সি তে ফাংশন

প্রোগ্রামিং এর সবচেয়ে মজার জিনিস হচ্ছে ফাংশন। এর আগে আমরা printf, scanf, toupper, tolower ইত্যাদি ফাংশন ব্যবহার করেছি। এগুলো প্রোগ্রামিং ল্যাঙ্গুয়েজের এর সাথে দিয়ে দেওয়া হয়েছে যেন আমরা সহজেই প্রোগ্রাম লিখতে পারি। এ অধ্যায় আমরা শিখব কিভাবে নিজের প্রয়োজন মত ফাংশন লিখে ফেলা যায়।

ফাংশন হচ্ছে পুনরায় ব্যবহার যোগ্য কোড ব্লক। যা একটি নির্দিষ্ট কাজ করতে পারে। ফাংশন ভালো ভাবে জানলেই প্রোগ্রামিং এর একটা বিশাল অংশ শেখা শেষ হয়ে যায়। লেখা যায় নিজের ইচ্ছে মত কোড।

ফাংশন লেখার নিয়মঃ
একটা ফাংশন নিচের মত করে লেখা হয়ঃ

```
return_type function_Name (parameters){
    function body;
    return value;
}
```

- return_Type হচ্ছে ফাংশনটি কি ধরনের ডেটা রিটার্ন করবে তা। যেমন int, char, float, double ইত্যাদি।
- ফাংশনের তো একটা নাম থাকতে হবে তাই না? যে নাম দিয়ে ফাংশনটিকে কল করে আমরা ব্যবহার করব। Function_Name হচ্ছে ফাংশনের নাম।
- Parameters হচ্ছে ফাংশনকে আমরা কি কি ডেটা দিব। এখানে এক বা একাদিক Parameter আমরা দিতে পারি। কোন কোন ফাংশনে কোন Parameter নাও থাকতে পারে। এটা নির্ভর করে কি ধরনের ফাংশন লিখা হচ্ছে তার উপর। একের অধিক Parameter থাকলে তাদেরকে কমা দিয়ে লিখতে হয়।
- Function Body তে কিছু কোড লিখি আমরা। এটাই ফাংশনের মূল অংশ। ফাংশনটি মূল কাজ এখানে করে।
- কাজ শেষে ফাংশনটি কি রিটার্ন করবে তাই return দিয়ে পাস করা হয়।

আমরা ছোট্ট একটা ফাংশনের কথা চিন্তা করি। যেমন একটা নাম্বারের বর্গ বা square বের করার ফাংশন। যে ফাংশনে প্যারামিটার হিসেবে আমরা একটা নাম্বার দিব। ফাংশনটি আমাদের ঐ নাম্বারটির বর্গ রিটার্ণ করবে। ফাংশনটি আমরা লিখতে পারি এভাবেঃ

```
int square(int num){
    int result = num * num;
    return result;
}
```

একটা ফাংশনের সব গুলো অংশই উপরের প্রগ্রামে রয়েছে। যেমন ফাংশনটির প্রথমে int দেখে আমরা বলতে পারি এটি একটি int ডেটা রিটার্ণ করবে। এরপর হচ্ছে ফাংশনটির নাম। পরে হচ্ছে ফাংশনটির প্যারামিটার। যেখানে ফাংশনটিকে কল করার সময় আমরা একটা ইন্টিজার নাম্বার দিব।

ফাংশনটির বডিতে আমরা একটা ইন্টিজার ভ্যারিয়েবল ডিক্লেয়ার করেছি, যার নাম result। এবং যার মধ্যে ফাংশনটির প্যারামিটারে পাওয়া ইন্টিজার ভ্যারিয়েবলটি নিজের সাথে নিজেকে গুণ দিয়েছি। কারণ আমরা জানি, একটি নাম্বারকে ঐ নাম্বার দিয়ে গুন করলে নাম্বারটির বর্গ বা স্কয়ার পাওয়া যায়। এবং শেষে result ভ্যারিয়েবলটি রিটার্ণ করেছি।

ফাংশনকে কল করাঃ

কোন ফাংশনকে ব্যবহার করার জন্য তাকে কল করতে হয়। কল করার জন্য আমরা শুধু ঐ ফাংশনটির নাম লিখি। এভাবেঃ

function_name(argument)

ফাংশনটি কাজ করার জন্য যে যে ডেটা লাগবে, তা আমাদের পাস করতে হয় argument হিসেবে। যেমন আমরা উপরে square নামে যে ফাংশনটি লিখেছি, যেখানে ইন্টিজার নাম্বার argument হিসেবে পাস করতে হবে। এভাবেঃ

Square(5);

এখানে আমরা যে কোন ইন্টিজার নাম্বার পাস করতে পারব। অন্য কোন ডেটা পাস করলে ফাংশনটি কাজ করবে না। নিচে সম্পুর্ণ একটি প্রোগ্রামঃ

```
#include<stdio.h>

int square(int num){
    int result = num * num;
    return result;
}


int main(void){

  int x = square(5);

  printf("Square is: %d", x);

  return 0;
}
```

প্রথমে আমরা ফাংশনটি লিখেছি। এরপর main ফাংশন লিখেছি। main নিজেও একটা ফাংশন। প্রতিটা প্রোগ্রামে main নামে একটা ফাংশন থাকতে হয়। কম্পাইলার প্রোগ্রামটি রান করার সময় সবার আগে এই main ফাংশনটাকে খুঁজে বের করে। মেইন ফাংশনে গিয়ে দেখে যে int x নামে একটা ভ্যারিয়েবল নেওয়া হয়েছে। এবং এর মান হচ্ছে square(5) নামে একটা ফাংশন। এরপর squire নামক ফাংশনটিকে খুঁজে বের করে।

আমরা যখন মেইন ফাংশন থেকে square ফাংশনকে কল করেছি, তখন আমরা একটা নাম্বার দিয়ে দিয়েছি। যদি এই নাম্বারটা না দিতাম, প্রোগ্রাম রান করত না। মানে আমরা একটা ইন্টিজার ভ্যালু ফাংশনটিকে দিতে হবে। এখানে আমরা শুধু ইন্টিজার নাম্বারই দিতে পারব। কারণ আমরা ফাংশনটি লেখার সময় বলে দিয়েছি, ফাংশনটি প্যারামিটার হিসেবে একটা ইন্টিজার নাম্বার নিবে। অন্য কিছু argument হিসেবে দিলে আমাদের ফাংশনটি পাগলামি শুরু করে দিতে পারে!

এখন square ফাংশনটিকে কল করার সময় যে কোন নাম্বার পাস করে দিলেই সে আমাদের ঐ নাম্বারটির স্কয়ার রিটার্ণ করবে। আর রিটার্ণটি আমরা একটা ভ্যারিয়েবলে রেখেছি। এরপর তা প্রিন্ট করেছি।

এবার ফাংশনটির দিকে একটু তাকাই, আমরা কি ফাংশনটিকে আরেকটু ছোট করে লিখতে পারি? হ্যা, পারি। আমরা ফাংশনটির ভেতর কোন ভ্যারিয়েবল না ডিক্লেয়ার করে নিচের মত করে ফাংশনটিকে লিখতে পারিঃ

```
int square(int num){
    return num * num;
}
```

আবার আমরা printf ফাংশন থেকেও ফাংশনটিকে কল করতে পারি। নিচে পূর্ণাঙ্গ প্রোগ্রামঃ

```
#include<stdio.h>

int square(int num){

    return num * num;
}


int main(void){

  printf("Square is: %d", square(5));

  return 0;
}
```

এটা থেকে আমরা জানলাম, আমরা একটা ফাংশনের ভেতর থেকে আরেকটা ফাংশনকে কল করতে পারি। একটার ভেতর থেকে আরেকটা, আরেকটার ভেতর থেকে আরেকটা, এভাবে যত খুশি তত [যতক্ষণ পর্যন্ত আমরা নিজেরা নিজেদের কোড বুঝতে পারি]।

আপনার মাথায় একটা প্রশ্ন উঁকি দিতে পারে। প্যারামিটার কি, আর আরগুমেন্ট কি। এ দুইটার মধ্যে পার্থক্য কি। ফাংশন কল করার সময় আমরা যে ডেটা পাস করি, তাই হচ্ছে আর্গুমেন্ট। যেমন square(5) এর ক্ষেত্রে 5 হচ্ছে আর্গুমেন্ট। আর ফাংশন লেখার সময় আমরা যে ভ্যারিয়েবল ডিক্লেয়ার করি, তা হচ্ছে প্যারামিটার। যেমন square(int num) এ num হচ্ছে প্যারামিটার।

প্যারামিটার ছাড়া ফাংশনঃ

কিছু কিছু ফাংশন এমন হতে পারে যে যেগুলোতে কোন প্যারামিটার নাও থাকতে পারে। তাই সেগুলোকে কল করার সময় কোন আর্গুমেন্ট পাস করতে হয় না। যেমন pi নামক একটা ফাংশন। যেটাকে কল করলে আমাদের pi এর মান 3.1416 রিটার্ণ করবে। কিন্তু তাকে কল করার সময় কোন আর্গুমেন্ট পাস করতে হবে না।

```
#include<stdio.h>

float pi(){

    return 3.1416;
}


int main(void){

    printf("%f", pi());

    return 0;
}
```

রিটার্ণ টাইপ ছাড়া ফাংশনঃ

কিছু কিছু ফাংশন এমন হতে পারে, যেগুলো কিছু নাও রিটার্ণ করতে পারে। সে গুলো লেখার সময় আমরা রিটার্ণ টাইপের যায়গায় লিখি void। যেমন happy নামক একটা ফাংশন। যার মনে কোন দুঃখ নেই। যখনই তাকে কল করা হয়, তখনই সে বলেঃ I am happy!

```
#include<stdio.h>

void happy(){
    printf("I am Happy!");
}


int main(void){
    happy();
    return 0;
}
```

এতক্ষণ যে ফাংশন গুলো লিখেছি, সেগুলো অনেক সহজ সহজ ছিল। আমরা ফাংশন ব্যবহার করে অনেক বড় বড় প্রোগ্রাম লিখে ফেলতে পারব। যেমন বাস্তব জীবনে আমাদের একটা সংখ্যার ফ্যাক্টোরিয়াল বের করতে হয়তে পারে। Factorial কি তা তো আমরা জানি, তাই না? কোন সংখ্যার ফ্যাক্টোরিয়াল হচ্ছে শূন্য ছাড়া ঐ সংখ্যাটি থেকে ছোট সকল পূর্ণসংখ্যার গুনফল হচ্ছে ফ্যাক্টোরিয়াল। n পূর্ণসংখ্যা হয়, তাহলে n এর ফ্যাক্টোরিয়াল প্রকাশ করা হয় এভাবেঃ n! যেমন 5! এর মান হবে 120

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120.$$

এবার আমরা একটা পূর্ণসংখ্যার Factorial বের করার একটি প্রোগ্রাম লিখিঃ

```
#include <stdio.h>

long int factorial(int n){
    int i;
    long int result =1;
    if(n>1){
        for(i=2; i<=n; i++)
        result = result*i;
    }
    return result ;
}

int main()
{
    int num;
    scanf("%d", &num);
    printf("Factorial of %d is : %d", num,  factorial(num));
    return 0;
}
```

এখানে আমরা factorial একটি ফাংশন লিখছি যা ফ্যাক্টোরিয়াল বের করতে পারে। এই ফাংশনকে আমরা যে কোন একটা নাম্বার আরগুমেন্ট হিসেবে দিলে সে আমাদের ঐ সংখ্যার ফ্যাক্টোরিয়াল রিটার্ণ করবে।

একের অধিক প্যারামিটার সহ ফাংশনঃ

আমরা চাইলে একটা ফাংশনে যত ইচ্ছে তত গুলো প্যারামিটার ব্যবহার করতে পারি। যেমন নিচের প্রোগ্রামটি দেখিঃ

```
#include <stdio.h>

int max(int x, int y){

    return (x > y) ? x : y;
}

int main()
{

    printf("Max is %d", max(8,12));

    return 0;

}
```

এখানে আমরা একটা ফাংশন লিখেছি, যার প্যারামিটার হিসেবে রয়েছে দুইটা ইন্টিজার ভ্যারিয়েবল। এরপর মেইন ফাংশন থেকে ফাংশনটিকে কল করার সময় দুইটা নাম্বার দিয়ে দিয়েছি। ফাংশনটি রিটার্ণ করেছে ঐ দুইটা নাম্বারের মধ্যে বড় নাম্বারটি।

ফাংশন প্রোটোটাইপঃ

আমরা এতক্ষণ যে ফাংশন গুলো লিখছি তা main এর আগে লিখছি। আমরা ইচ্ছে করলে main এর পরেও লিখতে পারি। তবে তার জন্য main ফাংশন এর আগে তাকে ডিক্লেয়ার করতে হবে। যাকে বলে Function Prototype। ফাংশন প্রোটোটাইপ নিচের মত করে লিখতে হয়ঃ

return_type function_name (parameters);

আমরা square নামে যে ফাংশনটি লিখেছি এই অধ্যায়ের শুরুতে, তা যদি আমরা main ফাংশনের পরে লিখি, তাহলে আমদের নিচের মত করে লিখতে হবেঃ

```
#include<stdio.h>

int square  (int num);

int main(void){

  printf("Square is: %d", square(5));

  return 0;
}



int square(int num){

    return num * num;
}
```

আমরা তো বুঝে গিয়েছি ফাংশন প্রোটোটাইপ কি, কিভাবে লিখতে হয়। এবার আরেকটু কমপ্লেক্স প্রোগ্রাম লিখে ফেলি।

১ থেকে বড় যেকোন সংখ্যা তখনই প্রাইম বা মৌলিক সংখ্যা, যখন তা ১ এবং ঐ সংখ্যা ছাড়া আর কোন সংখ্যা দিয়ে বিভাজ্য নয়। যেমন 2, 3, 5, 7, 11, 13, 17, 19 ইত্যাদি। আমরা একটা ফাংশন লিখব, যেটাকে আমরা একটা নাম্বার দিব। আর ফাংশনটি বলবে, এটি প্রাইম নাকি প্রাইম না। সম্পূর্ণ প্রোগ্রামঃ

```
#include <stdio.h>

void check_prime(int num);

int main()
{
    int a;
    scanf("%d", &a);

    check_prime(a);

```

```
        return 0;
}

void check_prime(int num){
        int i,count=0;
        for(i=2;i<=num/2;i++){
            if(num%i==0){
                count++;
                break;
            }
        }
        if(count==0 && num!= 1)
        printf("%d is a prime number",num);
        else
        printf("%d is not a prime number",num);
}
```

এখানে আমরা প্রাইম নাম্বার চেক করার একটা প্রোগ্রাম লিখছি।

প্রোগ্রামে শুরুতেই আমরা বলে দিয়েছি যে check_prime রয়েছে আমাদের এই প্রোগ্রামে যা কোন কিছু রিটার্ণ করবে না।

আমরা ইউজার থেকে একটা নাম্বার নিয়েছি। ঐ নাম্বারটি check_prime এ আর্গুমেন্ট হিসেবে পাস করেছি। আমাদের ফাংশনটি এরপর নাম্বারটি প্রাইম কি প্রাইম না, তা জানিয়েছে।

রিকার্শন বা রিকার্শিভ ফাংশনঃ

কোন কিছু যদি নিজেকে নিজে পুনরায় ডাকে, করে তাই হচ্ছে **রিকারশন/ Recursion।**

একটা খালি মাঠে গিয়ে নিজেকে নিজে ডাকলে বা একটা বিশাল হল রুমে নিজের নাম ধরে ডাকা ডাকি করলে রিকার্শন অনুভব করা যাবে।

গুগলে Recursion লিখে সার্চ করে দেখো। বার বার লেখা উঠবে Did you mean: Recursion। বানান ঠিক থাকার পর ও এটা দেখাবে। গুগল মজা করে একটা রিকারশন বসিয়ে দিয়েছে Recursion সার্চ টার্ম এর উপর।

Recursive Algorithm হচ্ছে যে অ্যালগরিদম নিজেকে নিজে কল করে, তা। কম্পিউটার প্রোগ্রামিং এ রিকারসিভ অ্যালগরিদম ব্যবহার করে কোন প্রোগ্রামে রিকারশন ব্যবহার করা হয়। বিভিন্ন প্রোগ্রামিং ল্যাঙ্গুয়েজে একটি ফাংশন নিজেকে কল করার মাধ্যমে রিকারশন এর প্রয়োগ করা হয়। সি প্রোগ্রামিং এ কিভাবে রিকার্শন বা রিকার্শিভ ফাংশন লিখা যায়, তাই দেখব।

রিকার্সিভ ফাংশন কি তা তো এখন সহজেই বলা যাচ্ছে তাই না? যে ফাংশন নিজেকে নিজে কল করে, তাই হচ্ছে রিকার্সিভ ফাংশন।

রিকারশনের সুবিধে হচ্ছে কোড সহজ করে, অনেক গুলো কোড লেখার পরিবর্তে কয়েক লাইনের কোড দিয়ে একটা সমস্যা সমাধান করা যায়। নিচের সুডোকোডটি দেখিঃ

```
void f() {
    f() …
}
```

এটা একটা রিকার্সিভ ফাংশন, কারণ ফাংশনটি নিজেকে নিজে কল করেছে। এটাকে বলে সরাসরি কল। আবার ফাংশন সরাসরি কল না করেও এমন একটা ফাংশনকে কল করতে পারে, যে ফাংশনটি প্রথম ফাংশনকে কল করে। নিচের উদারহণটি দেখিঃ

```
void f() {
    g() …
}
void g() {
    f() …
}
```

f ফাংশনটি g ফাংশনকে কল করেছে। আবার g ফাংশন f ফাংশনকে কল করেছে। এটাও রিকার্সিভ ফাংশন।

আমরা একটা রিকার্সিভ ফাংশন লিখি, যেটা ইউজার থেকে একটা নাম্বার নিবে, তারপর ঐ সংখ্যা থেকে এর পর ১ থেকে ঐ সংখ্যা পর্যন্ত সকল সংখ্যা প্রিন্ট করবে। এটা সিম্পল একটা প্রোগ্রাম। তবে আমরা এ প্রোগ্রামটি নতুন ভাবে লিখব। রিকারশন ব্যবহার করে। প্রোগ্রামটি যেহেতু সিম্পল, তাহলে আমরা একটু ভালো করে দেখলেই বুঝতে পারব। আর প্রোগ্রামটি কিভাবে কাজ

করে, তা বুঝতে পারলেই আমরা রিকার্শন বুঝে ফেলব। এরপর আমরা কমপ্লেক্স সব প্রোগ্রাম রিকারশন ব্যবহার করে সহজেই লিখে ফেলতে পারব। ফাংশনটি **Pseudo code [সুডো কোড]** এ আগে লিখি, এরপর সি প্রোগ্রামিং এ ইমপ্লিমেন্টেশন দেখব।

সুডো মানে মিথ্যে। সুডো Pseudo code হচ্ছে একটা প্রোগ্রাম বা একটা অ্যালগরিদমের ধাপ গুলো সাধারণ মানুষের ব্যবহার উপযোগি করে লেখা কিছু কোড। এগুলো কোন প্রোগ্রামিং ল্যাঙ্গুয়েজ ব্যবহার করে লেখা হয় না। এমন ভাবে লেখা হয় যেন মানুষ বুঝতে পারে। নিচে printInt নামে একটি ফাংশনের সুডো কোড দেওয়া হলো, যা ১ থেকে ঐ সংখ্যা পর্যন্ত সকল সংখ্যা প্রিন্ট করবে।

```
printInt( k ) {
    if (k == 0) {
        return 0;
    }
    print(k ); // calling itself
    printInt( k - 1 );
}
```

ফাংশনটি প্যারামিটার হিসেবে একটা ইন্টিজার নিবে। এরপর চেক করবে সংখ্যাটা কি ০? যদি শুন্য হয়, ঐখানেই ফাংশনের কাজ শেষ হবে। যদি ০ না হয় তাহলে ইন্টিজারটি প্রিন্ট করবে। এবং ঐ ইন্টিজার সংখ্যাটি থেকে ১ বিয়োগ করে আবার printInt কে কল করবে। মানে নিজেকে নিজে কল করবে।

এখন আমরা যদি printInt ফাংশনে 2 পাস করি, তাহলে ফাংশনটির তিনটে কপি তৈরি হবে। একটা হচ্ছে k এর মান 2 এর জন্য, একটা হচ্ছে k এর মান 1 এর জন্য। আর একটা হচ্ছে k এর মান ০ এর জন্য।

যখন k এর মান 2
printInt(int k) {
if (k == 0) {
return 0;
}
print(k);
printInt(k – 1);
}প্রিন্ট করবে 2
	যখন k এর মান 1
	printInt(int k) {
	if (k == 0) {
	return 0;
	}
	print(k);
	printInt(k – 1);
	}
	প্রিন্ট করবে 1
		যখন k এর মান 0
		printInt(int k) {
		if (k == 0) {
		return 0;
		}
		print(k);
		printInt(k – 1);
		}
		k এর মান ০ হওয়ায় কিছুই প্রিন্ট করবে না।

এভাবে এখন আমরা যদি printInt ফাংশনে 5 পাস করি, তাহলে ফাংশনটির পাঁচটি কপি তৈরি হবে। অর্থাৎ যে সংখ্যা পাস করব, তত সংখ্যক বার ফাংশনটির কপি তৈরি হবে। আর এভাবেই রিকারশন কাজ করে।

এবার সি প্রোগ্রামিং এ ইমপ্লিমেন্টেশন দেখিঃ

```
#include <stdio.h>

void printInt( int k ) {
    if (k == 0) {
        return 0;
    }
    printf( "%d \n", k );
    printInt( k - 1 );

}


int main(){
    int i;
    printf("Enter a number: ");
    scanf("%d", &i );
    printInt(i);

    return 0;
}
```

প্রোগ্রামটিতে প্রথমে আমরা আমাদের printInt ফাংশনটি লিখেছি। এর পর মেইন ফাংশনের ভেতর একটা ইন্টিজার ডিক্লেয়ার করেছি। তা ইউজার থেকে ইনপুট নিয়েছি। এরপর printInt ফাংশনে ইন্ট্রিজারটি পাস করেছি। আর printInt হচ্ছে রিকার্সিভ ফাংশন। যে নিজেকে নিজে কল করে আমাদের কাজ করে দিচ্ছে।

আরেকটা সিম্পল প্রোগ্রাম রিকারশন ব্যবহার করে লেখার চেষ্টা করি। যেমন একটা সংখ্যা ইউজার থেকে ইনপুট নিবে, তারপর ১ থেকে ঐ সংখ্যা পর্যন্ত সকল সংখ্যার যোগফল প্রিন্ট করবে। যখন ইনপুট হিসেবে 4 থাকবে তখন সাধারণ একটা প্রোগ্রাম যোগফল নির্নয় করবে নিচের মত করেঃ

sum(4) = 1+2+3+4

যখন ইনপুট হিসেবে থাকবে 5 তখনঃ

sum(5) = 1+2+3+4+5

যখন ইনপুট হিসেবে থাকবে 6 তখনঃ

sum(6) = 1+2+3+4+5+6

যার মানে হচ্ছেঃ

sum(6) = sum(5) + 6 [sum(5)=(1+2+3+4+5)]
sum(5) = sum(4) + 5 [sum(4)=(1+2+3+4)]
sum(4) = sum(3) + 4 [sum(3)=(1+2+3)]
sum(3) = sum(2) + 3 [sum(2)=(1+2)]
sum(2) = sum(1) + 2 [sum(1)=(1)]
sum(1) = sum(0) + 1 [sum(0)=(0)]
sum(0) = 0

উপরের স্টেপ গুলো ৬টা সংখ্যার জন্য। এখন আমরা n তম সংখ্যার যোগফলের জন্য সহজ একটা ইকুয়েশন লিখে ফেলতে পারি, sum(n) = sum(n-1) + n.

যখন ০ হবে, তখন প্রিন্ট করবে ০, আর যখন n হবে, তখন প্রিন্ট করবে sum(n) = sum(n-1) + n.

আর এটা সুডো কোডে লিখলেঃ

```
sum( int n ) {
    if( n == 0 ) return 0;
    else return n + sum( n-1 );
}
```

সি পোগ্রামে এর প্রয়োগ করলেঃ

```
#include <stdio.h>

int sumFunc( int n ) {
    if( n == 0 )
        return 0;
    else
    return n + sumFunc( n-1 );
}


int main(){
    int n,sum;
    printf("Enter the value of n: ");
    scanf("%d",&n);
    sum = sumFunc(n);
    printf("Sum of n numbers: %d",sum);


    return 0;
}
```

রিকার্শনের আরেকটা উদারহণ দেখি, এবার দেখব রিকার্শন ব্যবহার করে Factorial বের করার উপায়। এই অধ্যায়ের শুরুতে আমরা ফ্যাক্টোরিয়াল সম্পর্কে জেনেছি।

১ এর ফ্যাক্টোরিয়াল ১।

২ এর ফ্যাক্টোরিয়াল ১*২ = ২

৩ এর ফ্যাক্টোরিয়াল হচ্ছে ১*২*৩ = ৬।

৪ এর ফ্যাক্টোরিয়াল হচ্ছে ১*২*৩*৪ = ২৪।

৫ এর ফ্যাক্টোরিয়ালঃ ১*২*৩*৪*৫ = ১২০।

n তম সংখ্যার জন্য আমরা এভাবে লিখতে পারিঃ factorial(n) = (n * factorial(n-1)); আর যেহেতু factorial(0) = 1, আমরা factorial এর জন্য একটা রিকার্সিভ ফাংশনের সুডো কোড লিখে ফেলতে পারিঃ

```
factorial(n) {
    if (n == 0)
    return 1;
    else
    return (n * factorial(n-1));
}
```

এখানে n == 0 হলে 1 রিটার্ণ করার কারণ হচ্ছে factorial(0) = 1।

আমরা যদি factorial(3) এর মান বের করি, ফাংশনটি এভাবে কাজ করবেঃ

factorial(3) =
3 * factorial(2)
3 * 2 * factorial(1)
3 * 2 * 1 * factorial(0)
3 * 2 * 1 * 1
= 6

প্রোগ্রামিং এ আমরা নিচের মত করে প্রয়োগ করতে পারিঃ

```c
#include <stdio.h>
int factorial(int n) {
    if (n == 0)
    return 1;
    else
    return (n * factorial(n-1));
}

int main(){
    int n,result;
    printf("Enter the value of n: ");
    scanf("%d",&n);
    result = factorial(n);
    printf("Factorial is : %d",result);

    return 0;
}
```