current = right[current]
		Continue
>> stack = 8
>> current = 6
>> prints = 1 | 3
Left[current] != nullptr :
	Stack.push(current)
	Current = left[current]
	Continue
>> stack = 6 | 8
>> current = 4
>> prints = 1 | 3
left[current] == nullptr
** Print(4)
right[current] == nullptr
stack != empty :
	current = stack.pop() => current = 6
	** Print(6)
	Rignt[current] != nullptr :
		Current = right[current]
		Continue
>> stack = 8
>> current = 7
>> prints = 1 | 3 | 4 | 6
left[current] == nullptr
** Print(7)
right[current] == nullptr
stack != empty :
	current = stack.pop() => current = 8
	** Print(8)
	Rignt[current] != nullptr :
		Current = right[current]
		Continue
>> stack = empty     >> current = 10     >> prints = 1 | 3 | 4 | 6 | 7 | 8
… … … …
```

امیرحسین نجفی زاده ۹۸۳۱۰۶۵