






সূচিপত্র

পরিচিতি	0
প্রাথমিক ধারণা	1
কম্পিউটার প্রোগ্রামিং কি?	1.1
সি এর জন্মকথা	1.2
প্রয়োজনীয় সফটওয়্যার	1.3
প্রথম প্রোগ্রাম	1.4
কিছু সাধারণ সি প্রোগ্রাম	1.5
কন্ট্রোল স্টেটমেন্ট	2
if - else স্টেটমেন্ট	2.1
switch case স্টেটমেন্ট	2.2
for লুপ	2.3
while লুপ	2.4
do - while লুপ	2.5
ফাংশন	3
ইউজার ডিফাইনড ফাংশন	3.1
অ্যারে	4
ওয়ান ডাইমেনশনাল অ্যারে	4.1
মাল্টি ডাইমেনশনাল অ্যারে	4.2
পয়েন্টার	5
স্ট্রাকচার	6
ফাইল অপারেশন	7

কোর্স এর মূল পাতা | [HowToCode মূল সাইট](#) | [সবার জন্য প্রোগ্রামিং ব্লগ](#) | [পিডিএফ ডাউনলোড](#)

বাংলায় সি প্রোগ্রামিং শিক্ষা

[gitter](#) [join chat](#)

47		Jakir Hossain https://github.com/jakirseu	19		Md. Al-Amin alamin.opu10@gmail.com
15		Md Khaled Ben Islam https://github.com/mdkhaledben			
12		Rashedul Kabir https://github.com/ekush	6		Nuhil Mehdy nuhil@nuhil.net

সংক্ষেপ

সি (C) একটি বহুল ব্যবহৃত কম্পিউটার প্রোগ্রামিং ল্যাংগুয়েজ। বেশীরভাগ শিক্ষা প্রতিষ্ঠানে আন্ডারগ্রাজুয়েট (Undergraduate) প্রোগ্রামে আবশ্যিক বিষয় (core subject) হিসাবে এটি পড়ানো হয়ে থাকে। শিক্ষা প্রতিষ্ঠানগুলোতে সি পড়ানো হয় মূলত শিক্ষার্থীদের প্রোগ্রামিং এর ভিত রচনার জন্য। আর সফটওয়্যার শিল্পে সি ব্যবহৃত হয় সাধারণত পারফরমেন্স ক্রিটিক্যাল অ্যাপ্লিকেশন বানানোর জন্য।

সি এত বেশী জনপ্রিয় এর বিশেষ কিছু বৈশিষ্ট্যের জন্য। এটি দিয়ে একাধারে যেমন আমাদের ব্যবহৃত ভাষা ইংরেজী এর মত (High Level Language) করে প্রোগ্রাম লেখা যায়, তেমনি দরকার হলে মেশিনের ব্যবহৃত ভাষা এর মত (Assembly Language) করেও প্রোগ্রাম লেখা যায়। এজন্য আমরা দৈনন্দিন জীবনে ব্যবহৃত সফটওয়্যার বানাতে যেমন সি এর ব্যবহার দেখতে পাই তেমনি সিস্টেম সফটওয়্যার বানাতেও সি এর ব্যবহার চোখে লাগার মত।

সি তে লেখা প্রোগ্রাম গুলো সাধারণত অন্যান্য ভাষায় লেখা প্রোগ্রাম এর চেয়ে ইফিসিয়েন্ট (efficient) ও ফাস্ট (fast) হয়। এছাড়া সি তে লেখা প্রোগ্রাম/কোড সামান্য পরিবর্তন করে (কখনও না করেও) বিভিন্ন অপারেটিং সিস্টেম তে চালানো যায়। যেমন- আমি হয়ত উইন্ডোজ অপারেটিং সিস্টেম তে চালানোর উপযোগী করে প্রোগ্রাম লিখলাম, সেটা আমি চাইলে লিনাক্স তেও চালাতে পারব। এরকম আরো বহুবিধ সুবিধার কারণেই প্রোগ্রামাররা এখনো সি কে ভুলে যেতে পারে নাই, যেগুলো আমরা ধীরে ধীরে দেখতে পারব।

আশা করছি সি এর প্রাথমিক ধারণা থেকে শুরু করে অ্যাডভান্সড, ব্যবহারিক প্রয়োগ এবং অন্যান্য বিষয় গুলো নিয়েও এই কোর্সে আলোচনা করা সম্ভব হবে।

এই বইটি কাদের জন্যে:

কম্পিউটার প্রোগ্রামিং নিয়ে যাদের প্রবল আগ্রহ আছে এবং নিজে নিজে শিখতে আগ্রহী - এই বইটি মূলত তাদের জন্য। তবে ধরে নেওয়া হচ্ছে যে, শিক্ষার্থী কম্পিউটার এর সাথে ভালভাবে পরিচিত এবং ইন্টারনেট থেকে কোন কিছু খুঁজে নিতে স্বাচ্ছন্দ্য বোধ করে।

ওপেন সোর্স

এই বইটি মূলত স্বেচ্ছাশ্রমে লেখা এবং বইটি সম্পূর্ণ ওপেন সোর্স। এখানে তাই আপনিও অবদান রাখতে পারেন লেখক হিসেবে। আপনার কন্ট্রিবিউশান গৃহীত হলে অবদানকারীদের তালিকায় আপনার নাম যোগ করে দেওয়া হবে।

এটি মূলত একটি [গিটহাব রিপোজিটোরি](#) যেখানে এই বইয়ের আর্টিকেল গুলো মার্কডাউন ফরম্যাটে লেখা হচ্ছে। রিপোজিটোরিটি ফর্ক করে পুল রিকুয়েস্ট পাঠানোর মাধ্যমে আপনারাও অবদান রাখতে পারেন। তবে আপনি চাইলে কোন পরিবর্তন, পরিবর্ধন কিংবা সংশোধনের জন্য সরাসরি মূল লেখকের সাথেও যোগাযোগ করতে পারেন [mdkhaledben \[at\] gmail.com](mailto:mdkhaledben[at]gmail.com) - এই ঠিকানায়।



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).

প্রাথমিক ধারণা

যারা কম্পিউটার এর সাথে পরিচিত কিন্তু প্রোগ্রামিং বা সি ল্যাংগুয়েজ সম্পর্কে খুব একটা ধারণা নেই, এই চ্যাপ্টার টি মূলত তাদের জন্য। এখানে কম্পিউটার প্রোগ্রামিং এর পাশাপাশি সি ল্যাংগুয়েজ এবং সি ল্যাংগুয়েজ তে প্রোগ্রামিং করতে গেলে যেসব সফটওয়্যার লাগে সে সম্পর্কে ধারণা দেয়া হয়েছে।

সি এর সম্বন্ধে ।

- সি উদ্ভাবন করা হয় UNIX অপারেটিং সিস্টেম লিখার জন্য।
- সি হচ্ছে বি ল্যাংগুয়েজের (B Language) উত্তরাধিকারী, বি উদ্ভাবিত হয় ১৯৭০ সালে।
- সি ১৯৮৮ সালে American National Standard Institute (ANSI) দ্বারা ফরমালাইজ করা হয়।
- বর্তমানে জনপ্রিয় লিনাক্স ওস এবং আরডিবিএমএস মাইএসকিউএল (RDBMS MySQL) সি তে লিখা।

কেন সি ব্যবহার করব ।

সি প্রথমদিকে সিস্টেম ডেভেলপমেন্ট এর জন্য ব্যবহার করা হত। অন্যভাবে বলতে গেলে সি দিয়ে অপারেটিং সিস্টেম বানানোর জন্য প্রোগ্রাম লিখা হত। সিস্টেম ডেভেলপমেন্টের জন্য সি ব্যবহার করা হয় কারন এর দ্বারা লিখা প্রোগ্রাম, অ্যাসেম্বলি প্রোগ্রামের মত দ্রুত কাজ করে। সি দিয়ে লিখা কিছু সফটওয়্যারের উদাহরন :

- অপারেটিং সিস্টেম ।
- বিভিন্ন প্রোগ্রামিং ল্যাংগুয়েজের এর কম্পাইলার ।
- অ্যাসেম্বলার ।
- টেক্সট ইডিটর ।
- নেটওয়ার্ক ড্রাইভার ।
- ডাটাবেজে ।
- ল্যাংগুয়েজ ইন্টারপ্রিটার ।

মজার ব্যাপর হল, সি এর কম্পাইলার ও সি তে লিখা।

সি প্রোগ্রাম

একটা সি প্রোগ্রাম ৩ লাইন থেকে কয়েক হাজার লাইন হতে পারে। সি তে প্রোগ্রাম লিখা ফাইলের এক্সটেনসন হবে .c। আপনি সি প্রোগ্রাম লিখার জন্য নোটপ্যাড, জি ইডিট, ভিম ইত্যাদি টেক্সট ইডিটর ব্যবহার করতে পারেন। আমরা পরবর্তিতে এ সম্বন্ধে আরো বিস্তারিত আলোচনা করব।

কম্পিউটার প্রোগ্রামিং কি?

আমাদেরকে প্রতিদিনই কিছু না কিছু কাজ করতে হয়। কিছু কাজ আমরা নিজেরা করি, আবার কিছু কাজ অন্যদেরকে দিয়ে করিয়ে নেই। অন্যদের কে দিয়ে কোন কাজ করানোর সময় আমরা তাকে হয়ত মুখে বলে দেই যে কি কি করতে হবে এবং কিভাবে করতে হবে অথবা লিখে দেই। এজন্য আমরা সাধারণত এমন ভাষা ব্যবহার করি যেটা আমরাও বুঝতে পারি, আবার যাকে বলব সেও বুঝতে পারে। সেটা হতে পারে বাংলা, ইংরেজী কিংবা অন্য কোন ভাষা।

একইরকম ভাবে আমরা যখন কম্পিউটার কে দিয়ে কোন কাজ করিয়ে নিতে যাব তখন কম্পিউটার কেও জানাতে হবে কি কাজ সে করবে এবং কিভাবে করবে। সেটা হতে পারে দুটি নাম্বার যোগ করার কাজ কিংবা একটি গান বাজানোর কাজ। আমরা সাধারণত কম্পিউটারকে লিখে জানাই যে কি কি কাজ করতে হবে, কিভাবে করতে হবে।

কম্পিউটারকে এভাবে লিখে জানানোই কম্পিউটার প্রোগ্রামিং, লেখার জন্য যে ভাষা গুলো ব্যবহার করা হয় সেগুলো কম্পিউটার এর ভাষা বা প্রোগ্রামিং ল্যাংগুয়েজ এবং যা লেখা হয় সেগুলো প্রোগ্রাম বা কম্পিউটার প্রোগ্রাম নামে পরিচিত। অ্যাসেম্বলি, সি, জাভা এরকমই কিছু প্রোগ্রামিং ল্যাংগুয়েজ।

এবার আমরা দেখব কম্পিউটার কে দিয়ে দুটি নাম্বার যোগ করিয়ে নিতে গেলে বিভিন্ন ভাষায় কি কি লিখতে হয়। এই মুহূর্তে এই প্রোগ্রাম গুলো না বুঝলেও চলবে। এগুলো দেয়ার উদ্দেশ্য হল কম্পিউটার এর ভাষা কেমন হয় সে সম্পর্কে ধারণা পাওয়া।

অ্যাসেম্বলি (Assembly) তে দুটি পূর্ণ সংখ্যা যোগ করা-

```
.MODEL SMALL
.STACK 100H

.DATA
number_1 DW 2
number_2 DW 3
result DW ?
message DB 'Summation of two number is $'

.CODE
MAIN PROC

    MOV AX, @DATA
    MOV DS, AX

    MOV AX, number_1
    ADD AX, number_2
    MOV result, AX

    LEA DX, message
    MOV AH, 9
    INT 21H

    MOV AH, 2
    MOV DL, result
    INT 21H

    MOV AX, 4C00H
    INT 21H

MAIN ENDP
END MAIN
```

সি (C) তে দুটি পূর্ণ সংখ্যা যোগ করা-

```
#include<stdio.h>

int main(){

    int number_1, number_2, result;

    number_1 = 2;
    number_2 = 3;
    result = number_1 + number_2;

    printf("Summation of two number is %d", result);

    return 0;
}
```

জাভা (Java) তে দুটি পূর্ণ সংখ্যা যোগ করা-

```
import java.lang.*;

class Adder
{
    public static void main(String args[])
    {
        int number_1, number_2, result;

        number_1 = 2;
        number_2 = 3;
        result = number_1 + number_2;

        System.out.printf("Summation of two number is %d", result);
    }
}
```


যেভাবে সি (C) প্রোগ্রামিং ল্যাঙ্গুয়েজ এর জন্ম হয়

সেই ১৯০০ শতকের দিকে চার্লস ব্যাবেজের হাত ধরে প্রথম কম্পিউটার এর সূচনা। এরপর থেকে প্রতিদিন একটু একটু করে উন্নত হয়ে আমরা পেয়েছি আজকের আধুনিক কম্পিউটার। এই কম্পিউটার আসলে একটা যন্ত্র বা জড় বস্তু যেটা ০ আর ১ ছাড়া কিছুই বুঝে না। অর্থাৎ কম্পিউটার শুধুমাত্র বিদ্যুৎ আছে অথবা নাই, এই দুইটা অবস্থা বুঝতে পারে। যে শুধুমাত্র দুইটা শব্দ বা অবস্থা বুঝতে পারে তার সাথে ইন্টারেক্ট করা বা তাকে দিয়ে কাজ করানো সত্যিই কঠিন।

এই জন্য সবাই সেই ১৯০০ শতক থেকেই চেষ্টা করে যাচ্ছে এমন একটা ভাষা তৈরি করার যেটা ব্যবহার করে আমরা সহজেই কম্পিউটার এর সাথে ইন্টারেক্ট করতে পারি। এই চেষ্টারই ফলস্বরূপ আজকের এত কম্পিউটার প্রোগ্রামিং ল্যাঙ্গুয়েজ।

১৯৬৬ সালের দিকে ক্যামব্রিজ বিশ্ববিদ্যালয়ের মার্টিন রিচার্ড ডিজাইন করেন বিসিপিএল (BCPL) নামের প্রোগ্রামিং ল্যাঙ্গুয়েজ। এরপর ১৯৬৯ সালে আমেরিকার বেল ল্যাবরেটরিতে কেন থমসন ও ডেনিশ রিচি মিলিত ভাবে ডিজাইন করেন বি (B) নামের আর একটি প্রোগ্রামিং ল্যাঙ্গুয়েজ। এরপর আসে সেই মাহেন্দ্রক্ষণ যখন সি নামের একটি কম্পিউটার প্রোগ্রামিং ল্যাঙ্গুয়েজ এর জন্ম হয়। ১৯৬৯ থেকে ১৯৬৩ সাল এর মাঝামাঝি। আবারও সেই বেল ল্যাবরেটরিতে ডেনিশ রিচি সি ল্যাঙ্গুয়েজ এর সূচনা করেন। তখন এটি মূলত ইউনিক্স কে টার্গেট করে ডিজাইন করা হয়েছিল এবং প্রায় ১৯৭৮ সাল পর্যন্ত এর ব্যবহার বেল ল্যাবরেটরি এর মধ্যেই সীমিত ছিল। ১৯৭৮ সালে কারনিহান (Kernighan) এবং রিচি (Ritchie) "The C Programming Language" নামে সি ল্যাঙ্গুয়েজ এর বিস্তারিত বিবরণ দিয়ে একটা প্রকাশনা বের করেন, যা "K & R C" নামেও পরিচিত। এর পরেই মূলত বেল ল্যাবরেটরি এর বাহিরে সি এর ব্যবহার শুরু হয়।

কম্পিউটার প্রফেশনালরা সেই সময় সি ব্যাপক ভাবে ব্যবহার করে শুরু করে। নির্দিষ্ট ও আন্তর্জাতিক কোন স্ট্যান্ডার্ড না থাকার কারনে অনেকেই সি ল্যাঙ্গুয়েজ কে বিভিন্নভাবে ব্যবহার করতে থাকেন। অনেক টা আমাদের বাংলা ভাষা ব্যবহারের মত। বিভিন্ন এলাকায় বিভিন্ন রকম। এজন্য ১৯৮০ সালের মাঝামাঝি আমেরিকান ন্যাশনাল স্ট্যান্ডার্ড ইন্সটিটিউট (ANSI X3J11 committee) সি ল্যাঙ্গুয়েজ এর জন্য একটা স্ট্যান্ডার্ড ডেফিনিশন নির্ধারণ করে দেয় যা ANSI C নামেই বেশি পরিচিত। এরপর ১৯৯৯ সালের দিকে ইন্টারন্যাশনাল অর্গানাইজেশন অব স্ট্যান্ডার্ডাইজেশন সি এর নতুন ডেফিনিশন বা স্ট্যান্ডার্ড (ISO/IEC 9899:1999) তৈরি করে দেয় যা C99 নামে পরিচিত। এরপর প্রায় ১১ বছর পর সি এ জন্য ISO নতুন স্ট্যান্ডার্ড ISO/IEC 9899:2011 ডিফাইন করে দেয় যা C11 নামেও পরিচিত।

আমরা চেষ্টা করব সি এর সর্বশেষ স্ট্যান্ডার্ড এর উপর বেস করে বইটা লেখা।

তথ্যসূত্রঃ <http://cm.bell-labs.com/who/dmr/chist.html>

প্রয়োজনীয় সফটওয়্যার

প্রোগ্রামিং করতে গেলে কি ধরনের সফটওয়্যার লাগে

আমরা কম্পিউটার ব্যবহার করে যাই করি না কেন তা কোন না কোন সফটওয়্যার ব্যবহার করেই করি। যেমন গান শুনতে গেলে কোন না কোন মিডিয়া প্লেয়ার ব্যবহার করি, তা হতে পারে উইন্ডোজ মিডিয়া প্লেয়ার কিংবা কে এম প্লেয়ার। আবার কোন কিছু লেখা লেখির কাজে ব্যবহার করি মাইক্রোসফট ওয়ার্ড কিংবা এ ধরনেরই কোন না কোন সফটওয়্যার।

একইভাবে প্রোগ্রামিং করার জন্যও বিভিন্ন ধরনের সফটওয়্যার ব্যবহৃত হয়।

ক) প্রোগ্রাম লেখার জন্য ব্যবহৃত সফটওয়্যার

খ) লিখিত প্রোগ্রাম চালানোর উপযোগী করার জন্য ব্যবহৃত সফটওয়্যার

ক) প্রোগ্রাম লেখার জন্য ব্যবহৃত সফটওয়্যারঃ সি তে প্রোগ্রাম লেখার জন্য বিশেষ কোন সফটওয়্যার এর দরকার নাই। যে কোন এডিটর সফটওয়্যার (যেমন- নোটপ্যাড, নোটপ্যাড++, জিইডিটি) ব্যবহার করেই আমরা কোড লিখতে পারি। তবে সহজে কোডিং করা অর্থাৎ কোড লেখার জন্য আরেক ধরনের সফটওয়্যার ব্যবহার করতে পারি যেগুলো সাধারণত আইডিই (IDE) বা ইন্টিগ্রেটেড ডেভলপমেন্ট এনভায়রনমেন্ট নামে পরিচিত। যেমন- কোডব্লোকস, ভিজুয়াল স্টুডিও, টার্বো সি আইডিই।

খ) লিখিত প্রোগ্রাম চালানোর উপযোগী করার জন্য ব্যবহৃত সফটওয়্যারঃ প্রোগ্রামিং ল্যান্সুয়েজ ব্যবহার করে লেখা কোন প্রোগ্রাম চালানোর উপযোগী করার জন্য সাধারণত দুই ধরনের সফটওয়্যার ব্যবহার করা হয়-

১। কম্পাইলার (যেমন - জিএনইউ সি কম্পাইলার বা জিসিসি কম্পাইলার, মাইক্রোসফটের ভিজুয়াল সি কম্পাইলার এবং ইন্টারপ্রেটিভ সি কম্পাইলার)

২। ইন্টারপ্রেটার পিকক (picoc) সি ইন্টারপ্রেটার, জাভাস্ক্রিপ্ট তে ব্যবহৃত ইন্টারপ্রেটার)

কম্পাইলার টাইপের সফটওয়্যার গুলো সাধারণত পুরো একটা প্রোগ্রাম কে ইনপুট হিসাবে নেয় সেগুলোতে কোন ইরর বা ত্রুটি আছে কিনা দেখে, না থাকলে প্রোগ্রাম টাকে একটা ইন্টারমেডিয়েট অবজেক্ট কোডে রূপান্তরিত করে। পরবর্তীতে সেই অবজেক্ট কোড টাকে এক্সিকিউটেবল কোডে রূপান্তরিত করা হয়, যা রান করলে কোন একটা কাজ সম্পূর্ণ হয়।

অন্যদিকে ইন্টারপ্রেটার টাইপের সফটওয়্যার গুলো সাধারণত প্রোগ্রাম এর এক একটা লাইন কে ইনপুট হিসাবে নেয়, তাতে কোন ভুল বা ত্রুটি আছে কিনা দেখে, আর না থাকলে তা এক্সিকিউট করে।

সি প্রোগ্রামিং এর জন্য কি সফটওয়্যার ব্যবহার করব

সি মূলত একটা কম্পাইল্ড ল্যান্গুয়েজ অর্থাৎ সি তে প্রোগ্রামিং করতে হলে কোন না কোন কম্পাইলার ব্যবহার করতে হবে। আমরা এক্ষেত্রে ব্যবহার করব জিসিসি কম্পাইলার। যেটি আমরা চাইলে ইন্টারনেট থেকে ফ্রি ডাউনলোড করতে পারব নিচের ওয়েবসাইট থেকে। যদিও কোডব্লকস একটা আইডিই, আমরা চাইলে এর সাথেই জিসিসি কম্পাইলারও প্যাকেজ হিসাবে ডাউনলোড করতে পারব। আমরা যারা নতুন তাদের জন্য কোডব্লকস এবং জিসিসি কম্পাইলার প্যাকেজ আকারে ডাউনলোড করাই সুবিধাজনক।

<http://www.codeblocks.org/downloads/binaries>

উপরের ওয়েবসাইট তে যাই। এখানে উইন্ডোজ (Windows), লিনাক্স (Linux), ম্যাক (Mac OS X) এর জন্য বাইনারী ফাইল দেয়া আছে যেগুলো ডাউনলোড করে সরাসরি ইন্সটল করা যাবে। যারা উইন্ডোজ ব্যবহার করি তারা তিনটা ফাইল ডাউনলোড করার অপশন দেখতে পাব। i) codeblocks-13.12-setup.exe, ii) codeblocks-13.12mingw-setup.exe iii) codeblocks-13.12mingw-setup-TDM-GCC-481.exe। প্রতিনিয়ত সফটওয়্যার যেহেতু আপডেট হয় সে জন্য ডাউনলোড করার সময় ফাইল গুলোর "13.12" সংখ্যাটা নাও মিলতে পারে। না মিললে চিন্তিত হবার কোন কারণ নেই।

ডাউনলোড করার পর ইন্সটল করার পালা। এই ফাইল গুলো বাইনারী ফাইল, মানে রান করার উপযোগী, সে জন্য ইন্সটল করার পদ্ধতিও আর দশটা সফটওয়্যার ইন্সটল করার মত। (ফাইল টাতে ডাবল ক্লিক করা, এরপর কয়েকবার Next Button তে ক্লিক করা, সব শেষে Finish Button ক্লিক করা)

লিনাক্সে কমান্ড লাইন ব্যবহার করে কোডব্লকস ইন্সটল করার পদ্ধতি।

প্রথমে আমরা সফটওয়্যার পেকেজলিস্ট আপডেট করে নিব।

```
sudo apt-get update
```

তারপর কোডব্লকস চালানোর জন্য একটি সাহায্যকারী পেকেজ ইন্সটল করব। এটি মূলত g++ ইন্সটল করে।

```
sudo apt-get install build-essential
```

এখন আমরা কোডব্লকস ইন্সটল করার জন্য নিচের কমান্ডটি চালাবো।

```
sudo apt-get install codeblocks
```

ইন্সটল করা হয়ে গেলে টার্মিনাল থেকে `codeblocks` এই কমান্ডটি দিলেই কোডব্লকস রান করবে। আপনি চাইলে অ্যাপলিকেশন মেনু থেকেও কোডব্লকস রান করতে পারবেন।

প্রথম প্রোগ্রাম

প্রথমে কোডব্লোকস রান করি। তারপর নিচের কোডটুকু লিখি।

```
#include <stdio.h>
int main()
{
    printf("Hello Programming");
    return 0;
}
```

প্রোগ্রামটিকে first_program.c নামে সেভ করি। তারপর কোডব্লোকসের Build মেনু থেকে Build and run কমান্ডটিতে ক্লিক করি অথবা কিবোর্ড থেকে F9 বাটন চাপি। তাহলে নতুন একটা টার্মিনাল উন্ডোতে নিচের মত আউটপুট দেখতে পাবো।

Output

```
Hello Programming
```

নোট: প্রতিটা সি প্রোগ্রাম main() ফাংশন থেকে এক্সিকিউশন শুরু করে। printf() ফাংশনটি কোটেশনের ভিতরে থাকা কনটেন্ট প্রিন্ট করে। ফাংশন নিয়ে পরবর্তিতে আমরা আরও বিস্তারিত আলোচনা করব।

কন্ট্রোল স্টেটমেন্ট (প্রাথমিক ধারণা)

প্রোগ্রামিং মানে হচ্ছে লজিক। আর আমাদের লজিক গুলো কম্পিউটারকে জানাতে ব্যবহার করা হয় কন্ট্রোল স্টেটমেন্টস (Control Statements)।

এর আগে আমরা যত গুলো প্রোগ্রাম দেখেছি, সব গুলো শুধু মাত্র কিছু ডেটা ইনপুট নিয়েছে বা কিছু ডেটা আউটপুট দিয়েছে। কিন্তু কোন লজিক্যাল তুলনা করি নি। আমাদের যে সব প্রোগ্রাম লিখতে হবে, সে গুলোতে অনেক লজিক্যাল তুলনা করতে হবে। যেমন আমরা ৩টি সংখ্যার মধ্যে সবচেয়ে ছোট বা সবচেয়ে বড় সংখ্যা নির্ণয়ের ফ্লো চার্ট যদি দেখি, তাহলে দেখতে পাবো ঐখানে কিছু জায়গায় হ্যা অথবা না স্টেপ রয়েছে। আর ঐ ধরনের স্টেপ গুলোকে বলে লজিক্যাল স্টেপ। লজিক্যাল স্টেপ গুলোকে প্রোগ্রামে প্রকাশ করা হয় এই কন্ট্রোল স্টেটমেন্ট দিয়ে।

সি তে মূলত চার ধরনের কন্ট্রোল স্টেটমেন্ট রয়েছে। এগুলো হল

1. ডিসিশন মেকিং স্টেটমেন্ট
2. সিলেকশন স্টেটমেন্ট
3. ইটারেশন স্টেটমেন্ট
4. জাম্প স্টেটমেন্ট

সিনট্যাক্স হিসেবে কন্ট্রোল স্টেটমেন্ট গুলোর মধ্যে রয়েছেঃ

1. if-else Statement
2. while Statement
3. do-while statement
4. for statement ইত্যাদি।

সামনের অংশগুলোতে আমরা এসব কন্ট্রোল স্টেটমেন্ট নিয়ে আলোচনা করব। এগুলো জেনে আমরা সত্যিকারের প্রোগ্রামিং জগতে প্রবেশ করতে যাচ্ছি। যে যত সহজে কন্ট্রোল স্টেটমেন্ট গুলো দিয়ে নিজের লজিক গুলো কোডে পরিনত করতে পারবে, সে দ্রুত প্রোগ্রামিং এ ভালো করতে পারবে।

স্টেটমেন্ট এবং ব্লক

কন্ট্রোল স্টেটমেন্ট নিয়ে আগানোর আগে আমরা একটু ব্লক আর স্টেটমেন্ট এর ধারণাটা পরিষ্কার করে নেব। সাধারণত যে কোন এক্সপ্রেশনের শেষে সেমিকোলন ; দিলে সেটি একটি স্টেটমেন্ট হয়ে যায়। যেমন

```
x = 5;
i++;
printf("The result is ...");
```

উপরের তিনটি এক্সপ্রেশনই আলাদা আলাদা স্টেটমেন্ট।

একাধিক স্টেটমেন্টকে একসাথে `{` এবং `}` দিয়ে কম্পাউন্ড স্টেটমেন্ট বা ব্লক তৈরি করা হয়। ব্লকের বৈশিষ্ট্য হল ব্লকের সিনট্যাক্স গুলো একসাথে এক্সিকিউট হয়। কন্ট্রোল স্টেটমেন্ট নিয়ে কাজ করতে হলে বিভিন্ন যায়গায় বিভিন্ন লজিকের উপর ভিত্তি করে আমাদেরকে বিভিন্ন ব্লক এক্সিকিউট করা লাগতে পারে।

প্রাথমিক ধারণা

সি তে `if-else` হল ডিসিশন মেকিং স্টেটমেন্ট। প্রোগ্রামের বিভিন্ন যায়গায় আমাদের লজিকাল ডিসিশন নিতে হতে পারে এবং সেগুলোর উপর ভিত্তি করে আমাদের বিভিন্ন কাজ করা লাগতে পারে। এধরনের অপারেশনের জন্য `if-else` ব্যবহার করা হয়। `if-else` এর সিনট্যাক্স স্ট্রাকচার নিচের মত।

```
if(expression is true)
{
    statement;
}
else
{
    statement;
}
```

খুব সাধারণভাবে বলা যায় আমরা `if` দিয়ে একটি স্টেটমেন্ট টু নাকি ফলস সেটি দেখব এবং তারপর সেটি সত্য হলে আমরা কিছু স্টেটমেন্ট এক্সিকিউট করব আর না হলে অন্য কিছু স্টেটমেন্ট এক্সিকিউট করব। `if-else` আমাদের প্রোগ্রামটির কাজের ধারাকে বিভিন্ন ব্রাঞ্চ বা শাখায় ভাগ করে দিতে পারে। এখানে লক্ষ্যণীয় যে `if-else` এ মূলত `if` ব্লকটি প্রধান। অনেক সময় আমাদের `else` ব্লকটি একেবারে নাও লাগতে পারে। সেক্ষেত্রে শুধুমাত্র `if` ব্লকেই কাজ হয়ে যায়।

```
if(expression is true)
{
    statement;
}
```

আরেকটি বিষয় হল `if` সিনট্যাক্স এক্সপ্রেশনের নিউম্যারিক ভ্যালু নিয়ে কাজ করে। তাই সবসময় `expression != 0` না দিয়ে সরাসরি `expression` দিলেও হয়। অর্থাৎ নিচের দুইটি `if` ই কাজ করে একইভাবে।

```
if(expression !=0)
{
    statement;
}

if(expression)
{
    statement;
}
```

if-else এর ব্যবহার

আসুন এবার আমরা দেখি `if-else` কোথায় ব্যবহার করা যায়। আগেই বলেছি এটি ডিসিশন নিতে ব্যবহার করা হয়। আমরা একটি ছোট প্রোগ্রাম দিয়ে এর প্রয়োগ দেখব। এখানে আমরা দুইটি সংখ্যা ইউজারের কাছে জানতে চাইবো এবং সে দুটির মধ্যে কোনটি বড় আর কোনটি ছোট সেটি বের করব।

```
#include<stdio.h>

int main(void)
{

    int num1, num2;

    printf("Enter two numbers to compare :\n");

    scanf("%d %d", &num1, &num2);

    if(num1>num2) {
        printf("%d is greater than %d", num1, num2);
    }
    else {
        printf("%d is greater than %d", num2, num1);
    }

    return 0;
}
```

এখানে আমরা দুইটি সংখ্যা নিয়েছি এবং প্রথমে `if` দিয়ে আমরা দেখছি `num1` কি `num2` এর চাইতে বড় কি-না। যদি বড় হয় তাহলে `if` ব্লকটি এক্সিকিউট হবে এবং প্রিন্ট করবে `num1` বড়। এবং যদি `num1` বড় না হয় তাহলে `else` ব্লকটি কাজ করে এবং প্রিন্ট করবে `num2` বড়।

নেস্টেড if-else

আগের কোডটিতে আমাদের লজিক ফ্লো বেশ সাধারণ ছিল, আমাদের খালি একটি ডিসিশন নিতে হয়েছে। কিন্তু বাস্তবে এর চাইতে অনেক বেশি জটিল লাজিকাল ডিসিশন নিয়ে আমাদের কাজ করতে হতে পারে এবং একটি ডিসিশনের উপর ভিত্তি করে আরও ডিসিশন নিতে হতে পারে অর্থাৎ একটি `if` বা একটি `else` এর ভেতর আরও এক বা একাধিক `if-else` ব্লক থাকতে পারে। এধরনের কাজ করার জন্য আমরা `if-else` কে নেস্টেড করে ব্যবহার করতে পারি। উদাহরণ হিসেবে আমরা আগের উদাহরণটিকে একটু পাল্টিয়ে দেখব। দুটির বদলে আমরা এবার তিনটি সংখ্যা নেব এবং দেখব কোনটি সবচেয়ে বড়।


```
#include<stdio.h>

int main(void)
{

int num1,num2,num3;

printf("Enter three numbers to compare :\n");
scanf("%d %d %d", &num1, &num2, &num3);

if(num1>num2) { //if num1 is greater than num2
    if(num1>num3) { //checking with num3
        printf("%d is the largest number.", num1);
    }
    else {
        printf("%d is the largest number.", num3);
    }
}
else {
    if(num2>num3){ //checking num2 with num3
        printf("%d is the largest number.", num2);
    }
    else {
        printf("%d is the largest number.", num3);
    }
}

return 0;
}
```

এখানে আমরা যেটি করেছি সেটি হল, প্রথমে আমরা দেখেছি `num1` কি `num2` এর চাইতে বড় কি না। যদি বড় হয় তাহলে আমরা আরেকটি `if` ব্লকে `num1` কে `num3` এর সাথে তুলনা করেছি। আবার যদি প্রথম `if` এক্সপ্রেশনটি টু না হয় অর্থাৎ যদি `num1` ছোট হয়, তাহলে আমরা `else` ব্লকে গিয়ে আবার `num2` কে `num3` এর সাথে তুলনা করেছি। এরকমভাবে আমরা নেস্টেড `if-else` ব্যবহার করে আরো জটিল ধরনের লজিক নিয়ে কাজ করতে পারি।

সুইচ-কেস (switch case)

একটা ভ্যালু এর উপর নির্ভর করে অনেক গুলো স্টেটমেন্ট থেকে একটা স্টেটমেন্ট এক্সিকিউট করার জন্য switch case ব্যবহার করা হয়। switch case সাধারণত নিচের মত করে লেখা হয়ঃ

```
1 switch ( variable ) {
2     case expression 1:
3         statement;
4         break;
5     case expression 2:
6         statement;
7         break;
8     case expression 3:
9         statement;
10        break;
11    default:
12        statement;
13        break;
14
15 }
```

switch_case_form_c hosted with ♥ by GitHub

[view raw](#)

এখানে যদি switch (variable) এর variable টির মান expression 1 এর সাথে মিলে, তাহলে case expression 1 এর statement এক্সিকিউট হবে। যদি variable টির মান expression 2 এর সাথে মিলে, তাহলে case expression 2 এর statement এক্সিকিউট হবে। যদি variable টির মান expression 3 এর সাথে মিলে, তাহলে case expression 3 এর statement এক্সিকিউট হবে। যদি কোনটির সাথেই না মিলে, তাহলে default এর statement টি এক্সিকিউট হবে। এখানে যত ইচ্ছে তত গুলো case যুক্ত করা যাবে। আর case এর স্টেটমেন্ট শেষে break; যুক্ত করতে হয়। break; মানে হচ্ছে আমাদের কাজ শেষ, এবার switch case থেকে বের হতে পারি। break; টা খুব গুরুত্বপূর্ণ।

এখনো একটু জটিল মনে হতে পারে বিষয়টা, আমরা একটা উদাহরন দেখলে অনেক সহজ হয়ে যাবে এই switch case স্টেটমেন্টটি। আমরা একটা প্রোগ্রাম লিখব এমন, যেখানে যদি আমরা r ইনপুট দি, তাহলে লেখা উঠবে You select Red, যদি w ইনপুট দি, তাহলে লেখা উঠবে You select White. যদি b ইনপুট দি, তাহলে লেখা উঠবে You select Black. আর প্রোগ্রামটা লিখব আমরা switch case ব্যবহার করে।

```
1 #include <stdio.h>
2 int main ()
3 {
4     char colorCode;
5     printf("Enter first word of Red, White or Black: \n");
6     scanf("%c", &colorCode);
7 }
```

```

8
9  switch ( colorCode ) {
10     case 'r' :
11         printf("You select Red.");
12         break;
13
14     case 'w':
15         printf("You select White.");
16         break;
17
18     case 'b':
19         printf("You select Black.");
20         break;
21
22     default:
23         printf("Wrong choose!");
24         break;
25 }
26 return 0;
27 }

```

switth_case_example_c_1 hosted with ❤ by GitHub

[view raw](#)

উপরের প্রোগ্রামটি রান করি, তারপর r, w, b এ তিনটার মধ্যে যে কোন একটা ইনপুট দিলে ঐ কালারটা দেখবে। আর যদি আমরা অন্য কোন কারেকটার ইনপুট দি, তাহলে লেখা উঠবে Wrong choose!। switch case এর switch (colorFirsWord) এর ভেতরে যে ভ্যারিয়েবলটি রয়েছে, তার মান যদি case 'r' এর সাথে মিলে, মানে colorFirsWord এর মান r হয়, তাহলে প্রোগ্রামটি এ স্টেটম্যান্টটি এক্সিকিউট করবেঃ printf("You select Red."); এরপরের স্টেটম্যান্ট হচ্ছে break; মানে হচ্ছে আমরা যে কাজ করার জন্য switch case এর ভেতরে প্রবেশ করেছি, তা শেষ হয়েছে। switch case থেকে এবার আমরা বের হতে পারি। break দিয়ে switch case ঐখানেই বন্ধ করে দেওয়া হয়।

একই ভাবে যদি আমরা w ইনপুট দি, তাহলে case 'r' এর সাথে মিলিয়ে দেখবে। যেহেতু আমরা w ইনপুট দিয়েছি, প্রথম case এর সাথে মিলে না। পরের case দেখবে। পরের case এ এসে দেখবে case 'w' তে ইনপুটের সাথে মিল পেয়েছে, তাই এর পরে থাকা স্টেটম্যান্টটি এক্সিকিউট করবে। printf("You select White."); এবং এর পর break দিয়ে switch case থেকে বের হবে।

একই ভাবে আমরা যদি b ইনপুট দি, তাহলে উপরের দুইটা case এ কোন মিল পাবে না, তাই ঐ case গুলোর স্টেটম্যান্ট গুলোও এক্সিকিউট হবে না। শুধু মাত্র case 'b' এর স্টেটম্যান্ট গুলো এক্সিকিউট হবে।

যদি আমরা অন্য কোন কারেকটার ইনপুট দি, তাহলে তার জন্য রয়েছে ডিফল্ট ভ্যালু। তখন লেখা উঠবে Wrong choose!

বিঃ switch (colorCode) এর এখানে যে কোন Expression আমরা লিখতে পারি। এমন একটা উদাহরন আমরা দেখব। আবার case এ আমরা একের অধিক স্টেটম্যান্ট লিখতে পারব। তার উদাহরণ একটু পরই দেখব।

উপরের প্রোগ্রামটিতে আমরা যদি বড় হাতে R, W বা B ইনপুট দি, তাহলে লেখা উঠবে Wrong choose!। এখন আমরা উপরের প্রোগ্রামটিকে আরেকটু মডিফাই করব, যেন ছোট বা বড় হাতে colorCode ইনপুট দিলে উভয় ক্ষেত্রেই আমাদের সঠিক আউটপুট দেয়।

```
1  #include <stdio.h>
2  int main ()
3  {
4  char colorCode;
5  printf("Enter first word of Red, White or Black: \n");
6  scanf("%c", &colorCode);
7
8
9  switch ( colorCode ) {
10     case 'r' :
11     case 'R' :
12         printf("You select Red.");
13         break;
14
15     case 'w':
16     case 'W' :
17         printf("You select White.");
18         break;
19
20     case 'b':
21     case 'B' :
22         printf("You select Black.");
23         break;
24
25     default:
26         printf("Wrong choose!");
27         break;
28 }
29 return 0;
30 }
```

swich_case_example_c_2 hosted with ♥ by GitHub

[view raw](#)

এখানে আমরা লক্ষ্য করলে দেখব আমরা এখন break এর ব্যবহার ছাড়া এক সাথে দুই বার case ব্যবহার করেছি। যেমনঃ

case 'r' : case 'R' :

এভাবে break ছাড়া একের অধিক case লিখলে সে গুলো OR অপারেশনের মত কাজ করে। একটা সত্য হলে ঐ case গুলোর পরের Expression গুলো এক্সিকিউট হবে। এবং break পাওয়া পর্যন্ত অপেক্ষা করবে।

switch case এর ভেতরে যে কোন কোডই রান করানো যায়, লুপ চালানো, ফাংশান কল করা সহ সব কিছু। প্রথমে একটা প্রোগ্রাম লিখব, যেখানে switch case এর ভেতরে আমরা একবার for লুপ ব্যবহার করব, একবার while লুপ ব্যবহার করব। এটা বুঝানোর জন্য যে আমরা ইচ্ছে করলে যে কোন কোডই রান করাতে পারি switch case এর মধ্যেঃ

```

1  #include <stdio.h>
2  int main ()
3  {
4  int code;
5  int i = 0;
6  printf("Enter 1 to print even integers or enter 2 to print odd integers: ");
7  scanf("%d", &code);
8
9
10 switch ( code ) {
11     case 1 :
12         while(i<=100){
13             printf("%d \n", i);
14             i=i+2;
15         }
16         break;
17
18
19     case 2 :
20         for(i =1; i<=100; i=i+2)
21             printf("%d \n", i);
22
23         break;
24
25
26     default:
27         printf("Wrong choose!");
28         break;
29 }
30 return 0;
31 }
```

swithc_case_example_c_3 hosted with ❤ by GitHub

[view raw](#)

প্রোগ্রামটা আমাদের 1 অথবা 2 ইনপুট দিতে বলবে। 1 ইনপুট দিলে ১ থেকে ১০০ পর্যন্ত বেজোড় সংখ্যা গুলো প্রিন্ট করবে। আর 2 ইনপুট দিলে ১ থেকে ১০০ পর্যন্ত জোড় সংখ্যা গুলো প্রিন্ট করবে।

আমরা ইচ্ছে করলে switch case এর ভেতর থেকে ফাংশন ও কল করতে পারি। নিচের প্রোগ্রামটি দেখিঃ

```

1  #include <stdio.h>
2  int main ()
3  {
4      void playGame(){
```

```

5     printf("You choose to Play the game.");
6     }
7     void closeGame(){
8         printf("You choose to Close the game.");
9         }
10
11
12     int code;
13
14     printf("Enter 1 to play the Game.\nEnter 2 to close the Game :");
15     scanf("%d", &code);
16
17
18     switch ( code ) {
19         case 1 :
20             playGame();
21             break;
22
23
24         case 2 :
25             closeGame();
26             break;
27
28
29         default:
30             printf("Wrong choose!");
31             break;
32     }
33     return 0;
34 }

```

swicth_case_example_c_4 hosted with ❤ by GitHub

[view raw](#)

আমরা দুইটা ফাংশন তৈরি করেছি, playGame() এবং closeGame() । ব্যবহারকারী 1 ইনপুট দিলে playGame() ফাংশনটি কল হবে, এবং 2 ইনপুট দিলে closeGame() ফাংশন কল হবে... এভাবে নিজের মত করে নিজের ক্রিয়েটিভিটি ব্যবহার করে আমরা প্রয়োজন মত আমাদের প্রোগ্রামে switch case ব্যবহার করতে পারি । শুভ প্রোগ্রামিং :)

ফর-লুপ (for loop)

লুপিং এর কাজে সবচেয়ে বেশি ব্যবহৃত হয় for loop. এ for loop এর তিনটি অংশ রয়েছে। তার আগে আমরা দেখেনি for loop সাধারণ ব্যবহার নিয়ম।

for(expression1;Expression2;Expression3)Statement

বিঃ এখানে প্রত্যেকটি Expression ; (সেমিকোলন) দিয়ে আলাদা করে দিতে হবে।

এখানে প্রথম expression1 হচ্ছে for loop এর প্রথম অংশ। এটি দ্বারা একটি প্রাথমিক মান দেওয়া হয়। যাকে বলা হয় initial অংশ। এটি পুরো লুপিং প্রক্রিয়াকে নিয়ন্ত্রণ করে।

দ্বিতীয় অংশটি অর্থাৎ Expression2 দ্বারা একটি শর্ত দেওয়া হয়। লুপটি কতক্ষণ পর্যন্ত চলবে তা এটি নির্ণয় করে। Expression2 তে সাধারণত একটি logical expression থাকে যা শুধু সত্য মিথ্যে বুঝতে পারে। যদি সত্য হয় তাহলে 1 রিটার্ন করে আর যদি মিথ্যে হয় তাহলে 0 রিটার্ন করে। এটি যদি 0 ছাড়া অন্য কোন মান রিটার্ন করে তাহলে লুপটি চলবে, আর যদি 0 রিটার্ন করে তাহলে লুপটি আর চলবে না।

Expression3 কাজ হচ্ছে আমরা প্রথমে যে প্রাথমিক মান নিলাম তাকে আমাদের ইচ্ছে মত মডিফাই করা। এটি প্রত্যেক লুপের শেষ ধাপে কাজ করে।

আর আগেই বলছি লুপটি ততক্ষণই চলবে যতক্ষণ পর্যন্ত Expression2 মিথ্যে বা 0 রিটার্ন না করে।

For loop সম্পর্কে আমরা এতক্ষণ অনেক কিছু জানলাম, এবার প্রোগ্রামে এটাকে কিভাবে ব্যবহার করব তা দেখি। তার জন্য একটি প্রোগ্রাম লিখি যা 1 থেকে 10 পর্যন্ত সংখ্যাগুলো প্রিন্ট করবে। আপনাদের জন্য নিচের প্রোগ্রামটি। এটার আউটপুট কি হবে কিভাবে হবে তা বের করুন।

```
1 #include<stdio.h>
2 int main(void)
3 {
4     int i;
5     for(i=0;i<=10;i++)
6         printf("%d\n",i);
7 }
8
```

for_c1 hosted with ❤ by GitHub

[view raw](#)

এর আউট পুট হচ্ছেঃ 0 1 2 3 4 5 6 7 8 9 10 এখানে আমরা একটি integer variable নিয়েছি। for loop এর প্রথম Expression এ আমরা এর প্রাথমিক মান নিলাম 0. প্রাথমিক মানটি for loop এর দ্বিতীয় অংশ অর্থাৎ logical অংশ দ্বারা যাচাই না হয়েই for লুপের ভেতরে থাকা স্টেটমেন্টটি এক্সিকিউট করবে। এবং প্রথম Expression কাজ শেষ হয়ে যাবে। এর আর কোন কাজ নেই। Print করার পর এবার তৃতীয় অংশ অর্থাৎ Expression3 এখানে i++ অংশ দ্বারা মডিফাই হবে। আমরা জানি i++ এর মানে হচ্ছে i = i+1 সুতরাং এখানে i এর মান এক বাড়বে এবং 0 থেকে 1 হবে। এবার দ্বিতীয় অংশ Expression2 এখানে এসে i<=10 অংশ দ্বারা

লজিক্যাল যাচাই হবে। এখানে যাচাই করবে যে i এর মান 10 বা 10 থেকে ছোট কিনা। যেহেতু এখন i এর মান ১০ থেকে ছোট তাই লুপটা আবার চলবে। এবং দ্বিতীয় বার এসে 1 প্রিন্ট করবে। আবার Expression3 তে এসে মডিফাই হবে। আগের লুপ থেকে i এর মান পেয়েছি 1 এখন আবার 1 এর সাথে এক বেড়ে 2 হবে (এ অংশ $i++$ দ্বারা)। আবার দ্বিতীয় অংশ $i \leq 10$ অংশ দ্বারা লজিক্যাল যাচাই হবে। যেহেতু 2, 10 থেকে ছোট তাই আবার 2 প্রিন্ট করবে। এভাবে প্রত্যেক ধাপ শেষ করবে। যখন i এর মান বেড়ে 11 হয়ে যাবে তখন আর লুপ চলবে না। এবং আমাদের প্রোগ্রামটি শেষ হবে। আচ্ছা, আরেকটা প্রোগ্রাম লিখি। ছোট কালে কোন দুষ্টিমি করলে যে আমাদের শাস্তি দেওয়া হতো একশ বার লিখতে, আমি আর দুষ্টিমি করব না। আমরা এবার তা লিখব প্রোগ্রাম লিখে। এবং for লুপ ব্যবহার করে।

```
1 #include<stdio.h>
2 int main(void)
3 {
4     int i;
5     for(i=0;i<=100;i++)
6         printf("ami r dustumi korbo na. \n");
7     return 0;
8 }
```

for-c-3 hosted with ❤ by GitHub

[view raw](#)

এখানে আগের প্রোগ্রামের থেকে একটু পার্থক্য হচ্ছে আগে আমরা i এর মান প্রিন্ট করেছি। এখানে আমরা একটা লাইন প্রিন্ট করেছি "ami r dustumi korbo na." for লুপ এর ভেতরের Expression2 তে লিখেছি $i \leq 100$ । মানে i এর মান যতক্ষণ না পর্যন্ত ১০০ হচ্ছে, ততক্ষণ পর্যন্ত এই লুপটি চলবে। Expression3 তে i এর মান আমরা প্রতিবার ১ করে বাড়িয়ে দিয়েছি। এবার আমরা আরেকটি প্রোগ্রাম লিখি। এবার এক থেকে ৫০ পর্যন্ত বেজোড় সংখ্যাগুলো বের করার একটা প্রোগ্রাম লিখি।

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int i;
5     for(i=0;i<=50;i++)
6     {
7         if(i%2==1)
8             printf("%d\n",i);
9     }
10    return 0;
11 }
```

for_c2 hosted with ❤ by GitHub

[view raw](#)

আগের প্রথম প্রোগ্রামের মতই i এর মান আমরা প্রিন্ট করেছি। তবে একটা কন্ডিশন দিয়েছি এখানে। for লুপের ভেতর প্রতিবার চুকবে। চুকানোর পর if দিয়ে একটা কন্ডিশন চেক করবে। $\text{if}(i \% 2 == 1)$ মানে হচ্ছে i কে দুই দ্বারা ভাগ করলে ভাগ শেষ যদি ১ থাকে, তাহলে if কন্ডিশনের ভেতরের স্টেটমেন্ট $\text{printf}("%d\n", i);$ দিয়ে i এর মান প্রিন্ট হবে। আর না হলে কিছুই হবে না। এখানে কি করছি কি, একটা লুপের ভেতর আরেকটা লুপ ব্যবহার করেছি। আমরা ইচ্ছে করলে এমন একটা লুপের ভেতর আরেকটা, আরেকটা ভেতর আরেকটা এমন ইচ্ছে মত ব্যবহার করতে পারি। যেমন আমরা এবার একটা for লুপের ভেতর আরেকটা for লুপ ব্যবহার করব। তবে তার আগে উপরের প্রোগ্রামটি আরো সহজে কিভাবে লেখা যায়, তা দেখি। আমরা ইচ্ছে করলে ১ থেকে ৫০ পর্যন্ত বেজোড় সংখ্যাগুলো নিচের মত করেও

বের করতে পারিঃ

```

1  #include<stdio.h>
2  int main(void)
3  {
4  int i;
5  for(i=1;i<=50;i=i+2)
6      printf("%d\n",i);
7
8  return 0;
9  }
```

for-c-4 hosted with ❤ by GitHub

[view raw](#)

এখানে করছি কি i এর প্রথম মান ধরে নিয়েছি ১। এক একটা বিজোড় সংখ্যা। তা প্রিন্ট করবে। এরপর ১ এর সাথে ২ যোগ করে দিলেই তো হবে ৩, তা বিজোড় সংখ্যা। তা প্রিন্ট করবে। এরপর ৩ এর সাথে ২ যোগ করে দিলে হবে ৫, তা বিজোড় সংখ্যা। তা প্রিন্ট করবে। আমরা লুপের Expression3 তে লিখছি i=i+2। লুপের Expression2 তে কন্ডিশন দিয়েছি i<=50। মানে যতক্ষণ না পর্যন্ত i এর মান ৫০ এর বেশি হবে, ততক্ষণ পর্যন্ত লুপটি চলবে।। এবার একটা for লুপের ভেতরে আরেকটা for লুপ ব্যবহার করে একটা প্রোগ্রাম লিখিঃ

```

1  #include<stdio.h>
2  int main(void){
3  int i, j;
4
5  for(i=0;i<=5;i++){
6      for (j=0;j<=i;j++){
7          printf("%d ", j);
8      }
9      printf("\n");
10 }
11
12 return 0;
13 }
14
```

nested-for-c hosted with ❤ by GitHub

[view raw](#)

এখানে আমরা দুইটা ভ্যারিয়েবল নিয়েছি। i এবং j. প্রথম for লুপের ভেতর i এর ইনিশিয়াল মান দিয়েছি ০, কন্ডিশন দিয়েছি i<=5 এবং i এর মান 1 করে বাড়িয়েছি। মানে হচ্ছে প্রথম for লুপটি ৫ বার এক্সিকিউট হবে। প্রোগ্রামটি আউটপুট দিবেঃ ``c 0 0 1 0 1 2 0 1 2 3 0 1 2 3 4 0 1 2 3 4 5 `` দ্বিতীয় ফর লুপে j=0 দিয়ে ইনিশিয়াল মান দিয়েছি ০, এরপর j<=i দিয়ে কন্ডিশন দিয়েছি। এবং শেষে j++ দিয়ে j এর মান বাড়িয়েছি। কন্ডিশন অনুযায়ী দ্বিতীয় ফরলুপটি কতবার এক্সিকিউট হবে তা নির্ভর করবে প্রথম ফর লুপের উপর। যেমন প্রথম বার i এর মান ০। তাই প্রথমবার দ্বিতীয় ফর লুপ চলবে একবার। দ্বিতীয় বার i এর মান ১, তাই দ্বিতীয় বার দ্বিতীয় লুপ চলবে দুই বার। তৃতীয় বার দ্বিতীয় ফর লুপ চলবে ৩ বার। এবার পঞ্চম বার দ্বিতীয় লুপ চলবে ৫বার। প্রথম বার দ্বিতীয় লুপ প্রিন্ট করবে ০, দ্বিতীয়বার প্রিন্ট করবে 0 1. তৃতীয় বার 0 1 2. এভাবে পঞ্চম বার প্রিন্ট করবে 0 1 2 3 4 5. এখন আমরা যদি প্রথম ফর লুপের কন্ডিশন পরিবর্তন করে দশ করে দি, তাহলে আউটপুট দিবেঃ

```

main.c X
1  #include<stdio.h>
2  int main(void){
3      int i, j;
4
5      for(i=0;i<=10;i++){
6          for (j=0;j<=i;j++){
7              printf("%d ", j);
8          }
9          printf("\n");
10     }
11
12     return 0;
13 }
14
C:\Users\Jack\Documents\test\bin\Debug\test.exe
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9 10
Process returned 0 (0x0)   execution time : 0.026 s
Press any key to continue.

```

do while দিয়ে আমরা কয়েকটা সংখ্যার গড় বের করার প্রোগ্রাম লিখেছি এর আগে। এবার আমরা তা for লুপ ব্যবহার করে লিখবঃ

```

1  #include <stdio.h>
2  int main ()
3  {
4      int total_no, count;
5      float number, average, sum =0;
6
7      printf("How many numbers? ");
8      scanf("%d", &total_no);
9
10     for(count = 1; count <= total_no; count++){
11         printf("Enter number %d : ", count);
12         scanf("%f", &number);
13         sum +=number;
14     }
15
16     average = sum / total_no;
17     printf("Average is : %f\n", average);
18
19     return 0;
20 }

```

for-average-c hosted with ❤ by GitHub

[view raw](#)

হোয়াইল-লুপ (while loop)

while লুপের সাধারণ ফরম হচ্ছেঃ

while (expression) statement

expression বলতে একটা কন্ডিশন দেওয়া হয়। যতক্ষণ পর্যন্ত এই কন্ডিশনটি সত্য হবে, ততক্ষণ পর্যন্ত while লুপটি চলবে। ছোট্ট একটা প্রোগ্রাম লিখিঃ

```
1 #include <stdio.h>
2 int main ( ){
3
4     int number = 0;
5     while (number <=9){
6         printf("%d \n", number);
7         number++;
8     }
9 }
```

while_c1 hosted with ❤ by GitHub

[view raw](#)

উপরের প্রোগ্রামে আমরা number নামে একটা ইন্টিজার ভ্যারিয়েবল নিয়েছি। যার প্রাথমিক মান হচ্ছে ০। এরপর আমরা while লুপ লিখছি। (number <=9) এটা হচ্ছে কন্ডিশন। যতক্ষণ পর্যন্ত এ কন্ডিশনটা সত্য হবে, ততক্ষণ পর্যন্ত ব্র্যাকেটের ভেতরের {...} কোড গুলো এক্সিকিউট হবে। ব্র্যাকেটের ভেতরে আমরা প্রথমে নাম্বারটি প্রিন্ট করেছি। পরের লাইনে number++; দিয়ে নাম্বারটির মান এক করে বাড়িয়ে দিয়েছি।

প্রথমে number ভ্যারিয়েবলটির ভ্যালু ছিল ০... প্রথমে while লুপের ভেতর ঢুকে নাম্বারটি প্রিন্ট করল। এরপর number++; দিয়ে নাম্বারের ভ্যালু এক বাড়িয়ে দিল। এখন number ভ্যারিয়েবল এর মান ১ এরপর আবার লুপের প্রথমে ফিরে গেলো। গিয়ে কন্ডিশনটি (number <=9) চেক করল। number ভ্যারিয়েবলের মান কি ৯ অথবা এর থেকে ছোট? যেহেতু নাম্বার ভ্যারিয়েবলের মান ১, এবং ৯ থেকে ছোট। তাই আবার লুপের ভেতরে ঢুকবে। এবং আবার প্রিন্ট করবে। এবার প্রিন্ট করবে 1. আবার number++; দিয়ে ভ্যালু এক বাড়িয়ে দিবে, মান হবে ২। এবং আবার লুপের প্রথমে গিয়ে চেক করবে। যখন দেখবে ৯ থেকে ছোট, তখন লুপের ভেতরে ঢুকবে। এবং আবার number টি প্রিন্ট করবে। ১০ বার লুপে ঢুকবে।

শেষ বার যখন number++ দিয়ে এক মান বাড়িয়ে দিবে, তখন number ভ্যারিয়েবল এর মান হবে 10 এবং যখন কন্ডিশনটি চেক করবে, তখন দেখবে number ভ্যারিয়েবলের মান ৯ থেকে বেশি। তখন আর while লুপটি কাজ করবে না। প্রোগ্রামটি শেষ হবে।

আমরা আরেকটা প্রোগ্রাম লিখতে পারি। ছোট বেলায় দুঃখ করলে মাঝে মাঝে শিক্ষক বা বাড়িতে তো ১০০ বার লিখতে দিত, "আমি আর দুঃখ করব না।" তখন যদি আমরা প্রোগ্রামিং জানতাম, আমাদের এত কষ্ট করতে হতো না। কয়েক লাইনের কোড লিখলেই ১০০ বার লেখা হয়ে যেতো। চাইলে তখন ১০০০ বার বা ১লক্ষ বার ও লিখে দেওয়া যেত। while লুপ দিয়ে এই প্রোগ্রামটি লিখি। প্রোগ্রামের ভেতর বাংলায় লিখলাম "ami r dustumi korbo na"

```

1  #include <stdio.h>
2  int main ( ) {
3
4  int number = 1;
5  while (number <=100){
6      printf("ami r dustumi korbo na. \n");
7      number++;
8  }
9  }
```

while_c2 hosted with ♥ by GitHub

[view raw](#)

প্রোগ্রামটি রান করলে দেখব, কনসোলে ১০০ বার লেখা উঠে, আমি আর দুঃখ করব না।

এখানে এর আগের প্রোগ্রামের মত একই কাজই করা হয়েছে। আগের প্রোগ্রামে নাম্বার ভ্যারিয়েবলটি প্রিন্ট করা হয়েছে। এখন প্রিন্ট করা হয়েছে একটি স্ট্রিং।

প্রতিবার স্ট্রিংটি প্রিন্ট করার পর number ভ্যারিয়েবল এর মান এক করে বাড়িয়ে দেওয়া হয়েছে। এবং while লুপের প্রথমে গিয়ে কন্ডিশনটি চেক করা হয়েছে। যতক্ষণ পর্যন্ত দেখল number এর মান ১০০ থেকে ছোট বা সমান, ততক্ষণ পর্যন্ত লুপটি চলেছে। এবং যখন number এর মান ১০১ এক হয়েছে, তখন লুপের কাজ শেষ হয়েছে।

while লুপ দিয়ে আরেকটি প্রোগ্রাম লিখতে পারি। 1 থেকে 100 পর্যন্ত বেজোড় সংখ্যা গুলো প্রিন্ট করতে পারি। নিচের প্রোগ্রামটি দেখিঃ

```

1  #include <stdio.h>
2  int main ( )
3  {
4  int number = 0;
5  while (number <=100){
6      printf("%d \n", number);
7      number = number + 2;
8  }
9  }
```

while_c3 hosted with ♥ by GitHub

[view raw](#)

এক এর থেকে দুই যোগ করলে হবে তিন। একটা বিজোড় সংখ্যা। তিনের সাথে আবার দুই যোগ করলে হবে ৫, আরেকটি বিজোড় সংখ্যা। এভাবে প্রতিবার ভ্যারিয়েবল এর মান দুই বাড়িয়ে দিলেই আমরা বিজোড় সংখ্যা গুলো পেতে পারি। উপরের প্রোগ্রামে সে কাজটিই করা হয়েছে। `number = number + 2;` দিয়ে প্রতিবার নাম্বার ভ্যারিয়েবলটির মান দুই করে বাড়িয়ে দেওয়া হয়েছে।

প্রথমে ভ্যারিয়েবলটির মান ছিল 1, `int number = 1;` এর কথা বলছি। এরপর লুপের ভেতরে ঢুকলো। এবং কন্ডিশন চেক করা হল। যেহেতু কন্ডিশন সত্য, তাই লুপের ভেতরের কোড গুলো রান করল। `number` টি প্রিন্ট করল। পরের লাইনে `number` ভ্যারিয়েবল টির মান দুই বাড়িয়ে দেওয়া হলো। এভাবে এক সময় যখন দেখল `number` ভ্যারিয়েবলটির মান 100 থেকে বেশি হয়ে গেলো, তখন লুপটি কাজ করা শেষ করল।

উপরের প্রোগ্রামটিকে একটু মডিফাই করেই তো জোড় সংখ্যা গুলো বের করতে পারি তাই না? তাহলে তা করে ফেলুন। এরপর কमेंটে আমাকে জানাতে পারেন।

ডু-হোয়াইল-লুপ (do while loop)

কিছু একটা কর, যতক্ষণ পর্যন্ত একটা কন্ডিশন সত্য হয়। এমন প্রোগ্রাম লিখতে আমরা do while ব্যবহার করি। do while লুপের সাধারণ ফরম হচ্ছেঃ

do statement while (expression);

expression বলতে একটা কন্ডিশন দেওয়া হয়। যতক্ষণ পর্যন্ত এই কন্ডিশনটি সত্য হবে, ততক্ষণ পর্যন্ত এই do while লুপটি চলবে এবং এই statement এক্সিকিউট হতে থাকবে। একটি স্টেটমেন্ট এক্সিকিউট করার জন্য আমাদের দ্বিতীয় ব্র্যাকেট ব্যবহার করতে হবে না। কিন্তু যদি আমরা একের অধিক স্টেটমেন্ট এক্সিকিউট করতে চাই, তাহলে আমাদের দ্বিতীয় ব্র্যাকেট ব্যবহার করতে হবে। তখন লিখতে হবে এমন করেঃ

```
do{
    statement 1;
    statement 2;
    statement 3;
    ....
} while (expression);
```

ছোট্ট একটা প্রোগ্রাম লিখিঃ

```
1  #include <stdio.h>
2  int main ()
3  {
4      int number = 0;
5      do {
6          printf("%d \n", number);
7          number++;
8      } while (number <=9);
9  return 0;
10 }
```

do-while-1 hosted with ❤ by GitHub

[view raw](#)

উপরের প্রোগ্রামে আমরা number নামে একটা ইন্টিজার ভ্যারিয়েবল নিয়েছি। যার প্রাথমিক মান হচ্ছে ০। এরপর আমরা do while লুপ লিখছি। do এর পর দ্বিতীয় ব্র্যাকেটের মধ্যে আমরা যে স্টেটমেন্ট গুলো এক্সিকিউট করতে হবে, সে গুলো লিখেছি। এরপর লিখছি while (number <=9) এটা হচ্ছে কন্ডিশন। যতক্ষণ পর্যন্ত এ কন্ডিশনটা সত্য হবে, ততক্ষণ পর্যন্ত ব্র্যাকেটের ভেতরের {...} কোড গুলো এক্সিকিউট হবে। ব্র্যাকেটের ভেতরে আমরা প্রথমে নাম্বারটি প্রিন্ট করেছি। পরের লাইনে number++; দিয়ে নাম্বারটির মান এক করে বাড়িয়ে দিয়েছি। এরপর এক সময় number এর মান 10 হয়ে গেছে। তখন while (number <=9) এ এসে দেখল কন্ডিশনটি মিথ্যে হয়ে গেছে। মানে number এর মান ৯ এর থেকে বড়, তখন আর do ভেতরে আর ঢুকবে না এবং তার ভেতরের কোড গুলোও এক্সিকিউট করবে না।

উপরের প্রোগ্রামটি আমরা আরেকটু ছোট করে লিখতে পারে। একই ভাবে কাজ করবে। শুধু ব্রাকেটটি সরিয়ে ফেলছি।

```

1  #include <stdio.h>
2  int main ()
3  {
4      int number = 0;
5      do
6          printf("%d \n", number++);
7      while (number <=9);
8  return 0;
9  }
10

```

do-while-c-2 hosted with ❤ by GitHub

[view raw](#)

আমরা যদি একটি মাত্র স্টেটমেন্ট এক্সিকিউট করতে চাই, তাহলে আমাদের দ্বিতীয় ব্রাকেট ব্যবহার না করলেও হবে।

do while ব্যবহার করে আমরা কিছু সংখ্যার গড় বের করব। প্রোগ্রামটি প্রথম ব্যবহারকারীকে জিজ্ঞেস করব, কয়টা সংখ্যার গড় ব্যবহার করতে চায়। এরপর এক এক করে সবগুলো সংখ্যা ইনপুট নিবে। তারপর গড় দেখাবে।

```

1  #include <stdio.h>
2  int main ()
3  {
4      int total_no, count = 1;
5      float number, average, sum =0;
6
7      printf("How many numbers? ");
8      scanf("%d", &total_no);
9
10
11     do{
12
13         printf("Enter number %d : ", count);
14         scanf("%f", &number);
15         sum +=number;
16         count++;
17
18     }while (count <=total_no);
19
20     average = sum / total_no;
21     printf("Average is : %f\n", average);
22
23     return 0;
24 }

```

average-do-while hosted with ❤ by GitHub

[view raw](#)

আমরা অনেক গুলো ভ্যারিয়েবল ডিক্লেয়ার করেছি। `total_no` হচ্ছে কয়েটা নাম্বারের গড় বের করব, তার জন্য। এরপর আমরা `do while` লুপে প্রবেশ করেছি। লুপ ততক্ষণই চলবে যতক্ষণ না পর্যন্ত আমাদের সব গুলো নাম্বার ইনপুট নেওয়া হয়। তার জন্য আমরা একটা `count` ভ্যারিয়েবল নিয়েছি। যার প্রথম মান হচ্ছে ১। এর পর প্রতিবার আমরা একটা সংখ্যা ইনপুট নিব, একবার করে এই `count` এর মান বাড়িয়ে দিব।

`number` নামক ভ্যারিয়েবল দিয়ে আমরা এরপর প্রতিটা সংখ্যা ইনপুট নিচ্ছি। ইনপুট নিয়ে সেগুলো সব যোগ করছি। ছোট্ট একটা লাইন দিয়ে `sum += number;` যার মানে হচ্ছে `sum = sum + number;`

যখন দেখেছি `count` এর মান `total_no` এর থেকে বড় হয়ে গেছে, তার মানে হচ্ছে আমাদের সকল সংখ্যা ইনপুট নেওয়া হয়ে গেছে। তাই আমরা `do while` থেকে বের হয়ে গিয়েছি। এরপর গড় বের করেছি। তারপর প্রিন্ট।

প্রাথমিক ধারণা

প্রোগ্রামিং এর ভাষায় ফাংশন হল একটি নির্দিষ্ট ধরনের কাজ করে এমন কতগুলো ইন্সট্রাকশনের সমষ্টি। আমরা সি তে কোন কিছু মনিটরে প্রিন্ট করে দেখাতে চাইলে `printf()` ফাংশনটি ব্যবহার করি। `printf()` আমাদের দেয়া লিখাটি নিয়ে সেটিকে মনিটরে দেখায়। কিংবা আমরা যদি কোন সংখ্যার বর্গমূল জানতে চাইলে তাহলে `math.h` থেকে `sqrt()` ফাংশনটি ব্যবহার করে সেটি জানতে পারি। এগুলোর সবগুলোই এক একটি ফাংশন। এই ফাংশনগুলো বিভিন্ন লাইব্রেরিতে দেওয়া আছে আমাদের কাজকে সহজ করার জন্য। এর বাইরে আমরা যদি চাই আমাদের নির্দিষ্ট কোন কাজের জন্য একটি ফাংশন লিখতে হবে আমরা সেটিও করতে পারি।

প্রশ্ন আসতে পারে আমরা কেন নিজেরা ফাংশন লিখতে যাবো? ধরুন আপনাকে একটি বড় প্রোগ্রাম লিখতে হচ্ছে যেখানে বার বার আপনাকে কয়েকটি সংখ্যার গড় বের করতে হবে। গড় বের করার জন্য প্রদত্ত সংখ্যাগুলোকে যোগ করে মোট যতটি সংখ্যা আছে তা দিয়ে ভাগ করলেই আমরা গড় পাই। কিন্তু প্রোগ্রামের মধ্যে যতবার আপনাকে গড় বের করতে হবে প্রতিবার তার জন্য একই ধরনের কোড বার বার লিখাটা বেশ ঝামেলা। আর এভাবে লিখতে থাকলে প্রোগ্রামটির কোডের গঠন ও বেশ অগোছালো হয়ে যায়।

এই বিষয়টি আমরা সমাধান করতে পারি সহজেই গড় বের করার জন্য একটি ফাংশন লিখে। ফাংশন লিখলে আমরা একবারেই ঠিক করে দেব কিভাবে গড় হিসাব করতে হয় এবং তারপরে আমরা যেখানেই গড় বের করতে যাবো সেখানেই আমরা এই ফাংশনটি কল করতে পারবো এবং সহজেই গড় পেয়ে যাবো। একই সাথে আমাদের প্রোগ্রামটির কোডও বেশ গোছানো থাকবে কারণ আমরা বার বার একই ধরনের কোড লিখবো না।

সি তে দুই ধরনের ফাংশন রয়েছে।

1. লাইব্রেরি ফাংশন `printf()`, `sin()`, `sqrt()`
2. ইউজার ডিফাইনড ফাংশন

লাইব্রেরি ফাংশন গুলো আমরা সব সময়ই ব্যবহার করি কম বেশি। আমাদের এই অধ্যায়ের মূল আলোচনা হবে ইউজার ডিফাইনড ফাংশন নিয়ে।

এবারের অধ্যায়ে আমরা দেখব কীভাবে ইউজার ডিফাইন ফাংশন লিখা যায়। প্রথমেই দেখে নিই সি তে ফাংশনের গঠন কেমন হয়। উদাহরণ হিসেবে আমরা একটি ফাংশন লিখব যেটি একটি আয়তক্ষেত্রের ক্ষেত্রফল বের করে। শুরুতেই দেখে নিই ফাংশনের গঠন কেমন হয়।

```
return_type function_name(parameter list)
{
    Body of the function
    ...
}
```

আসুন এটি একটু ব্যখ্যা করি।

return_type :

রিটার্ন টাইপ হল ফাংশন কি ধরনের ড্যালা আপনাকে ফেরত দিবে। আমরা সাধারণত ফাংশন লিখি কারণ আমরা ফাংশনে কিছু একটা কাজ করে তার ফলাফলটি ফেরত আনতে চাই। রিটার্ন টাইপ হল ফাংশনটি যে ধরনের ড্যালা ফেরত দেবে তার ডেটা টাইপ। এটি সি এর যে কোন ডাটা টাইপ হতে পারে। সব ফাংশনই যে ড্যালা রিটার্ন করবে এমন নয়। যদি কোন ফাংশন কোন ড্যালা রিটার্ন না করে তাহলে তার রিটার্ন টাইপ হবে `void`

function_name

ফাংশনগুলোর একটি নির্দিষ্ট নাম থাকতে হবে। এই ফাংশন নেম ধরেই আমরা ফাংশনটিকে প্রোগ্রামের বিভিন্ন যায়গায় কল করব। ফাংশন নেম আমরা আমাদের পছন্দমত দিতে পারি। তবে ভাল প্র্যাকটিস হল ফাংশনের নামগুলো এমনভাবে দেওয়া যাতে সেটি পড়ে বোঝা যায় ফাংশনটি কি করে। যেমন ক্ষেত্রফল বের করার ফাংশনটির নাম আমরা `finn_area()` দিতে পারি আবার `klmn()` ও দিতে পারি। দুই ক্ষেত্রেই ফাংশনটি ভেতরের ইনস্ট্রাকশন অনুযায়ী কাজ করবে। শুধু একটি ক্ষেত্রে এটি বোঝা সহজ যে ফাংশনটি কি করে আরেকটিতে না।

Parameter list

ফাংশনটিতে আমরা যে ড্যালাগুলো নিয়ে কাজ করতে পাঠাবো সেগুলো থাকে প্যারামিটার লিস্ট এ। প্যারামিটার লিস্ট এ আমরা বিভিন্ন ড্যারিয়েবল ফাংশনে পাঠাই এবং এখানে ড্যারিয়েবলের ডেটা টাইপ বলে দিতে হয়। ফাংশন কল করার সময় এই ড্যালাগুলো ইনপুট হিসেবে দিতে হয়। কোন ইনপুট না থাকলে `void` দেওয়া হয় প্যারামিটার হিসেবে। প্যারামিটার গুলোকে সরারি ড্যারিয়েবল হিসেবে ফাংশনের ভেরত থেকে কল করা যায়।

function body

ফাংশন বডিতে আমরা আমরা ফাংশনটি কি কি ইনস্ট্রাকশন এক্সিকিউট করবে সেটি লিখব। যেমন ক্ষেত্রফল বের করার জন্য আমরা দৈর্ঘ্য ও প্রস্থ গুণ করব। এরপর আমরা ক্ষেত্রফলটি ফেরত পাঠাবো বা রিটার্ন দেব। এটিই হবে ফাংশনের রিটার্ন ড্যালা। রিটার্ন ড্যালায় ডেটা টাইপ এবং `return_type` একই হতে হবে।

এবার আসুন আমরা আমাদের ফাংশনটি লিখি। আমরা বলেছি আমরা এমন একটি ফাংশন লিখবো যেটি দিয়ে আয়তক্ষেত্রের ক্ষেত্রফল বের করা যাবে।

```
#include<stdio.h>

int find_area(int len, int wid) //function declaration
{
    int area;

    area = len*wid;

    return area; //returning the result
}

int main(void)
{

    int length, width, area;

    printf("Enter the length and width of the rectangle: \n");
    scanf("%d", &length);
    scanf("%d", &width);

    area = find_area(length, width); //calling the function and getting the value back

    printf("The area of the rectangle is %d",area);

    return 0;
}
```

এখানে প্রথমে আমরা `find_area` নামের একটি ফাংশন লিখেছি যেটির রিটার্ন টাইপ `int` এবং এটি দুটি প্যারামিটার গ্রহণ করে। একটি হল `len` যেটি একটি `int` ভ্যারিয়েবল, অপরটি `wid` এটিও `int`। এরপর আমরা ফাংশনটির ভেতরে একটি ভ্যারিয়েবল নিয়েছি যেটির নাম `area` এটিতে আমরা ক্ষেত্রফল রাখবো। এরপর আমরা ক্ষেত্রফল বের করে সেটিকে রিটার্ন করেছি। `main()` ফাংশনের ভেতর থেকে আমরা আয়তক্ষেত্রের দৈর্ঘ্য ও প্রস্থ নিয়ে `find_area()` ফাংশনটিকে কল করেছি, এবং কল করার সময় `length` ও `width` কে প্যারামিটার হিসেবে দিয়েছি। এবং রিটার্ন ভ্যালুটি `area` নামের একটি ভ্যারিয়েবলে ফেরত নিয়েছি এবং সেটি প্রিন্ট করেছি। কল করা এবং ভ্যালু ফেরত নেওয়ার কাজটি হয়েছে `area = find_area(length, width);` স্টেটমেন্টে।

বেস্ট প্র্যাকটিস

এই প্রোগ্রামটি ঠিকমত কাজ করলেও আমরা এখানে একটি বেস্ট প্র্যাকটিস ফলো করিনি। এখানে আমরা ফাংশনটি `main` এর আগে লিখেছি। যদিও এটি একটি ছোট ফাংশন, কিন্তু ফাংশনটি যদি অনেক বড় হত এবং এরকম আরও বেশ কয়েকটি ফাংশন থাকতো তাহলে `main` ফাংশনটি খুজে বের করার জন্য আমাদের স্ক্রল করে অনেক নিচে নামতে হত এবং প্রোগ্রামটিও একটু অগোছালো মনে হতো। এধরনের ঝামেলা এড়ানোর জন্য সাধারণত ফাংশন গুলোকে `main` এর পরে লিখা হয়। কিন্তু ফাংশন ডেফিনেশনের আগে সেটিকে কল করলে কম্পাইলার এর

দেখাবে। এজন্য শুরুতেই ফাংশন প্রোটোটাইপ ডিক্লেয়ার করে দেওয়া হয়। প্রোটোটাইপ জিনিসটি একটি প্লেসহোল্ডারের মতই যেটি কম্পাইলারকে বলে দেয় সামনে কোথাও এরকম একটি ফাংশন ডিফাইন করা আছে। তাহলে আমরা দেখি কীভাবে এটি করতে হয়।

```
#include<stdio.h>

int find_area(int len, int wid); //function prototype declaration

int main(void)
{

    int length, width, area;

    printf("Enter the length and width of the rectangle: \n");
    scanf("%d", &length);
    scanf("%d", &width);

    area = find_area(length, width);

    printf("The area of the rectangle is %d",area);

    return 0;
}

int find_area(int len, int wid) //function declaration
{

    //function definition
    int area;

    area = len*width;

    return area; //returning the result
}
```

এখানে আমরা শুরুতেই `int find_area(int len, int wid);` লাইনটি দিয়ে ফাংশন প্রোটোটাইপ বলে দিয়েছি। এখানে আমরা ফাংশন বডি লিখি নি। কিন্তু এখন কম্পাইলার জানে সামনে কোথাও ফাংশন বডি ও ডেফিনিশন লিখা আছে।

এভাবে কোড লিখলে সেটির রিডাবিলিটিও ভালো থাকে কারণ আমরা শুরুতেই যেনে যাই সামনে কী কী ফাংশন আছে এবং আমরা দ্রুত `main` ফাংশনে গিয়ে দেখে নিতে পারি কোন ফাংশন কিভাবে কল করা হয়েছে।

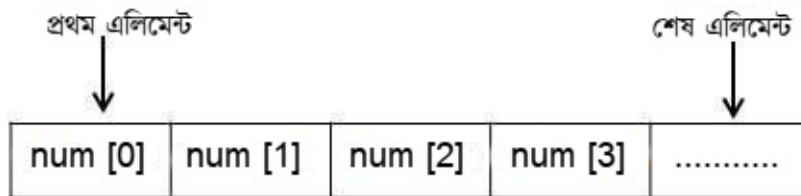
প্রাথমিক ধারণা

অ্যারে কি?

যে কোন ধরনের প্রোগ্রামিং এর বেলায় একটি সাধারণ চ্যালেঞ্জ হল একই ধরনের অনেকগুলো ডেটা নিয়ে কাজ করা। যেমন আপনাকে হয়ত তিন হাজার মানুষের হোম ডিস্ট্রিক্ট নিয়ে কাজ করতে হবে বা তিনশত ছাত্র-ছাত্রীর একটি বিষয়ের প্রাপ্ত নম্বর নিয়ে কাজ করতে হবে। অল্প সংখ্যক ডেটা হলে আমরা দুই একটি ভ্যারিয়েবল নিয়ে কাজগুলো করতে পারি। কিন্তু এরকম অজস্র ডেটা হলে তার জন্য অজস্র ভ্যারিয়েবল তৈরি করাটা বেশ ঝামেলার।

এই ধরনের ডেটা নিয়ে কাজ করার জন্য সি প্রোগ্রামিং এ আছে অ্যারে। খুব সহজে বলতে গেলে অ্যারে হল একই ডেটা টাইপের একই ধরনের ডেটার একটি নির্দিষ্ট সিরিজ।

যেহেতু অ্যারে একটি সিরিজ, তাই এর শুরু এবং শেষ আছে। অ্যারে বানানোর সময়ই এই সাইজ বলে দিতে হয় কারণ অ্যারের সাইজের সাথে মেমোরি অ্যালোকেশনের ব্যাপার আছে। অ্যারেগুলোর একটি নির্দিষ্ট সাইজ থাকে এবং এদের ইনডেক্সিং শুরু হয় জিরো বা শূন্য থেকে।



উপরের চিত্রটিতে আমরা একটি অ্যারের সাধারণ রিপ্রেজেন্টেশন দেখতে পারছি।

সি প্রোগ্রামিং এ অ্যারে দুই প্রকার।

- ওয়ান ডাইমেনশনাল অ্যারে
- মাল্টি ডাইমেনশনাল অ্যারে

উপরের ছবিটি একটি ওয়ান ডাইমেনশনাল অ্যারের উদাহরণ। এখানে একটি সিরিজে সবগুলো ভ্যারিয়েবল বা এলিমেন্ট রাখা হয়। মাল্টি ডাইমেনশনাল অ্যারে হতে পারে যখন আমরা ম্যাট্রিক্স নিয়ে কাজ করতে চাই বা যদি অ্যারেতে একাধিক ইনডেক্সিং দরকার হয়।

পরের অংশগুলোতে আমরা অ্যারে নিয়ে কীভাবে কাজ করতে তা দেখব।

অ্যারে ডিক্লেয়ারেশন

সি তে অ্যারে ব্যবহার করার আগে সেগুলোকে ডিক্লেয়ার করে নিতে হয় সাধারণ ভ্যারিয়েবলের মতই। অ্যারে ডিক্লেয়ারেশনের সিনট্যাক্স হচ্ছে

```
dataType Array_Name [ size ];
```

এখানে `dataType` হচ্ছে অ্যারেটি কী ধরনের ডেটা রাখবে তার টাইপ। তারপরে অ্যারের নাম, এবং সবশেষে `size` হচ্ছে অ্যারেতে এলিমেন্টের সংখ্যা।

অ্যারে অ্যাকসেস করতে হলে অ্যারের ইনডেক্স নাম্বারটি নির্দিষ্ট করে বলতে হয়। অ্যারের ইনডেক্সিং শুরু হয় জিরো বা শূন্য থেকে। আসুন আমরা অ্যারে নিয়ে বেসিক একটি প্রোগ্রাম দেখি। এখানে আরা অ্যারে ডিক্লেয়ার করব, সেটিতে ভ্যালু রাখব, প্রিন্ট করব এবং কয়েকটি সাধারণ অপারেশন দেখব।

```
#include<stdio.h>

int main(void)
{

    int age[5] = {18, 30, 50, 47, 57}; //array declaration and assigning values

    printf("%d", age[0]); //Printing the first element
    printf("\n%d", age[3]); //Printing the 4th element

    age[2] = 80; //changing a value

    printf("\n%d", age[2]); //printing the changed value

    printf("\nEnter age of person 1 :");
    scanf("%d", &age[0]); //using scanf to insert a value in the array
    printf("The newly assigned age for person 1 is : %d", age[0]);

    return 0;
}
```

এখানে আমরা একটি অ্যারে ডিক্লেয়ার করে সেটিতে পাঁচটি ভ্যালু অ্যাসাইন করলাম। এরপর প্রথম ভ্যালুটি প্রিন্ট করলাম `printf("%d", age[0]);` দিয়ে। `printf("%d", age[3]);` দিয়ে আমরা চতুর্থ ভ্যালুটি প্রিন্ট করেছি। এরপর আমরা তৃতীয় ভ্যালু যার ইনডেক্স 2 সেটিকে পাশ্চাত্যেছি এবং প্রিন্ট করেছি। তারপরে আমরা `scanf` ব্যবহার করে একটি ভ্যালু নিয়ে প্রথম ইনডেক্সে রেখেছি এবং সেটি প্রিন্ট করেছি।

অ্যারের ভ্যালুগুলোকে অ্যাকসেস করার জন্য `for` লুপ ব্যবহার করা একটি প্রচলিত প্র্যাকটিস। আমরা আরেকটি উদাহরণ দেখব যেখানে আমরা `for` লুপ ব্যবহার করে অ্যারে অ্যাকসেস করছি এবং কয়েকটি সংখ্যার গড় বের করছি।

```
#include<stdio.h>

int main(void)
{

    int expenses[5];
    int i;
    int sum = 0;
    float average = 0;

    printf("Enter your expenses for last 5 days: ");

    for(i=0; i<=4; i++) {
        scanf("%d", &expenses[i]); //assigning all the values
    }

    for (i=0; i<=4; i++) {
        sum = sum + expenses[i];
    }

    average = sum/5;

    printf("Your average expenses for last five days is : %f", average);

    return 0;
}
```

এই প্রোগ্রামটিতে আমরা আমাদের গত পাঁচদিনের গড় ব্যয় বের করছি। আমরা `expenses` নামের একটি অ্যারে এখানে ডিক্লেয়ার করেছি এবং এরপর একটি `for` লুপ ব্যবহার করে `scanf()` দিয়ে আমরা ভ্যালুগুলো অ্যারেতে নিয়েছি। আবার আরেকটি `for` লুপ দিয়ে আমরা `sum` বের করেছি এবং তারপরে `average` বের করে প্রিন্ট করেছি। এখানে একটি বিষয় লক্ষ্যনীয় যে আমরা `for` লুপের ইনডেক্স `i` কে অ্যারের ইনডেক্স হিসেবে ব্যবহার করেছি `expenses[i]` দিয়ে।

মাল্টি ডাইমেনশনাল অ্যারে

মাল্টি ডাইমেনশনাল অ্যারে নামটিই বলে দেয় যে এই অ্যারেগুলোতে একের অধিক ডাইমেনশন বা ইনডেক্সিং আছে। আমাদের সুবিধার জন্য আমরা টু-ডাইমেনশনাল অ্যারে নিয়ে কাজ করব এই অধ্যায়ে। অ্যারেতে এর চাইতে বেশি ডিরেকশনও থাকতে পারে, কিন্তু সেগুলো একটু অ্যাডভান্সড লেভেলের কাজ হয়ে যায়। টু-ডাইমেনশনাল অ্যারেকে আমরা এরকম টেবল বা ম্যাট্রিক্স আকারেও দেখাতে পারি। এবং স্বাভাবিকভাবেই এই অ্যারের ইনডেক্সিংও শুরু হয় ০,০ থেকে। নিচের টেবলটি একটি টু-ডাইমেনশনাল অ্যারের রিপ্রেজেন্টেশন হিসেবে ধরা যায়।

a [0,0]	a[0,1]	a[0,2]
a [1,0]	a [1,1]	a [1,2]
a [2,0]	a [2,1]	a [2,2]
a [3,0]	a [3,1]	a [3,2]

এবার আসুন আমরা দেখি এধরনের অ্যারে নিয়ে আমরা কিভাবে কাজ করব। এই অ্যারে ডিক্লেয়ার করতে হয় সাধারণ অ্যারের মতই। আসুন দেখে নিই কিভাবে মাল্টি ডাইমেনশনাল অ্যারে ডিক্লেয়ার ও ড্যালাু অ্যাসাইন করতে হয়।

```
int a[3][4] = {
    {0, 1, 2, 3} ,    // values for first row, index 0
    {4, 5, 6, 7} ,    // values for second row, index 1
    {8, 9, 10, 11}    // values for third row, index 2
};

int b[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11}; // this works too
```

উপরের কোডগুলোতে আমরা দেখলাম কিভাবে মাল্টি ডাইমেনশনাল অ্যারে বানাতে হয়। এই অ্যারে অ্যাকসেস করার নিয়ম ওয়ান ডাইমেনশনাল অ্যারের মতই। আসুন দেখি for লুপ ব্যবহার করে কিভাবে আমরা এই ধরনের অ্যারে অ্যাকসেস করতে পারি।


```
#include <stdio.h>

int main (void)
{

int a[3][4] = {
    {0, 1, 2, 3} ,    // values for first row, index 0
    {4, 5, 6, 7} ,    // values for second row, index 1
    {8, 9, 10, 11}    // values for third row, index 2
};
int i, j;

// output each array element's value
for ( i = 0; i < 3; i++ )
{
    for ( j = 0; j < 4; j++ )
    {
        printf("a[%d][%d] = %d", i,j, a[i][j] );
    }

    printf("\n");
}
return 0;
}
```

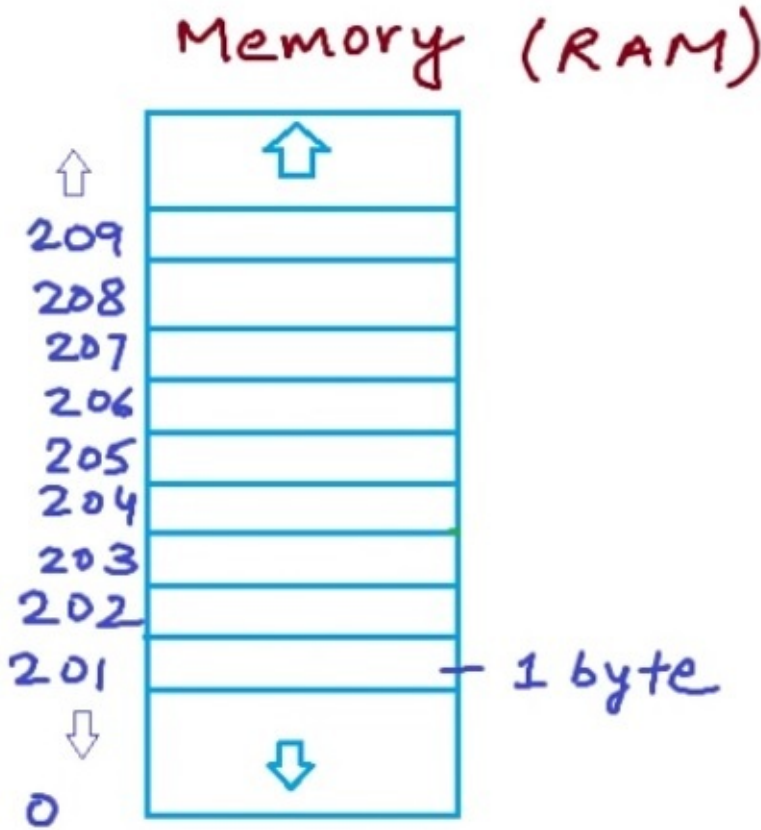
এই উদাহরণে আমরা দেখতে পাচ্ছি আমরা কিভাবে `for` লুপ ব্যবহার করে আমরা টু-ডাইমেনশনাল অ্যারে অ্যাকসেস করছি।

পয়েন্টার কি?

সি ল্যান্ডুয়েজের একটি বেশ শক্তিশালী ফিচার হল পয়েন্টার। পয়েন্টার দিয়ে ফাংশনের ভেতর থেকে ডায়ারিয়েবল বা অ্যারে পাস্টানো, মেমোরি অ্যালোকেশন সহ বিভিন্ন অ্যাডভান্সড কাজ করা যায়। আমরা অ্যারের ব্যবহার দেখেছি আগের অধ্যায়ে। অ্যারের একটি বিষয় হল এটির সাইজ প্রথমেই বলে দিতে হয়, পয়েন্টার ব্যবহার করে আমরা চাইলে ডায়নামিক অ্যারে তৈরি করতে পারি যেটির সাইজ প্রোগ্রাম চলার সময়ে ঠিক হবে এবং সে অনুযায়ী মেমোরি অ্যালোকেশন হবে। পয়েন্টার প্রোগ্রামিং এ দারুন একটি টুল। পয়েন্টার সম্পর্কে জানার আগে কিছু ব্যাসিক জিনিস জানা যাক, যেগুলো বুঝতে কাজে দিবে।

ডায়ারিয়েবল গুলো কিভাবে কম্পিউটার মেমরিতে/র‍্যাম এ স্টোর হয়?

র‍্যাম এর এক একটি সেল এক একটি বাইট। আর প্রত্যেকটা বাইট এর একটি করে এড্রেস রয়েছে। আর প্রতিটা বাইটে ৮টি করে বিট রয়েছে।



আমরা যখন বলি আমাদের র‍্যাম 8 Giga byte, তখন আমাদের কম্পিউটারের র‍্যামে মোট 8 000 000 000 bytes ডেটা স্টোর করা যাবে, এবং এদের প্রত্যেকের একটি করে এড্রেস রয়েছে। প্রথমটি ০ পরের টি ১, এর পরের টি ২ এভাবে বাড়তে থাকে। যদিও কম্পিউটার এ এড্রেস গুলো রিপ্রেজেন্ট করে হেক্সাডেসিমেল নাম্বার সিস্টেমে।

আমরা যখন একটি ভ্যারিয়েবল ডিক্লেয়ার করার পর যখন প্রোগ্রামটি এক্সিকিউট/রান করি তখন কম্পিউটার ঐ ভ্যারিয়েবল এর জন্য কিছু মেমরি এলোকেট করে। কত বাইট মেমরি এলোকেট করবে, তা নির্ভর করে ঐ ভ্যারিয়েবল এর ডেটা টাইপ এবং কম্পাইলার এর উপর।

সাধারনত কম্পাইলার গুলো একটা int এর জন্য 2 byte মেমরি এলোকেট করে। তেমনি একটি char ভ্যারিয়েবলের জন্য 1 byte মেমরি এলোকেট করে। floating-point নাম্বার এর জন্য 4 byte মেমরি এলোকেট করে।

যেমন যখন কম্পিউটার দেখে এমন একটি ডিক্লোরেশন int a; তখন এটি বুঝতে পারে এটি একটি ইন্টিজার ভ্যারিয়েবল এবং এর জন্য ২ বাইট মেমরি এলোকেট করা দরকার। তখন র‍্যাম এর খালি যায়গা থেকে এটি এই ইন্টিজারের জন্য ২ বাইট মেমরি এলোকেট করে।

আমরা সহজেই একটি ভ্যারিয়েবলের মেমরি লোকেশন বের করতে পারি, নিচের প্রোগ্রামটি দেখা যাকঃ

```
#include <stdio.h>
int main()
{
    int a =5;
    printf("Memory address of variable a is: %x",&a);
    return 0;
}
```

উপরের প্রোগ্রামটি রান করালে এমন কিছু দেখাবেঃ Memory address of variable a is: 2686732 । এক কম্পিউটারে এক এক মান দেখাবে। এবং একবার এক এক ভ্যালু দেখাবে। কারণ যতবারই আমরা প্রোগ্রামটি রান করি, প্রতিবারই ভ্যারিয়েবলটির জন্য মেমরিতে একটা জায়গা বরাদ্দ করা হয়। আর ঐ জায়গার এড্রেসটা প্রতিবারই পরিবর্তন হয়।

কোন ভ্যারিয়েবল এর এর মেমরি এড্রেস জানার জন্য & [ampersend] ব্যবহার করা হয়। যাকে address-of operator [&] ও বলা হয়। যা দিয়ে আমরা অন্য একটি ভ্যারিয়েবল এর এড্রেস বা মেমরি লোকেশন পেতে পারি।

যখন আমরা প্রোগ্রামটি রান করি, তখন কম্পিউটার র‍্যাম এর খালি যায়গা থেকে ভ্যারিয়েবল a এর জন্য ২ বাইট মেমরি এলোকেট করে। কম্পিউটার অটোমেটিকেলি তখন a এর জন্য 2686732 এবং 2686733 নং সেল এলোকেট করে রাখে। আর মেমরি এড্রেস জানার জন্য শুধু মাত্র শুরুর এড্রেস জানলেই হয়। আমরা যখন a এর মেমরি এড্রেস প্রিন্ট করেছি, তখন শুধু শুরুর এড্রেস 2686732 ই পেয়েছি। যদি ও a ভ্যারিয়েবল এর জন্য 2686732 এবং 2686733 মেমরি এলোকেট করা হয়েছে এবং এর মান 5 এই দুই সেলে স্টোর করে রাখা হয়েছে। এখন আমরা যদি a এর মান পরিবর্তন করে অন্য আরেকটা ভ্যালু রাখি, যেমন 8, তখন র‍্যামের 2686732 এবং 2686733 এ দুটো সেল এর মান ও পরিবর্তন হয়ে যাবে এবং এ দুটো সেলে 5 এর পরিবর্তে 8 স্টোর হবে। এবার পয়েন্টার কি জানা যাক।

পয়েন্টার হচ্ছে একটা ভ্যারিয়েবল যার ভ্যালু হচ্ছে আরেকটি ভ্যারিয়েবল এর মেমরি লোকেশন। পয়েন্টার একটা ডেটা, অ্যারে বা ভ্যারিয়েবল এর কম্পিউটার মেমরি লোকেশন রিপ্রেজেন্ট করে বা পয়েন্ট করে। অন্যান্য ভ্যারিয়েবল এর মত পয়েন্টার ভ্যারিয়েবল ব্যবহার করার আগে কম্পিউটার/ কম্পাইলারকে বলতে হবে এটা একটি পয়েন্টার ভ্যারিয়েবল। নিচের মত করে একটি পয়েন্টার ভ্যারিয়েবল ডিক্লেয়ার করে।

```
data_type *name;
```

যেমন integer পয়েন্টারের জন্যঃ `int *i;`

asterisk [*] একটি ভ্যারিয়েবলের আগে ব্যবহার করে পয়েন্টার হিসেবে ডিক্লেয়ার করা হয়। যাকে indirection operator বা value-at-address operator বলা হয়। এখানে আরো কিছু ডেটা টাইপ এর পয়েন্টার ডিক্লেয়ারেশন এর উদাহরন দেওয়া হলোঃ

```
int*ip; /* pointer to an integer */
double*dp; /* pointer to a double */
float*fp; /* pointer to a float */
char*ch /* pointer to a character */
```

আমরা এখন দেখব কিভাবে পয়েন্টার ব্যবহার করতে হয় একটি প্রোগ্রামে।

```
#include <stdio.h>
int main ()
{
    int a = 5; /* variable declaration */
    int *ip; /* pointer variable declaration */
    ip = &a; /* store address of "a" in pointer variable*/
    printf("Address of a variable: %xn", &a );
    /* address stored in pointer variable */
    printf("Address stored in ip variable: %xn", ip );
    return 0;
}
```

এখানে আমরা একটি ভ্যারিয়েবল `a` ডিক্লেয়ার করেছি। এরপর একটি পয়েন্টার ভ্যারিয়েবল ডিক্লেয়ার করেছি। তারপর পয়েন্টার ভ্যারিয়েবলে `a` এর মেমরি এড্রেস রেখেছি। তারপর `&` অপারেটর দিয়ে `a` ভ্যারিয়েবল এর এড্রেস প্রিন্ট করে দেখলাম। এবং পয়েন্টার ভ্যারিয়েবল এর ভ্যালু প্রিন্ট করে দেখলাম। উভয় এর মান ই একই।

আমরা ইচ্ছে করলে এখন `ip` পয়েন্টার ভ্যারিয়েবল দিয়ে `a` এর মান বের করতে পারি।

```
#include <stdio.h>
int main ()
{
    int a = 5;
    int *ip;
    ip = &a;
    /* access the value using the pointer */
    printf("Value of *ip variable: %dn", *ip );
    return 0;
}
```

আমরা যখন প্রগ্রামটি রান করব, তখন ip যে ভ্যারিয়েবলটির এড্রেস শো করবে, তার মান প্রিন্ট করবে। লক্ষ্যকরি, যখন আমরা পয়েন্টার ভ্যারিয়েবল দিয়ে কোন ভ্যারিয়েবল এর এড্রেস বের করতে চাইবো, তখন শুধু পয়েন্টার ভ্যারিয়েবল লিখলেই হবে। কিন্তু যখন আমরা পয়েন্টার ভ্যারিয়েবল দিয়ে মূল ভ্যারিয়েবল এর ভ্যালু বের করতে চাইবো, তখন পয়েন্টার ভ্যারিয়েবল এর আগে * যোগ করতে হবে। যেমন প্রথম প্রোগ্রামে আমরা ip [পয়েন্টার ভ্যারিয়েবল] প্রিন্ট করায় আমরা এড্রেস পেয়েছি। এবং পরের প্রোগ্রামে ip এর আগে একটা দিয়ে ip প্রিন্ট করায় আমরা মূল ভ্যারিয়েবলের মান পেয়েছি।

স্ট্রাকচার – struct

Structures সি প্রোগ্রামিং এর দারুণ একটা বিষয়। আমরা ডেটা টাইপ সম্পর্কে জানি, int, char, float ইত্যাদি। Structures দিয়ে আমরা নিজেদের মত করে ডেটা স্ট্রাকচার তৈরি করে নিতে পারি। যেমন অ্যারে হচ্ছে একটা ডেটা স্ট্রাকচার। যেখানে শুধু আমরা একই ডেটা টাইপ এর ডেটা রাখতে পারি। কিন্তু স্ট্রাকচার তৈরি করে আমরা এক সাথে int, char, float ইত্যাদি ভিন্ন ভিন্ন ডেটা এক সাথে রাখতে পারি। নিজেদের প্রয়োজন মত, ইচ্ছে মত।

int অ্যারেতে শুধু ইন্টিজার ভ্যালুই রাখতে পারব। char অ্যারেতে শুধু কারেকটারই রাখতে পারব। কিন্তু Structures ব্যবহার করে আমরা নিজেদের মত করে ডেটা স্ট্রাকচার তৈরি করতে পারব। এবং এখানে ইচ্ছে মত একের অধিক ভিন্ন ভিন্ন ডেটা রাখতে পারব।

Structures নিচের মত করে ডিফাইন করা হয়।

```
struct book
{
    int no;
    char name;
};
```

struct কীওয়ার্ড দিয়ে Structures ডিফাইন করা হয়। এরপর লিখছি book, যা হচ্ছে আমাদের নিজস্ব স্ট্রাকচারের নাম। এরপর দ্বিতীয় ব্র্যাকেটের মধ্যে স্ট্রাকচারের মেম্বার গুলো। এখানে একটা মেম্বার হচ্ছে ইন্টিজার, যেখানে আমরা বই এর ক্রমিক নাম্বার রাখব। আরেকটা হচ্ছে কারেকটার, যেখানে আমরা বইটির নাম রাখব।

Structures ডিফাইন করার পর তা ব্যবহার করার জন্য ডিক্লেয়ার করতে হয়। একটা ইন্টিজার ভ্যারিয়েবল ব্যবহারের জন্য যেমন আগে তা ডিক্লেয়ার করতে হয়, তেমনি।

ডিক্লেয়ার করার জন্য নিচের মত করে লিখতে হয়: **struct book myBook;**

যেখানে struct দিয়ে বুঝায় আমরা একটা স্ট্রাকচার ডিক্লেয়ার করতে যাচ্ছি, এরপর book, যা দিয়ে বুঝাচ্ছে আমরা কোন struct টা ডিক্লেয়ার করতে যাচ্ছি। এরপর হচ্ছে আমরা কি নামে আমাদের তৈরি স্ট্রাকচারটা ব্যবহার করব।

ডিক্লেয়ার করার পর স্ট্রাকচার ব্যবহার করতে হবে। তো আমরা আমাদের তৈরি myBook স্ট্রাকচার ব্যবহার করব।

myBook এর দুইটা মেম্বার। একটা হচ্ছে no আরেকটা name.

এখন আমরা বই এর নাম্বার এবং নাম সেট করব। তার জন্য লিখতে হবে: **myBook.no = 3; myBook.name = 'C';**

এবার আমাদের সেট করা বই এর নাম এবং নং প্রিন্ট করতে চাইলে:

```
printf( "Book No : %dn", myBook.no);
printf( "Book Name : %cn", myBook.name);
```

আমরা ছোট ছোট কোড লিখেছি, এবার পুরো প্রোগ্রামটি লিখে ফেলিঃ

```

1  #include <stdio.h>
2
3  struct book{
4      int no;
5      char name;
6  };
7
8  int main ()
9  {
10     struct book myBook;
11     myBook.no = 3;
12     myBook.name = 'C';
13
14     printf( "Book No : %d\n", myBook.no);
15     printf( "Book Name : %c\n", myBook.name);
16
17     return 0;
18 }
```

struc hosted with ♥ by GitHub

[view raw](#)

typedef ব্যবহার করে আমরা আমাদের স্ট্রাকচারের instanc তৈরি করার সময় struct কীওয়ার্ড ব্যবহার ছাড়াই আমাদের তৈরি স্ট্রাকচার ব্যবহার করতে পারি। তার জন্য আমাদের উপরের স্ট্রাকচারটা নিচের মত করে লিখতে হবেঃ

```

typedef struct {
    int no;
    char name;
} book;
```

আগের থেকে পার্থক্য হচ্ছে আমরা এখানে নতুন একটা কীওয়ার্ড ব্যবহার করেছি, typedef। তারপর লিখছি struct। এবং আমাদের স্ট্রাকচারের একবারে শেষের দিকে লিখেছি আমাদের স্ট্রাকচারের নাম।

এখন আমরা আমাদের এই book স্ট্রাকচারের instance তৈরি করার জন্য struct কীওয়ার্ড ব্যবহার ছাড়াই তৈরি করতে পারব, যেমনঃ book myBook;

যেভাবে আমরা একটা ভ্যারিয়েবল ডিক্লেয়ার করি, ঠিক সেভাবে। আগের প্রোগ্রামটা **typedef** ব্যবহার করলে হয়ঃ

```

1  #include <stdio.h>
2
3  typedef struct book{
4      int no;
5      char name;
6  } book;
7
8  int main ()
```

```

9  {
10     book myBook;
11     myBook.no = 3;
12     myBook.name = 'C';
13
14     printf( "Book No : %d\n", myBook.no);
15     printf( "Book Name : %c\n", myBook.name);
16
17     return 0;
18 }

```

struct_c2 hosted with ❤ by GitHub

[view raw](#)

আচ্ছা, স্ট্রাকচার এর সুবিধে হচ্ছে একবার ডিফাইন করার পর ইচ্ছে মত instance তৈরি করে নিতে পারি আমরা। আগের দুইটি প্রোগ্রামে আমরা একটা স্ট্রাকচার ডিফাইন করেছি, এবং এরপর একটা মাত্র স্ট্রাকচার ডিক্লেয়ার করেছি। এখন আমরা আরেকটা প্রোগ্রাম লিখব, যেখানে একটা স্ট্রাকচারের তিনটে instance তৈরি করব।

```

1  #include <stdio.h>
2
3  typedef struct book{
4      int no;
5      char name;
6  } book;
7
8  int main ()
9  {
10     book book1;
11     book book2;
12     book book3;
13
14     book1.no = 1;
15     book1.name = 'A';
16
17
18     book2.no = 2;
19     book2.name = 'B';
20
21
22     book3.no = 3;
23     book3.name = 'C';
24
25     printf( "Book No : %d\n", book1.no);
26     printf( "Book Name : %c\n \n", book1.name);
27
28     printf( "Book No : %d\n", book2.no);
29     printf( "Book Name : %c\n \n", book2.name);

```



```

30
31     printf( "Book No : %d\n", book3.no);
32     printf( "Book Name : %c\n", book3.name);
33
34
35
36
37     return 0;
38 }

```

struct_c3 hosted with ❤ by GitHub

[view raw](#)

book এর instance তৈরি করার সময় আমরা ভিন্ন ভিন্ন লাইনে না লিখে একই লাইনে লিখতে পারি। যেমনঃ

```

book book1;
book book2;
book book3;

```

এর পরিবর্তে আমরা লিখতে পারিঃ **book book1,book2, book3;**

আমরা book এর তিনটে instance তৈরি করেছি মাত্র। এবং ম্যানুয়ালি সব গুলো মেম্বারে ডেটা সেট করেছি। কিন্তু আমাদের এমন প্রোগ্রাম লিখতে হবে, যেখানে আমাদের ১০০ বা ১০০০ বা আরো বেশি instance তৈরি করতে হবে। তখন কি করব? এমন প্রোগ্রামটা দেখতে কি বিশিষ্ট না দেখাবে, তাই না? কিন্তু না, আমরা এখানে কন্ডিশনাল ব্যবহার করতে পারব। লুপ দিয়ে সব গুলোতে ডেটা সেট করতে পারব। আবার লুপ দিয়ে সব গুলো থেকে ডেটা বের করে প্রিন্ট করতে পারব।

আমরা সিম্পল একটা প্রোগ্রাম লিখব এ জন্য। আসলে এ প্রোগ্রামটির জন্য স্ট্রাকচার ব্যবহার করতে হয় না। তারপর ও লুপ ব্যবহার করে কিভাবে স্ট্রাকচারের ভিন্ন ভিন্ন মেম্বার এক্সেস করতে হয়, তার একটা উদাহরণ দেখব।

প্রোগ্রামটিতে আমরা ০ থেকে ১০০ এর বর্গমূল [square root] এর একটা চার্ট তৈরি করব।

```

1  #include <stdio.h>
2
3  typedef struct squareRoot{
4      int number[100];
5      double root[100];
6      } squareRoot;
7
8  int main ()
9
10 {
11     int i =0;
12     squareRoot squareRoot1;
13
14     // setting data
15     for (i =0; i<=100; i++){

```

```
16
17     squareRoot1.number[i] = i;
18     squareRoot1.root[i] = sqrt(i);
19 }
20
21 // printing data
22
23 for (i =0; i<=100; i++){
24     printf( "Square Root of %d ", squareRoot1.number[i]);
25     printf( "is : %f\n", squareRoot1.root[i]);
26 }
27
28     return 0;
29 }
```

struct_c4 hosted with ♥ by GitHub

[view raw](#)

এখানে `sqrt()` হচ্ছে একটা লাইব্রেরী ফাংশান। যার মধ্যে একটা নাম্বার দিলে ঐ নাম্বারটির বর্গমূল রিটার্ন করে। বাকি কোড গুলো তো সহজ। প্রথমে আমরা `for` লুপ ব্যবহার করে আমাদের `squareRoot1` এর বিভিন্ন মেম্বারে ডেটা সেট করেছি। এপর আবার লুপ দিয়ে সেগুলো প্রিন্ট করেছি। এবার তো আমরা স্ট্রাকচার ব্যবহার করে কমপ্লেক্স কোড লিখতে পারব, তাই না? শুভ প্রোগ্রামিং...

4.2. ফাইল থেকে ইনপুট এবং আউটপুট

একটা ফাইল নিয়ে কাজ করার জন্য তা ডিক্লেয়ার করতে হয়। ডিক্লেয়ার করা হয় FILE পয়েন্টার দিয়ে। যেমনঃ

FILE *MyFile;

FILE বড় হারের অক্ষরে লিখতে হয় এবং MyFile হচ্ছে পয়েন্টার ডেরিয়েবল। এটা মূলত একটা বাফার তৈরি করে কম্পিউটার মেমরি এবং ঐ ফাইল এর মধ্যে। পয়েন্টার ডেরিয়েবল তৈরি করার পর আমরা ফাইলটি ওপেন করতে পারব। তার জন্য fopen ফাংশান ব্যবহার করতে হয়। যা সাধারনত লেখা হয় এমনঃ MyFile = fopen(file-name, file-type); । ফাইল এর নাম এবং টাইফ দুটি স্টিং। এবং ফাইল টাইফ হচ্ছে ফাইলটা কোন মুড এ ওপেন হবে তা। যেমন Read Only, Write Only অথবা দুটিই ইত্যাদি। যেমনঃ

MyFile = fopen ("myfile.txt","w");

file-type নিচের টেবিলের যে কোন একটা হতে পারেঃ

r	শুধু মাত্র ফাইলটি রিড করার জন্য ওপেন করা।
w	ফাইলটিতে কিছু রাইট করার জন্য ওপেন করা। ফাইলে যদি আগে কিছু থাকে তাহলে তা মুছে যাবে। ফাইল যদি না থাকে, তাহলে অটোমেটিক ভাবে তৈরি হবে।
a	ফাইলের শেষে কিছু রাইট করার জন্য ওপেন করা। যদি ফাইলে আগে কিছু থাকে, তাহলে তার শেষে রাইট হবে। ফাইল যদি না থাকে, তাহলে অটোমেটিক ভাবে তৈরি হবে।
r+	ফাইলটি রিড করা বা রাইট করার জন্য ওপেন করা। এবং ফাইলের শুরুতে কিছু লেখা।
w+	ফাইলটি রিড করা বা রাইট করার জন্য ওপেন করা।
a+	ফাইলটি রিড করা বা রাইট করার জন্য ওপেন করা। এবং ফাইলের শেষে কিছু লেখা।

ফাইল ওপেন করার জন্য তাতে কিছু লেখার জন্য fputs ফাংশান ব্যবহার করা হয়। যেমনঃ

fputs ("Writing to a file using 'fopen' example.",MyFile);

ফাইলটি রিড বা রাইট করা হলে ফাইলোটি বন্ধ বা ক্লোজ করতে হয়, তার জন্য ব্যবহার করা হয় fclose ফাংশানঃ

fclose (MyFile);

```
#include <stdio.h>

int main ()
{
    FILE * MyFile;
    MyFile = fopen ("myfile.txt", "w");
    if (MyFile!=NULL)
    {
        fputs ("Writing to a file using 'fopen' example.", MyFile);
        fclose (MyFile);
    }
    return 0;
}
```

ফাইল থেকে ডেটা পড়াঃ

ফাইল থেকে ডেটা পড়ার জন্য fscanf ব্যবহার করা হয়। নিচের কোডগুলো দেখুনঃ

```
#include <stdio.h>
int main ()
{
    FILE * MyFile;
    char string[10];
    MyFile = fopen ("myfile.txt", "r+");

    while(! feof(MyFile))
    {
        fscanf(MyFile, "%s", string);

        printf("%s ", &string);
    }
    fclose (MyFile);
    return 0;
}
```

এখানে feof দিয়ে end-of-file চেক করা হয়েছে। অর্থাৎ যতক্ষণ পর্যন্ত ফাইলের মধ্যে কোন ডেটা থাকবে ততক্ষণ পর্যন্ত ফাইলটির ডেটা গুলো fscanf দিয়ে রিড করা হবে। fscanf এর তিনটা প্যারামিটার রয়েছে। fscanf(file-name, data-type, variable);

file-name হচ্ছে ফাইলের নাম। যে ফাইল থেকে ডেটা পড়া হবে। data-type হচ্ছে ফাইলের ডেটা টাইপ। বা কোন টাইপে ডেটা গুলো পড়া হবে। এখানে char টাইপের ডেটা পড়া হয়েছে। ইচ্ছ করলে int অথবা floating point ডেটা পড়া যাবে।

variable হচ্ছে ভেরিয়েবলের নাম, যেখানে ডেটা গুলো ফাইল থেকে পড়ে সংরক্ষিত থাকবে।

আমরা এ পর্যন্ত যে ফোল্ডারে আমাদের সি প্রোগ্রাম টা রয়েছে তা থাকে ফাইলটি ওপেন করছি বা রিড করছি। ইচ্ছে করলে আমরা যে কোন ডিরেক্টরি থেকে ফাইলটি ওপেন করতে পারি। file-name এর জাগায় পুরো ফাইল পাথ দিলেই হবে। নিচের উদাহরণটি দেখুনঃ

```
#include <stdio.h>

int main ()
{
    FILE * MyFile;
    char string[10];
    MyFile = fopen ("myfile.txt", "r+");

    while(! feof(MyFile))
    {
        fscanf(MyFile, "%s", string);
        printf("%s ", &string);
    }
    fclose (MyFile);
    return 0;
}
```

ফাইল নিয়ে কাজ করা অনেক সহজ তাই না?

টেবিলে আগেই বলছি যে w মুডে ফাইলটি ওপেন করলে তার মধ্যের সকল ডেটা মুছে যাবে এবং নতুন করে ডেটা রাইট হবে। কিন্তু আমরা যদি আগের ডেটা না মুছে আগের ডেটার নিচে নতুন ডেটা লিখতে চাই তার জন্য ব্যবহার করব a মুড।

নিচের কোড টি কয়েকবার রান করিয়ে দেখুনঃ

```
#include <stdio.h>

int main ()
{
    FILE * MyFile;
    MyFile = fopen ("F:\MyFolder\myfile.txt", "a");
    if (MyFile!=NULL)
    {
        fputs ("Writing to a file using 'fopen' example.", MyFile);
        fclose (MyFile);
    }
    return 0;
}
```

এতটুকুই, অন্যান্য মুড আপনারা টাই করে দেখুন। অন্য কোন ডেটা টাইফ সংরক্ষন করে দেখুন। অন্য ডেটা টাইফের ডেটা গুলো আবার রিড করার চেষ্টা করুন।

