

## Data Structure and algorithms

### ডাটা স্ট্রাকচার এবং এলগরিদম

**ডাটা স্ট্রাকচার:** এটি একটি পদ্ধতি যার সাহায্যে কম্পিউটার মেমোরিতে নির্দিষ্ট উপায়ে ডাটাকে সাজানো এবং সংরক্ষণ করা হয়। আমরা তো অ্যারে এবং অবজেক্ট ডাটা স্ট্রাকচারের ব্যবহার জানি। যখনই আমাদের একটু বেশি ডাটা নিয়ে কাজ করার দরকার পরেছে তখনই আমরা অ্যারে এবং অবজেক্টের ব্যবহার করেছি।

কিন্তু যখন অনেক অনেক বেশি ডাটা নিয়ে কাজ করার দরকার পরবে তখন আমাদের আলাদাভাবে ডাটা সংরক্ষণ এবং সাজাতে হবে। এবং ডাটার মাঝে সম্পর্ক তৈরি করে বিভিন্ন অপারেশনের ব্যবস্থা করতে হবে আর এই জন্যই আমাদের ডাটা স্ট্রাকচারের ব্যবহার ভালোভাবে জানতে হবে।

তাছাড়াও কোনো একটা প্রোগ্রাম ডিজাইনের জন্যও ডাটা স্ট্রাকচার(Data Structure) প্রধান বিষয় তখন আমাদের জানতে হবে যে কোন ডাটা স্ট্রাকচার টি ব্যবহার করতে হবে।

**বিভিন্ন ধরনের ডাটা স্ট্রাকচার:** বিভিন্ন ধরনের অ্যাপ্লিকেশনের উপর নির্ভর করে ডাটা স্ট্রাকচার(Data Structure) এর শ্রেণীবিভাগ করা হয়েছে। ডাটা স্ট্রাকচার(Data Structure) কে প্রধানত ২ ভাগে ভাগ করা হয়েছে।

- প্রিমিটিভ ডাটা স্ট্রাকচার(Data Structure)
- নন-প্রিমিটিভ ডাটা স্ট্রাকচার(Data Structure)

প্রিমিটিভ ডাটা স্ট্রাকচার(Data Structure) গুলো হইলো –

- String
- Boolean
- number (integer, float)
- null

- undefined
- symbol

ইত্যাদি।

## **নন-প্রিমিটিভ ডাটা স্ট্রাকচার(Data Structure) - আবার দুইভাবে বিভক্ত :-**

- 1। লিনিয়ার ডাটা স্ট্রাকচার(Data Structure)
- 2। নন - লিনিয়ার ডাটা স্ট্রাকচার(Data Structure)

**১। লিনিয়ার ডাটা স্ট্রাকচার(Data Structure) -** যেসব ডাটা-স্ট্রাকচারে ডাটা গুলো একটি নির্দিষ্ট সিকুয়েন্সিয়াল(sequential) অর্ডারে(order) থাকে অর্থাৎ ডাটা সমূহ নির্দিষ্ট ভাবে সাজানো গোছানো অবস্থায় রাখা হয় তাকেই লিনিয়ার ডাটা স্ট্রাকচার(Data Structure) বলে।

যেমন –

1. অ্যারে(array)
2. লিংকড লিস্ট(linked list)
3. স্ট্যাক(stack)
4. কিউ(queue)

**২। নন-লিনিয়ার ডাটা স্ট্রাকচার(Data Structure) -** যেসব ডাটা-স্ট্রাকচারে ডাটা গুলো কোনো নির্দিষ্ট সিকুয়েন্সিয়াল(sequential) অর্ডারে(order) থাকে নাহ অর্থাৎ ডাটা সমূহ এলোমেলো ভাবে থাকে তাকেই নন-লিনিয়ার ডাটা স্ট্রাকচার(Data Structure) বলে।

যেমন –

1. ট্রি(tree)

## 2. গ্রাফ(graph)

**ডাটা অপারেশন ইন ডাটা স্ট্রাকচার:-** কোনো ডাটা স্ট্রাকচার(Data Structure) ইমপ্লিমেন্ট বা ব্যবহার করতে বিভিন্ন ধরনের অপারেশনের প্রয়োজন হতে পারে তার মধ্যে কিছু কমন অপারেশন গুলো হলো –

- ট্রাভার্সিং (traversing) - কোনো রেকর্ডকে একবার এক্সেস এবং প্রসেস করাকে ট্রাভার্সিং (traversing) বলে।
- সার্চিং (searching) - কোনো ডাটা স্ট্রাকচার থেকে নির্দিষ্ট আইটেমকে খুঁজে বের করাকে সার্চিং (searching) বলে।
- ইনসার্টিং (inserting) - কোনো ডাটা স্ট্রাকচারে নতুন আইটেম যুক্ত করাকে ইনসার্টিং (inserting) বলে।
- ডিলিটিং (deleting) - কোনো ডাটা স্ট্রাকচার থেকে রিমুভ করাকে ডিলিটিং (deleting) বলে।
- সোর্টিং (sorting) - ডাটাসমূহকে মানের ক্রমানুসারে সাজানোকে সোর্টিং বলে।

## অ্যালগরিদম (algorithm) –

অ্যালগরিদম হচ্ছে সমস্যা সমাধানের সুনির্দিষ্ট এবং ধারাবাহিক কিছু ধাপ। যে ধাপগুলো অনুসরণ করে একটি নির্দিষ্ট সমস্যা সমাধান করা হয়।

যেমন ধরুন আজ দুপুরে আপনি থিচুরি রান্না করতে চাচ্ছেন কিন্তু থিচুরি কিভাবে রান্না করতে হয় আপনি জানেন নাহ তাই যে পারে তার থেকে আপনি একটি রেসিপি নিলেন নিচের মতো –

১। শুরু

২। প্রথমে মুগ ডাল সামান্য ভেজে নিয়ে পানিতে ধুয়ে ফেলুন এবং চাল ও মুশরী ডালের সাথে মিশিয়ে নিন।

৩। চাল ও ডাল গুলো মিশিয়ে ভাল করে ধুয়ে নিয়ে পানি ঝারিয়ে নিন এবং এর পর মূল রান্নায় নেমে পড়ুন।

৪। এখন তেল গরম করে তাতে এলাচি ও দারুচিনি দিন।

৫। এবার পেঁয়াজ কুঁচি ও কাঁচা মরিচ দিন। কাঁচা মরিচ তেলে ফুটে উঠে তাই সাবধানে বা চিরে দিতে পারেন।

৬। এবার আদা, রসুন, মরিচ গুড়া ও হলুদ গুড়া দিয়ে দিন। এই সময়ে এক চা চামচ লবন দিন। রঙ বেশি কড়া চাইলে সামান্য হলুদ বেশি দিতে পারেন।

-----

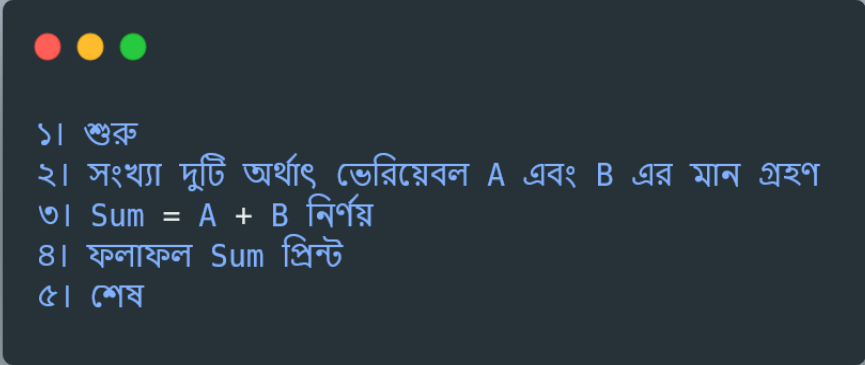
-----

-----

\*। শেষ

উপরের রেসিপি অনুযায়ী আপনাকে খিচুরি রান্না করতে হবে। একটি স্টেপ যদি আপনি বাদ দেন অথবা উলটা পাল্ট করে করেন তাহলে কিন্তু আর খিচুরি হবে নাহ। অর্থাৎ একটা নির্দিষ্ট সমস্যাকে সমাধান করার জন্য, নির্দিষ্ট কিছু নিয়মকানুন বা ধাপ অনুসরণ করতে হয়।

তবে একটা কথা বলে রাখা ভালো যে, একটি সমস্যা অনেক ভাবেই সমাধান করা যায়। তাই কোনো সমস্যার অ্যালগরিদম অনেক ভাবেই থাকতে পারে কিন্তু সেই অ্যালগরিদমটি যেনো ঠিকঠাক ভাবে কাজ করে।



```
১। শুরু  
২। সংখ্যা দুটি অর্থাৎ ভেরিয়েবল A এবং B এর মান গ্রহণ  
৩।  $Sum = A + B$  নির্ণয়  
৪। ফলাফল Sum প্রিন্ট  
৫। শেষ
```

আর্টিকেলটি শেষ করার পূর্বে চলুন দুটি সংখ্যার যোগফল নির্ণয়ের অ্যালগরিদম কিভাবে লিখতে হয় তা শিখি

## অ্যালগরিদম(algorithm) লিখার নিয়ম

অ্যালগরিদম ও ডাটা স্ট্রাকচার সম্পর্কে গত পর্বে আলোচনা করা হয়েছে। যদি আপনি পড়ে নাহ থাকেন তাহলে এখানে পড়ুন। অ্যালগরিদম কি বা কাকে বলে তা নিয়ে আর নাহ বলি কারণ গত পর্বে আলোচনা করা হয়েছে। এখন আসি কিভাবে আমরা অ্যালগরিদম লিখতে পারি ?

অ্যালগরিদম লেখার বিভিন্ন পদ্ধতি আছে। অনেকরই মনে প্রশ্ন থাকতে পারে যে, অ্যালগরিদম কোন প্রোগ্রামিং ভাষা দিয়ে লিখতে হয় ? উত্তর হচ্ছে, অ্যালগরিদম আপনি যে কোনো ভাষায় লিখতে পারেন। কোনো একটা অ্যালগরিদম আপনি বর্ণনামূলক ভাবে লিখতে পারেন আবার সুডোকোড আকারে লিখতে পারেন।

এবং আপনি বাংলায় ও লিখতে পারেন। আমরা জানি যে অ্যালগরিদম হচ্ছে সমস্যা সমাধানের সুনির্দিষ্ট কিছু ধাপ। যা মানুষ

বুঝতে পারলেই হইলো । আপনি যখন কোনো একটা অ্যালগরিদম বুঝতে পারবেন তখন খুব সহজেই আপনি যেকোনো প্রোগ্রামিং ভাষা দিয়ে ইমপ্লিমেন্ট করতে পারবেন।

এখন চলুন কয়েকটা সমস্যার অ্যালগরিদম কি ভাবে লিখতে হয় তা দেখি ।

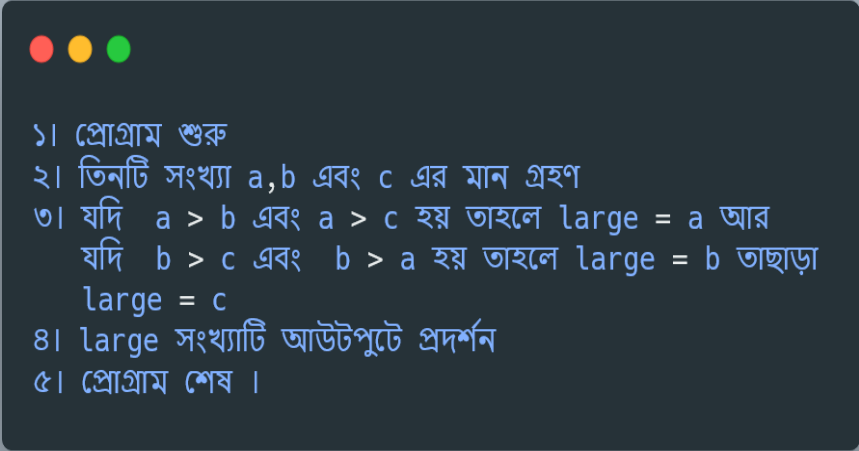
### সমস্যা ০১ : তিনটি সংখ্যার মধ্যে বৃহত্তম সংখ্যাটি নির্ণয়ের অ্যালগরিদম লিখ –

এখানে একটা কথা বলে রাখি যে, যেকোনো সমস্যার অ্যালগরিদম লিখার সময় তার শুরু এবং শেষ থাকবেই । তারপর সমস্যা অনুযায়ী অ্যালগরিদম লিখতে হবে ।

ধরে নিই যে, আমাদের কাছে ১০,২০,৩০ এই তিনটি সংখ্যা আছে, এখানে কিন্তু আমরা বুঝতে পারছি যে সবচেয়ে বড় সংখ্যাটি হলো ৩০। কিন্তু উপরোক্ত প্রব্লেমে আমাদের তিনটি সংখ্যার কোনটিই দেওয়া নেই । তাই যেকোনো তিনটি সংখ্যাই হতে পারে । তাহলে এখানে আমাদেরকে তিনটি সংখ্যায় ইউজার থেকে ইনপুট নিতে হবে অথবা ফাংশনের প্যারামিটার হিসেবে পাস করতে হবে ।

এখন আমরা ধরে নিতে পারি যে, তিনটি সংখ্যা  $a, b$  এবং  $c$  যা ইউজারের কাছে থেকে আসবে । আমাদের ইনপুট নেওয়া কিন্তু শেষ । এরপর আমাদের লজিক ব্যবহার করে তিনটি সংখ্যার ভেতর বড় সংখ্যাটি বের করতে হবে । অর্থাৎ, যদি  $a > b$  এবং  $a > c$  হয় তাহলে বড় সংখ্যাটি  $a$  আর যদি  $b > c$  এবং  $b > a$  হয় তাহলে বড় সংখ্যাটি হবে  $b$  তাছাড়া বড় সংখ্যাটি হবে  $c$  ।

যেহেতু বড় সংখ্যাটি পেয়ে গেলাম সেইটা আউটপুটে দেখাবো এবং প্রোগ্রামটি শেষ করবো । এখন যদি এইটার অ্যালগরিদম অর্থাৎ সুনির্দিষ্ট ধাপসমূহ লিখি তাহলে –



```
১। প্রোগ্রাম শুরু  
২। তিনটি সংখ্যা a,b এবং c এর মান গ্রহণ  
৩। যদি a > b এবং a > c হয় তাহলে large = a আর  
    যদি b > c এবং b > a হয় তাহলে large = b তাছাড়া  
    large = c  
৪। large সংখ্যাটি আউটপুটে প্রদর্শন  
৫। প্রোগ্রাম শেষ ।
```

এইরকম হবে। আমি আগেই বলেছি যে, একটি সমস্যার অ্যালগরিদম অনেক ভাবেই লিখা যায় কিন্তু সেইটা অবশ্যই ঠিকঠাক কাজ করতে হবে।

আমাদের উপরোক্ত অ্যালগরিদমটি সঠিকভাবে কাজ করবে কি নাহ সেইটা বোঝার জন্য আপনি যেকোনো প্রোগ্রামিং ভাষা দিয়ে স্টেপ বাই স্টেপ ইমপ্লিমেন্ট করতে পারেন।

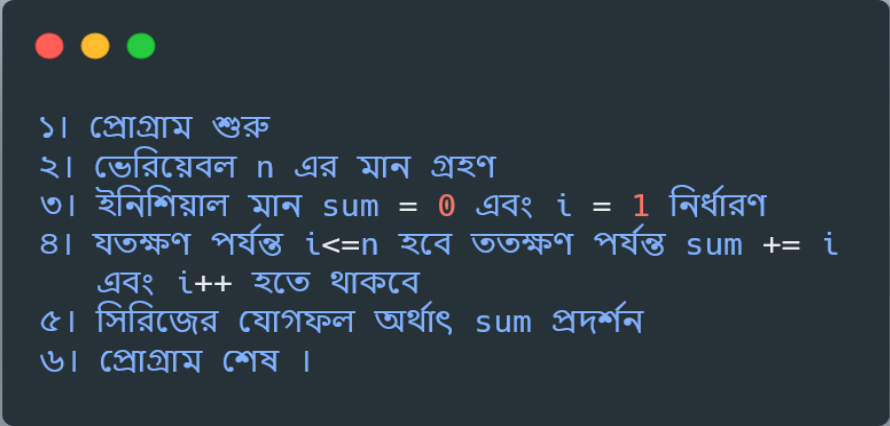
**সমস্যা ০২ :  $1 + 2 + 3 + \dots + n = ?$  সিরিজের যোগফল নির্ণয়ের অ্যালগরিদম লিখ –**

এখানে  $n$  তম সংখ্যার যোগফল নির্ণয়ের অ্যালগরিদম লিখতে বলা হয়েছে। তাহলে বরাবরের মতোই আমরা প্রোগ্রাম শুরু করবো। যেহেতু,  $n$  তম সংখ্যার যোগফল তাই আমাদেরকে  $n$  ইনপুট নিতে হবে।

তারপর sum নামে একটা ভেরিয়েবল নিবো যার ইনিশিয়াল মান 0 রাখবো। এবং যেহেতু লুপ ব্যবহার করবো তাই  $i$  এর ইনিশিয়াল মান 1 রাখবো। এরপর লুপের ভেতর কন্ডিশন রাখবো যে,  $i$  এর মান  $n$  এর চেয়ে যতক্ষণ পর্যন্ত কম বা সমান থাকবে ততক্ষণ পর্যন্ত sum

এর ভেতর  $sum + i$  এর মান অ্যাসাইন এবং  $i$  এর মান 1 করে বাড়তে থাকবে।

লুপ শেষে সিরিজের যোগফল  $sum$  প্রদর্শন করবে এবং প্রোগ্রাম শেষ হবে। তাহলে এর অ্যালগরিদম হবে –



- ১। প্রোগ্রাম শুরু
- ২। ভেরিয়েবল  $n$  এর মান গ্রহণ
- ৩। ইনিশিয়াল মান  $sum = 0$  এবং  $i = 1$  নির্ধারণ
- ৪। যতক্ষণ পর্যন্ত  $i \leq n$  হবে ততক্ষণ পর্যন্ত  $sum += i$  এবং  $i++$  হতে থাকবে
- ৫। সিরিজের যোগফল অর্থাৎ  $sum$  প্রদর্শন
- ৬। প্রোগ্রাম শেষ।

চলুন উপরোক্ত অ্যালগরিদমটি আমরা জাভাস্ক্রিপ্ট প্রোগ্রামিং ভাষা ব্যবহার করে ইমপ্লিমেন্ট করি –

```
const seriesSum = (n) => {  
  let sum = 0;  
  let i = 1;  
  while(i <= n){  
    sum += i;  
    i++;  
  }  
  console.log(sum)  
}
```



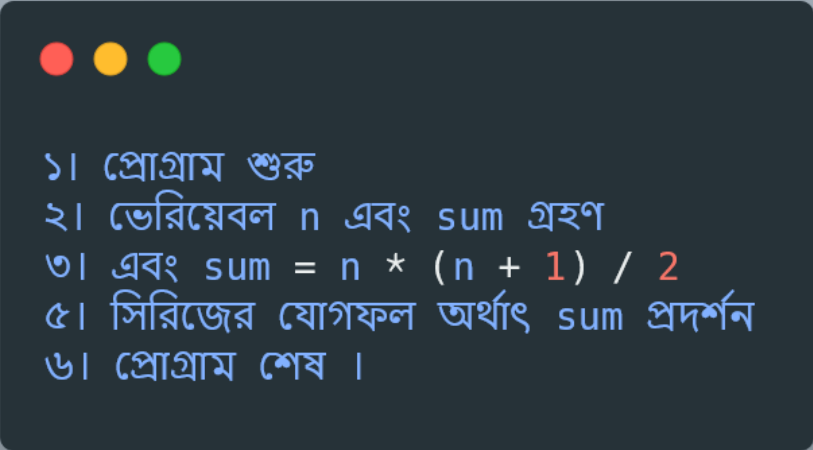
এখন আপনি উপরোক্ত অ্যালগরিদমটি দেখুন এবং নিচের প্রোগ্রামিং ভাষা ব্যবহার করা সমাধান দেখুন আশা করি বুঝতে পারবেন।  
উপরোক্ত অ্যালগরিদমকেই স্টেপ বাই স্টেপ ফলো করে নিম্নোক্ত প্রোগ্রামটি লিখা হয়েছে।

আমি হয়তোবা অনেকবারই বলেছি যে, একটি সমস্যা অনেক ভাবেই সমাধান বা অ্যালগরিদম উন্নয়ন করা যায় চলুন এইবার তা দেখে নেওয়া যাক। আমরা উপরোক্ত সমস্যাটিরই প্রোগ্রাম এবং অ্যালগরিদম উন্নয়ন করবো কিন্তু অন্যভাবে।

সমস্যাটি ছিলো  $1 + 2 + 3 + ..... + n = ?$  সিরিজের যোগফল নির্ণয়ের অ্যালগরিদম লিখ -

এইবার এই সমস্যাটির সমাধান করতে আমরা লুপ ব্যবহার না করে একটি ম্যাথম্যাটিক্সের সূত্র ব্যবহার করে করবো। সূত্রটি হইলো -  $n*(n+1)/2$

প্রথমে অ্যালগরিদম উন্নয়ন করি -



```
১। প্রোগ্রাম শুরু  
২। ভেরিয়েবল n এবং sum গ্রহণ  
৩। এবং  $sum = n * (n + 1) / 2$   
৫। সিরিজের যোগফল অর্থাৎ sum প্রদর্শন  
৬। প্রোগ্রাম শেষ।
```

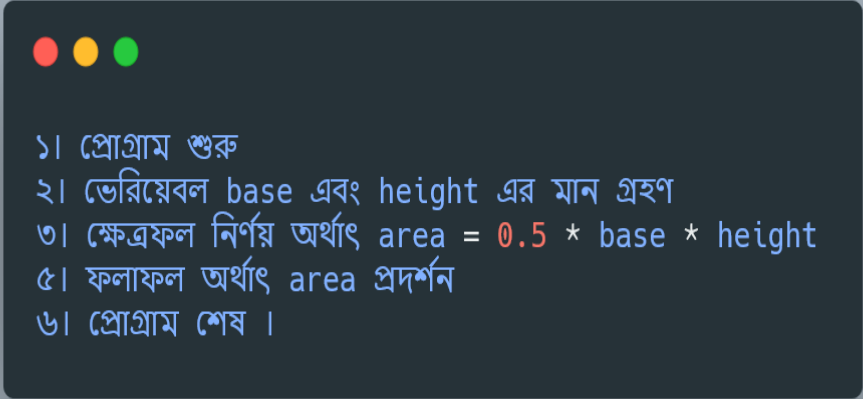
এখন আমরা এই অ্যালগরিদমের সাহায্যে প্রোগ্রাম লিখি-

```
const seriesSum = (n) => {  
  let sum;  
  sum = n * (n+1) / 2 ;  
  console.log(sum);  
}
```

আশা করি বুঝতে পেরেছেন। আমি জাভাস্ক্রিপ্ট ব্যবহার করে অ্যালগরিদম ইমপ্লিমেন্ট করেছি আপনি চাইলে অন্য যেকোনো প্রোগ্রামিং ভাষা ব্যবহার করতে পারেন।

আমরা জানি যে, সমকোণী ত্রিভুজ ক্ষেত্রের ক্ষেত্রফল =  $0.5 * \text{ভূমি} * \text{উচ্চতা}$ । প্রথমে প্রোগ্রাম শুরু করতে হবে এরপর ভূমি এবং উচ্চতার মান ইনপুট নিতে হবে।

এরপর ক্ষেত্রফল এর ভেতর ( $0.5 * \text{ভূমি} * \text{উচ্চতা}$ ) এর মান অ্যাসাইন করতে হবে। এবং ফলাফলের মান আউটপুটে প্রিন্ট করে প্রোগ্রাম শেষ করতে হবে। অর্থাৎ অ্যালগরিদমটি হবে,



```
১। প্রোগ্রাম শুরু  
২। ভেরিয়েবল base এবং height এর মান গ্রহণ  
৩। ক্ষেত্রফল নির্ণয় অর্থাৎ  $\text{area} = 0.5 * \text{base} * \text{height}$   
৫। ফলাফল অর্থাৎ area প্রদর্শন  
৬। প্রোগ্রাম শেষ।
```

## বিগ ও নোটেশন (Big O Notation)

কম্পিউটার সাইন্সে কোনো একটা অ্যালগরিদমের পারফরমেন্স(performance) বা কমপ্লিক্সিটি(complexity) কে বোঝাতে বিগ ও নোটেশন (Big O) ব্যবহার করা হয়। বিগ ও নোটেশন (Big O) স্পেসালি(specifically) অ্যালগরিদমের টাইম বা স্পেস কমপ্লিক্সিটির(complexity) ওয়ার্স্ট-কেস(worst-case) কে নির্দেশ করে ।

### ওয়ার্স্ট-কেস(worst-case) কি ?

[1,2,3,4,5,6]

ধরি, আমাদের কাছে উপরোক্ত অ্যারেটি আছে । এবং আমরা দেখতে পাইতেছি যে, অ্যারের সর্বশেষ আইটেমটি হলো 6 । এখন আমাদেরকে যদি কোনো একটা অ্যালগরিদম ব্যবহার করে 6 কে খুঁজতে বলা হয় তাহলে কিন্তু অন্য আইটেম গুলোর থেকে অপারেশন বেশি করতে হবে এবং সময়ও বেশি লাগবে, আর এটিই এই অ্যালগরিদমের ওয়ার্স্ট-কেস(worst-case)। অনেকেই ওয়ার্স্ট-কেস(worst-case) কে খারাপ কেসও বলে থাকে ।

প্রোগ্রামার যখন কোনো একটা অ্যালগরিদম লিখে তখন অবশ্যই তাকে পারফরমেন্স(performance) বা কমপ্লিক্সিটি(complexity) এর কথা মাথায় রেখে লিখতে হয়। একটা অ্যালগরিদম তো অনেক ভাবেই লিখা যায় কিন্তু কোন অ্যালগরিদমটি ব্যবহার করলে মেমরি এবং সময় কম নিবে তা আগে বুঝতে হবে । তবে একটা কথা বলে রাখা ভালো যে, অনেক সময় একটি অ্যালগরিদমের মেমরি বাঁচাতে অতিরিক্ত সময় খরচ করতে হয়, আবার সময় বাঁচাতে অতিরিক্ত মেমরি খরচ করতে হয় ।

কোনো একটি অ্যালগরিদম রান করতে কি পরিমাণ সময় নিবে বা অ্যালগরিদমটি কি পরিমাণ মেমরি নিবে এগুলোর হিসেব-নিকেশ বিগ ও নোটেশন (Big O) এর দ্বারা করা হয়।

**বিগ ও নোটেশন (Big O) ছাড়াও আরও দুইটি গাণিতিক ফাংশন আছে ।**

**১। থেটা( $\theta$ )**

এটি মূলত অ্যালগরিদমের এভারেজ কেস হিসেব করতে ব্যবহার করা হয়।

**২। ওমেগা( $\Omega$ )**

এটি মূলত অ্যালগরিদমের বেস্ট কেস হিসেব করতে ব্যবহার করা হয়।

**টাইম কমপ্লিক্সিটি(complexity) –**

অ্যালগরিদম বা প্রোগ্রামে টাইম কমপ্লিক্সিটি(complexity) অনেক গুরুত্বপূর্ণ বিষয় । সোজা-সাপ্টায় কোনো অ্যালগরিদম বা প্রোগ্রাম রান হতে কতটুকু সময় লাগলো তাকেই টাইম কমপ্লিক্সিটি(complexity) বলে। কোনো একটা প্রোগ্রাম রান হতে ঠিক কতটুকু সময় নিলো তা আমরা বিভিন্ন লাইব্রেরী ফাংশনের সাহায্যে বের করতে পারি । কিন্তু যদি আমরা একবার এর হিসেব-নিকেশ বুঝি তাহলে যেকোনো প্রোগ্রাম রান না করেই অর্থাৎ দেখলেই বলে দিতে পারবো যে সেই প্রোগ্রাম রান হতে কেমন সময় লাগতে পারে ।

**টাইম কমপ্লিক্সিটি(complexity) মাথায় রাখা কেনো দরকার-**

বিভিন্ন অনলাইন জাজে কোনো একটা পর্লেম পড়ার সময় দেখবেন যে, উপরে বা কোথাও টাইম লিমিট মেনশন করা থাকে । যার অর্থ

হইলো যে, আপনি যেভাবেই প্রোগ্রামটা লিখেন নাহ কেনো প্রোগ্রাম রান হতে মেনশন করা সময়ের বেশি যেনো নাহ লাগে।

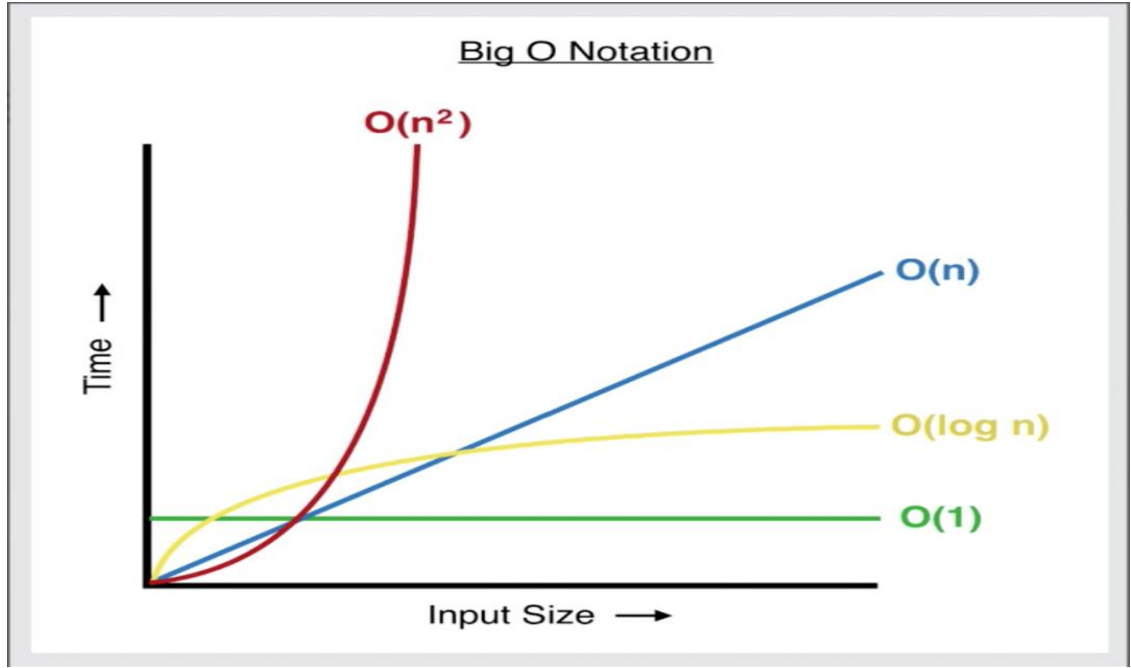
এছাড়াও, ইউজার এক্সপেরিয়েন্সের জন্য টাইম কমপ্লিক্সিটি(complexity) একটা প্রধান বিষয়। এখন আপনি একটি অ্যাপ্লিকেশন তৈরি করলেন দেখা গেলো যে, সেটির কোনো একটা পার্টে ইউ-আই বা ডাটা লোড হতে নির্দিষ্ট সময়ের থেকে অনেক বেশি সময় লাগতেছে। এখন আপনার কি মনে হয় যে ইউজার এটি ব্যবহার করবে? অবশ্যই নাহ করার সম্ভাবনাটা বেশিই।

### **স্পেস কমপ্লিক্সিটি(complexity) –**

অবশ্যই অ্যালগরিদম বা প্রোগ্রামে স্পেস কমপ্লিক্সিটি(complexity)ও গুরুত্বপূর্ণ বিষয়। কোনো অ্যালগরিদম বা প্রোগ্রাম ঠিক কতটুকু জায়গা বা মেমরি নিলো তাকেই স্পেস কমপ্লিক্সিটি(complexity) বলে।

কোনো অ্যালগরিদম বা প্রোগ্রামের জন্য আমাদের নির্দিষ্ট মেমরি ব্যবহার করতে হয়। কিন্তু আমরা যদি স্পেস কমপ্লিক্সিটির(complexity) হিসেব-নিকেশ বুঝি তাহলে সহজেই বলে দিতে পারবো যে একটি প্রোগ্রাম কার্জকর কার্যকর করতে আমাদের ঠিক কতটুকু মেমরি ব্যবহার করতে হবে।

### **Common orders in Big (O) –**



আগেই বলেছি যে, কোনো একটা সমস্যা অনেকভাবে অ্যালগরিদম বা প্রোগ্রাম লিখে সমাধান করা যায়। আর এখানেই আসে ওর্ডার অপারেশন অর্থাৎ সমস্যাটি সমাধান করার আগে জানতে হবে যে, কোন অ্যালগরিদমের কমপ্লিক্সিটি (complexity) কত? কম না বেশি। অবশ্যই একটি সমস্যা সমাধান করতে সবচেয়ে কম সময় বা মেমরি যাতে লাগে এরকম একটা অ্যালগরিদম লিখার চেষ্টা করতে হবে।

আর এসব বুঝতে কিছু ওর্ডার অপারেশন সমন্ধে জানতে হবে যেমন

—

- $O(1)$
- $O(n)$
- $O(!n)$

- $O(n^2)$
- $O(n^3)$
- $O(\log n)$

ইত্যাদি ইত্যাদি । এগুলো কে ওর্ডার অব তারপর ব্রাকেটসের ভেতর যা আছে সেইটা ধরে বলতে হয় যেমন -  $O(1)$  হলো ওর্ডার অব ওয়ান এবং  $O(\log n)$  ওর্ডার অব লগ এন ।

### **Constant Time: $O(1)$ -**

যখন কোনো প্রোগ্রামে অপারেশন কন্সট্যান্ট টাইম হয় সেই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব ওয়ান  $O(1)$ ।

ছোট্ট একটি উদাহরণ –

```
const a = 10;
```

```
const b = 5;
```

```
const sum = a+b;
```

```
const subtract = sum - 10;
```

উপরের এই প্রোগ্রামে -

a ভ্যারিয়েবলে 10 রাখা হয়েছে ।

b ভ্যারিয়েবলে 5 রাখা হয়েছে ।

sum ভ্যারিয়েবলে (a+b) এর মান রাখা হয়েছে ।

subtract ভ্যারিয়েবলে (sum - 10) এর মান রাখা হয়েছে।

এখানে a ও b ভ্যারিয়েবলের মান যতই রাখা হোক নাহ কেন আমাদের প্রোগ্রামে কিন্তু গাণিতিক অপারেশন মাত্র দুইবারই(২) হবে। অর্থাৎ a ও b ভ্যারিয়েবলের মান যদি ১০০ করে রাখি তবুও গাণিতিক অপারেশন মাত্র দুইবারই(২) হবে। তাই আমরা বলতে পারি এই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব ওয়ান  $O(1)$ ।

### **Linear Time: $O(n)$ –**

যখন কোনো প্রোগ্রামে কোনো কাজ একের অধিক বার কোনো ইনপুটের উপর নির্ভর করে হয় সেই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব এন  $O(n)$  যাকে Linear Time ও বলা হয়।

যেমন –

```
let sum = 0;
const summation = (n) => {
  for (let i = 0; i < n; i++) {
    sum += i;
  }
};
summation(10);
console.log(sum);
```



উপরোক্ত প্রোগ্রামে -

sum ভ্যারিয়েবলে 0 রাখা হয়েছে।

summation(n) ফাংশন যা n নামে একটি প্যারামিটার নিচ্ছে।

এবং ফাংশনের ভেতরে একটি লুপ আছে যা চলবে প্যারামিটার এর মান পর্যন্ত।

পরের লাইনে sum এর সাথে অ্যাসাইনমেন্ট ও যোগ অপারেশন হচ্ছে।

এখানে যদি n এর মান 100 পাঠানো হয় তাহলে কিন্তু অ্যাসাইনমেন্ট ও যোগ অপারেশন 100 বারই হবে সাথে sum এর মানও বাড়তে থাকবে। আর যদি n এর মান 1 হতো তাহলে একবারই অপারেশন গুলো হতো। এখন আমরা খুব সহজেই বলতে পারি যে, এই প্রোগ্রামের টাইম কমপ্লেক্সিটি(complexity) ওর্ডার অব এন  $O(n)$ ।

## **অ্যালগরিদমে টাইম কমপ্লেক্সিটি(complexity) - পর্ব (০১)**

অ্যালগরিদম বা প্রোগ্রামে টাইম কমপ্লেক্সিটি(complexity) অনেক গুরুত্বপূর্ণ বিষয়। সোজা-সাপ্টায় কোনো অ্যালগরিদম বা প্রোগ্রাম রান হতে কতটুকু সময় লাগলো তাকেই টাইম কমপ্লেক্সিটি(complexity) বলে। কোনো একটা প্রোগ্রাম রান হতে ঠিক কতটুকু সময় নিলো তা আমরা বিভিন্ন লাইব্রেরী ফাংশনের সাহায্যে বের করতে পারি। কিন্তু যদি আমরা একবার এর হিসেব-নিকেশ বুঝি

তাহলে যেকোনো প্রোগ্রাম রান না করেই অর্থাৎ দেখলেই বলে দিতে পারবো যে সেই প্রোগ্রাম রান হতে কেমন সময় লাগতে পারে ।

আপনি যদি বিগ ও নোটেশন (Big O) নোটেশন সম্পর্কে নাহ যেনে থাকেন তাহলে এখানে পড়ুন ।

আমরা জানি যে, কোনো একটা সমস্যা অনেক ভাবেই সমাধান করা যায় । তাই আমাদের কোনো একটা সমস্যা সমাধান করার সময় অবশ্যই মাথায় রাখতে হবে কোনটার টাইম কমপ্লিক্সিটি(complexity) কেমন এবং কোনটা বেশি স্মার্ট ।

এই জন্যই আমাদের টাইম কমপ্লিক্সিটি(complexity) বিষয়টা ভালো করে বুঝতে হবে ।

```
const ghurLoop = (n) => {  
  for(let i = 0; i<n; i++){  
    console.log("I am haba!")  
  }  
}
```

```
ghurLoop(10);
```

আচ্ছা উপরের প্রোগ্রামে I am haba! কতো বার প্রিন্ট হবে? যার প্রোগ্রামিং সম্পর্কে বেসিক নলেজ আছে সে খুব সহজেই বলে দিতে পারবে যে ১০ বার । কারণ, আমাদের লুপটি কিন্তু ১০ বার ঘুরবে কন্ডিশন অনুযায়ী যা আমরা প্যারামিটার হিসেবে পাস করেছি ।

এখন আমরা যদি ফাংশনের প্যারামিটার হিসেবে ১০ না দিয়ে ২০ দেয় তাহলে I am haba! কতো বার প্রিন্ট হবে? ২০ বার তাই তো । এইভাবে যদি ১০০ বা ২০০ বা ৫০০ প্যারামিটার হিসেবে পাঠায় তাহলে I am haba! ১০০ বা ২০০ বা ৫০০ বার প্রিন্ট হবে ।

এখানে কিন্তু প্যাঁরামিটার এর ওপর নির্ভর করে আমাদের লুপটি ঘুরবে । এখন আমরা বুঝতে পারছি যে, I am haba! n বার প্রিন্ট করবে ।

তাহলে আমরা খুব সহজেই বলে দিতে পারি যে, এই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব এন  $O(n)$  ।

### **Linear Time: ওর্ডার অব এন $O(n)$ –**

যখন কোনো প্রোগ্রামে কোনো কাজ একের অধিক বার কোনো ইনপুটের উপর নির্ভর করে হয় সেই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব এন  $O(n)$  যাকে Linear Time ও বলা হয়।

যেমন –

```
let sum = 0;
const summation = (n) => {
  for (let i = 0; i < n; i++) {
    sum += i;
  }
};
summation(10);
console.log(sum);
```

উপরোক্ত প্রোগ্রামে -

- sum ভ্যারিয়েবলে 0 রাখা হয়েছে ।
- summation(n) ফাংশন যা n নামে একটি প্যাঁরামিটার নিচ্ছে।

- এবং ফাংশনের ভেতরে একটি লুপ আছে যা চলবে প্যারামিটার এর মান পর্যন্ত ।
- পরের লাইনে sum এর সাথে অ্যাসাইনমেন্ট ও যোগ অপারেশন হচ্ছে।

এখানে যদি n এর মান 100 পাঠানো হয় তাহলে কিন্তু অ্যাসাইনমেন্ট ও যোগ অপারেশন 100 বারই হবে সাথে sum এর মানও বাড়তে থাকবে । অর্থাৎ এখানে n এর মান যত বৃদ্ধি পাবে প্রোগ্রামে অ্যাসাইনমেন্ট, যোগ অপারেশন ও sum এর মান ততই বাড়তে থাকবে এতে প্রোগ্রামের রানটাইমও বেড়ে যাবে। এখন আমরা খুব সহজেই বলতে পারি যে, এই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব এন  $O(n)$  ।

### **Constant Time: $O(1)$ –**

যখন কোনো প্রোগ্রামে অপারেশন কন্সট্যান্ট টাইম হয় সেই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব ওয়ান  $O(1)$ ।

ছোট্ট একটি উদাহরণ –

```
const a = 10;
```

```
const b = 5;
```

```
const sum = a+b;
```

```
const subtract = sum - 10;
```

উপরের এই প্রোগ্রামে -

- a ভ্যারিয়েবলে 10 রাখা হয়েছে ।
- b ভ্যারিয়েবলে 5 রাখা হয়েছে ।
- sum ভ্যারিয়েবলে (a+b) এর মান রাখা হয়েছে ।
- subtract ভ্যারিয়েবলে (sum - 10) এর মান রাখা হয়েছে ।

এখানে a ও b ভ্যারিয়েবলের মান যতই রাখা হোক নাহ কেন আমাদের প্রোগ্রামে কিন্তু গাণিতিক অপারেশন মাত্র দুইবারই(২) হবে । অর্থাৎ a ও b ভ্যারিয়েবলের মান যদি ১০০ করে রাখি তবুও গাণিতিক অপারেশন মাত্র দুইবারই(২) হবে । তাই আমরা বলতে পারি এই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব ওয়ান  $O(1)$ ।

আমরা উপরে ফর-লুপ(for loop) ব্যবহার করে n তম সংখ্যার যোগফল নির্ণয়ের একটা প্রোগ্রামে লিখেছি যার টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব এন  $O(n)$ ।

এখন আমরা আবার n তম সংখ্যার যোগফল নির্ণয়ের একটা প্রোগ্রাম লিখবো তবে এবার কোনো লুপ(loop) ব্যবহার করে নাহ সূত্রের সাহায্যে ।

সূত্রটি হলো -  $n * (n + 1) / 2 ;$

```
const sum = (n) => {  
  const result = n * (n + 1) / 2 ;  
  console.log(result);  
}  
sum(4);
```

আচ্ছা আপনি কি বলতে পারবেন নাহ এই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) কত ? আসলে এখানে হচ্ছে টা কি -

একটি যোগ, একটি গুণ, একটি ভাগ এবং একটি অ্যাসাইনমেন্ট এর কাজ হচ্ছে ।

তাছাড়া আর তেমন কিছু হচ্ছে নাহ । এখানে প্যারামিটার ১০, ১০০, ৩০০ যেইটাই পাস করা হোক না কেন একটি যোগ, একটি গুণ, একটি ভাগ এবং একটি অ্যাসাইনমেন্ট এরই কাজ হবে এর বেশি কিন্তু হবে নাহ । তাহলে আমরা বলতে পারি এই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) ওর্ডার অব ওয়ান  $O(1)$ ।

আরেকটি প্রোগ্রাম দেখা যাক –

```
const ghurLoop = (n) => {  
  for(let i = 0; i<n; i = i+2){  
    console.log("I am haba!")  
  }  
}
```

এই প্রোগ্রামের টাইম কমপ্লিক্সিটি(complexity) কত হতে পারে ?  
প্রথমে লক্ষ্য করুন যে, এইবার কিন্তু লুপে i এর মান ২ করে বাড়তেছে । তার মানে প্যারামিটার যত পাঠানো হবে তার অর্ধেক বার লুপটি চলবে এবং I am haba! প্রিন্ট করবে।

যেমন আমি যদি এখন প্যারামিটার ১০ পাঠায় তাহলে আউটপুট হবে –

Pass Parameter -

```
ghurLoop(10)
```

- OUTPUT

I am haba! I am haba! I am haba! I am haba! I am haba!

খেয়াল করছেন ? প্যারামিটারের অর্ধেক বার লুপটি ঘুরলো । অর্থাৎ প্যারামিটার ২০, ৫০, ২০০ যাই পাঠানো হোক না কেন তার অর্ধেক লুপটি ঘুরবে এবং I am haba! প্রিন্ট করবে ।

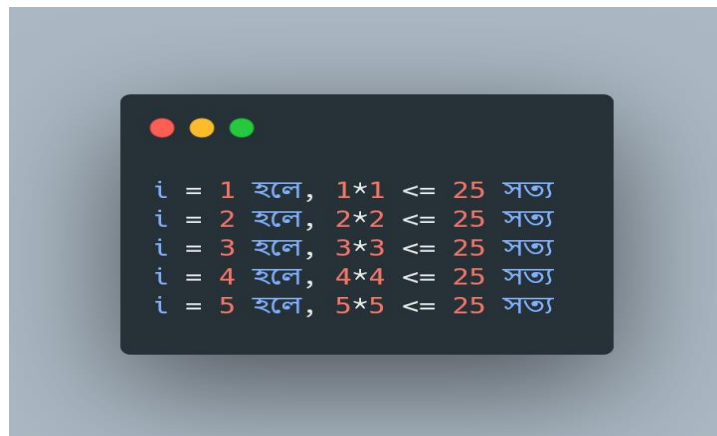
## অ্যালগরিদমে টাইম কমপ্লিক্সিটি(complexity) - পর্ব (০২)

ডাটা-স্ট্রাকচার এন্ড অ্যালগরিদমে টাইম কমপ্লিক্সিটি একটি গুরুত্বপূর্ণ বিষয় তাই আমাদের এইটা সম্পর্কে খুব ভালো ধারণা থাকতে হবে। গতপর্বে আমরা টাইম কমপ্লিক্সিটি সম্পর্কে বেশ কিছু জেনেছিলাম এবং শিখেছিলাম।

এই পর্বে আমরা আরও টাইম কমপ্লিক্সিটি সম্পর্কে জানবো। প্রথমেই একটি সহজ প্রোগ্রাম দিয়ে শুরু করা যাক

```
const printer = (n) => {  
  for(let i = 0; i*i<=n; i++){  
    console.log("shakil");  
  }  
}  
  
printer(25);
```

আপনি কি বলতে পারবেন, উপরোক্ত প্রোগ্রামের টাইম কমপ্লিক্সিটি কত? একটু লক্ষ্য করলে দেখবেন যে, এখানে  $i$  এর মান ১ করে বাড়তেছে কিন্তু কন্ডিশন  $i*i$  এইভাবে চেক করতেছে। তাহলে লুপটি কতবার চলবে আর shakil লিখাটি কতবার প্রিন্ট করবে?

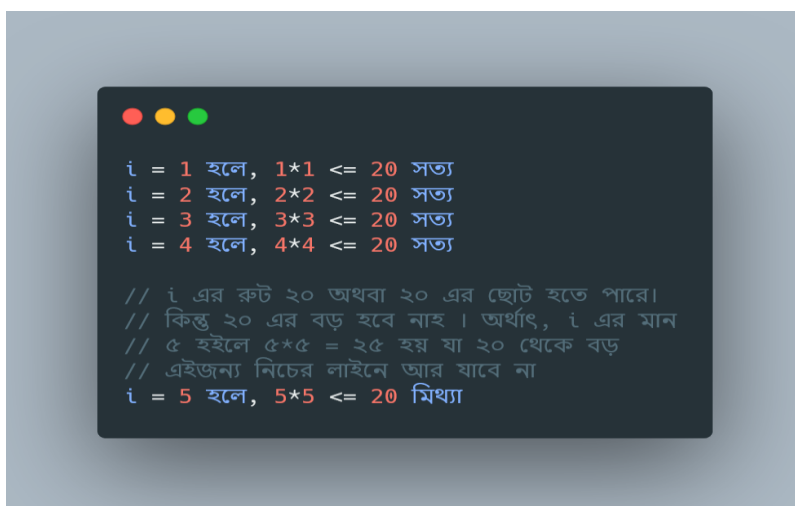


২৫ এর জন্য লুপটি ৫ বার ঘুরতেছে তাহলে shakil লিখাটি ৫ বার প্রিন্ট করবে। যদি প্যাঁরামিটার ৩৬ হয়তো তাহলে লুপটি ৬ বার চলতো অথবা প্যাঁরামিটার ৪৯ হয়তো লুপটি ৭ বার চলতো এবং লুপ অনুযায়ী shakil লিখাটি প্রিন্ট করতো।

এখানে একটা জিনিস খেয়াল করুন আমরা প্যাঁরামিটার হিসেবে যাই পাস করি নাহ কেন তার রুট(root)  $\sqrt{}$  সংখ্যক বার লুপটি চলতেছে।

তাহলে এই প্রোগ্রামের টাইম কমপ্লিক্সিটি হইলো অর্ডার অব রুট এন  $O(\sqrt{n})$ ।

আচ্ছা এইবার আমি যদি প্যাঁরামিটারে ২০ পাঠাই তাহলে কতবার লুপটি চলবে? একটু ভাবুন তো আশা করি আপনারা পেরেছেন। হ্যাঁ ৪ বার লুপটি চলবে –



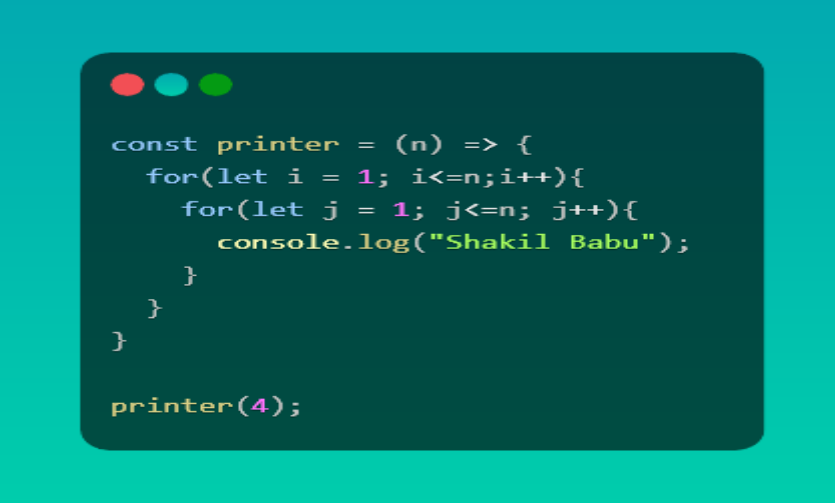
```
i = 1 হলে, 1*1 <= 20 সত্য
i = 2 হলে, 2*2 <= 20 সত্য
i = 3 হলে, 3*3 <= 20 সত্য
i = 4 হলে, 4*4 <= 20 সত্য

// i এর রুট ২০ অথবা ২০ এর ছোট হতে পারে।
// কিন্তু ২০ এর বড় হবে নাহ। অর্থাৎ, i এর মান
// ৫ হইলে ৫*৫ = ২৫ হয় যা ২০ থেকে বড়
// এইজন্য নিচের লাইনে আর যাবে না
i = 5 হলে, 5*5 <= 20 মিথ্যা
```

যখন i এর মান ১ তখন  $1*1 = 1$  যা ২০ থেকে ছোট। এভাবে i এর মান ৪ পর্যন্ত যাবে যা উপরোক্ত কন্ডিশনের সাথে ম্যাচ করে। তাহলে বলতে পারি যে, এই প্রোগ্রামের টাইম কমপ্লিক্সিটি অর্ডার অব রুট এন  $O(\sqrt{n})$ ।



এবার একটি নেস্টেড লুপসহ প্রোগ্রাম দেখা যাক।-



```
const printer = (n) => {  
  for(let i = 1; i<=n;i++){  
    for(let j = 1; j<=n; j++){  
      console.log("Shakil Babu");  
    }  
  }  
}  
  
printer(4);
```

### ফ্লোচার্ট (Flowchart)

অ্যালগরিদমের চিত্রাভিত্তিক রূপকেই ফ্লোচার্ট (Flowchart) বলে অর্থাৎ কোনো সমস্যা সমাধানের নির্দিষ্ট ধাপসমূহের চিত্রাভিত্তিক রূপই হলো ফ্লোচার্ট (Flowchart)।

কোনো একটা সমস্যা সমাধান করার সময় প্রথমে আমরা অ্যালগরিদম উন্নয়ন করি তারপর সেই অনুযায়ী যেকোনো একটা প্রোগ্রামিং ভাষা দিয়ে ইমপ্লিমেন্ট করি।

কিন্তু অনেক সময় আমাদের বেশ বড় সমস্যা সমাধান করার প্রয়োজন হয় তখন অ্যালগরিদমের পাশাপাশি যদি ফ্লোচার্ট (Flowchart) ও উন্নয়ন করি তাহলে প্রোগ্রাম ইমপ্লিমেন্ট করতে খানিকটা সহজ হয়। অথবা কোনো একটা সমস্যা সমাধানের অ্যালগরিদম ও ফ্লোচার্ট (Flowchart) উন্নয়ন করা থাকলে অন্য প্রোগ্রামার খুব সহজেই সেটি ইমপ্লিমেন্ট করতে পারবে।

তাছাড়া অ্যালগরিদমের চেয়ে দ্রুত ফ্লোচার্ট (Flowchart) দেখে প্রোগ্রামের মূল কার্যকারিতা বোঝা যায় এবং এই অনুযায়ী প্রোগ্রাম রচনা করা যায়।

### **ফ্লোচার্ট (Flowchart) এর প্রকারভেদ -**

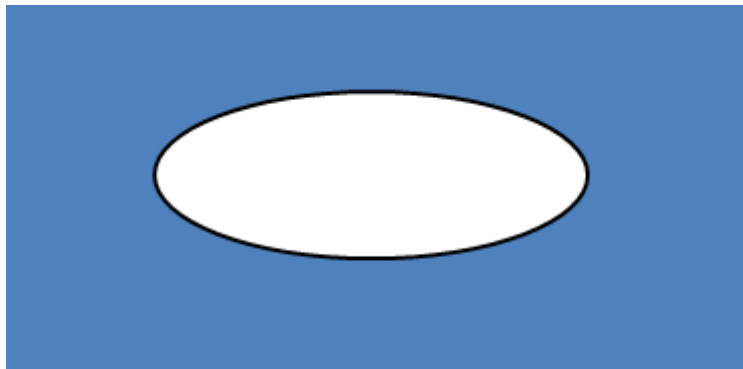
ফ্লোচার্ট (Flowchart) দুই প্রকার - ১। প্রোগ্রাম ফ্লোচার্ট এবং ২।  
সিস্টেম ফ্লোচার্ট

তবে আমরা কোনো প্রোগ্রামের জন্য কিভাবে ফ্লোচার্ট তৈরি করতে হয় তা জানবো অর্থাৎ প্রোগ্রাম ফ্লোচার্ট সম্পর্কে জানবো যা দিয়ে আমরা আমাদের প্রোগ্রাম রিলেটেড সমস্যার সমাধান করতে পারি।

### **প্রোগ্রাম ফ্লোচার্ট -**

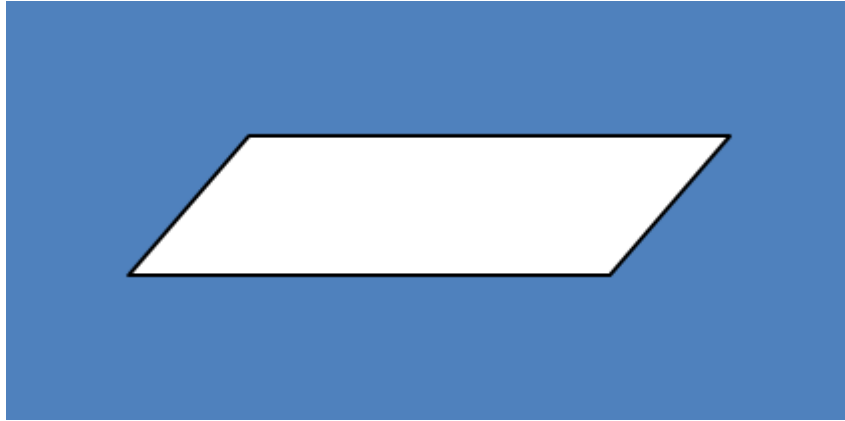
এ ফ্লোচার্ট অনুসারে প্রোগ্রাম রচনা করা হয়। তাছাড়াও প্রোগ্রামে ভুল নির্ণয়, সংশোধন এবং প্রোগ্রাম পরীক্ষার জন্য এ ফ্লোচার্ট ব্যবহার করা হয়। নিম্নে প্রোগ্রাম ফ্লোচার্ট এর কিছু কমন সিম্বল নিয়ে আলোচনা করা হলো।

#### **১। শুরু / শেষ প্রতীক-**



আমরা জানি যে, কোনো একটা অ্যালগরিদম লিখার সময় আমাদের প্রথমে প্রোগ্রাম শুরু লিখতে হয় এবং শেষে প্রোগ্রাম শেষ বা শুধু শেষ লিখলেই হয়। ফ্লোচার্টে(Flowchart) এই শুরু এবং শেষ করার জন্য উপরোক্ত ডিম্বাকৃতির প্রতীক ব্যবহার করা হয়। এবং এই সিম্বলের ভেতরে start এবং end লিখা হয়।

## ২। ইনপুট / আউটপুট প্রতীক -



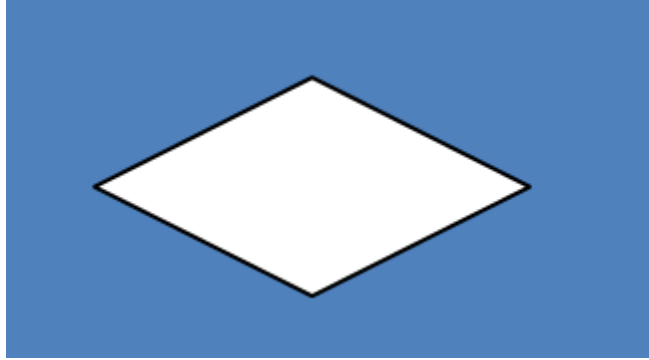
যখন কোনো ভ্যালু ইউজারের কাছে থেকে নেওয়া হয় অর্থাৎ ইনপুট এবং যখন কোনো ভ্যালু প্রিন্ট করা হয় অর্থাৎ আউটপুট মোদা কথা ইনপুট/ আউটপুট প্রকাশের জন্য উপরোক্ত সামান্তরিক প্রতীক ব্যবহার করা হয়।

## ৩। প্রসেস / প্রক্রিয়াকরণ প্রতীক -



বিভিন্ন ধরনের গাণিতিক ও যৌক্তিক অপারেশন সম্পন্ন করার জন্য এই আয়তাকার প্রতীক ব্যবহার করা হয়।

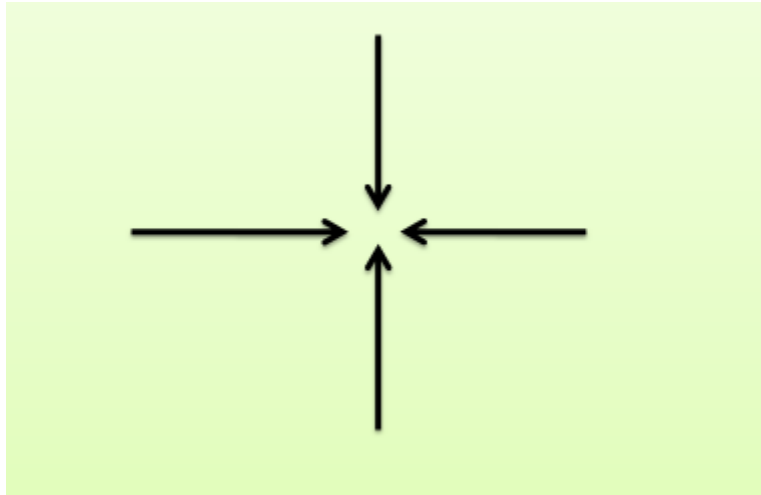
#### ৪। সিদ্ধান্ত গ্রহণ প্রতীক -



শর্ত বা কন্ডিশন সম্পর্কিত কার্যাদি সম্পাদনের ক্ষেত্রে এই ডায়মন্ড আকৃতির প্রতীক ব্যবহার করা হয়।

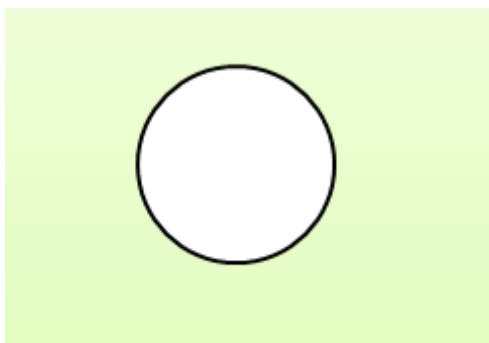
এখানে কন্ডিশন অনুযায়ী সিদ্ধান্ত Yes অথবা No হতে পারে।

#### ৫। প্রবাহ রেখা প্রতীক -



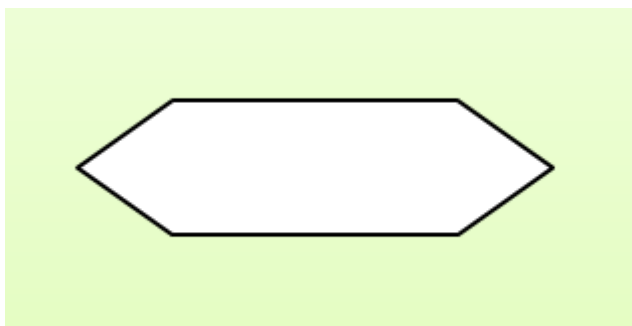
প্রোগ্রাম বা অ্যালগরিদমের ধাপসমূহের ধারাবাহিকতা নির্দেশ করার জন্য উপরোক্ত প্রবাহ রেখা ব্যবহার করা হয়। অর্থাৎ অ্যালগরিদমের ফ্লো ঠিক কোনদিকে তা বোঝানোর জন্যই এই অ্যারো প্রতীক গুলো ব্যবহার করা হয়।

## ৬। সংযোগ রেখা প্রতীক -



যখন কোনো বড় ফ্লোচার্ট (Flowchart) এক পৃষ্ঠায় ধরে নাহ তখন এই সংযোগ প্রতীক ব্যবহার করে বাকি অংশ অপর পৃষ্ঠায় আঁকা হয়।

## ৭। লুপ প্রতীক -

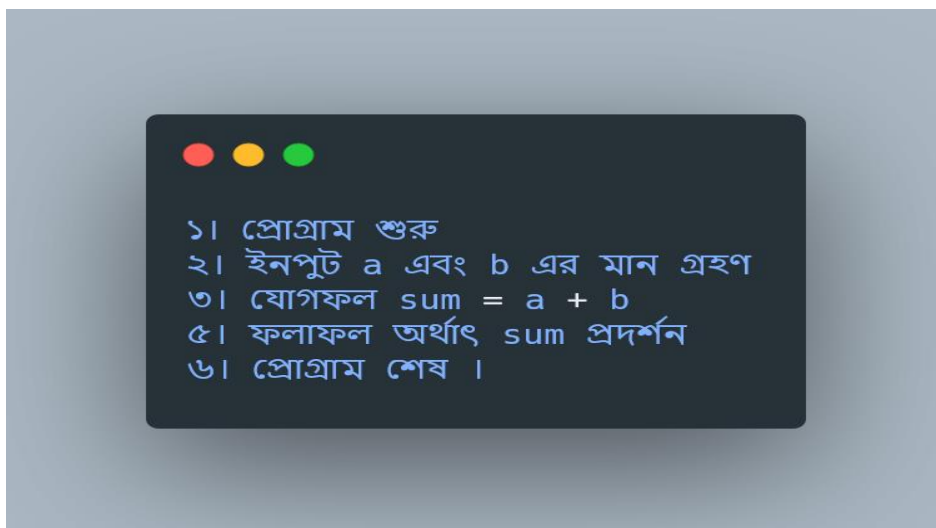


লুপিং স্টেটমেন্টসমূহকে লেখার জন্য এই ষড়ভুজ আকৃতির প্রতীক ব্যবহার করা হয়।

এছাড়াও প্রোগ্রাম ফ্লোচার্টে আরও কিছু প্রতীক আছে তবে সবচেয়ে বেশি উপরোক্ত ৭ টি ব্যবহার করা হয়। যাদের সাহায্যে কোনো সমস্যার ফ্লোচার্ট (Flowchart) তৈরি করা হয়। এখন আমরা উপরোক্ত প্রতীক গুলো দিয়ে দু-একটি ফ্লোচার্ট (Flowchart) তৈরি করা দেখবো।

**সমস্যা ০১ - দুইটি সংখ্যার যোগফল বের করার অ্যালগরিদম এবং ফ্লোচার্ট (Flowchart) দেখাও :-**

আমরা কিন্তু অবশ্যই দুটি সংখ্যার যোগফল বের করার অ্যালগরিদম লিখতে পারবো। যদি নাহ পারেন তাহলে আপনি গত পর্বগুলো ভালো ভাবে দেখেন নি বা পড়েন নি। তবুও আমি এই সহজ সমস্যার অ্যালগরিদম লিখতেছি –

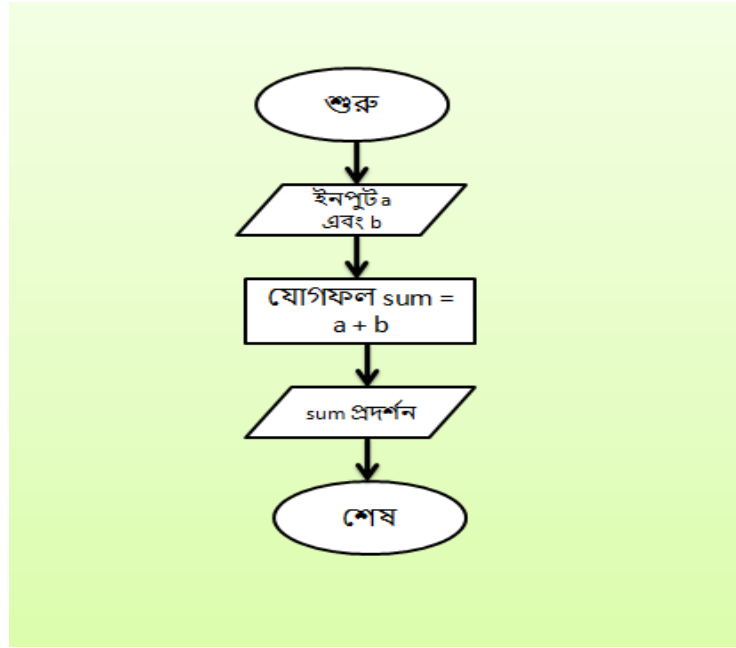


এখন আমরা এর ফ্লোচার্ট (Flowchart) কিভাবে বানাবো? অ্যালগরিদমের দিকে খেয়াল করুন প্রথম ধাপে কিন্তু প্রোগ্রাম শুরু হয়েছে। তাহলে আমরা জানি যে ফ্লোচার্টে (Flowchart) শুরু এবং শেষ বোঝাতে ডিম্বাকৃতির প্রতীক ব্যবহার করা হয়।

এখন অ্যালগরিদমের দ্বিতীয় ধাপে ইনপুটের মান গ্রহণ করা হয়েছে আমরা এটাও জানি যে ফ্লোচার্টে (Flowchart) ইনপুট / আউটপুট প্রকাশের জন্য সামান্তরিক প্রতীক ব্যবহার করা হয়।

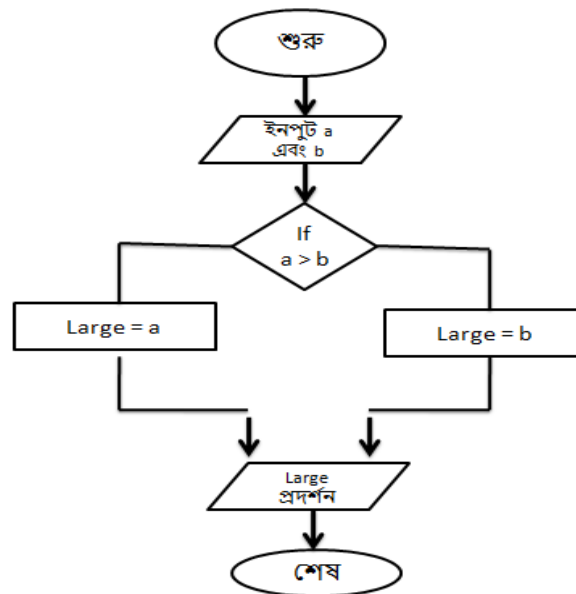
অ্যালগরিদমের তৃতীয় ধাপে যোগ করতেছে যা ফ্লোচার্টে (Flowchart) প্রসেস / প্রক্রিয়াকরণ বা আয়তাকার প্রতীক ব্যবহার করে নেওয়া হয়।

এখন যদি আমরা দুটি সংখ্যার যোগফল বের করার ফ্লোচার্ট (Flowchart) অঙ্কন করি তাহলে সেটি হবে এমন –



**সমস্যা ০২ - দুইটি সংখ্যার মধ্যে বড় সংখ্যা বের করার অ্যালগরিদম এবং ফ্লোচার্ট (Flowchart) দেখাও :-**

আশা করি এই সমস্যার অ্যালগরিদম আপনি খুব সহজেই লিখতে পারবেন। যদি নাহ পারেন তাহলে অ্যালগরিদম নিয়ে আগের পোস্ট গুলো পড়ুন। আমি এখন এই সমস্যার ফ্লোচার্ট (Flowchart) তৈরি করবো যা হবে এমন –



যদি এই সমস্যাটার অ্যালগরিদম আপনি নিজে লিখতে পারেন তাহলে উপরোক্ত ফ্লোচার্ট (Flowchart) বুঝতে আপনার বেগ পেতে হবে নাহ।

## ডাটা স্ট্রাকচার

অ্যারে (Array)-

তোমাকে বলা হল ৩ জন ছাত্রের বয়স ইনপুট নিয়ে তাদের গড় বের করতে। তুমি age0, age1, age2 নামের তিনটি int type এর ভ্যারিয়েবল ডিক্লেয়ার করলে। এরপর তাতে ইনপুট নিলে নিচের মত করেঃ

```
scanf("%d", &age0);  
scanf("%d", &age1);  
scanf("%d", &age2);
```

এরপর এদেরকে যোগ করে ৩ দিয়ে ভাগ করে গড় বের করলে। এ পর্যন্ত করতে কোন সমস্যা নাই।

সমস্যা শুরু হবে যদি বলি কোন ক্লাসের সবার বয়স ইনপুট নিয়ে স্টোর কর। তাদের বয়সের গড় প্রিন্ট কর। তখন ১০০ জন ছাত্রের জন্য age0, age1, age2,..., age99 পর্যন্ত এতগুলো variable declare করে এরপর তাতে ইনপুট নিয়ে যোগ করে গড় বের করা কিন্তু কোন সহজ কাজ নয়।

চিন্তা করো, যদি এমন একটা সিস্টেম করা যেত যে একটা ভ্যারিয়েবলের নাম রাখব ধরো age. এই age এর সাথে ছাত্রদের রোল নম্বর দিয়ে যার যার বয়স ইনপুট দিয়ে রাখ। অর্থাৎ প্রথমে যেই কাজটা করতে চাচ্ছিলাম সেটাই করব তবে অটোমেটিক ভাবে। age0, age1, age2... আলাদা আলাদা ভাবে ডিক্লেয়ার করে আলাদা আলাদা লাইনে ইনপুট নিতে হবে না। এই সিস্টেমটার নামই হচ্ছে অ্যারে (Array).

### Array: A Linear Data Structure

দুই ধরনের ডেটা স্ট্রাকচার রয়েছে। একটা হচ্ছে Linear Data Structure, আরেকটি হচ্ছে Nonlinear Data Structure.

লিনিয়ার ডেটা স্ট্রাকচার হচ্ছে এমন এক ধরনের স্ট্রাকচার যা মেমরিতে sequence অনুযায়ী স্টোর হয়। এই স্ট্রাকচারের ডেটাগুলো একটার পর একটা সিরিয়ালি সাজানো থাকে। এই লিনিয়ার ডেটা স্ট্রাকচারের দুই ধরনের



representation রয়েছে। একটা হচ্ছে Array. সোজা সাপটা ভাবে একই ডেটা টাইপের (int, float, double, char) ডেটাগুলো লাইন ধরে সাজানো থাকে অ্যারের মধ্যে। আরেকটা রিপ্রেজেন্টেশন হচ্ছে, লিস্টের element-গুলোর মধ্যকার সম্পর্ক। এর উদাহরন হচ্ছে Linked List.

নন লিনিয়ার ডেটা স্ট্রাকচারের উদাহরণ হিসেবে উল্লেখ করা যায় Tree স্ট্রাকচারের কথা। Tree-তে অ্যারের মত সিরিয়ালি ডেটা সাজানো থাকে না বা সম্ভবও না। কারণ এখানে root, শাখা-প্রশাখা ইত্যাদির হিসাব-নিকাশ রয়েছে।

অ্যারেকে এক কথায় সংজ্ঞায়িত করতে চাইলে এভাবে বলা যায়, নির্দিষ্ট সংখ্যক ডেটা স্টোর করার জন্য একটা স্ট্রাকচার যেখানে শুধুমাত্র এক ধরনের ডেটাই সংরক্ষণ করা যায়। অর্থাৎ অ্যারের একটা নির্দিষ্ট সাইজ থাকবে। এই সাইজের চেয়ে বেশি ডেটা কোন অ্যারে স্টোর করতে পারবে না। আর একই ধরনের ডেটাই স্টোর করতে হবে। int type ডেটা স্টোর করতে চাইলে সেই অ্যারেতে শুধুমাত্র int type এর ডেটাই স্টোর করা যাবে। সেখানে int, float, double, char ইত্যাদি মিক্স করে স্টোর করা যাবে না। যদি int type একটা ১০০ সাইজের অ্যারে ডিক্লেয়ার করি তাহলে এই অ্যারেতে সর্বোচ্চ ১০০ টা int-ই স্টোর করা যাবে।

যে কোন ডেটা স্ট্রাকচারেই data insert, traverse, update, delete, searching, sorting এর মত ব্যাসিক কিছু কাজ থাকে। এই পোস্টে অ্যারের **Declaration, Insertion, Traverse** এই অপারেশনগুলো দেখানোর চেষ্টা করা হবে।

## Array Declaration

প্রোগ্রামিং ল্যাঙ্গুয়েজ ভেদে অ্যারের ডিক্লেয়ারেশন একটু এদিক সেদিক হয়ে থাকে। এক্ষেত্রে সি প্রোগ্রামিং ল্যাঙ্গুয়েজে অ্যারের সকল অপারেশনগুলো দেখাবো। সি এর কোড বুঝলে যে কোন ল্যাঙ্গুয়েজেই অ্যারে ইমপ্লিমেন্ট করা যাবে।

সর্বোচ্চ ১০০ জন ছাত্রের বয়স যদি আমাদের স্টোর করে প্রসেস করার দরকার হয় সেক্ষেত্রে আমরা অ্যারেটা ডিক্লেয়ার করতে পারি নিচের মত করেঃ

```
.  
int age[100];  
.
```

int হচ্ছে ডেটা টাইপ। ধরে নিলাম বয়স হিসেবে শুধু পূর্ণ সংখ্যাই ইনপুট দেয়া হবে। তাই এখানে int টাইপের অ্যারে নিয়েছি। যদি ভগ্নাংশ নিয়ে কাজ করার

দরকার হয় সেক্ষেত্রে float বা double ডেটা টাইপের অ্যারে ডিক্লেয়ার করতে হবে। এই অ্যারেতে সর্বোচ্চ ১০০ টি পূর্ণ সংখ্যা স্টোর করা যাবে।

Array Representation. Photo Credit: [www.java67.com/](http://www.java67.com/)

### Array Initialization (insert)

প্রথমত দেখি যদি কিছু ফিক্সড ভ্যালু অ্যারেতে স্টোর করতে চাই তাহলে কিভাবে করা যায়।

```
age[0] = 45;  
age[1] = 17;
```

```
.  
. .  
.
```

উপরে দেখা যাচ্ছে age নামক অ্যারের প্রথম ইন্ডেক্সে একটা ভ্যালু (45) assign করা হয়েছে। age[0] অ্যারেটির প্রথম ইন্ডেক্স। সব ল্যাক্সুয়েজেই অ্যারের ইন্ডেক্সিং শুরু হয় শূন্য থেকে। সি ল্যাক্সুয়েজে অ্যারের প্রতিটা ইন্ডেক্স অ্যাক্সেস করতে হয় অ্যারের নাম দিয়ে এরপর 3rd bracket এর ভিতরে ইন্ডেক্সের নাম্বার লিখে। age অ্যারের সর্বশেষ ইন্ডেক্স হচ্ছে ৯৯। সর্বশেষ ইন্ডেক্স বা অ্যারের সর্বশেষ খোপে যদি কোন মান অ্যাসাইন করতে চাই তাহলে লিখতে হবে এভাবেঃ age[99] = 65;

একটা বিষয় লক্ষ্যনীয়, অ্যারের ইন্ডেক্স নাম্বার আর অ্যারের ইন্ডেক্সের ভ্যালু কিন্তু ভিন্ন জিনিস। age[1] = 17; বলতে বুঝায় age নামক অ্যারেতে যতগুলো ইন্ডেক্স বা খোপ আছে তাদের মধ্য থেকে ১ নাম্বার খোপে ১৭ মানটা বসিয়ে দাও। ১ হচ্ছে খোপের নাম্বার। এই নাম্বারিং এর মাধ্যমেই কিন্তু আমরা লিনিয়ার অ্যারে ইমপ্লিমেন্ট করতে পারছি। ০, ১, ২, ৩, ... এভাবে এই খোপের সংখ্যাগুলো বাড়ছে। আর ১৭ হচ্ছে ছাত্রের বয়স। যেটা অরিজিনাল ডেটা বা ভ্যালু। এক কথায় বললে ১৭ ভ্যালুটাকে age অ্যারের ১ নাম্বার ইন্ডেক্সে বসানোর জন্য age[1] = 17; লিখতে হবে। আশা করি ইন্ডেক্স নাম্বার আর ইন্ডেক্স ভ্যালু গুলিয়ে ফেলবে না আর।

এখন দেখব সি ল্যাক্সুয়েজ দিয়ে একটা অ্যারেতে কিভাবে ইনপুট নিতে হয়। অ্যারে নিয়ে কাজ করতে গেলে Loop এর পরিষ্কার ধারণা থাকতে হবে। যদি লুপের মধ্যে ঝামেলা থাকে তাহলে উচিত হবে লুপটা একটু রিভাইস দিয়ে এসে বাকি লেখাটা পড়া।

তোমরা চাইলে ম্যারাথন স্টাইলে ইনপুট নিতে পারো কোন রকমের লুপের ইউজ ছাড়াই।

```
scanf("%d", &age[0]);  
scanf("%d", &age[1]);  
scanf("%d", &age[2]);  
.  
.  
.
```

কিন্তু উপরের সিসটেমে কেউ অ্যাারেতে ইনপুট করে না। লুপের মাধ্যমেই ইনপুট করতে হয়। আমরা নিচে নির্দিষ্ট সংখ্যক ছাত্রের বয়স ইনপুট দেয়ার জন্য কোড লিখব। number\_of\_student একটি int type variable. এতে ইনপুট নেয়া হচ্ছে কতজন ছাত্রের বয়স ইনপুট নেয়া হবে।

```
scanf("%d" &number_of_student);  
  
for(i = 0; i<number_of_student; i++)  
{  
    scanf("%d", &age[i]);  
}
```

উপরের কোডে লুপের ভিতর ইনপুট নেয়ার কাজ চলতে থাকবে। একদম শুরুতে অ্যারের প্রথম ইন্ডেক্স 0-তে ইনপুট হবে। এরপর 1, 2, ... (number\_of\_student-1) পর্যন্ত সবগুলো ইন্ডেক্সে ইনপুট হবে।

সব সময় যে সিরিয়ালি সবগুলো ইন্ডেক্সেই মান ইনপুট নিতে হবে এমন না। তুমি চাইলে এক ঘর বাদ দিয়ে দিয়েও ইনপুট নিতে পারো। যেমনঃ age[0], age[2], age[4]... এগুলোতে ইনপুট নিবে, কিন্তু বাকিগুলোতে নিবে না। এখানে তোমাকে ইনপুট নেয়া শেখানো হল। পরবর্তীতে কখন কিভাবে কী কাজে লাগতে হবে সেটা তুমি সিদ্ধান্ত নিবা।

## Array Traversing

তুমি ১০০ সাইজের একটা অ্যারে ডিক্লেয়ার করলা। এরপর তাতে কিছু ডেটা রাখলা এরপর কাজ কী? এরপর হয় তোমাকে সেই ডেটাগুলো প্রিন্ট করতে হবে, বা নির্দিষ্ট কোন ডেটার উপর নির্দিষ্ট কোন কাজ করা লাগতে পারে। কোন একটা ডেটা সার্চ করার দরকার হতে পারে অথবা কোন একটা শর্ত অনুযায়ী ডেটাগুলোকে সাজানো লাগতে পারে। এর মানে হচ্ছে তোমার মেমরি সংরক্ষিত

ডেটাগুলোতে তুমি অ্যাক্সেস করবে বা ডেটাগুলোর উপর তুমি ভ্রমণ (traverse) করবে।

যেভাবে লুপ চালিয়ে প্রতিটা ইন্ডেক্সে ইনপুট নিয়েছিলাম, একই ভাবে লুপ চালিয়ে প্রতিটা ইন্ডেক্সে ট্রাভার্স করতে পারি। এই ট্রাভার্সের মাধ্যমে চাইলে কোন ডেটা প্রিন্ট করতে পারি, কোন ডেটার সাথে অ্যারিথমেটিক কোন অপারেশন ঘটাতে পারি। কোন ডেটা মুছে দিতে পারি ইত্যাদি।

এখন সবগুলো ইন্ডেক্সের ভ্যালুগুলো একেকটা লাইনে প্রিন্ট করতে চাইলে নিচের কোডটা লিখা যায়ঃ

```
for(i = 0; i < number_of_student; i++)
{
    printf("%d\n", age[i]);
}
```

তাহলে age অ্যারের 0-তম ইন্ডেক্স থেকে (number\_of\_student - 1)-তম ইন্ডেক্স পর্যন্ত সবগুলো ইন্ডেক্সের ভ্যালু পরপর লাইনে প্রিন্ট করে দিবে। যদি শেষ থেকে শুরুর element-গুলো প্রিন্ট করতে চাও তাহলে লুপটাকে একটু মডিফাই করলেই কাজ করবে। যথাঃ for(i = number\_of\_student - 1; i >= 0; i--){{}}. অর্থাৎ অ্যারের শেষ ইন্ডেক্স থেকে প্রথম ইন্ডেক্সের সবগুলো মান প্রিন্ট হবে।

তুমি ইচ্ছা করলে নির্দিষ্ট একটা ইন্ডেক্সের ভ্যালুও ইন্ডেক্স নাম্বারের সাহায্যে প্রিন্ট করতে পারোঃ

```
printf("%d", age[3]);
```

উপরের কোডে age array এর 3 নাম্বার ইন্ডেক্সের মানটা প্রিন্ট হবে।

কখনো যদি এমন হয় যে ইউজার লুপ ঘুরিয়ে ১০ টি ইন্ডেক্সের নাম্বার ইনপুট দিবে। সেই ইন্ডেক্সের নাম্বার ইনপুট দিবে সেই ইন্ডেক্সের ভ্যালু প্রিন্ট করতে হবে। তাহলে কী করবা?

```
for(i = 1; i <= 10; i++)
{
    scanf("%d", &index);

    printf("%d", age[index]);
}
```

উপরের কোডে প্রতিবার একটা ইন্ডেক্সের মান ইনপুট নেয়া হচ্ছে। পরের লাইনে সেই ইন্ডেক্স এর ভ্যালু প্রিন্ট করা হচ্ছে।

সব ছাত্রের বয়সের গড় বের করতে চাইলে কী করতে হবে? ধরো প্রথমে অ্যারেতে ইনপুট নেয়া হল। এরপর অ্যারেতে লুপ চালিয়ে ট্রাভার্স করব। প্রতিটা ইন্ডেক্সের ভ্যালু যোগ করব এরপর number\_of\_student দিয়ে ভাগ করব।

```
for(i = 0; i<number_of_student; i++)  
{  
    sum = sum + age[i];  
}
```

```
average = sum/number_of_student;
```

```
printf("%d", average);
```

### অ্যারে কম্প্রেশন:

খুবই সহজ কিন্তু দরকারি একটা টেকনিক নিয়ে আলোচনা করবো আজকে। STL এর ম্যাপ ব্যবহার করে আমরা অ্যারে কম্প্রেশন করবো। মনে করো কোনো একটা প্রবলেমে তোমাকে বলা হলো একটি অ্যারেতে ১ লাখটা সংখ্যা দেয়া থাকবে যাদের মান হবে ০ থেকে সর্বোচ্চ ১০০০। এখন যেকোনো আমি একটি সংখ্যা বললে তোমাকে বলতে হবে অ্যারের কোন কোন পজিশনে সংখ্যাটি আছে। যেমন মনে করো ইনপুট অ্যারেটা হলো:

১ ০ ০ ২ ৫ ২ ১ ০ ৪ ৫ ১ ২

খুবই সহজ সলিউশন হলো একটি ভেক্টর নেয়া। i তম পজিশনে x সংখ্যাটি পেলে ভেক্টরের x তম পজিশনে পুশ করে রাখবে i। তাহলে ইনপুট নেবার পর ভেক্টরগুলোর চেহারা হবে:

[0]->1 2 7

[1]->0 6 10

[2]->3 5 11

[3]->empty

[4]->8

[5]->4 9

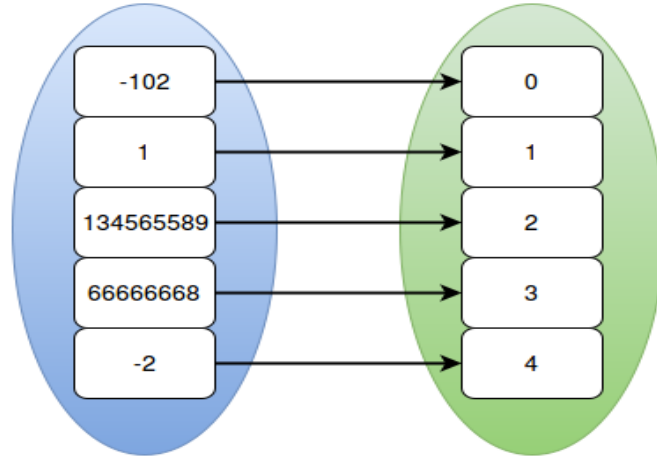
কোন সংখ্যা কোন কোন পজিশনে আছে তুমি পেয়ে গেলে। এখন যদি বলে 2 কোথায় কোথায় আছে তাহলে তুমি ২ নম্বর ইনডেক্সে লুপ চালিয়ে 3,5,11 প্রিন্ট করে দাও। 1001 সাইজের ২-ডি ভেক্টর দিয়েই কাজ হয়ে যাবে।

এখন তোমাকে বলা হলো সংখ্যাগুলো নেগেটিভও হতে পারে এবং সর্বোচ্চ  $2^{30}$  পর্যন্ত হতে পারে। এবার কি এই পদ্ধতি কাজ করবে? একটি সংখ্যা -100 বা  $2^{30}$  হলে তুমি সেটার পজিশনগুলো কোন ইনডেক্সে রাখবে? অবশ্যই তুমি এতো বড় ভেক্টর ডিক্লেয়ার করতে পারবেনা অথবা নেগেটিভ ইনডেক্স ব্যবহার করতে পারবেনা। ধরো ইনপুট এবার এরকম:

```
input[]={-102,1,134565589,134565589,-  
102,66666668,134565589,66666668,-102,1,-2}
```

একটা ব্যাপার লক্ষ্য করো, সংখ্যা দেয়া হবে সর্বোচ্চ  $10^5$  বা ১ লাখটা। তাহলে অ্যাারেতে মোট ভিন্ন ভিন্ন সংখ্যা হতে পারে কয়টা? অবশ্যই সর্বোচ্চ ১ লাখটা। আমরা প্রতিটি ভিন্ন ভিন্ন সংখ্যাকে ছোটো কিছু সংখ্যার সাথে one-to-one ম্যাপিং করে ফেলবো। উপরের অ্যাারেতে ভিন্ন ভিন্ন সংখ্যা গুলো হলো:

-102,1,134565589,66666668,-2



দুইটা উপায় আছে ম্যাপিং করার। একটা হলো STL এর map ব্যবহার করে। map এর কাজ হলো যেকোনো একটি সংখ্যা, স্ট্রিং বা অবজেক্টকে অন্য একটি মান অ্যাসাইন করা যেটা আমরা উপরের ছবিতে করেছি। আমাদের এই প্রবলেমে কোন সংখ্যা কার থেকে বড় বা ছোটো সেটা দরকার নাই তাই সর্ট না করেই ম্যাপিং করে দিতে পারি। নিচের কোডটা দেখো:

```
void compress() {  
    map < int, int > mymap;  
    int input[] = {  
        -102,  
        1,  
        134565589,  
        134565589,  
        -102,  
        66666668,  
        134565589,  
        66666668,  
        -102,  
        1,  
        -2
```

```

};

int assign = 0, compressed[100], c = 0, n = sizeof(input) / sizeof(int);
//array size;

for (int i = 0; i < n; i++) {
    int x = input[i];
    if (mymap.find(x) == mymap.end()) { //x not yet compressed
        mymap[x] = assign;
        printf("Mapping %d with %d\n", x, assign);
        assign++;
    }
    x = mymap[x];
    compressed[c++] = x;
}

printf("Compressed array: ");
for (int i = 0; i < n; i++) printf("%d ", compressed[i]);
puts("");
}

```

কোডের আউটপুট হবে:

Mapping -102 with 0

Mapping 1 with 1

Mapping 134565589 with 2

Mapping 66666668 with 3

Mapping -2 with 4

Compressed array: 0 1 2 2 0 3 2 3 0 1 4



আমরা ইনপুট অ্যারেতে যখনই একটু নতুন সংখ্যা পাচ্ছি সেটাকে একটা ভ্যালু অ্যাসাইন করে দিচ্ছি এবং আসল ভ্যালুকে ম্যাপের ভ্যালু দিয়ে রিপ্লেস করে আরেকটি অ্যারেতে রেখে দিয়েছি। এরপরে আমরা নতুন অ্যারেটা ব্যবহার করে প্রবলেমটা সলভ করতে পারবো। কুয়েরিতে যদি বলে -2 কোথায় কোথায় আছে বের করো তাহলে তুমি আসলে 4 কোথায় কোথায় আছে বের করবে কারণ `mymap[-2]=4`।

অনেক সময় কমপ্রেস করার পরেও কোন সংখ্যাটি কার থেকে বড় এটা দরকার হয়। যেমন তোমাকে বলতে পারে যে একটি ভ্যালু নেয়ার পর ছোটো কোনো ভ্যালু আর নিতে পারবেনা। এই তথ্যটা কিভাবে রাখবে উপরের পদ্ধতিতে ১ এর থেকে -২ এ অ্যাসাইন করা ভ্যালু বড় হয়ে গিয়েছে। যদি তুমি সেটা না চাও তাহলে আগে ভিন্ন ভিন্ন ভ্যালুগুলো আরেকটা অ্যারেতে স্টোর্ড করে নাও।

```
sorted[]={-102,-2,1,66666668,134565589}
```

এইবার sorted অ্যারেটা ব্যবহার করে ম্যাপিং করে ফেলো। এ ক্ষেত্রে আসলে যে সংখ্যাটি sorted অ্যারেতে যে পজিশনে আছে সেটাই হবে তার ম্যাপ করা ভ্যালু।

এটা যদি বুঝতে পারো তাহলে নিশ্চয়ই ম্যাপ ছাড়াই বাইনারি সার্চ করে তুমি কমপ্রেস করে ফেলতে পারবে অ্যারেটাকে। ইনপুট অ্যারের উপর লুপ চালাও, প্রতিটি x এর জন্য দেখো sorted অ্যারেতে x এর অবস্থান কোথায়। সেই মানটা তুমি আরেকটা অ্যারেতে রেখে দাও:

```
input[]={-102,1,134565589,134565589,-  
102,66666668,134565589,66666668,-102,1,-2}  
compressed_input[]={0,2,4,4,0,3,4,3,0,2,1}
```

কুয়েরির সময়ও বাইনারী সার্চ করে কমপ্রেস করার মানটা বের করে কাজ করতে হবে।

তুমি ম্যাপ বা বাইনারী সার্চ যেভাবে ইচ্ছা কমপ্রেস করতে পারো। প্রতিবার map এক্সেস করতে logn কমপ্লেক্সিটি লাগে যেটা বাইনারী সার্চের সমান। stl এ বিভিন্ন class, ডাইনামিক মেমরির ব্যবহারের জন্য map কিছুটা স্লো, তবে টাইম লিমিট খুব tight না হলে খুব একটা সমস্যা হবে না। map ব্যবহার করলে কোডিং সহজ হয়।

কোনো কোনো গ্রাফ প্রবলেমে নোডগুলো আর এজগুলো কে string হিসাবে ইনপুট দেয়। যেমন ধরো গ্রাফের ৩টা এজ হলো:

3

BAN AUS

AUS SRI

SRI BAN

map এ string কেও integer দিয়ে ম্যাপিং করা যায়। এই সুবিধাটা ব্যবহার করে প্রতিটি নোডকে একটি ভ্যালু অ্যাসাইন করবো:

```
map < string, int > mymap;
```

```
int edge, assign = 0;
```

```
cin >> edge;
```

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

```
    char s1[100], s2[100];
```

```
    cin >> s1 >> s2;
```

```
    if (mymap.find(s1) == mymap.end()) {
```

```
        printf("Mapping %s with %d\n", s1, assign);
```

```
        mymap[s1] = assign++;
```

```

}

if (mymap.find(s2) == mymap.end()) {
    printf("Mapping %s with %d\n", s2, assign);
    mymap[s2] = assign++;
}

int u = mymap[s1];
int v = mymap[s2];

cout << "Edge: " << u << " " << v << endl;
}

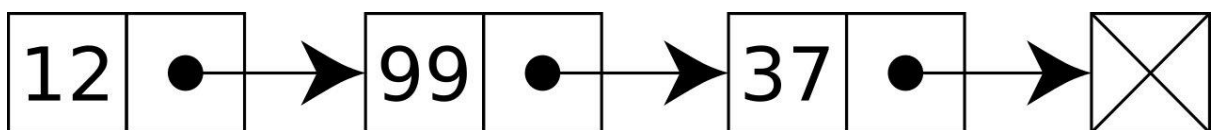
```

অনেক সময় বলা হতে পারে নোডগুলোর মান হবে  $-2^{30}$  থেকে  $2^{30}$  পর্যন্ত কিন্তু সর্বোচ্চ নোড হবে ১০০০০ টা, তখন আমরা নোডগুলোকে একই ভাবে ছোটো ভ্যালু দিয়ে ম্যাপিং করে ফেলবো।

2-d grid ও কমপ্রেস করে ফেলা যায় অনেকটা এভাবে। সেটা নিয়ে আরেকদিন আলোচনা করতে চেষ্টা করবো, তবে নিজে একটু চিন্তা করলেই বের করতে পারবে।

### ❖ লিংকড লিস্ট [Singly Linked List Create & Print in C]

জ্ঞানগুরু উইকিপিডিয়ার ভাষ্য মতে "A linked list is a linear collection of data elements, called nodes, each pointing to the next node by means of a pointer. It is a data structure consisting of a group of nodes which together represent a sequence." অর্থাৎ লিংকড লিস্ট হচ্ছে কিছু নোডের লিনিয়ার কালেকশন, যেই নোডগুলো একেকটা তার পরেরটাকে পয়েন্টারের মাধ্যমে পয়েন্ট করে। এটা একটা ডেটা স্ট্রাকচার, যেখানে নোডগুলো একত্রে একটা সিকোয়েন্স তৈরি করে থাকে।



মাথার উপর দিয়ে গেল তাই না? গেলে যাক! মূল ব্যাপারটা যেহেতু বুঝেই গেল এত গুরুগম্ভীর আলোচনাকে পাত্তা না দিলেও চলবে।

কোডিং পাঠে ঢ়ুকার আগে তেতার কছু বিষয়ে নলেজ থাকতে হবে। যেমনঃ অ্যারে, স্ট্রাকচার, পয়েন্টার ও হালকা পাতলা রিকার্সন। অ্যারের উপর বিস্তারিত লেখাগুলো পাওয়া যাবে এখানে। রিকার্সনের প্রাথমিক ধারণা পেতে পারো আমার ব্লগের রিকার্সন সিরিজ থেকে। বাকি টপিকগুলো ব্লগে এখনো লিখি নাই। গুগল করে শিখে ফেলতে পারো।

ফিরে যাই পোস্টের শুরুতে উল্লেখ করা উদাহরণে। সেখানে আমরা ঁকটা পোস্টের সাথে আরেকটা পোস্টকে লিংক করেছিলাম। পরের পোস্টের সাথে লিংক করেছিলাম সেই পোস্টের অ্যাড্রেসের মাধ্যমে। পোস্টের শেষে উল্লেখ করা অ্যাড্রেসে ক্লিক করলে সেই পোস্টটি পড়া যায়। অর্থাৎ কছু ডেটা পাওয়া যায়। অ্যাড্রেসের কাজ কিন্তু সেরেফ লিংক করা। মূল জিনিস বা মূল উদ্দেশ্য কিন্তু ব্লগের লেখাটা পড়া তাই না? তাহলে উদাহরণের এলিমেন্টগুলোকে দুই ভাগ করি। প্রথম ভাগ বা মূল জিনিস হচ্ছে ব্লগ পোস্টের কনটেন্ট বা লেখাগুলো। আর দ্বিতীয় অংশ হচ্ছে পরের পোস্টের ওয়েব অ্যাড্রেস যা শুধু লিংক করার জন্য ব্যবহৃত হবে। মনে করো প্রতিটা ব্লগ পোস্ট ঢুকলে তুমি বড় করে ঁকটা নাম্বার দেখতে পাবে। ঁটাই ধরে নাও মূল ডেটা। আর ঁই নাম্বারের পরে ছোট্ট করে পরের পোস্টের অ্যাড্রেস দেখতে পাবে।

তুমি যেহেতু স্ট্রাকচার জানো তাই বলছি। ঁই নাম্বার আর অ্যাড্রেসকে ঁকটা ডেটা হিসেবে কিভাবে উল্লেখ করা যায়? খুব সহজেই ঁকটা স্ট্রাকচার বানিয়ে ফেলতে পারো ঁভাবেঃ

```
struct blog_post
{
    int number;
    string address;
};
```

তাহলে blog\_post হচ্ছে ঁকটা স্ট্রাকচার। ঁটাতে রয়েছে ব্লগের মূল ডেটা (number) ঁবং পরের পোস্টের অ্যাড্রেস (address). ঁবার ঁই blog\_post টাইপের ডেটার কয়েকটা ভেরিয়েবল বানিয়ে ফেলি।

·  
blog\_post blog\_post1, blog\_post2, blog\_post3;  
·

তিনটা ব্লগ পোস্ট তৈরি করা হয়েছে। আমরা যদি প্রথমটার address ভেরিয়েবলে দ্বিতীয় পোস্টের লিংকটা রাখতে পারি, দ্বিতীয় পোস্টের address ভেরিয়েবলে তৃতীয় পোস্টের লিংক রাখতে পারি তাহলে কিন্তু প্রথমটার অ্যাড্রেসে ক্লিক করলে দ্বিতীয় পোস্ট, দ্বিতীয় পোস্টের নিচে থাকা অ্যাড্রেসে ক্লিক করলে তৃতীয় পোস্টটি পড়তে পারব। যেহেতু ৩ টাই মাত্র পোস্ট। তাই তৃতীয় পোস্টের অ্যাড্রেসে আপাতত NULL রেখে দিতে পারি। কারণ পরে আর কোন পোস্ট নাই। নতুন কোনো পোস্ট যোগ হলে তার লিংকটা রেখে দিব blog\_post3 এর address এ। আর নতুনটার address এ রেখে দিব NULL. এই আইডিয়াটাই লিংকড লিস্ট। এই আইডিয়া কাজে লাগিয়ে আমরা সত্যিকারের লিংকড লিস্ট ইমপ্লিমেন্ট করবো।

এক কথায় লিংকড লিস্টের সংজ্ঞা বলতে চাইলে বলা যায় লিংকড লিস্ট হচ্ছে কতগুলো স্ট্রাকচারের একটা লিস্ট। যেই স্ট্রাকচারগুলোর মধ্যে এক বা একাধিক ডেটা থাকতে পারে। এবং পরবর্তী স্ট্রাকচারের মেমরি অ্যাড্রেস থাকে। অন্যান্য ডেটা স্ট্রাকচারের মত লিংকড লিস্ট ডেটা স্ট্রাকচারেরও কিছু কমন অপারেশন রয়েছে। সেগুলো আমরা আস্তে আস্তে কভার করবো।

## Operations of Linked List

Create linked list

Traverse

Counting the list item

Print the full list

Search an item on list

Insert a new item on list

Delete an item from list

Concatenate two linked list

Types of Linked List:

Linear Singly Linked List

Circular Linked List

Doubly Linked List

Circular Doubly Linked List

এই পোস্টে প্রথম টাইপের লিঙ্কড লিস্ট নিয়েই আলোচনা করা হবে। পরবর্তী পোস্টে বাকিগুলো নিয়ে আলোকপাত করার ইচ্ছা আছে।

Problem Definition

তোমাকে একটা প্রোগ্রাম লিখতে হবে যেটা ডায়নামিক্যালি একটা int টাইপের লিস্ট তৈরি করতে পারে। অর্থাৎ ইউজার আগে থেকে ইনপুট দিবে না যে সে কয়টা এলিমেন্টের লিস্ট তৈরি করতে চায়। ইউজার হয়ত কখনো ৫ টা সংখ্যার লিস্ট তৈরি করবে, আবার কখনো ৫০০০ সংখ্যার লিস্ট তৈরি করবে। শর্ত হচ্ছে সে যতটা সংখ্যার লিস্ট তৈরি করবে ঠিক ততটুকু মেমরিই দখল করা যাবে। শুরুতেই তুমি অনেক বড় একটা অ্যারে ডিক্লেয়ার করে রাখলে হবে না। এক্ষেত্রে মেমরি খুব সীমিত। তাই প্রয়োজনের অতিরিক্ত ১ বাইটও খরচ করা যাবে না। Problem টি সলভ করতে হবে Linked List এর মাধ্যমে।

Solution

লিংকড লিস্ট যেহেতু একটা স্ট্রাকচারের লিস্ট। তাই শুরুতেই একটা স্ট্রাকচার বানিয়ে ফেলিঃ

```
struct linked_list
{
    int number;
    struct linked_list *next;
};
```

ডেটা হিসেবে এখানে আছে number. তোমাদের প্রয়োজন অনুসারে এখানে যতগুলো দরকার ডেটা নিতে পারো। next হচ্ছে এই linked\_list টাইপের স্ট্রাকচারের একটা পয়েন্টার ভেরিয়েবল। যে কিনা এই টাইপের একটা স্ট্রাকচারের মেমরি অ্যাড্রেস সংরক্ষণ করতে পারে।

main function এর উপরে, এই linked\_list স্ট্রাকচারের একটা global variable declare করি node নাম দিয়ে এভাবেঃ

```
.  
typedef struct linked_list node;
```

```
.  
typedef কী?
```

typedef এমন একটা keyword যার মাধ্যমে তুমি যে কোন টাইপের নতুন নামকরণ করতে পারবে। উদাহরণ দিলে ব্যাপারটা পরিষ্কার হবে।

```
{  
.  
    typedef char Book[100];  
    Book book1;  
    scanf("%s", book1);  
    printf("%s",book1);  
.  
}
```

char টাইপের একটা অ্যারে ডিক্লেয়ার করা হয়েছে Book[100] লিখে। এর শুরুতে typedef কীওয়ার্ডটা বসানো হয়েছে। এর পরের লাইনে দেখো, Book টাইপের একটা ভেরিয়েবল ডিক্লেয়ার করা হয়েছে। অর্থাৎ নতুন কোন ডেটাটাইপ না, কিন্তু আমাদের বুঝার সুবিধার্থে কোন একটা ভেরিয়েবলকেই আমরা ডেটাটাইপের মত করে ব্যবহার করতে পারি। বা Type define করে দিতে পারি।

ফিরে আসি লিংকড লিস্ট। প্রবলেমটা সলভ করার জন্য আমাদের procedure হচ্ছে, main function এ node এর একটা পয়েন্টার ভেরিয়েবল (head) তৈরি করা। যে কিনা লিস্টের প্রথম আইটেমের মেমরি অ্যাড্রেস সংরক্ষণ করবে। এরপর main function থেকে create ফাংশন কল করা হবে। প্যারামিটার হিসাবে পাঠানো হবে head-কে। এই head এর সাথে লিস্টের পরের আইটেমগুলো একটার পর একটা যুক্ত হতে থাকবে। head তৈরির কাজটা করা যায় এভাবেঃ

.

```
node *head; //node টাইপের ভেরিয়েবলের মেমরি অ্যাড্রেস সংরক্ষণ করবে head
```

```
head = (node *) malloc(sizeof(node)); //node টাইপের ভেরিয়েবলের মেমরি অ্যাড্রেস assign করা হয়েছে
```

.

malloc কী?

Dynamic memory allocation এর জন্য এই ফাংশনটি ব্যবহৃত হয়। আমরা একটা int type এর ভেরিয়েবল ডিক্লেয়ার করতে পারি int a; লিখে। এতে মেমরির যে কোন একটা অ্যাড্রেসে a এর জন্য মেমরি অ্যালোকেট করা হয়। কিন্তু কখনো যদি সরাসরি ভেরিয়েবল ডিক্লেয়ার না করে ভেরিয়েবলের মেমরি ডিক্লেয়ার করার দরকার হয় তখন আমরা malloc ব্যবহার করতে পারি।

```
{
```

```
    int *a;
```

```
    a = (int *) malloc (sizeof(int));
```

```
    printf("Memory address is %d\n",a);
```

```
    scanf("%d", a); //input to address "a". "a" is the memory address. So no need to use & sign
```

```
    printf("%d", *a); //"a" is memory address. but "*a" is the value of address "a"
```

```
}
```



আমাদের লিংকড লিস্টের ক্ষেত্রে প্রতিটা নতুন নতুন node লিস্টের সাথে জুড়ে দেয়ার সময় malloc ব্যবহার করে নতুন নোডের জন্য memory allocate করে হবে। আর মেমরি অ্যাড্রেসটা আগের নোডের next (মেমরি অ্যাড্রেস) variable এ অ্যাসাইন করে দিলেই তৈরি হয়ে যাবে লিংকড লিস্ট।

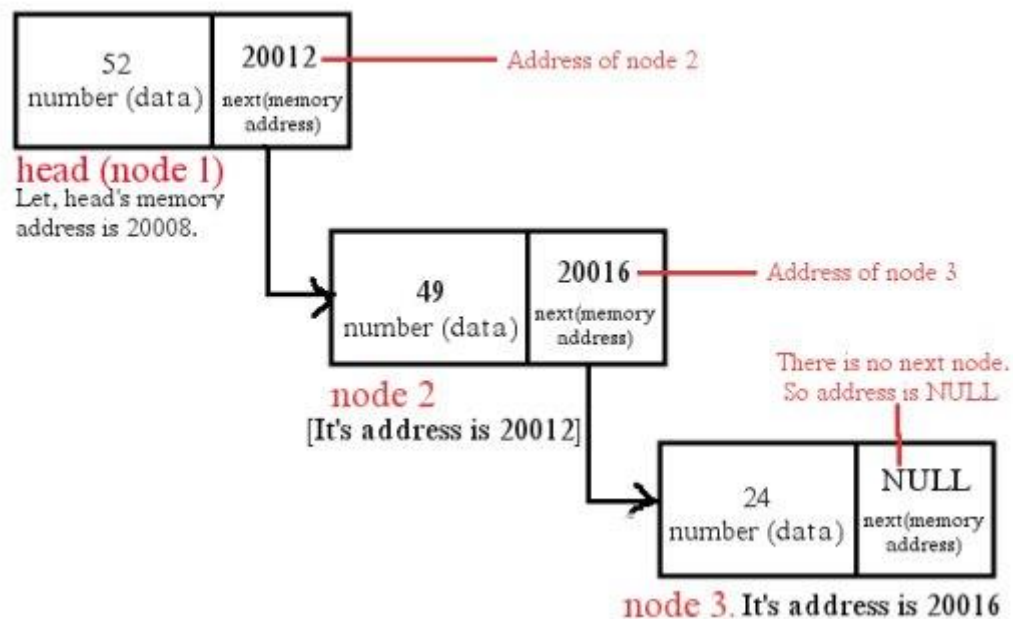
আগে বলে দেয়া procedure অনুযায়ী আমাদের head নোড তৈরি করা হয়ে গেছে। এখন create(head) ফাংশন কল করে এই head এর সাথে লেজ জুড়ে দেয়ার কাজ করতে হবে।

Create a Linked List

```
void create(node *myList)
{
    printf("Input a number. (Enter -99999 at end)\n");
    scanf("%d", &myList->number);
    if(myList->number == -99999)
        myList->next = NULL;
    else
    {
        myList->next = (node *) malloc(sizeof(node));
        create(myList->next);
    }
}
```

এই ফাংশনের প্যারামিটার হিসেবে পাঠানো হয়েছে একটা memory address. আর অ্যাড্রেসটা হচ্ছে node টাইপের একটা স্ট্রাকচারের। লিংকড লিস্টের ক্ষেত্রে প্রায় সব কাজই কিন্তু মেমরি অ্যাড্রেস নিয়ে করতে হবে। কোনো আইটেম add করা, delete করা ইত্যাদি অপারেশনগুলো করার উপায় এই মেমরি অ্যাড্রেস ধরে ধরে।

create() একটি রিকার্সিভ ফাংশন। এর base case হচ্ছে -99999 ইনপুট হওয়া। অর্থাৎ কখনো যদি -99999 ইনপুট করা হয় তাহলে ধরে নেয়া হবে লিস্টটা আর বড় হবে না। -99999 এর আগের সংখ্যাটিই লিস্টের সর্বশেষ আইটেম। base case সত্যি হলে current node এর pointer ভেরিয়েবল next = NULL করে দেয়া হবে। এতে বুঝা যাবে এর পরে আর কোন আইটেম নাই, এটিই সর্বশেষ আইটেম।



### লিংকড লিস্টের উদাহরণ

যদি base case সত্য না হয়, তাহলে `myList->next = (node *) malloc(sizeof(node));` এর মাধ্যমে নতুন একটা node এর জন্য মেমরি দখল করা হলো। এরপর সেই মেমরি অ্যাড্রেসটা দিয়ে আবার `create(myList->next);` কল করা হলো। যতক্ষণ -99999 ইনপুট না হবে ততক্ষণ এই রিকার্সিভ কল চলতেই থাকবে। এরই মাধ্যমে আমাদের লিস্ট তৈরির কাজ শেষ হলো।

Print the linked list

পুরো লিস্টটা প্রিন্ট করতে চাইলে main function থেকে print function কল করতে হবে। প্যারামিটার হিসাবে থাকবে লিস্টের শুরুর node বা head.

```
void print(node *myList)
```

```
{
```

```

printf("%d ", myList->number);
if(myList->next == NULL)
    return;
print(myList->next);
}

```

এটাও একটা recursive function. ফাংশন বডিতে তুকেই current node এর number-টা print করে দিবে। পরের নোডের অ্যাড্রেস রাখা আছে next নামক pointer variable এ। যদি এতে NULL পাওয়া যায় তার মানে হচ্ছে print করার মত এর পরে আর কোনো নোড নাই। তাই return করার মাধ্যমে ফাংশনের কাজ শেষ করা হচ্ছে। অন্যথায় পরের নোডের অ্যাড্রেস দিয়ে আবারো print() কল হচ্ছে। এভাবে পুরো লিস্টটি প্রিন্ট করা হচ্ছে।

Size of linked list

লিস্টে কতগুলো আইটেম আছে সেটা জানার জন্য এই ফাংশনটা কল করা যায়ঃ

```

int countListItem(node *myList)
{
    if(myList->next == NULL)
        return 0;
    return (1 + countListItem(myList->next));
}

```

প্রিন্ট করার মতই। তাই আর ব্যাখ্যায় গেলাম না।

