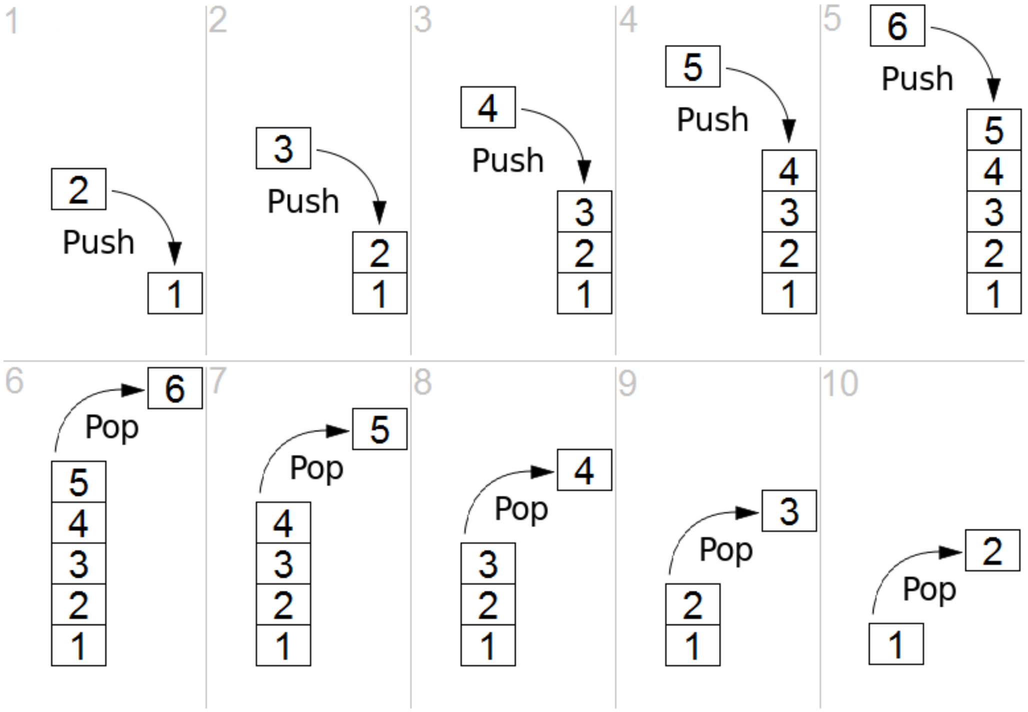
যেকোনো ডাটা স্ট্রাকচার কোর্সে একদম শুরুর দিকে যেসব ডাটা স্ট্রাকচার পড়ানো হয় তার মধ্যে স্ট্যাক অন্যতম। স্ট্যাককে বলা হয় **LIFO** বা লাস্ট-ইন-ফার্স্ট-আউট ডাটা স্ট্রাকচার। তুমি এভাবে চিন্তা করতে পারো, তোমার কাছে অনেকগুলো বই একটার উপর আরেকটা সাজানো আছে, তুমি চাইলে সবার উপরের বইটা সরিয়ে ফেলতে পারো(Pop), অথবা সবার উপরে আরেকটা বই রাখতে পারো(Push)। এটা হলো বইয়ের একটা স্ট্যাক। তুমি এই স্ট্যাকের উপরে ছাড়া কোনো জায়গায় বই ঢুকাতে পারবে না, উপরের বই ছাড়া কোনো বই সরাতে পারবে না, এগুলো করলে সেটা আর স্ট্যাক থাকবে না।

স্ট্যাক হলো কিছু বস্তুর(এলিমেন্ট) একটা সংগ্রহ। এখানে দুইরকম অপারেশন করা যায়:

- Push(new_element): স্ট্যাকের উপরে নতুন বস্তুটা রাখো। যদি শুরুতে স্ট্যাকটা খালি হয়, তাহলে প্রথম জায়গায় বস্তুটা রাখতে হবে।
- Pop(): স্ট্যাকের উপরের বস্তুটা সরিয়ে ফেলো। যদি শুরুতে স্ট্যাকটা খালি হয় তাহলে এই অপারেশনটা করা সম্ভব না।



চিত্র ১: স্ট্যাক এ পুশ এবং পপ অপারেশন (সোর্স: উইকিপিডিয়া)

চিত্র ১ এ দেখা যাচ্ছে কিভাবে স্ট্যাকে পুশ এবং পপ অপারেশন করতে হয়। স্ট্যাকে আরেকটা অপারেশন থাকতে পারে Peek(), যেটা ব্যবহার করে সবার উপরের বস্তুটা কি জানা যাবে বস্তুটা পপ না করেই।

স্ট্যাকের আকার যদি সীমিত হয়, যেমন যদি ১০টার বেশি বস্তু রাখার সামর্থ কোনো স্ট্যাকের না থাকে তাহলে ১১তম বস্তু পুশ করলে স্ট্যাকটা “ওভারফ্লো” স্টেট এ চলে যাবে এবং তোমার প্রোগ্রাম ক্র্যাশ করবে। একই ভাবে খালি স্ট্যাক থেকে কিছু পপ করার চেষ্টা করলে স্ট্যাক ‘আন্ডারফ্লো’ স্টেটে

চলে যাবে এবং প্রোগ্রাম ক্র্যাশ করবে। স্ট্যাক ব্যবহারের সময় এই দুই ব্যাপারে খুব সতর্ক থাকতে হয়।

স্ট্যাক ইমপ্লিমেন্ট করা খুব সহজ। সাধারণ অ্যারে ব্যবহার করেই নির্দিষ্ট আকারের স্ট্যাক ইমপ্লিমেন্ট করা যায়। সবার উপরের এলিমেন্টটা কোন ইনডেক্সে আছে সেটা একটা অতিরিক্ত ভ্যারিয়েবলে সেভ করে রাখতে হবে। নিচে পাইথনে একটা ইমপ্লিমেন্টেশন দেখানো হলো, একই ভাবে সি++ এও তুমি করতে পারবে:

```
class Stack:
    def __init__(self, max_size): #initialize a stack of max_size
        self.top_pointer = -1 #Keep track of top element using this
        self.stack = [None for x in range(max_size)]  #create a list of max_size

    def push(self, new_element):
        self.top_pointer = self.top_pointer + 1 #Move the pointer
        self.stack[self.top_pointer] = new_element #Add the new_element to the top

    def pop(self):
        last_element = self.stack[self.top_pointer]
        self.top_pointer = self.top_pointer - 1 #Move the pointer
        return last_element #Pop the last element

    def peek(self):
        return self.stack[self.top_pointer]

    def is_empty(self):
        return top.pointer == -1
```

এই কোডে আমি কোনো ওভারফ্লো বা আন্ডারফ্লো হ্যান্ডেল করি নি। তুমি কোডটার উন্নতি করতে চাইলে এ ব্যাপারগুলো নিয়ে কাজ করতে পারো। যদি তুমি নির্দিষ্ট আকারের স্ট্যাক না চাও তাহলে লিংকড-লিস্ট ব্যবহার করতে হবে।

স্ট্যাকে প্রতিটা অপারেশনের কমপ্লেক্সিটি O(1)O(1)

স্ট্যাকের ব্যবহার:

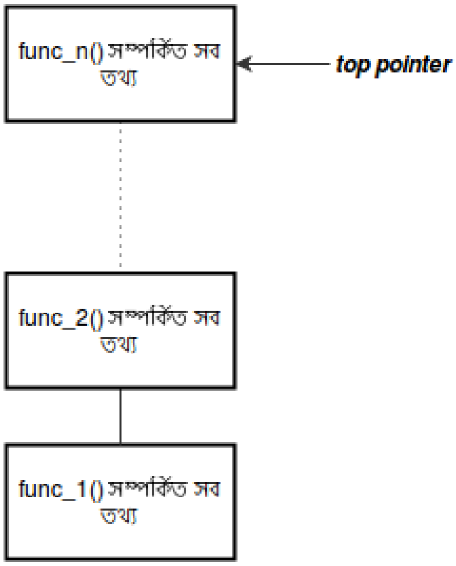
আমরা ব্রাউজারে প্রায়ই আগের ওয়েবসাইটে ফিরে যেতে “Back” বোতামে চাপ দেই। এটাও স্ট্যাক ব্যবহার করে তৈরি করা যায়। যখন নতুন ওয়েবসাইটে যাবে তখন আগের ওয়েবসাইটটাকে স্ট্যাকের উপরে রেখে দাও, ফিরে যেতে চাইলে স্ট্যাকের উপরের ওয়েব সাইটে ফেরত গিয়ে স্ট্যাক থেকে পপ করে দাও। এভাবে তুমি তোমার সফটওয়্যারে “undo” ফিচার বানাতে পারো।

প্রোগ্রামিং ল্যাংগুয়েজে ফাংশন কল করার সময় স্ট্যাকের খুব গুরুত্বপূর্ণ ব্যবহার আছে। মনে করো তুমি একটা ফাংশন func1 থেকে func2 কল করছো, সেখান থেকে আবার func3 কল করছো:

```
def func_3():
        return 42

def func_2():
        x=20
        y=func3()
        return x+y

def func_1():
        x=10
        y=func2()
        return x+y

print func1()
```

func_1() এ কিছু লোকাল ভ্যারিয়েবল আছে। তুমি যখন func_1() থেকে func_2() কে কল করছো তখন func_1() এর সব তথ্য একটা স্ট্যাকে ঢুকিয়ে রেখে নতুন ফাংশন func_2() কে লোড করা হয়। আবার যখন func_3() কে কল করছো তখন func_2() কে স্ট্যাকে ঢুকিয়ে রাখে হবে।

যদি $n+1n+1$ টা ফাংশন থাকে তাহলে যখন nn তম ফাংশন টা $n+1n+1$ তম ফাংশনকে কল করবে তখন স্ট্যাকের চেহারাটা হবে এরকম:



func_3() এর কাজ শেষ হবার পর আবার func2() কে মেমরিতে লোড করা হবে এবং স্ট্যাক থেকে পপ করে দেয়া হবে। এটাকে কল-স্ট্যাক বলা হয়। রিকার্সিভ ফাংশনও এভাবে আগের স্টেটগুলোকে স্ট্যাকে ঢুকিয়ে রাখে।

হ্যাকাররা এক ধরণের আক্রমণ মাঝে মাঝে ব্যবহার করে যাকে বলা হয় “স্ট্যাক স্ম্যাশিং”। আগ্রহী হলে এই আর্টিকেলটা পড়তে পারো।

স্ট্যাকের খুবই কমন একটা ব্যবহার হলো ব্রাকেট এর ব্যালেন্স বা ভারসাম্য ঠিক আছে সেটা পরীক্ষা করা (Parenthesis Balance)। মনে করো তোমাকে একটা ব্রাকেট সিকোয়েন্স S দেয়া হলো এরকম

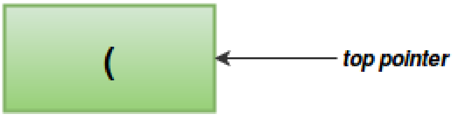
“(({})”। তোমাকে বলতে হবে সিকোয়েন্সটার ব্যালেন্স ঠিক আছে নাকি নেই। ব্যালেন্স ঠিক থাকবে যদি নিচের শর্ত গুলো পূরণ হয়:

-
  - যদি S একটা ০ দৈর্ঘ্যের স্ট্রিং হয়।
  - যদি A আর B দুইটা ব্যালেন্সড সিকোয়েন্স হয় তাহলে AB ও একটা ব্যালেন্সড সিকোয়েন্স। যেমন GA = “(){}” এবং B = “()” দুইটা ব্যালেন্সড সিকোয়েন্স হলে AB = “(){}()” ও ব্যালেন্সড।
  - যদি S একটা ব্যালেন্সড সিকোয়েন্স হয় তাহলে (S) অথবা {S} ও ব্যালেন্সড। যেমন S=”{}()” ব্যালেন্সড হলে (S) = “({}())” এটাও একটা ব্যালেন্সড সিকোয়েন্স।

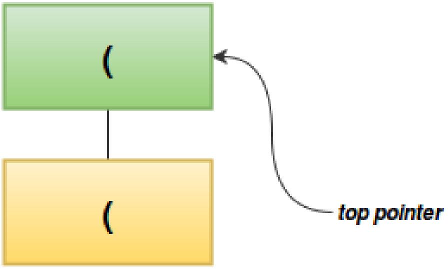
কিছু ভারসাম্যহীন সিকোয়েন্স হলো “((” , “({)}”, ()(}” ইত্যাদি।

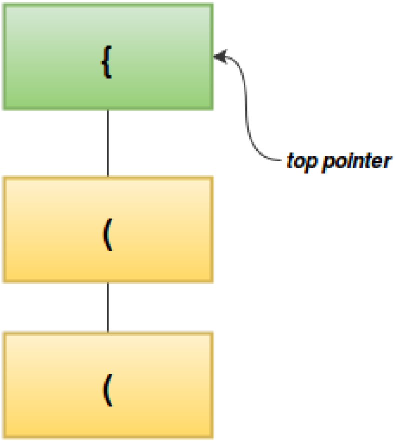
“(“ আর “{“ কে আমরা বলবো “ওপেন ব্রাকেট”, আর “)” এবং “}” কে বলবো “ক্লোজিং ব্রাকেট”। এখন আমরা S=”(({})” ব্যালেন্সড নাকি পরীক্ষা করবো স্ট্যাক ব্যবহার করে। আমরা বাম থেকে ডানে একটা একটা ক্যারেকটার নিয়ে কাজ করবো। কোনো “ওপেন ব্রাকেট” পেলেই সেটাকে স্ট্যাকে পুশ করবো।

এই সিকোয়েন্সের প্রথম ক্যারেকটার “(“, আমরা এটাকে একটা স্ট্যাকে পুশ করবো।



২য় এবং ৩য় ক্যারেকটার হলো “(“ এবং “{“। এদেরকেও পুশ করবো:

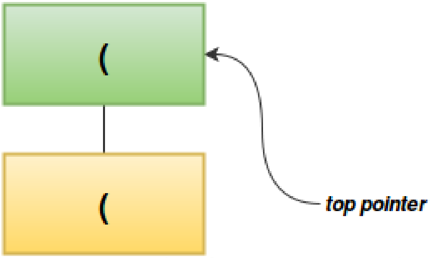




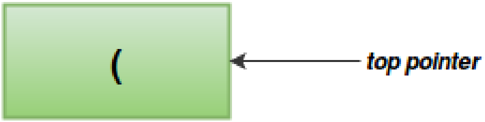
পরের ক্যারেকটারটা হলো “}”। ক্লোজিং ব্রাকেট যখন পাবো তখন আমরা দেখবো স্ট্যাকের উপরে কি আছে:

- যদি স্ট্যাকটা খালি হয় তাহলে সিকোয়েন্সটা ব্যালেন্সড না।
- যদি বর্তমান ক্যারেকটারটা “}” হয় তাহলে স্ট্যাকের উপরে অবশ্যই “{“ থাকতে হবে।
- যদি বর্তমান ক্যারেকটারটা “)” হয় তাহলে স্ট্যাকের উপরে অবশ্যই “(“ থাকতে হবে।

আমাদের স্ট্যাকের উপরে “{“ আছে যেটা “}” এর সাথে ম্যাচ করে। আমরা “}” নিয়ে কিছু করবো না কিন্তু “{“ কে স্ট্যাকের উপর থেকে পপ করে দিবো।



শেষ ক্যারেকটারটা হলো “)” যেটা “(“ এর সাথে ম্যাচ করে, আবার আমরা পপ করবো:



সব কাজ শেষে দেখা যাচ্ছে স্ট্যাকে এখনো একটা ব্রাকেট থেকে গেছে যেটা কারো সাথে মিলানো যাচ্ছে না! তারমানে আমাদের ব্রাকেট সিকোয়েন্সটা ব্যালেন্সড না। ব্যালেন্সড হলে সব ক্যারেকটার নিয়ে কাজ করার পর অবশ্যই স্ট্যাক খালি হয়ে যেত।

আমরা স্ট্যাক এর কোড ইতিমধ্যেই লিখেছি, এবার ব্রাকেট ব্যালেন্স করার কোডটা লিখে ফেলি:

```
def checkBalance(s):
    mystack=Stack(len(s))
    if s=="":
        return True
    for c in s:
        if c=="(" or c=="{":
            mystack.push(c) #push the opening bracket
        else:
            if mystack.is_empty():
                return False
            if c=="{" and mystack.peek()!="}": #the brackets dont match
                return False
            if c=="(" and mystack.peek()!=")": #the brackets dont matchs
                return False
            mystack.pop() #pop matching brackets

    if mystack.is_empty(): #stack must be empty at the end
        return True
    return False



print checkBalance("(){}")
print checkBalance("()(((")
print checkBalance("()(}")
```

অ্যালগোরিদমটা বুঝেছো নাকি পরীক্ষা করতে UVA 674 প্রবলেমটা সমাধান করে ফেলো।

স্ট্যাকের আরেকটা গুরুত্বপূর্ণ ব্যবহার হলো গাণিতিক ইকুয়েশনের মান বের করা, যেমন $(1+(2+5)\ast 3)+(5\ast 7)(1+(2+5)\ast 3)+(5\ast 7)$ এরকম একটা ইকুয়েশন দেয়া থাকলে মান বের করা খুব সহজ না। কিন্তু রিভার্স পলিশ নোটেশনে লিখলে স্ট্যাক দিয়ে খুব সহজে মান বের করা যায়। এটা এখন তুমি সহজেই উইকিপিডিয়া থেকে শিখে ফেলতে পারবে, আমি বিস্তারিত লিখবো না, আমার কাজ শুধু বেসিক জিনিসগুলো তোমাকে জানানো, কিন্তু ভালো করতে হলে নিজে কষ্ট করে শেখার কোনো বিকল্প নেই ☺

হ্যাপি কোডিং।