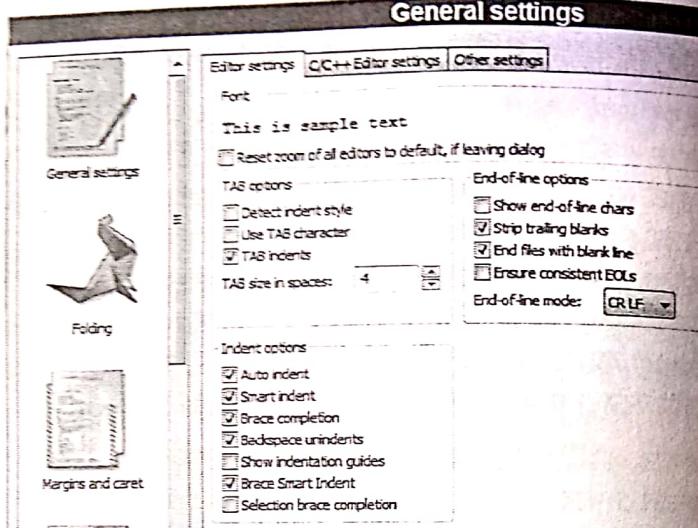


Editor-এ ক্লিক করে TAB Options-এ TAB indents চেক করো  
এবং TAB size in spaces 4 নাও (১.১৯ ছবিটি দেখো)।



ছবি ১. ১৯

এবারে তোমাদের জন্য একটি কাজ। একটি প্রোগ্রাম লেখো যেটি  
স্ক্রিনে প্রিন্ট করবে : I love my country, Bangladesh।

প্রোগ্রামটি টাইপ করার পরে অবশ্যই কম্পাইল ও রান করবে।  
কম্পাইল করার আগে সেভ করতে ভুলবে না।

## অধ্যায় দুই : ডাটা টাইপ, ইনপুট ও আউটপুট

এ অধ্যায়ে আমরা কিছু ছোট ছোট প্রোগ্রাম লিখব। সবগুলো প্রোগ্রাম  
অবশ্যই কম্পিউটারে চালিয়ে দেখবে এবং একটু পরিবর্তন করে  
কম্পাইল ও রান করার চেষ্টা করবে।

আমাদের প্রথম প্রোগ্রামটি হবে দুটি সংখ্যা যোগ করার প্রোগ্রাম।  
এখন কথা হচ্ছে, সংখ্যাগুলো তো কম্পিউটারের মেমোরিতে রাখতে  
হবে, সেই জটিল কাজটি কীভাবে করব? চিন্তা নেই! সব প্রোগ্রামিং  
ল্যাঙ্গুয়েজে ভেরিয়েবল বলে একটি জিনিস আছে, যেটি কোনো নির্দিষ্ট  
মান ধারণ করার জন্য ব্যবহার করা হয়। ভেরিয়েবলের একটি নাম  
দিতে হয়, তারপর ভেরিয়েবল = কোনো মান লিখে দিলে  
ভেরিয়েবলের ভেতর সেটি ঢুকে যায়। এটির সঙ্গে গাণিতিক  
সমীকরণের কিন্তু কোনো সম্পর্ক নেই।

চলো, প্রোগ্রামটি লিখে রান করাই, তারপর ব্যাখ্যা করা যাবে।

```
#include <stdio.h>
```

```
int main()
```

```
{
    int a;
    int b;
    int sum;
    a = 50;
    b = 60;
    sum = a + b;
    printf("Sum is %d", sum);
    return 0;
}
```

প্রোগ্রাম : ২.১

প্রোগ্রামটি রান করাও, তুমি স্ক্রিনে দেখবে: Sum is 110।

এখানে a, b, sum তিনটি ভেরিয়েবল (variable) আলাদা সংখ্যা ধারণ করে। প্রথমে আমাদের বলে দিতে হবে যে a, b, sum নামে সেটিও বলে দিতে হবে। int a; দিয়ে আমরা কম্পাইলারকে বলছি a নামে একটি ভেরিয়েবল এই প্রোগ্রামে আছে যেটি একটি পূর্ণসংখ্যা (integer)-এর মান ধারণ করার জন্য ব্যবহার করা হবে। এই কাজটিকে বলে ভেরিয়েবল ডিক্লারেশন। আর int হচ্ছে ডাটা টাইপ, যেটি দেখে সি-এর কম্পাইলার বুঝবে যে এতে ইন্টিজার টাইপ ডাটা থাকবে। আরও বেশ কিছু ডাটা টাইপ আছে, সেগুলো আমরা আন্তে আন্তে দেখব। আমরা চাইলে একই টাইপের ভেরিয়েবলগুলো ডিক্লেয়ার করার সময় আলাদা লাইনে না লিখে একসঙ্গে কমা দিয়েও নিখতে পারতাম, যেমন: int a, b, sum;। আর লক্ষ করো যে ভেরিয়েবল ডিক্লারেশনের শেষে সেমিকোলন ব্যবহার করতে হয়।

এরপর আমি দুটি স্টেটমেন্ট লিখেছি:

a = 50;

b = 60;

এখানে a-এর মান 50 আর b-এর মান 60 বলে দিলাম (assign করলাম), যতক্ষণ না এটি আমরা পরিবর্তন করছি, কম্পাইলার a-এর মান 50 আর b-এর মান 60 ধরবে।

পরের স্টেটমেন্ট হচ্ছে: sum = a + b;। এতে বোঝায়, sum-এর মান হবে a + b-এর সমান, অর্থাৎ a ও b-এর যোগফল যে সংখ্যাটি হবে সেটি আমরা sum নামের ভেরিয়েবলে রেখে দিলাম (বা assign করলাম)।

এবারে যোগফলটি মনিটরে দেখাতে হবে, তাই আমরা printf ফাংশন ব্যবহার করব।

printf("Sum is %d", sum);

এখানে দেখো printf ফাংশনের ভেতরে দুটি অংশ। প্রথম অংশে "Sum is %d" দিয়ে বোঝানো হয়েছে ক্রিনে প্রিন্ট করতে হবে

Sum is এবং তার পরে একটি ইন্টিজার ভেরিয়েবলের মান যেটি %d-এর জায়গায় বসবে। তারপর কমা দিয়ে আমরা sum লিখে বুঝিয়ে দিয়েছি যে %d-তে sum-এর মান প্রিন্ট করতে হবে। ইন্টিজারের জন্য যেমন %d ব্যবহার করলাম, অন্য ধরনের ভেরিয়েবলের জন্য অন্য কিছু লিখতে হবে, যেটি আমরা ব্যবহার করতে করতে শিখব। খুব ভালো হতো যদি আমি এখন একটি চার্ট লিখে দিতাম যে সি ল্যাঙ্গুয়েজে কী কী ডাটা টাইপ আছে, সেগুলো কী দিয়ে লেখে এবং প্রিন্ট করার জন্য কী করতে হবে আর তোমরা সেই চার্ট মুখস্থ করে ফেলতে। কিন্তু শুধু শুধু মুখস্থ করার কোনো দরকার নেই, মুখস্থ করার প্রবণতা চিন্তাশক্তি কমায় আর প্রোগ্রামারদের জন্য চিন্তা করার ক্ষমতা খুবই গুরুত্বপূর্ণ।

আমরা ওপরের প্রোগ্রামটি চাইলে এভাবেও লিখতে পারতাম:

```
#include <stdio.h>
int main()
{
    int a, b, sum;√
    a = 50;√
    b = 60;√
    sum = a + b;√
    printf("Sum is %d", sum);
    return 0;
}
```

প্রোগ্রাম : ২.২

আবার ভেরিয়েবল ডিক্লেয়ার করার সময় একই সঙ্গে অ্যাসাইনও করা যায় :

```
#include <stdio.h>
int main()
{
}
```

```

int a = 50, b = 60, sum;
sum = a + b;
printf("Sum is %d", sum);
return 0;
}

```

প্রোগ্রাম : ২.৩

এখন তোমাদের জন্য একটি প্রশ্ন। নিচের প্রোগ্রামটির আউটপুট কী হবে?

```

#include <stdio.h>
int main()
{
    int x, y;
    x = 1;
    y = x;
    x = 2;
    printf("%d", y);
    return 0;
}

```

প্রোগ্রাম : ২.৪

কী মনে হয়? আউটপুট 1 নাকি 2? আউটপুট হবে 1, কারণ প্রথমে কম্পাইলার দেখবে, x-এর মান 1 অ্যাসাইন করা হয়েছে ( $x = 1$ )। তারপর x-এর মানটি আবার y-এ অ্যাসাইন করা হয়েছে ( $y = x$ )। এখন y-এর মান 1। তারপর আবার x-এর মান 2 অ্যাসাইন করা হয়েছে। কিন্তু তাতে y-এর মানের কোনো পরিবর্তন হবে না। প্রোগ্রামিংয়ে  $y = x$ ; আসলে কোনো সমীকরণ নয় যে এটি সব সময় সত্য হবে। '=' চিহ্ন দিয়ে একটি ভেরিয়েবলে নির্দিষ্ট কোনো মান রাখা হয়।

এবারে নিচের প্রোগ্রামটি দেখো :

```

#include <stdio.h>
int main()
{
    int a = 50, b = 60, sum;
    sum = a + b;
    printf("%d + %d = %d", a, b, sum);
    return 0;
}

```

প্রোগ্রাম : ২.৫

প্রোগ্রামটি মনিটরে কী প্রিন্ট করে? চালিয়ে দেখো।  $\text{printf}("%d + %d = %d", a, b, sum);$  না লিখে  $\text{printf}("%d + %d = %d", b, a, sum);$  লিখে প্রোগ্রামটি আবার চালাও। এখন জিনিসটি চিন্তা করে বুঝে নাও।

লেখাপড়া করার সময় আমাদের মনে নানা বিষয়ে নানা প্রশ্ন আসে, যার উত্তর আমরা বইতে খুঁজি, শিক্ষককে জিজ্ঞাসা করি, ইন্টারনেটে খুঁজি বা চিন্তা করে যুক্তি দাঁড় করিয়ে উত্তরটি বের করি। আমাদের দুর্ভাগ্য যে বেশিরভাগ ছেলেমেয়েই শেষ কাজটি করে না, কারণ নিজে নিজে চিন্তা করতে একটু সময় লাগে ও পরিশ্রম হয়, সেই সময় আর শ্রম তারা দিতে চায় না। আর আমাদের অভিভাবক, শিক্ষক ও শিক্ষাব্যবস্থা চিন্তা করার জন্য কোনো পুরস্কার দেয় না, বরং মুখস্ত করার জন্যই পুরস্কৃত করে।

যা-ই হোক, প্রোগ্রামিংয়ের ব্যাপারে যখনই মনে কোনো প্রশ্ন আসবে, সঙ্গে সঙ্গে একটি প্রোগ্রাম লিখে ফেলবে। দেখো তোমার কম্পাইলার কী বলে। ধরা যাক, আমরা যদি  $\text{int}$  টাইপের ভেরিয়েবলে দশমিকযুক্ত সংখ্যা (বাস্তব সংখ্যা বা real number) ব্যবহার করতাম, তাহলে কী হতো?

```

#include <stdio.h>
int main()
{
    int a = 50.45, b = 60, sum;
    sum = a + b;
    printf("%d + %d = %d", a, b, sum);
    return 0;
}

```

প্রোগ্রাম : ২.৬

এখানে a-এর মান 50.45 ব্যবহার করলাম। এবারে প্রোগ্রাম চালাও, দেখো কী হয়। আবার মনে যদি প্রশ্ন আসে যে, main ফাংশনের শেষ লাইনে return 0; না লিখলে কী হয়? তাহলে return 0; ছাড়া প্রোগ্রাম চালিয়ে দেখো কী হয়।

আউটপুট হবে :  $50 + 60 = 110$

সি কম্পাইলার a-এর মান 50 ধরেছে, যদিও আমরা 50.45 অ্যাসাইন করেছি। এই ব্যাপারটিকে বলে টাইপ কাস্ট (type cast)। বাস্তব সংখ্যা রাখার জন্য সি ভাষায় double নামের ডাটা টাইপ রয়েছে। টাইপ কাস্ট করে double সংখ্যাটিকে int-এ নেওয়া হয়েছে, এটি অটোমেটিক হয়। আবার কম্পাইলারকে বলেও দেওয়া যায় : int a = (int) 50.45।

int a = 50.99; এখানে a-এর মান হবে 50। int a = -50.9; লিখলে a-এর মান হয় -50। এক কথায় বললে double থেকে int-এ টাইপ কাস্ট করলে দশমিকের পরের অংশটি বাদ পড়ে যাবে।

আরেকটি কথা। যেই ভেরিয়েবলকে টাইপ কাস্ট করা হচ্ছে, তার মান কিন্তু পরিবর্তন হয় না। টাইপ কাস্ট করা মানটি একটি ভেরিয়েবলে রাখা যায়। নিচের প্রোগ্রামটি কম্পিউটারে চালালেই বুঝতে পারবে।

```

#include <stdio.h>
int main()

```

```

{
    int n;
    double x;
    x = 10.5;
    n = (int)x;
    printf("Value of n is %d\n", n);
    printf("Value of x is %lf\n", x);
    return 0;
}

```

প্রোগ্রাম : ২.৭

প্রোগ্রামের আউটপুট দেখো। x-এর মান কিন্তু পরিবর্তন হয়নি। আর বুঝতেই পারছ যে বাস্তব সংখ্যা রাখার জন্য সি-তে যে double টাইপের ভেরিয়েবল ব্যবহার করা হয়, তা প্রিন্ট করার সময় %lf (। এখানে ইংরেজি ছোট হাতের L) ব্যবহার করতে হয়।

int ডাটা টাইপে তো কেবল পূর্ণ সংখ্যা রাখা যায়। কিন্তু সেটি কি যেকোনো পূর্ণসংখ্যা? উভয়ের জন্য একটি প্রোগ্রাম লিখি:

```

#include <stdio.h>
int main()
{
    int a;
    a = 1000;
    printf("Value of a is %d", a);
    a = -21000;
    printf("Value of a is %d", a);
    a = 10000000;
    printf("Value of a is %d", a);
    a = -10000000;
    printf("Value of a is %d", a);
}

```

৩৫

```

a = 100020004000503;
printf("Value of a is %d", a);
a = -4325987632;
printf("Value of a is %d", a);
return 0;
}

```

প্রোগ্রাম : ২.৮

এখানে আমরা a-তে বিভিন্ন সংখ্যা অ্যাসাইন করলাম। সব মান কি ঠিকঠাক আসছে? আসেনি। কেন আসেনি সেটি ব্যাখ্যা করার আগে একটি কথা বলে নিই। পরপর এতগুলো printf-এর কারণে তোমার কম্পিউটারের ক্রিনে নিচয়ই দেখতে একটু অস্বচ্ছতার লাগছে। প্রতিটি printf তোমরা এভাবে লিখতে পারো : printf("Value of a is %d\n", a);। এখন printf ফাংশনে “-এর ভেতর \n লিখলে কী হয় সেটি আমি বলব না। প্রোগ্রামটি চালালেই বুঝতে পারবে।

a-এর সবগুলো মান কিন্তু ঠিকভাবে দেখা যায়নি, যেসব মান - 2147483648 থেকে 2147483647 পর্যন্ত, কেবল সেগুলোই ঠিকঠাক প্রিণ্ট হবে। কারণ এই রেঞ্জের বাইরের সংখ্যা int টাইপের ভেরিয়েবলে রাখা যায় না। এটি হলো int টাইপের সংখ্যার সীমা। সিতে int টাইপের ডাটার জন্য মেমোরিতে চার বাইট (byte) জায়গা ব্যবহৃত হয়। চার বাইট মানে বক্সিং বিট (1 byte = 8 bit)। প্রতি বিটে দুটি জিনিস রাখা যায়, 0 আর 1। দুই বিটে রাখা যায় চারটি সংখ্যা (00, 01, 10, 11)। তাহলে 32 বিটে রাখা যাবে  $2^{32}$  টা সংখ্যা অর্থাৎ 4294967296টি সংখ্যা। এখন অর্ধেক ধনাত্মক আর অর্ধেক ঋণাত্মক যদি রাখি, তাহলে -2147483648 থেকে -1 পর্যন্ত মোট 2147483648 সংখ্যা আবার 0 থেকে 2147483647 পর্যন্ত মোট 2147483648 সংখ্যা, সর্বমোট 4294967296টি সংখ্যা। আশা করি, হিসাবটা বুঝতে পেরেছ।

এখন আমরা যোগ করার প্রোগ্রামটি লিখব যেটি সব বাস্তব সংখ্যা (real number) যোগ করতে পারবে। তোমাদের মনে করিয়ে দিই,

পূর্ণসংখ্যা হচ্ছে, -3, -2, -1, 0, 1, 2, 3 ইত্যাদি। আর বাস্তব সংখ্যা হচ্ছে -5, -3, -2.43, 0, 0.49, 2.92 ইত্যাদি (সংখ্যারেখার ওপর সব সংখ্যাই কিন্তু বাস্তব সংখ্যা)।

```

#include <stdio.h>
int main()
{
    double a, b, sum;
    a = 9.5;
    b = 8.743;
    sum = a + b;
    printf("Sum is: %lf\n", sum);
    printf("Sum is: %0.2lf\n", sum);
    return 0;
}

```

প্রোগ্রাম : ২.৯

প্রোগ্রামটি কম্পাইল এবং রান করো। আউটপুট হবে নিচের দুই লাইন:

Sum is: 18.243000

Sum is: 18.24

%lf ব্যবহার করায় প্রথম লাইনে দশমিকের পরে ছয় ঘর পর্যন্ত প্রিন্ট হয়েছে। আবার দ্বিতীয় লাইনে দশমিকের পরে দুই ঘর পর্যন্ত প্রিন্ট হয়েছে, কারণ %0.2lf লিখেছি (তিনি ঘর পর্যন্ত প্রিন্ট করতে চাইলে %0.3lf লিখতাম, আবার দশমিক অংশ প্রিন্ট করতে না চাইলে %0.0lf)। double টাইপের ডাটা রাখা যায়। বিস্তারিত হিসাব বুঝতে হলে কম্পিউটার-বিজ্ঞানসংক্রান্ত আরও কিছু জ্ঞানবুদ্ধির দরকার, তাই আমি আর এখন সেদিকে যাচ্ছি না।

এখন আমরা আমাদের প্রোগ্রামে এমন ব্যবহাৰ রাখতে চাই, যাতে কোন দুটি সংখ্যা যোগ কৰতে হবে সেটি আমরা কোডের ভেতৰ লিখব না, ব্যবহারকাৰীৰ কাছ থেকে ইনপুট আকাৰে জেনে নেব। ব্যবহারকাৰীৰ (মানে যে প্রোগ্রামটি ব্যবহাৰ কৰছে) কাছ থেকে ইনপুট নেওয়াৰ জন্য আমরা scanf ফাংশন ব্যবহাৰ কৰব (সি-তে আৱণ ফাংশন আছে এই কাজেৰ জন্য)। তাহলে দেৱি না কৰে প্রোগ্রাম লিখে ফেলি :

```
#include <stdio.h>
int main()
{
    int a, b, sum;
    scanf("%d", &a);
    scanf("%d", &b);
    sum = a + b;
    printf("Sum is: %d\n", sum);
    return 0;
}
```

প্ৰোগ্ৰাম : ২.১০

প্রোগ্রামটি রান কৰলে দেখবে ফাঁকা স্ক্ৰিন (blank screen) আসে। তখন তুমি একটি সংখ্যা লিখবে, তাৰপৰ স্পেস (space) বা এন্টাৰ (enter) দিয়ে আৱেকটি সংখ্যা লিখবে। তাৰপৰ আবাৰ এন্টাৰ চাপলে যোগফল দেখতে পাৰবে।

তোমৰা নিচয়ই scanf ফাংশনেৰ ব্যবহাৰ শিখে ফেলেছ। scanf("%d", &a); এখানে ডবল কোটেশনেৰ ভেতৰে %d দিয়ে scanf-কে বলে দেওয়া হচ্ছে যে একটি ইন্টিজাৰ বা int টাইপেৰ ভেরিয়েবলেৰ মান পড়তে হবে (ব্যবহাৰকাৰী কি-বোৰ্ড থেকে ইনপুট দেবে)। আৱ দেখো &a-এৰ আগে এমপাৰসেন্ড (&) চিহ্ন ব্যবহাৰ কৰা হয়েছে, &a দিয়ে বোৰানো হয়েছে, যে সংখ্যাটি ইনপুট দেওয়া হবে সেটি a ভেরিয়েবলেৰ মাঝে অ্যাসাইন হবে। তোমৰা যখন সি

38

আৱেকটু ভালোভাৰে শিখবে, তখন &a-এৰ প্ৰকৃত অৰ্থ বুৰাতে পাৰবে, আপাতত আমৰা ব্যবহাৰেৰ দিকেই মনোযোগ দিই। a এবং b-এৰ মান একটি scanf ফাংশন দিয়েও নেওয়া যেত এভাৰে : scanf("%d %d", &a, &b);। ভেৱিয়েবলেৰ আগে & চিহ্ন না দিলে কী সমস্যা? নিচেৰ প্রোগ্রামটি রান কৰাৰ চেষ্টা কৰো, কিছু একটি এৱেৰ পাৰে। এই মুহূৰ্তে এৱেৱটা ব্যাখ্যা কৰছি না, কাৰণ ব্যাখ্যাটা একটু জটিল আৱ এখন বোৰাতে গেলে তোমৰা ভুল বুৰাতে পাৰো এবং পৰে আমাকে গালমন্দ কৰবে।

```
#include <stdio.h>
int main()
{
    int a, b, sum;
    scanf("%d", &a);
    scanf("%d", &b);
    sum = a + b;
    printf("Sum is: %d\n", sum);
    return 0;
}
```

প্ৰোগ্ৰাম : ২.১১

এখন আমৰা যদি ইনপুট হিসেবে ইন্টিজাৰ না নিয়ে ডবল টাইপেৰ ডাটা নিতে চাইতাম তাহলে কী কৰতে হতো? scanf-এ %d-এৰ বদলে %lf ব্যবহাৰ কৰলেই চলত। তোমৰা প্ৰোগ্রামটি লিখে ফেলো এবং দেখো ঠিকঠাক রান হয় কি না। তাৰপৰে বাকি অংশ পড়া শুৱ কৰো।

আসলে ঠিকঠাক রান হবে না, কাৰণ ডাটা টাইপও পৰিবৰ্তন কৰতে হবে। মানে int না লিখে double লিখতে হবে। প্ৰোগ্রামটি ঠিক কৰে আবাৰ চালাও।

বইতে যখনই আমি কোনো প্ৰোগ্ৰাম লেখতে বলব সেটি যত সহজ কিংবা কঢ়িনই মনে হোক না কেন, সেটি কম্পিউটাৰে লিখে কম্পাইল

39

ও রান করতে হবে। এ কাজ না করে সামনে আগানো যাবে না। মনে রাখবে, গাড়ি চালানো শেখার জন্য যেমন গাড়ি চালানোর কোনো বিকল্প নেই, সাঁতার শেখার জন্য যেমন সাঁতার কাটার বিকল্প নেই, তেমনই প্রোগ্রামিং শেখার জন্য প্রোগ্রামিং করার কোনো বিকল্প নেই, শুধু বই পড়ে প্রোগ্রামার হওয়া যায় না।

এবার আমরা আরেক ধরনের ডাটা টাইপ দেখব, সেটি হচ্ছে char (character) টাইপ। তো এই character টাইপের চরিত্র হচ্ছে একে মেমোরিতে রাখার জন্য মাত্র এক বাইট জায়গার দরকার হয়। সাধারণত যেকোনো অক্ষর বা চিহ্ন রাখার জন্য এই টাইপের ডাটা ব্যবহার করা হয়। তবে সেই অক্ষরটা ইংরেজি বর্ণমালার অক্ষর হতে হবে, অন্য ভাষার অক্ষর char টাইপের ভেরিয়েবলে রাখা যাবে না। নিচের প্রোগ্রামটি কম্পিউটারে লিখে রান করাও :

```
#include <stdio.h>
int main()
{
    char ch;
    printf("Enter the first letter of your
name: ");
    scanf("%c", &ch);
    printf("The first letter of your name
is: %c\n", ch);
    return 0;
}
```

প্রোগ্রাম : ২.১২

কোড দেখে বুঝতেই পারছ, char টাইপের জন্য printf এবং scanf ফাংশনের ভেতরে %c ব্যবহার করতে হয়। আরেকটি ফাংশন আছে getchar, এটি দিয়েও char টাইপের ডাটা রিড করা যায়। নিচের প্রোগ্রামটি দেখো :

```
#include <stdio.h>
```

৮০

```
int main()
{
    char ch;
    printf("Enter the first letter of your
name: ");
    ch = getchar();
    printf("The first letter of your name
is: %c\n", ch);
    return 0;
}
```

প্রোগ্রাম : ২.১৩

এটা রান করাও। আগের প্রোগ্রামটির মতো একই কাজ করবে। getchar ফাংশন একটি অক্ষর পড়ে সেটি ch ভেরিয়েবলের ভেতরে অ্যাসাইন করে দিল। আর সরাসরি কোনো কিছু char টাইপ ভেরিয়েবলে রাখতে চাইলে যে অক্ষর বা চিহ্ন রাখবে তার দুই পাশে সিঙ্গেল কোটেশন চিহ্ন দেবে। যেমন : char c = 'A';

এখন নিচের প্রোগ্রামটি দেখো :

```
#include <stdio.h>
int main()
{
    int num1, num2;
    printf("Please enter a number: ");
    scanf("%d", &num1);
    printf("Please enter another number:
");
    scanf("%d", &num2);
    printf("%d + %d = %d\n", num1, num2,
num1+num2);
    printf("%d - %d = %d\n", num1, num2,
81
```

```

    num1 - num2);
    printf("%d * %d = %d\n", num1, num2,
num1 * num2);
    printf("%d / %d = %d\n", num1, num2,
num1 / num2);
    return 0;
}

```

প্রোগ্রাম : ২.১৪

এটি কম্পাইল ও রান করাও। এটি দেখে নিচয়ই বুঝতে পারছ বিয়োগ, গুণ ও ভাগের কাজ কীভাবে করতে হয়। এবারে তোমাদের কাজ হচ্ছে চারটি। এক, num1 ও num2-এর মধ্যেকার যোগ, বিয়োগ, গুণ, ভাগের কাজটি printf ফাংশনের ভেতরে না করে আগে করা এবং মানটি অন্য একটি ভেরিয়েবলে রেখে দেওয়া। এর জন্য একটি প্রোগ্রাম লিখে ফেলো। দ্বিতীয় কাজ হচ্ছে প্রোগ্রামটি ডবল টাইপের ভেরিয়েবল ব্যবহার করে করো। তৃতীয় কাজ হচ্ছে, num2-এর মান 0 দিয়ে দেখো কী হয়। চতুর্থ কাজটি হচ্ছে printf ফাংশনের ভেতরে ডবল কোটেশনের ভেতরে যেই +, -, \*, / চিহ্ন আছে সেগুলো সরাসরি ব্যবহার না করে একটি char টাইপ ভেরিয়েবলে রেখে তারপর ব্যবহার করা। চারটি কাজ ঠিকমতো করার পরে নিচের প্রোগ্রামটি দেখো :

```

#include <stdio.h>
int main()
{
    int num1, num2, value;
    char sign;
    printf("Please enter a number: ");
    scanf("%d", &num1);
    printf("Please enter another number:
");

```

```

    scanf("%d", &num2);
    value = num1 + num2;
    sign = '+';
    printf("%d %c %d = %d\n", num1, sign,
num2, value);
    value = num1 - num2;
    sign = '-';
    printf("%d %c %d = %d\n", num1, sign,
num2, value);
    value = num1 * num2;
    sign = '*';
    printf("%d %c %d = %d\n", num1, sign,
num2, value);
    value = num1 / num2;
    sign = '/';
    printf("%d %c %d = %d\n", num1, sign,
num2, value);
    return 0;
}

```

প্রোগ্রাম : ২.১৫

প্রোগ্রামটি দেখলেই বুঝতে পারবে কী কাজ করে। তবে শুধু দেখে বুঝলেই হবে না। কোড করে কম্পাইল ও রান করো।

সি ল্যাঙ্গুয়েজে আরও বেশ কিছু ডাটা টাইপ রয়েছে। ইনপুট ও আউটপুটের জন্যও রয়েছে নানা পদ্ধতি, যা তোমরা আস্তে আস্তে শিখবে (সব হয়তো এ বইয়ে থাকবে না, সি-এর অন্য বই পড়লে জানতে পারবে)। আপাতত যা শিখেছ, তা দিয়েই তোমরা অনেক প্রোগ্রাম লিখে ফেলতে পারবে।

এখন একটি মজার এবং দরকারি জিনিস বলে রাখি। প্রোগ্রামের কোডের ভেতরে তুমি নিজের ভাষাও ব্যবহার করতে পারো। এটিকে

বলে কমেন্ট (comment) করা। কম্পাইলার কমেন্টগুলোকে প্রোগ্রামের অংশ ধরবে না। এক লাইনের কমেন্ট হলে // চিহ্ন দিয়ে কমেন্ট শুরু করতে পারো। আর একাধিক লাইন থাকলে /\* দিয়ে শুরু এবং \*/ দিয়ে শেষ করতে হবে। নিচের প্রোগ্রামটি কিন্তু ঠিক্যাক কম্পাইল ও রান হবে।

```
#include <stdio.h>
```

```
int main()
{
    // test program - comment 1
    printf("Hello ");
    /* We have printed Hello,
    now we shall print World.

    Note that this is a multi-line comment
*/
    printf("World"); // printed world
    return 0;
}
```

প্রোগ্রাম : ২.১৬

এবারে একটি প্রশ্ন, (যেটি সি-এর টেক্সট বইয়ে এই চ্যাপ্টারের শুরুতেই বলে দিত), ভেরিয়েবলগুলোর নামকরণের নিয়মকানুন কী? ভেরিয়েবলের নাম এক বা একাধিক অক্ষরের হতে পারে, অক্ষরগুলো হতে পারে a থেকে z, A থেকে Z, 0 থেকে 9 এবং \_ (আন্ডারস্কোর বা আন্ডারবার)। তবে একাধিক অক্ষরের ক্ষেত্রে প্রথম অক্ষরটা অক্ষ (ডিজিট) হতে পারবে না। তুমি একটি প্রোগ্রামে int 7d; লিখে দেখো কম্পাইলার কী বলে। আর ভেরিয়েবলের নামগুলো অর্থপূর্ণ হলে ভালো হয়। যেমন, যোগফল রাখার জন্য ভেরিয়েবলের নাম sum হলেই ভালো, যদিও y নাম দিলেও প্রোগ্রাম চলে। অর্থপূর্ণ নাম দিলে বুঝতে সুবিধা হয়, ভেরিয়েবলটা কী উদ্দেশ্যে ব্যবহার করা হয়েছে।

## অধ্যায় তিনি : কন্ডিশনাল লজিক

তোমরা অনেকেই হয়তো জানো যে “চাচা চৌধুরীর বুদ্ধি কম্পিউটারের চেয়েও প্রখর”! এটি শুনে প্রথম প্রথম চাচা চৌধুরীর ওপর ভঙ্গ-শুঙ্গ অনেক বেড়ে গেলেও একটু চিন্তা করলেই তোমরা বুঝতে পারবে যে আসলে তোমাদের সবার বুদ্ধি কম্পিউটারের চেয়ে প্রখর। আসলে কম্পিউটারের তো কোনো বুদ্ধিই নেই। প্রোগ্রামাররা যেভাবে প্রোগ্রাম লিখে দেয় কম্পিউটার সেভাবে কাজ করে। এখন আমরা প্রোগ্রামিংয়ের সহজ অর্থচ খুব গুরুত্বপূর্ণ একটি বিষয় শিখব। সেটি হচ্ছে কন্ডিশনাল লজিক। কোন শর্তে কী করতে হবে, সেটি প্রোগ্রাম লিখে কম্পিউটারকে বোঝাতে হবে। কথা না বাড়িয়ে আমরা প্রোগ্রাম লিখা শুরু করে দিই। তোমরা কিন্তু অবশ্যই প্রতিটি প্রোগ্রাম কম্পিউটারে চালিয়ে দেখবে।

```
#include <stdio.h>
```

```
int main()
{
    int n;
    n = 10;
    if(n >= 0) {
        printf("The number is positive\n");
    }
    else {
        printf("The number is negative\n");
    }
    return 0;
}
```

প্রোগ্রাম : ৩.১

ওপরের প্রোগ্রামটির কাজ কী? n-এর বিভিন্ন মান (যেমন: 0, -10, -2, 5, 988 ইত্যাদি) বসিয়ে তোমরা প্রোগ্রামটি চালাও। দেখা যাচ্ছে

যে এর কাজ হচ্ছে n ধনাত্মক (positive) না ঋণাত্মক (negative) সেটি নির্ণয় করা। কোনো সংখ্যা ধনাত্মক হতে গেলে একটি শর্ত পূরণ করতে হয়। সেটি হচ্ছে তাকে শূন্যের সমান বা তার চেয়ে বড় হতে হয়। তাহলে আমাদের লজিকটি দাঁড়াচ্ছে এ রকম যে, “n যদি শূন্যের সমান বা বড় হয়, তবে n ধনাত্মক, না হলে n ঋণাত্মক”。 এই ব্যাপারটি সি ল্যাঙ্গুয়েজে প্রকাশ করতে হয় if এবং তার সঙ্গে else ব্যবহার করে। if-এর ভেতর একটি শর্ত (কন্ডিশন) লিখে দিতে হয়, যা সত্য হলেই কেবল তার ভেতরের অংশের কাজ হয় (মানে if-এর পর যে দ্বিতীয় বন্ধনী {} ব্যবহার করা হয়েছে তার ভেতরের সব কাজ)। আর কন্ডিশনটা লিখতে হয় প্রথম বন্ধনীর ভেতরে। if-এর ভেতরে যেই কন্ডিশনটা আছে সেটি যদি মিথ্যা হয়, তবে else-এর ভেতরের (দ্বিতীয় বন্ধনীর ভেতরে) অংশের কাজ হয়। সব প্রোগ্রামিং ল্যাঙ্গুয়েজেই এটি আছে, তবে লেখার ধরন হয়তো আলাদা।

এখন আমাদের দেখতে হবে, কন্ডিশনগুলো কীভাবে লিখতে হবে? তোমরা একক্ষণে জেনে গেছ যে “বড় কিংবা সমান” এই তুলনা করার জন্য >= চিহ্ন ব্যবহার করা হয়। “ছোট কিংবা সমান”-এর জন্য ব্যবহার করে <= চিহ্ন। দুটি সংখ্যা একটি আরেকটির সমান কি না সেটি পরীক্ষার জন্য ব্যবহার করে == চিহ্ন (লক্ষ করো এখানে দুটি সমান চিহ্ন আছে। শুরুর দিকে অনেকেই সমান কি না পরীক্ষার জন্য ভুল করে = (একটি সমান চিহ্ন যা দিয়ে আসলে কোনো ভেরিয়েবলে কোনো কিছু অ্যাসাইন করা হয়) ব্যবহার করে বিপদে পড়ে যায়)। দুটি সংখ্যা পরস্পর অসমান কি না, এটি পরীক্ষার জন্য != চিহ্ন ব্যবহার করে। আর ছোট কিংবা বড় পরীক্ষার জন্য < আর > চিহ্ন ব্যবহার করতে হয়। আরও ব্যাপার-স্যাপার আছে। একবারে সব না শিখে চলো আস্তে আস্তে প্রোগ্রাম লিখে শেখা যাক। এখানে ইন্ডেন্টেশনের ব্যাপারটিও কিন্তু খেয়াল কোরো। if কিংবা else-এর ভেতরের ইন্ডেন্টেশনের সব লাইন কিন্তু if বা else যেখানে শুরু, তার চার ঘর (চারটি স্পেস) ডান থেকে শুরু হয়েছে।

আমরা ওপরের প্রোগ্রামটি এভাবেও লিখতে পারতাম :

```
#include <stdio.h>
```

```
int main()
{
    int n;
    n = 10;
    if(n < 0) {
        printf("The number is negative\n");
    }
    else {
        printf("The number is positive\n");
    }
    return 0;
}
```

প্রোগ্রাম : ৩.২

এখানে আমরা প্রথমে পরীক্ষা করেছি যে n শূন্যের চেয়ে ছোট কি না। যদি ছোট হয়, তাহলে n নেগেটিভ; আর সেটি না হলে (সেটি না হওয়া মানে n অবশ্যই শূন্যের সমান বা বড়) n পজিটিভ।

তোমাদের মধ্যে যারা একটু খুঁতখুঁতে স্বত্বাবের, তারা নিচয়ই ভাবছ যে শূন্য তো আসলে পজিটিভ বা নেগেটিভ কোনোটাই না। শূন্যের চেয়ে বড় সব সংখ্যা হচ্ছে পজিটিভ আর ছোট সব সংখ্যা হচ্ছে নেগেটিভ। কম্পিউটারকে সেটি বোঝাতে গেলে আমরা নিচের প্রোগ্রামটি লিখতে পারি :

```
#include <stdio.h>
int main()
{
    int n = 10;
    if(n < 0) {
        printf("The number is negative\n");
    }
}
```

```

    }
    else if (n > 0) {
        printf("The number is positive\n");
    }
    else if (n == 0) {
        printf("The number is zero!\n");
    }
    return 0;
}

```

প্রোগ্রাম : ৩.৩

প্রোগ্রামটি একটু ব্যাখ্যা করা যাক :

if( $n < 0$ ): এখানে আমরা প্রথমে পরীক্ষা করেছি  $n$  শূন্যের চেয়ে ছোট কি না। ছোট হলে তো কথাই নেই। আমরা বলে দিচ্ছি যে সংখ্যাটি নেগেটিভ।

else if( $n > 0$ ): আর যদি ছোট না হয়, তবে  $n$  শূন্যের চেয়ে বড় কি না সেটি পরীক্ষা করেছি if( $n > 0$ )। এই শর্ত সত্যি হলে সংখ্যাটি পজিটিভ।

else if( $n == 0$ ): আর  $n > 0$  যদি সত্যি না হয় তবে কোন শর্তটি বাদ রইল? দুটি সমান কি না সেটি পরীক্ষা করা। তাহলে আমরা পরীক্ষা করেছি যে  $n$  শূন্যের সমান কি না এবং সমান হলে বলে দিচ্ছি যে এটি শূন্য।

দুটি সংখ্যা তুলনা করার সময় প্রথমটা যদি দ্বিতীয়টির চেয়ে বড় না হয়, ছোটও না হয়, তাহলে তারা অবশ্যই সমান। তাই তৃতীয় কন্ডিশনটা আসলে আমাদের দরকার নেই। আমরা প্রথম দুটি কন্ডিশন মিথ্যা হলেই বলে দিতে পারি যে  $n$ -এর মান শূন্য।

```

#include <stdio.h>
int main()

```

```

{
    int n = 10;
    if(n < 0) {
        printf("The number is negative\n");
    }
    else if (n > 0) {
        printf("The number is positive\n");
    }
    else {
        printf("The number is zero!\n");
    }
    return 0;
}

```

প্রোগ্রাম : ৩.৪

আবার সব সময় যে if ব্যবহার করলেই সঙ্গে else কিংবা else if ব্যবহার করতে হবে, এমন কোনো কথা নেই। নিচের প্রোগ্রামটি দেখো :

```

#include <stdio.h>
int main()
{
    int number = 12;
    if(number > 10) {
        printf("The number is greater than
ten\n");
    }
    return 0;
}

```

প্রোগ্রাম : ৩.৫

এখানে কেবল দেখা হচ্ছে যে সংখ্যাটির মান দশের চেয়ে বড় কি না।  
নিচের প্রোগ্রামটি দেখে বলো তো আউটপুট কী হবে?

```
#include <stdio.h>
int main()
{
    int n = 10;

    if (n < 30) {
        printf("n is less than 30.\n");
    }
    else if(n < 50) {
        printf("n is less than 50.\n");
    }
    return 0;
}
```

প্রোগ্রাম : ৩.৬

আউটপুট হবে : n is less than 30. যদিও else if(n < 50)  
এটিও সত্য কিন্তু যেহেতু if (n < 30) সত্য হয়ে গেছে, তাই এর সঙ্গে  
বাকি যত else if কিংবা else থাকবে, সেগুলো আর পরীক্ষা করা  
হবে না। এখন তোমরা নিশ্চয়ই নিচের প্রোগ্রামটির আউটপুট বলতে  
পারবে।

```
#include <stdio.h>
int main()
{
    int n = 10;
    if (n < 30) {
```

৫০

```
        printf("n is less than 30.\n");
    }
    if(n < 50) {
        printf("n is less than 50.\n");
    }
    return 0;
}
```

প্রোগ্রাম : ৩.৭

এখন আমরা আরেকটি প্রোগ্রাম লিখব। কোনো সংখ্যা জোড় না  
বেজোড় সেটি নির্ণয় করার প্রোগ্রাম। কোনো সংখ্যা জোড় কি না সেটি  
বোঝার উপায় হচ্ছে সংখ্যাটিকে 2-এর ভাগ করা। যদি ভাগশেষ শূন্য  
হয়, তাহলে সংখ্যাটি জোড়; আর ভাগশেষ শূন্য না হলে সেটি  
বেজোড়। সি ল্যাঙ্গুয়েজে ভাগশেষ বের করার জন্য মডুলাস অপারেটর  
(modulus operator) আছে, যেটাকে '%' চিহ্ন দিয়ে প্রকাশ করা  
হয়। তাহলে আর চিন্তা নেই।

শুরুতে একটি সংখ্যা নেব : int number;

এবাবে number-এর জন্য একটি মান ঠিক করি : number = 5;

এখন number-কে 2 দিয়ে ভাগ করলে যে ভাগশেষ পাব সেটি বের  
করি : remainder = number % 2;

এখন if-এর সাহায্যে remainder-এর মান পরীক্ষা করে আমরা  
সিদ্ধান্তে পৌঁছে যেতে পারি। remainder-এর কেবল দৃটি মানই  
সন্তুষ্ট- 0 আর 1। পুরো প্রোগ্রামটি লিখে ফেলি :

```
#include <stdio.h>
int main()
{
    int number, remainder;
    number = 5;
```

৫১

```

remainder = number % 2;
if(remainder == 0) {
    printf("The number is even\n");
}
else {
    printf("The number is odd\n");
}
return 0;
}

```

প্রোগ্রাম : ৩.৮

প্রোগ্রামটি remainder ভেরিয়েবল ব্যবহার না করেও সেখা ষষ্ঠী :

```

#include <stdio.h>
int main()
{
    int number = 9;

    if(number % 2 == 0) {
        printf("The number is even\n");
    }
    else {
        printf("The number is odd\n");
    }
    return 0;
}

```

প্রোগ্রাম : ৩.৯

আচ্ছা, আমাদের যদি কেবল  
আমরা কী করতাম? else

ংখ্যা নির্ণয়  
দিয়ে দি

তোমাদের জন্য এখন একটি ছেট পরীক্ষা। মডুলাস অপারেটর  
ব্যবহার না করে ভাগশেষ বের করতে পারবে? একবার করে গুণ,  
ভাগ ও বিয়োগ (\*, /, -) ব্যবহার করে কিন্তু কাজটি করা যায়। তোমরা  
সেটি করার চেষ্টা করতে পারো।

এবার আরেকটি প্রোগ্রাম দেখা যাক। কোনো একটি অক্ষর ছেট  
হাতের (small letter বা lower case letter) নাকি বড় হাতের  
(capital letter বা upper case letter), সেটি বের করতে হবে।  
এর জন্য সবচেয়ে সহজ সমাধানটা হতে পারে এই রকম যে আমরা  
একটি character টাইপের ভেরিয়েবলের ডেতের অক্ষরটা রাখতে  
পারি। তারপর একে একে সেটিকে 26টি lower case letter এবং  
26টি upper case letter-এর সঙ্গে তুলনা করে দেখতে পারি।  
যখনই মিলে যাবে, তখনই বলে দেওয়া যায়, অক্ষরটা কোন ধরনের।

```

char ch = 'p';
if (ch == 'a') {
    printf("%c is lower case\n", ch);
}
else if (ch == 'A') {
    printf("%c is upper case\n", ch);
}
else if (ch == 'b') {
    printf("%c is lower case\n", ch);
}
if (ch == 'B') {
    printf("%c is upper case\n", ch);
}

```

```

    }
else if (ch == 'c')
{
    printf("%c is lower case\n", ch);
}
else if (ch == 'C')
{
    printf("%c is upper case\n", ch);
}

```

এভাবে চলবে।

কিন্তু এই সমস্যার সমাধান করার জন্য এত কোড লেখার কোনো দরকার নেই। এটি সহজে করার জন্য আমাদের জানতে হবে অ্যান্ড অপারেটরের (AND operator) ব্যবহার। সি ল্যাঙ্গুয়েজে একে '&&' চিহ্ন দিয়ে প্রকাশ করা হয়। নিচের কোডটি দেখলেই তোমরা এর কাজ বুঝে যাবে।

```

#include <stdio.h>
int main()
{
    char ch = 'W';
    if(ch >= 'a' && ch <= 'z') {
        printf("%c is lower case\n", ch);
    }
    if(ch >= 'A' && ch <= 'Z') {
        printf("%c is upper case\n", ch);
    }
    return 0;
}

```

### প্রোগ্রাম : ৩.৯

'&&'-এর বাঁ পাশে একটি কন্ডিশন এবং ডান পাশে একটি কন্ডিশন থাকবে, এবং দুটি কন্ডিশন সত্য হলেই সম্পূর্ণ কন্ডিশনটা সত্য হবে।  $ch \geq 'a' \&\& ch \leq 'z'$  এটি পুরোটা একটি কন্ডিশন। এখন &&-এর বাঁ দিকে একটি কন্ডিশন আছে  $ch \geq 'a'$  আর ডান দিকে আরেকটি কন্ডিশন  $ch \leq 'z'$ । দুটি কন্ডিশনই যদি সত্য হয়, তাহলে পুরো কন্ডিশনটা সত্য হবে। এখন কম্পিউটার প্রতিটি অক্ষর বোঝার জন্য যে কোড ব্যবহার করে, তাতে a-এর চেয়ে b-এর মান এক বেশি, b-এর চেয়ে c-এর মান এক বেশি, c-এর চেয়ে d-এর মান এক বেশি ... এরকম। তাই কোনো অক্ষর lower case হলে সেটি অবশ্যই 'a'-এর সমান কিংবা বড় হতে হবে। আবার সেটি 'z'-এর সমান কিংবা ছোট হতে হবে। একইভাবে A-এর চেয়ে B-এর মান এক বেশি, B-এর চেয়ে C-এর মান এক বেশি ... এরকম। তাই কোনো ক্যারেক্টোরের মান 'A' থেকে 'Z'-এর মধ্যে হলে আমরা বলতে পারি যে সেটি upper case। 'A'-এর সমান কিংবা বড় হতে হবে এবং 'Z'-এর সমান কিংবা ছোট হতে হবে।

আরেকটি ব্যাপার। দ্বিতীয় if-এর আগে else ব্যবহার করা উচিত। তাহলে কম্পাইলার প্রথম if-এর ভেতরের শর্ত সত্য হলে আর পরের if-এর কন্ডিশন পরীক্ষা করবে না। তাতে সময় বাঁচবে।

```

#include <stdio.h>
int main()
{
    char ch = 'k';
    if(ch >= 'a' && ch <= 'z') {
        printf("%c is lower case\n", ch);
    }
    else if(ch >= 'A' && ch <= 'Z') {
        printf("%c is upper case\n", ch);
    }
}

```

```

    }
    return 0;
}

```

প্রোগ্রাম : ৩.১০

আশা করি, তোমরা '&&'-এর ব্যবহার বুঝে গেছ।

এখন আরেকটি অপারেটরের ব্যবহার দেখব। সেটি হচ্ছে অর (OR)। একে প্রকাশ করা হয় '||' চিহ্ন দিয়ে (পরপর দুটি ||)। '&&'-এর ক্ষেত্রে যেমন দুই পাশের শর্ত সত্য হলেই সম্পূর্ণ শর্ত সত্য হয়, '||'-এর ক্ষেত্রে যেকোনো এক পাশের শর্ত সত্য হলেই সম্পূর্ণ শর্ত সত্য হয়।

নিচের প্রোগ্রামটির আউটপুট কী হবে? কোড দেখে বলতে না পারলে প্রোগ্রামটি চালাও।

```

#include <stdio.h>

int main()
{
    int num = 5;

    if(num >= 1 || num <= 10) {
        printf("yes\n");
    }
    else {
        printf("no\n");
    }
    return 0;
}

```

প্রোগ্রাম : ৩.১১

এটির আউটপুট হবে yes। এখন num-এর মান 50 করে দাও। আউটপুট কী হবে?

এবারেও আউটপুট yesই হবে। কারণ num-এর মান 50 হলে, প্রথম শর্তটি সত্য হবে (num >= 1) আর দ্বিতীয় শর্তটি (n <= 10) মিথ্যা হবে। তবে আমরা যেহেতু দুটি শর্তের মাঝে '||' ব্যবহার করেছি, তাই যেকোনো একটি শর্ত সত্য হলেই সম্পূর্ণ শর্তটি সত্য হবে।

এখন আরও একটি সমস্যা। কোনো অক্ষর vowel নাকি consonant, সেটি নির্ণয় করতে হবে। আমরা জানি, vowelগুলো হচ্ছে a, e, i, o, u। এখন কোনো ক্যারেক্টার এই পাঁচটির মধ্যে পড়ে কি না সেটি নির্ণয় করার জন্য যদি আমরা এমন শর্ত দিই: ch >= 'a' && ch <= 'U' তাহলে কিন্তু হবে না। কারণ তাহলে a থেকে u পর্যন্ত সব অক্ষরের জন্যই শর্তটি সত্যি হবে কিন্তু আমাদের দরকার নির্দিষ্ট কিছু অক্ষর। তাই শর্তটি আমরা এভাবে লিখতে পারি :

```

if(ch == 'a' || ch == 'e' || ch == 'i' ||
ch == 'o' || ch == 'u') {
    printf("%c is vowel\n", ch);
}
else {
    printf("%c is consonant\n", ch);
}

```

তাহলে এবার সম্পূর্ণ প্রোগ্রামটি তোমরা লিখে ফেলতে পারো।

## অধ্যায় চার : লুপ (Loop)

তোমরা এই মধ্যে প্রোগ্রামের মধ্যে বিভিন্ন ধরনের শর্ত (condition) ব্যবহার করতে শিখে গেছ। এইসব শর্ত দিয়ে বিভিন্ন প্রোগ্রাম তৈরি করাও হয়তো শুরু করে দিয়েছ। খুব ভালো কথা। কিন্তু এখন আমরা আরেকটি সমস্যা ও তার সমাধানের পথ খুঁজব। একটি প্রোগ্রাম লিখতে হবে, যেটি 1 থেকে 10 পর্যন্ত সব পূর্ণসংখ্যা মনিটরে দেখাবে (প্রতি লাইনে একটি সংখ্যা থাকবে)। খুবই সহজ সমস্যা এবং সমাধানও অত্যন্ত সহজ। আমি জানি, তোমরা এক মিনিটের মধ্যেই নিচের প্রোগ্রামটি লিখে ফেলবে :

```
#include <stdio.h>
int main()
{
    printf("1\n");
    printf("2\n");
    printf("3\n");
    printf("4\n");
    printf("5\n");
    printf("6\n");
    printf("7\n");
    printf("8\n");
    printf("9\n");
    printf("10\n");

    return 0;
}
```

প্রোগ্রাম : 8.1

এখনে আমরা 1 থেকে 10 পর্যন্ত সবগুলো সংখ্যা প্রিন্ট করে দিয়েছি।

অবশ্য একটি printf() ব্যবহার করেও কাজটি করা যেত :  
printf("1\n2\n3\n4\n5\n6\n7\n8\n9\n10\n");  
আবার প্রোগ্রামটি এভাবেও লেখা যেত। n একটি ইন্টজার ভেরিয়েবল, যার মান আমরা প্রথমে 1 বসাব। তারপর n-এর মান প্রিন্ট করব। তারপর n-এর মান এক বাড়াব (n = n + 1 অথবা সংক্ষেপে, n++ লিখে)।

```
int n = 1;
printf("%d\n", n);
n = n + 1;
printf("%d\n", n);
n = n + 1;
printf("%d\n", n);
n = n + 1;
/* এভাবে মোট দশ বার */
আবার n-এর মান 1 বাড়ানোর কাজটি কিন্তু এক লাইনেই সেরে ফেলা যায়।
```

```
printf("%d\n", n);
n = n + 1;
```

এর পরিবর্তে আমরা লিখতে পারি :

```
printf("%d\n", n++);
```

যা-ই হোক, এ তো গেল 1 থেকে 10 পর্যন্ত প্রিন্ট করা। কিন্তু আমাদের যদি 1 থেকে 100, বা 1000, বা 10000 পর্যন্ত প্রিন্ট করতে বলা হতো তাহলে আমরা কী করতাম? ওপরে যে পদ্ধতি অবলম্বন করা হয়েছে সেটি তো অবশ্যই করা যেত। কিন্তু আমি জানি, তোমরা কেউই এত কষ্ট করতে রাজি নও।

এ সমস্যা সমাধানের জন্য সব প্রোগ্রামিং ল্যাঙ্গুয়েজেই লুপ (loop) বলে একটি পদ্ধতি রয়েছে। এটি দিয়ে একই কাজ বারবার করা যায়। লুপের মধ্যে একটি শর্ত বসিয়ে দিতে হয়, যেটি পূরণ না হওয়া পর্যন্ত প্রোগ্রামটি লুপের ভেতরের কাজ বারবার করতে থাকবে। সি ল্যাঙ্গুয়েজে দুটি জনপ্রিয় লুপ হচ্ছে while এবং for। আমরা এখন while ব্যবহার করে ওই প্রোগ্রামটি লিখব।

```
#include <stdio.h>
int main()
{
    int n = 1;
    while(n <= 10) {
        printf("%d\n", n);
        n++;
    }
    return 0;
}
```

প্রোগ্রাম : 8.২

কী চমৎকার! এখন আমরা চাইলে 10-এর বদলে যেকোনো সংখ্যা বসাতে পারি, যে সংখ্যা বসাব 1 থেকে সেটি পর্যন্ত প্রিন্ট হবে। while লুপে প্রথম বন্ধনীর ভেতর শর্ত লিখে দিতে হয়। প্রোগ্রাম সেই শর্ত পরীক্ষা করে। যতক্ষণ পর্যন্ত শর্তটি সত্য হয় ততক্ষণ পর্যন্ত লুপের ভেতরের কাজগুলো চলতে থাকে। লুপের ভেতরের কাজগুলো থাকবে দ্বিতীয় বন্ধনীর ভেতর। যেমন এখানে লুপের ভেতরে আমরা দুটি কাজ করেছি। n-এর মান প্রিন্ট করেছি আর তারপর n-এর মান 1 বাড়িয়েছি। n-এর মান 1 করে বাড়তে থাকলে একসময় এটি 11 হবে আর তখন n <= 10 এই শর্তটি মিথ্যা হয়ে যাবে (কারণ 11 > 10)। আর প্রোগ্রামটি লুপ থেকে বের হয়ে আসবে। অর্থাৎ, শর্তটি যখনই মিথ্যা হবে তখনই লুপ থেকে বের হয়ে যাবে।

উডনেশনের ব্যাপারটি খেয়াল করো। লুপের ভেতরের অংশের

কোড চার ঘর ডান দিক থেকে শুরু হয়েছে।

এবারে তোমাদের জন্য একটি প্রশ্ন। বলো তো, নিচের প্রোগ্রামটির আউটপুট কী হবে?

```
#include <stdio.h>
int main()
{
    int n = 1;
    while(n <= 10) {
        printf("%d\n", n);
    }
    n++;
    return 0;
}
```

প্রোগ্রাম : 8.৩

এটাও কি 1 থেকে 10 পর্যন্ত সব সংখ্যা প্রিন্ট করবে? দেখা যাক। প্রোগ্রামটি রান করাও। আউটপুট কী?

মনিটরে প্রতি লাইনে 1 প্রিন্ট হচ্ছে এবং প্রোগ্রামটি বন্ধ হচ্ছে না। খুবই দুঃখের বিষয়। দেখা যাক দুঃখের পেছনে কারণটা কী।

int n = 1; প্রথমে প্রোগ্রামটি n-এর মান 1 বসাবে।

তারপর while লুপে গিয়ে শর্ত পরীক্ষা করবে। আমরা শর্ত দিয়েছি n <= 10 মানে n-এর মান 10-এর ছোট বা সমান। এই শর্ত তো সত্য, কারণ n-এর মান 1। তারপর প্রোগ্রামটি n-এর মান প্রিন্ট করবে printf("%d\n", n);। তারপর কি n-এর মান 1 বাড়বে? বাড়বে না, কারণ আমরা দ্বিতীয় বন্ধনী শেষ করে দিয়েছি '}' চিহ্ন দিয়ে (মানে লুপ শেষ)। তার মানে প্রোগ্রামটি আবার শর্ত পরীক্ষা করবে, আবার n-এর মান প্রিন্ট করবে...এভাবে চলতেই থাকবে। কারণ n-এর মান যেহেতু বাড়ছে না, n <= 10 শর্তটি সব সময় সত্যই রয়ে যাচ্ছে— কখনো

মিথ্যা হচ্ছে না।

এখন তোমরা while লুপ নিয়ে বিভিন্ন ধরনের গবেষণা চালিয়ে যেতে পারো। সব সময় সত্য হয়, এমন শর্ত ব্যবহার করে তোমার কম্পিউটারকে ব্যস্ত রাখতে পারো। while(1){...} এখানে শর্ত হিসেবে 1 ব্যবহার করা হয়েছে। কম্পিউটার 1 বলতে বোবো সত্য। সুতরাং লুপের ভেতরের কাজগুলো সব সময় চলতে থাকবে, বন্ধ হবে না। while(1 == 1){...} ও একই আচরণ করবে।

তবে এখন আমি তোমাদের একটি দরকারি জিনিস বলে রাখি, যেটি দিয়ে তোমরা জোর করে লুপ থেকে বের হয়ে যেতে পারবে। সেটি হচ্ছে break স্টেটমেন্ট। কথা না বাড়িয়ে একটি প্রোগ্রাম লিখলেই ব্যাপারটি পরিষ্কার হয়ে যাবে।

```
#include <stdio.h>

int main()
{
    int n = 1;
    while(n <= 100) {
        printf("%d\n", n);
        n++;
        if(n > 10) {
            break;
        }
    }
    return 0;
}
```

প্রোগ্রাম : 8.8

এই প্রোগ্রামটি কী করবে? 1 থেকে 10-এর পর্যন্ত প্রিন্ট করবে। যদিও while-এর ভেতর আমরা বলেছি যে শর্ত হচ্ছে  $n \leq 100$ , কিন্তু লুপের ভেতরে আবার বলে দিয়েছি যে যদি  $n > 10$  হয়, তাহলে

break; মানে বের হয়ে যাও, বা লুপটি ভেঙে দাও। break সব সময় যেই লুপের ভেতর থাকে সেটির বাইরে প্রোগ্রামটিকে নিয়ে আসে। সুতরাং n-এর মান 10 প্রিন্ট হওয়ার পরে এর মান এক বাড়বে ( $n++$ ); অর্থাৎ n-এর মান হবে 11। আর তখন  $n > 10$  সত্য হবে, ফলে প্রোগ্রামটি if কন্ডিশনের ভেতরে চুকে যাবে। সেখানে গিয়ে সে দেখবে তাকে break করতে বলা হয়েছে, তাই সে লুপের বাইরে চলে যাবে। break-এর উল্টা কাজ করে, এমন একটি স্টেটমেন্ট হচ্ছে continue;। কোনো জায়গায় continue ব্যবহার করলে লুপের ভেতরে continue-এর পরের অংশের কাজ আর হয় না। নিচের প্রোগ্রামটি কোড করে কম্পাইল ও রান করো :

```
#include <stdio.h>

int main()
{
    int n = 0;

    while (n < 10) {
        n = n + 1;
        if (n % 2 == 0) {
            continue;
        }
        printf("%d\n", n);
    }
    return 0;
}
```

প্রোগ্রাম : 8.9

এই প্রোগ্রামটি 1 থেকে 10-এর মধ্যে কেবল বেজোড় সংখ্যাগুলো প্রিন্ট করবে। জোড় সংখ্যার বেলায় continue ব্যবহার করার কারণে প্রোগ্রামটি printf("%d\n", n); স্টেটমেন্ট এক্সিকিউট না করে লুপের পরবর্তী ধাপের কাজ শুরু করবে।

এবার আমরা আরেকটি প্রোগ্রাম লিখব। ছোটবেলায় যে নামতাঙ্গলো তোমরা শিখেছ সেগুলো এখন আমরা প্রোগ্রাম লিখে কম্পিউটারের মনিটরে দেখব। চলো 5-এর নামতা দিয়ে শুরু করা যাক। আমাদের প্রোগ্রামের আউটপুট হবে এরকম :

$$\begin{aligned} 5 \times 1 &= 5 \\ 5 \times 2 &= 10 \\ 5 \times 3 &= 15 \\ 5 \times 4 &= 20 \\ 5 \times 5 &= 25 \\ 5 \times 6 &= 30 \\ 5 \times 7 &= 35 \\ 5 \times 8 &= 40 \\ 5 \times 9 &= 45 \\ 5 \times 10 &= 50 \end{aligned}$$

তোমরা নিশ্চয়ই এখন অনেকগুলো printf ফাংশন লেখা শুরু করবে না। লুপের সাহায্যে প্রোগ্রামটি লিখে ফেলবে :

```
#include <stdio.h>
int main()
{
    int n = 5;
    int i = 1;

    while (i <= 10) {
        printf("%d X %d = %d\n", n, i,
n*i);
        i = i + 1;
    }
}
```

return 0;

}

প্রোগ্রাম : 8.৬

এতক্ষণ আমরা while লুপ ব্যবহার করলাম। এবার চলো for লুপ ব্যবহার করতে শিখি। 5-এর নামতার প্রোগ্রামটি যদি আমরা for লুপ ব্যবহার করে লিখি, তাহলে সেটির চেহারা দাঁড়াবে :

```
#include <stdio.h>
int main()
{
    int n = 5;
    int i;
    for(i = 1; i <= 10; i = i + 1) {
        printf("%d X %d = %d\n", n, i,
n*i);
    }
    return 0;
}
```

প্রোগ্রাম : 8.৭

for লুপের প্রথম বন্ধনীর ভেতর তিনটি অংশ লক্ষ করো। প্রতিটি অংশ সেমিকোলন (;) দিয়ে আলাদা করা হয়েছে। প্রোগ্রামটি যখন লুপের ভেতর ঢোকে, তখন প্রথম সেমিকোলনের আগে আমরা যে কাজগুলো করতে বলব, সেগুলো একবার করবে। যেমন, এখনে i-এর মান 1 বসাবে। তারপর দ্বিতীয় অংশের কাজ করবে। দ্বিতীয় অংশে সাধারণত শর্ত ব্যবহার করা হয় (while লুপে প্রথম বন্ধনীর ভেতর আমরা যে কাজটি করি আরকি)। ওপরের প্রোগ্রামে আমরা দ্বিতীয় অংশে i <= 10 শর্তটি ব্যবহার করেছি। এই শর্ত যদি মিথ্যা হয় তাহলে প্রোগ্রামটি লুপ থেকে বেরিয়ে আসবে। আর যদি সত্য হয় তাহলে লুপের ভেতরের কাজগুলো করবে এবং তারপর for লুপের সেই প্রথম বন্ধনীর ভেতর তৃতীয় অংশে যে কাজগুলো করতে বলা হয়েছে

মেঙ্গলো করবে। তারপর আবার দ্বিতীয় অংশে এসে শর্ত পরীক্ষা করবে। প্রথম অংশের কাজ কিন্তু আর হবে না। তো আমাদের প্রোগ্রামটি আবার লক্ষ করো।  $i \leq 10$  সত্য, কারণ  $i$ -এর মান 1। তারপর  $i = i + 1$  স্টেটমেন্ট এক্সিকিউট হবে ( $i$ -এর মান এক বেড়ে যাবে)। তারপর আবার  $i \leq 10$  শর্তটি হিসেবে যাবে ( $i$ -এর মান এখন 2)। 10 সত্য না মিথ্যা সেটি পরীক্ষা করা হবে ( $i$ -এর মান এখন 2)। তারপর আবার লুপের ভেতরের কাজ হবে ( $printf()$ )। এভাবে যতক্ষণ না  $i \leq 10$  শর্তটি মিথ্যা হচ্ছে ততক্ষণ লুপের ভেতরের কাজ চলতে থাকবে।  $i$ -এর মান এক এক করে বেড়ে বেড়ে যখন 11 হবে তখন শর্তটি মিথ্যা হবে আর প্রোগ্রামটি লুপ থেকে বের হয়ে আসবে।

for লুপের প্রথম বন্ধনীর ভেতরের তিনটি অংশই যে ব্যবহার করতে হবে, এমন কোনো কথা নেই। কোনো অংশ ব্যবহার করতে না চাইলে আমরা সেটি ফাঁকা রেখে দিতে পারি, তবে সেমিকোলন কিন্তু অবশ্যই দিতে হবে। যেমন, আমরা যদি  $i$ -এর মান আগেই নির্ধারণ করে দিই, তাহলে সেটি লুপের ভেতর না করলেও চলে।

```
int i = 1;
for(; i<= 10; i = i + 1) {
    printf("%d X %d = %d\n", n, i, n*i);
}
```

যদি তিনটি অংশের কোনোটিই লিখতে না চাই, তবে পুরো প্রোগ্রামটি এভাবে লেখা যায় :

```
#include <stdio.h>
int main()
{
    int n = 5;
    int i = 1;

    for( ; ; ) {
        printf("%d X %d = %d\n", n, i,

```

```
    n*i);

    i = i + 1;
    if (i > 10) {
        break;
    }
}

return 0;
```

প্রোগ্রাম : ৮.৮

এখন আমরা আরেকটি কাজ করব। for লুপ ব্যবহার করে 5-এর নামতায় যে গুণ করেছি ( $n^i$ ) সেটি না করে কেবল যোগ করে প্রোগ্রামটি লিখব। তোমরা কি অবাক হচ্ছ যে নামতার প্রোগ্রাম আবার গুণ ছাড়া কীভাবে হবে? আমরা কিন্তু  $5 \times 3$ -কে লিখতে পারি  $5 + 5 + 5$ । আমি কী করতে যাচ্ছি তা বুঝতে পারছ নিশ্চয়ই। প্রোগ্রামটি লিখে ফেলি :

```
#include <stdio.h>
int main()
{
    int m, n = 5;
    int i;

    m = 0;
    for(i = 1; i <= 10; i = i + 1) {
        m = m + n;
        printf("%d X %d = %d\n", n, i, m);
    }
    return 0;
}
```

প্রোগ্রাম : 8.৯

প্রোগ্রামটিতে আমরা গুণ না করে যোগ করলাম। কম্পাইল ও রান করে দেখো। কাজ করবে ঠিকঠাক। কোনো সংখ্যার গুণিতকগুলো যেমন গুণ করে বের করা যায়, তেমনই যোগ করেও করা যায়। আমরা যদি কোনো প্রোগ্রামে দেখি যে গুণ না করে যোগ করলেই কাজ হচ্ছে, তাহলে যোগ করাই ভালো। কারণ কম্পিউটারের প্রসেসর একটি যোগ করতে যে সময় নেয়, একটি গুণ করতে তার চেয়ে অনেক বেশি সময় নেয়। যদিও তুমি হয়তো প্রোগ্রাম রান করার সময় তা বুঝতে পারো না। কম্পিউটারের প্রসেসর সম্পর্কে বিস্তারিত লেখাপড়া করলে বিষয়টা জানতে পারবে। আপাতত এটি জানলেই চলবে যে একটি গুণ করার চেয়ে একটি যোগ করা ভালো, কারণ যোগ করতে কম্পিউটার অপেক্ষাকৃত কম সময় নেয়।

তো আমরা for লুপ শিখে ফেললাম। এখন আমরা চেষ্টা করব শুধু নির্দিষ্ট একটি সংখ্যার নামতা না লিখে 1 থেকে 20 পর্যন্ত সবগুলো সংখ্যার নামতা একবারে লিখে ফেলতে। অর্থাৎ n-এর মান 5 নির্দিষ্ট না করে 1 থেকে 20 পর্যন্ত হবে। এটি করার একটি বোকা পদ্ধতি (নাকি ঢোরা পদ্ধতি?) হচ্ছে নামতা লেখার অংশটি বারবার কপি-পেস্ট করা। কিন্তু আমরা এটি করব লুপের ভেতর লুপ ব্যবহার করে। একটি লুপের সাহায্যে n-এর মান 1 থেকে 20 পর্যন্ত এক করে বাড়াব। আর তার ভেতর n-এর একটি নির্দিষ্ট মানের জন্য নামতাটা লিখব।

```
#include <stdio.h>
int main()
{
    int n, i;
    for(n = 1; n <= 20; n = n + 1) {
        for(i = 1; i <= 10; i = i + 1) {
            printf("%d X %d = %d\n", n, i,
n*i);
        }
    }
}
```

৬৮

```
}
```

```
return 0;
```

```
}
```

প্রোগ্রাম : 8.১০

এখন তোমরা প্রোগ্রামটি চালাও। তারপর তোমাদের কাজ হবে গুণ না করে কেবল যোগ ব্যবহার করে প্রোগ্রামটি লেখা।

আমরা এখানে একটি for লুপের ভেতর আরেকটি for লুপ, যাকে নেস্টেড লুপও (nested loop) বলে, সেটি ব্যবহার করলাম। তো আমরা চাইলে for লুপের ভেতর for বা while অথবা while লুপের ভেতর for বা while লুপ একাধিকবার ব্যবহার করতে পারি। অবশ্য সেটি কখনোই চার বা পাঁচবারের বেশি দরকার হওয়ার কথা না। নেস্টেড লুপ দিয়ে আমরা এখন আরেকটি প্রোগ্রাম লিখব। 1, 2, 3 এই তিনটি সংখ্যার সব বিন্যাস (permutation) বের করার প্রোগ্রাম। বিন্যাসগুলো ছোট থেকে বড় ক্রমে দেখাতে হবে অর্থাৎ প্রোগ্রামটির আউটপুট হবে এই রুক্ম :

1, 2, 3

1, 3, 2

2, 1, 3

2, 3, 1

3, 1, 2

3, 2, 1

এই প্রোগ্রাম অনেকভাবে লেখা যেতে পারে, কিন্তু আমরা এখন পর্যন্ত যতটুকু প্রোগ্রামিং শিখেছি, তাতে নেস্টেড লুপের ব্যবহারই সবচেয়ে ভালো সমাধান।

এখানে আমরা প্রথম সংখ্যাটির জন্য একটি লুপ, দ্বিতীয় সংখ্যাটির জন্য প্রথম লুপের ভেতরে একটি লুপ এবং তৃতীয় সংখ্যাটির জন্য দ্বিতীয় লুপের ভেতর আরেকটি লুপ ব্যবহার করব।

```
#include <stdio.h>
```

৬৯

```

int main()
{
    int a, b, c;
    for (a = 1; a <= 3; a++) {
        for (b = 1; b <= 3; b++) {
            for (c = 1; c <= 3; c++) {
                printf ("%d, %d, %d\n", a,
b, c);
            }
        }
    }
    return 0;
}

```

প্রোগ্রাম : 8.11

এখন প্রোগ্রামটি রান করলে আমরা এই রকম আউটপুট পাব:

```

1, 1, 1
1, 1, 2
1, 1, 3
1, 2, 1
1, 2, 2
1, 2, 3
1, 3, 1
1, 3, 2
1, 3, 3
2, 1, 1
2, 1, 2

```

```

2, 1, 3
2, 2, 1
2, 2, 2
2, 2, 3
2, 3, 1
2, 3, 2
2, 3, 3
3, 1, 1
3, 1, 2
3, 1, 3
3, 2, 1
3, 2, 2
3, 2, 3
3, 3, 1
3, 3, 2
3, 3, 3

```

কিন্তু আমরা তো আসলে এই রকম জিনিস চাহিই না। a-এর মান যখন  
 1 তখন b ও c-এর মান 1 হবে না, আবার b ও c-এর মানও সমান  
 হবে না। মানে a, b ও c আলাদা হবে। তাহলে আমরা লুপের ভেতর  
 শর্তগুলো একটু পরিবর্তন করব। দ্বিতীয় লুপের শর্ত  $b \leq 3$ -এর সঙ্গে  
 আরেকটি শর্ত জুড়ে দেব,  $b \neq a$ ।  $b \leq 3 \&& b \neq a$  মানে b-  
 এর মান 3-এর চেয়ে ছোট বা সমান হবে এবং b-এর মান a-এর  
 মানের সমান হবে না। তৃতীয় লুপে আমরা এখন শর্ত দেব,  $c \leq 3$   
 $\&& c \neq a \&& c \neq b$ , মানে “c-এর মান 3-এর ছোট বা সমান  
 হতে হবে” এবং “c-এর মান 3-এর মানের সমান হওয়া চলবে না”  
 এবং “c-এর মান b-এর মানের সমান হলেও চলবে না”。 তাহলে  
 আমাদের প্রোগ্রামটির চেহারা দাঁড়াবে এই রকম :

```

#include <stdio.h>
int main()
{
    int a, b, c;
    for (a = 1; a <= 3; a++) {
        for (b = 1; b <= 3 && b != a; b++) {
            for (c = 1; c <= 3 && c != a && c != b; c++) {
                printf ("%d, %d, %d\n", a, b, c);
            }
        }
    }
    return 0;
}

```

প্রোগ্রাম : ৪.১২

রান করলে আমরা আউটপুট কী দেখব?

3, 2, 1

মাত্র একটি লাইন! আমরা প্রোগ্রামটি ঠিক করতে গিয়ে কামেলা পাকিয়ে ফেলেছি মনে হচ্ছে। তোমরা কি একটু চিন্তা করে কামেলা কারণ বের করতে পারবে?

প্রথমে a-এর মান 1 তাই a <= 3 সত্য। প্রোগ্রামটি প্রথম লুপের ভেতর চুকে গেল। তারপর দ্বিতীয় লুপের ভরতে b-এর মান 1। b <= 3 সত্য। কিন্তু b != a মিথ্যা। কারণ a ও b-এর মান তো সমান, দুটোর মানই 1। তাই প্রোগ্রামটি আর দ্বিতীয় লুপের ভেতর চুকবে না। এরপর a-এর মান 1 বাড়ল (a++)। a <= 3 সত্য (a-এর মান 2)। এখন দ্বিতীয় লুপ শুরু হবে। b-এর মান 1। এবাবে b <= 3 এবং b != a দুটি শর্তই সত্য। প্রোগ্রামটি দ্বিতীয় লুপের ভেতর চুকে যাবে।

দ্বিতীয় লুপের ভরতে C-এর মান 1। c <= 3 সত্য, c != a সত্য কিন্তু c != b মিথ্যা (দুটোর মানই 1)। তাই প্রোগ্রামটি দ্বিতীয় লুপ থেকে বের হয়ে যাবে— কেবল তিনটি শর্ত সত্য হলেই প্রোগ্রামটি দ্বিতীয় লুপের ভেতর চুকবে এবং a, b ও c-এর মান প্রিণ্ট করবে। এভাবে কিছুক্ষণ গবেষণা করলে তোমরা দেখবে যে যখন a-এর মান 3, b-এর মান 2 এবং c-এর মান 1, তখনই কেবল সব শর্ত সত্য হয় আর আমরা আউটপুট পাই : 3, 2, 1।

আসলে দ্বিতীয় লুপে আমরা b-এর মান 3-এর মানের সমান হলে লুপ থেকে বের হয়ে যাচ্ছি। সেই কাজটি করা ঠিক হচ্ছে না। আমাদের উচিত দুটো মান সমান হলে পরবর্তী মানের জন্য চেষ্টা করা। আর মান দুটো সমান না হলেই কেবল পরবর্তী কাজ করা। তাহলে আমরা লিখতে পারি:

```

for (b = 1; b <= 3; b++) {
    if (b != a) { /* b-এর মান a-এর মানের সমান না হলেই
    ভেতরের অংশে প্রোগ্রামটি চুকবে। */
        for (c = 1; c <= 3; c++) {
            if (c != a && c != b) { /* c-এর মান a-এর মানের সমান
            না হলে এবং c-এর মান b-এর মানের সমান না হলেই কেবল
            ভেতরের অংশে প্রোগ্রামটি চুকবে। */
                printf ("%d, %d, %d\n", a, b, c);
            }
        }
    }
}

```

তাহলে আমাদের পুরো প্রোগ্রামটি দীঢ়াচ্ছে এই রকম :

```

#include <stdio.h>
int main() {

```

```

int a, b, c;
for (a = 1; a <= 3; a++) {
    for (b = 1; b <= 3; b++) {
        if (b != a) {

            for (c = 1; c <= 3; c++) {
                if (c != b && c != a) {
                    printf ("%d, %d, %d\n",
a, b, c);
                }
            }
        }
    }
}
return 0;
}

```

প্রোগ্রাম : 8.13

প্রোগ্রামটি চালালে আমরা নিচের আউটপুট দেখব, যেটি আমরা চাচ্ছিলাম।

1, 2, 3  
 1, 3, 2  
 2, 1, 3  
 2, 3, 1  
 3, 1, 2  
 3, 2, 1

যাক, অবশ্যে আমাদের সমস্যার সমাধান হলো। তবে আমরা কিন্তু আরও সহজেই সমাধান করতে পারতাম, এভাবে—

```

#include <stdio.h>
int main()
{
    int a, b, c;
    for (a = 1; a <= 3; a++) {
        for (b = 1; b <= 3; b++) {
            for (c = 1; c <= 3; c++) {
                if(b != a && c != a && c != b) {
                    printf ("%d, %d, %d\n",
a, b, c);
                }
            }
        }
    }
    return 0;
}

```

প্রোগ্রাম : 8.14

এখানে আমাদের বেশি চিন্তা করতে হলো না। কেবল প্রিন্ট করার সময় a, b, c তিনটির মান পরম্পরারের সমান নয়, সেটি নিশ্চিত করে নিলেই হলো! বুদ্ধিটা ভালোই, তবে এটির চেয়ে আমাদের আগের প্রোগ্রামটি কম্পিউটারকে দিয়ে কম কাজ করায়, তাই চলতেও কম সময় লাগে, বা কম্পিউটারের ভাষায় বললে রান টাইম (run time) কম। আসলে একটি প্রোগ্রাম চলতে কেমন সময় লাগবে সেটি নির্ভর করে মূলত প্রোগ্রামটি মোট কয়টি অ্যাসাইনমেন্ট অপারেশন আর কয়টি কম্পারিজন অপারেশন করল তার ওপর।

## অধ্যায় পাঁচ : একটুখানি গণিত

এই অধ্যায়ে আমরা প্রোগ্রামিংয়ের নতুন কিছু শিখব না। এখন পর্যন্ত আমরা যতটুকু প্রোগ্রাম শিখেছি, তা দিয়েই কিছু সহজ-সরল গণিতিক সমস্যার সমাধান করব।

১)  $x + y = 15$ ,  $x - y = 5$  হলে  $x$  ও  $y$ -এর মান কত?

সমীকরণ দুটি যোগ করলে পাই  $2x = 20$ , বা  $x = 10$ । আবার বিয়োগ করলে পাই,  $2y = 10$ , বা  $y = 5$ । এখন একটি প্রোগ্রাম লিখতে হবে, যেখানে  $x + y$  ও  $x - y$ -এর মান দেওয়া থাকবে,  $x$  ও  $y$ -এর মান বের করতে হবে। আমি প্রোগ্রামটি একটু পরে লিখে দেব। এর মধ্যে তুমি নিজে লেখার চেষ্টা করো। সহজ প্রোগ্রাম।

২)  $4x + 5y = 14$ ,  $5x + 6y = 17$  হলে  $x$  ও  $y$ -এর মান কত?

সমীকরণ দুটিকে আমরা এভাবে লিখতে পারি :

$a_1x + b_1y = c_1$ ,  $a_2x + b_2y = c_2$ । তোমরা বিভিন্নভাবে এর সমাধান করতে পারো। এর মধ্যে দুটি জনপ্রিয় উপায় হচ্ছে প্রতিস্থাপন (substitution) ও নির্ণয়করে (determinant) সাহায্যে সমাধান। পদ্ধতিগুলো জানা না থাকলে ক্লাস এইট বা নাইনের গণিত বই দেখো। সমাধান করলে দেখবে,

$x = (b_2c_1 - b_1c_2) / (a_1b_2 - a_2b_1)$  এবং  $y = (a_1c_2 - a_2c_1) / (a_1b_2 - a_2b_1)$ । এখন  $a_1$ ,  $a_2$ ,  $b_1$ ,  $b_2$ ,  $c_1$ ,  $c_2$ -এর জায়গায় নির্দিষ্ট মান বসিয়ে দিলেই  $x$  ও  $y$ -এর মান পেয়ে যাবে।

এই ধরনের সমীকরণ সমাধানের জন্যও আমরা একটি প্রোগ্রাম লিখব, যার ইনপুট হবে  $a_1$ ,  $a_2$ ,  $b_1$ ,  $b_2$ ,  $c_1$ ,  $c_2$  এবং আউটপুট হবে  $x$  ও  $y$ -এর মান। এটিও সহজ প্রোগ্রাম। নিজে চেষ্টা করো।

আশা করি, তোমরা দুটি সমস্যারই সমাধান নিজে করে ফেলতে পারবে। এখন আমি প্রথম সমস্যার কোড দিচ্ছি :

```
#include <stdio.h>
int main()
```

```
{  
    double x, y, x_plus_y, x_minus_y;  
    printf("Enter the value of x + y: ");  
    scanf("%lf", &x_plus_y);  
    printf("Enter the value of x - y: ");  
    scanf("%lf", &x_minus_y);  
    x = (x_plus_y + x_minus_y) / 2;  
    y = (x_plus_y - x_minus_y) / 2;  
    printf("x = %0.2lf, y = %0.2lf\n", x,  
y);  
    return 0;  
}
```

প্রোগ্রাম : ৫.১

সমাধান খুবই সহজ। তবে লক্ষ করো যে আমি ভেরিয়েবলের ডাটা টাইপ int ব্যবহার না করে double ব্যবহার করেছি।

এবাবে দ্বিতীয় সমস্যার কোড :

```
#include <stdio.h>
int main()
{
    double a1, a2, b1, b2, c1, c2, x, y;
    printf("a1 = ");
    scanf("%lf", &a1);
    printf("a2 = ");
    scanf("%lf", &a2);
    printf("b1 = ");
    scanf("%lf", &b1);
    printf("b2 = ");
    scanf("%lf", &b2);
    x = (b1*c2 - b2*c1) / (a1*b2 - a2*b1);
    y = (a1*c2 - a2*c1) / (a1*b2 - a2*b1);
}
```

```

    scanf("%lf", &b2);
    printf("c1 = ");
    scanf("%lf", &c1);
    printf("c2 = ");
    scanf("%lf", &c2);
    x = (b2 * c1 - b1 * c2) / (a1 * b2 - a2
    * b1);
    y = (a1 * c2 - a2 * c1) / (a1 * b2 - a2
    * b1);
    printf("x = %0.2lf, y = %0.2lf\n", x,
    y);
    return 0;
}

```

প্রোগ্রাম : ৫.২

এটিও সহজ প্রোগ্রাম! তবে তোমরা দেখো,  $(a1 * b2 - a2 * b1)$ -এর মান আমি দুবার বের করেছি ( $x$ -এর মান বের করার সময়, আবার  $y$ -এর মান বের করার সময়)। কাজটি একবারেই করা যেত এবং একবারে করলেই ভালো, তাহলে আমাদের প্রোগ্রাম দুটি গুণ ও একটি বিয়োগের কাজ করবে। আবার  $(a1 * b2 - a2 * b1)$ -এর মান যদি শূন্য হয়, তাহলে একটি ঝামেলা হয়ে যাচ্ছে, কারণ কোনো কিছুকে তো শূন্য দিয়ে ভাগ করা যায় না। তাই ওই মানটি শূন্য হলে আসলে সমীকরণের কোনো সমাধান নেই। এবার প্রোগ্রামটি আরও ভালোভাবে লিখে ফেলি।

```

#include <stdio.h>
int main()
{
    double a1, a2, b1, b2, c1, c2, d, x, y;
    printf("a1 = ");
    scanf("%lf", &a1);

```

৭৮

```

    printf("a2 = ");
    scanf("%lf", &b2);
    printf("b1 = ");
    scanf("%lf", &b1);
    printf("b2 = ");
    scanf("%lf", &b2);
    printf("c1 = ");
    scanf("%lf", &c1);
    printf("c2 = ");
    scanf("%lf", &c2);
    d = a1 * b2 - a2 * b1;
    if ((int) d == 0) {
        printf("Value of x and y can not be
determined.\n");
    }
    else {
        x = (b2 * c1 - b1 * c2) / d;
        y = (a1 * c2 - a2 * c1) / d;
        printf("x = %0.2lf, y = %0.2lf\n",
        x, y);
    }
    return 0;
}

```

প্রোগ্রাম : ৫.৩

এখনে একটি ব্যাপার খেয়াল করো। আমি if-এর ভেতর লিখেছি  $(int) d == 0$ । এখনে আমি প্রথমে  $d$  (যা একটি double টাইপের ডেরিয়েবল)-কে ইন্টিজারে টাইপ কাস্ট করে তারপর তার মানটি 0-এর সমান কি না, তা পরীক্ষা করেছি। পরীক্ষাটা এভাবেও করা যেত :

৭৯

if ( $d == 0.0$ ) তবে এতে মাঝে মাঝে ঝামেলা হয়, ফ্লোটিং পয়েন্ট-সংক্রান্ত হিসাব-নিকাশের জন্য। তোমরা কম্পিউটার আর্কিটেকচার নিয়ে লেখাপড়া করলে বিষয়টা বুঝতে পারবে।

তোমাদের মনে একটি প্রশ্ন আসতে পারে যে এই সহজ সমস্যাগুলো প্রোগ্রামিং করে সমাধান করে কী লাভ? আসলে একবার প্রোগ্রাম লিখে ফেলার পরে কিন্তু আর সমাধান করতে হয় না। তারপর শুধু ইনপুট দেবে, প্রোগ্রামটি নিজেই সমস্যার সমাধান করে তোমাকে আউটপুট দেবে।

৩) আমি যদি তোমাকে দশ হাজার টাকা খণ্ড দিই 35% সুদে এবং টাকাটা পাঁচ বছর সময়ের মধ্যে তোমাকে সুদে-আসলে পরিশোধ করতে বলি, তাহলে পাঁচ বছরে মোট কত টাকা তোমার দিতে হবে এবং প্রতি মাসে কত টাকা দিতে হবে? খণ্টা যদি জটিল কিছু না হয়, তাহলে তোমার মোট পরিশোধ করতে হবে  $10000 + 10000 * 35 / 100$  টাকা। এই সহজ-সরল খণ্ডের জন্য একটি প্রোগ্রাম লিখে ফেলা যাক:

```
#include <stdio.h>

int main()
{
    double loan_amount, interest_rate,
    number_of_years, total_amount,
    monthly_amount;

    printf("Enter the loan amount: ");
    scanf("%lf", &loan_amount);

    printf("Enter the interest rate: ");
    scanf("%lf", &interest_rate);

    printf("Number of years: ");
    scanf("%lf", &number_of_years);

    total_amount = loan_amount +
    loan_amount * interest_rate / 100.00;
```

৮০

```
monthly_amount = total_amount /
(number_of_years * 12);
printf("Total amount: %0.2lf\n",
total_amount);
printf("Monthly amount: %0.2lf\n",
monthly_amount);
return 0;
}
```

প্রোগ্রাম : ৫.৮

আমাদের ফর্মুলাতে একটু সমস্যা আছে। আসলে 35% সুদ দিতে হলে সেটা বার্ষিক সুদ হবে। অর্থাৎ প্রতিবছর মোট খণ্ডের ওপর 35% সুদ দেওয়া লাগবে। তাহলে দেখা যাচ্ছে পাঁচ বছরে তোমার মোট পরিশোধ করতে হবে  $10000 + 10000 * 35 * 5 / 100$  টাকা। এখন এই ফর্মুলা অনুযায়ী প্রোগ্রাম লিখে ফেলো।

তবে বাস্তবে খণ্ডের হিসাব-নিকাশ কিন্তু এত সরল নয়। তুমি ব্যাংক থেকে খণ্ড নিতে গেলেই সেটি টের পাবে।

৪) পদার্থবিজ্ঞানের একটি সমস্যার সমাধান করা যাক।

কোনো বস্তু u আদিবেগে (initial velocity) এবং a ত্বরণে (acceleration) যাত্রা শুরু করল (ত্বরণের মান সব সময় a থাকবে, বাড়বে বা কমবে না)। t সময় পরে এর বেগ যদি v হয় তাহলে  $2t$  সময়ে বস্তুটি কত দূরত্ব অতিক্রম করবে? (সমস্যাটি দিয়েছেন শাহরিয়ার মঙ্গুর, এটি ভ্যালাডলিড অনলাইন জার্জের 10071 নম্বর সমস্যা)।

$2t$  সময়ে অতিক্রান্ত দূরত্ব হবে  $v \times 2t$ । এটি প্রমাণ করে ফেলো। তারপর আবার পড়া শুরু করো।

নবম-দশম শ্রেণীর পদার্থবিজ্ঞান বইতে তোমরা দৃষ্টি সূত্র পাবে :

$$v = u + at$$

$$s = ut + 0.5 at^2 \quad (\text{এখানে } s \text{ হচ্ছে } t \text{ সময়ে অতিক্রান্ত দূরত্ব})$$

৮১

তাহলে 21 সহয় পরে অতিক্রান্ত দূরত্ব হবে

$$u \times 21 + 0.5 \times a \times (21)^2 = u \times 21 + 0.5 \times a \times 41^2 = u \times 21 + a \times 21^2 = 21(u + a) = 21v$$

এখন, তোমার একটি প্রোগ্রাম লিখতে হবে, যেখানে  $v$  ও  $l$ -এর মান ইনপুট হিসেবে দেওয়া হবে, 21 সহয় অতিক্রান্ত দূরত্ব নির্ণয় করতে হবে। প্রোগ্রামটি নিচে নিখে দেখে ফেলো।

(১)  $1 + 2 + 3 + \dots + 998 + 999 + 1000$  এই ধারার সমষ্টি কত?

তোমরা ধারা ধারা ঘোশফলের সূত্র জানো, তারা টাট করে বলে দিতে পারবে, এই ধারাটির ঘোশফল হচ্ছে  $1000 \times 1001 / 2$ । তাহলে এব জন্য একটি প্রোগ্রাম লিখে ফেলো যাক, যেখানে শেষ গদের মান হবে ইনপুট আব আউটপুট হবে ঘোশফল।

```
#include <stdio.h>
int main()
{
    int n, sum;
    scanf(" %d ", &n);
    sum = (n * (n + 1)) / 2;
    printf(" Summation is %d(%d) ", sum);
    return 0;
}
```

প্রোগ্রাম : ৫.৫

ধারা ঘোশফল নির্ণয়ের সূত্র জানা না থাকলে আমরা নৃশ ব্যবহার করে প্রোগ্রামটি লিখতে পারি।

```
#include <stdio.h>
int main()
```

```
int l, n, sum;
scanf(" %d ", &n);
for(l = 1, sum = 0; l <= n; l++)
{
    sum = sum + l;
}
printf(" Summation is %d(%d) ", sum);
return 0;
```

প্রোগ্রাম : ৫.৬

সূত্রবাঁধ ধারা ধারা সমষ্টি নিয়ে আব চিন্তা নেই। সুই যদি একটি পদের মান তাৰ আশেৰ পদেৰ কেঁফে কত করে বাঢ়ছে, সেটি বেৰ কৰতে পাবো, তাহলেই নৃশ ব্যবহাৰ করে ঘোশফল বেৰ করে ফেলতে পাবো। তবে সূত্র বেৰ কৰতে শাৰলো নৃশ ব্যবহাৰ না কৰাই ভালো। কাৰণ প্ৰথম প্ৰোগ্ৰামটি দেখো (যেখানে সূত্র ব্যবহাৰ কৰেছি); যেখানে একটি ঘোশ, একটি তুল আৰ একটি ভাগ কৰতে হয়েছে,  $n$ -এর মান ধৰ বড়ই হোক না কেন। আব দ্বিতীয় প্ৰোগ্ৰাম (যেখানে নৃশ ব্যবহাৰ কৰেছি)  $n$ -এর মান ধৰ, ততকাৰ ঘোশ কৰতে হয়েছে, আবাৰ সেই ঘোশফলটি sum ভেবিয়েৰলে বাখতে হয়েছে (ভেবিয়েৰলে কোনো মান বাখতেও কিন্তু একটি সহজ নাশে)।

এখন তোমাদেৰ একটি সহজ প্ৰোগ্রাম লিখতে হবে: আব  $n$ -সহযোগ ধনাত্মক বেজোড় সংখ্যাৰ ঘোশফল নিৰ্ণয়েৰ প্ৰোগ্ৰাম।  $n$ -এর মান হবে ইনপুট, আব ঘোশফল হবে আউটপুট।

(২) আমাদেৰ এবাৰকাৰ প্ৰোগ্ৰামটি হবে কালৰিয়ান (Celsius) থেকে ফাৰেনহাইট (Farenheit) কল্পনাৰ কৰাৰ প্ৰোগ্ৰাম।

কেলৰিয়ানকে ফাৰেনহাইট কল্পনাৰে সূত্র হচ্ছে:  ${}^{\circ}F = ({}^{\circ}C \times 1.8) + 32$ ।

```
#include <stdio.h>
```

তাহলে  $2t$  সময় পরে অতিক্রান্ত দূরত্ব হবে

$$u \times 2t + 0.5 \times a \times (2t)^2 = u \times 2t + 0.5 \times a \times 4t^2 = u \times 2t + a \times 2t^2 = 2t(u + at) = 2tv$$

এখন, তোমার একটি প্রোগ্রাম লিখতে হবে, যেখানে  $v$  ও  $t$ -এর মান ইনপুট হিসেবে দেওয়া হবে,  $2t$  সময়ে অতিক্রান্ত দূরত্ব নির্ণয় করতে হবে। প্রোগ্রামটি নিজে নিজে লিখে ফেলো।

৫)  $1 + 2 + 3 + \dots + 998 + 999 + 1000$  এই ধারার সমষ্টি কত?

তোমরা ধারার যোগফলের সূত্র জানো, তারা চট করে বলে দিতে পারবে, এই ধারাটির যোগফল হচ্ছে  $1000 \times 1001 / 2$ । তাহলে এর জন্য একটি প্রোগ্রাম লিখে ফেলা যাক, যেখানে শেষ পদের মান হবে ইনপুট আর আউটপুট হবে যোগফল।

```
#include <stdio.h>
int main()
{
    int n, sum;
    scanf("%d", &n);
    sum = (n * (n + 1)) / 2;
    printf("Summation is %d\n", sum);
    return 0;
}
```

প্রোগ্রাম : ৫.৫

ধারার যোগফল নির্ণয়ের সূত্র জানা না থাকলে আমরা লুপ ব্যবহার করে প্রোগ্রামটি লিখতে পারি।

```
#include <stdio.h>
int main()
{
```

```
int i, n, sum;
scanf("%d", &n);
for(i = 1, sum = 0; i <= n; i++) {
    sum = sum + i;
}
printf("Summation is %d\n", sum);
return 0;
```

প্রোগ্রাম : ৫.৬

সুতরাং ধারার সমস্যা নিয়ে আর চিন্তা নেই। তুমি যদি একটি পদের মান তার আগের পদের চেয়ে কত করে বাঢ়ছে, সেটি বের করতে পারো, তাহলেই লুপ ব্যবহার করে যোগফল বের করে ফেলতে পারবে। তবে সূত্র বের করতে পারলে লুপ ব্যবহার না করাই ভালো। কারণ প্রথম প্রোগ্রামটি দেখো (যেখানে সূত্র ব্যবহার করেছি)। সেখানে একটি যোগ, একটি গুণ আর একটি ভাগ করতে হয়েছে,  $n$ -এর মান যত বড়ই হোক না কেন। আর দ্বিতীয় প্রোগ্রামে (যেখানে লুপ ব্যবহার করেছি)  $n$ -এর মান যত, ততবার যোগ করতে হয়েছে, আবার সেই যোগফলটি sum ভেরিয়েবলে রাখতে হয়েছে (ভেরিয়েবলে কোনো মান রাখতেও কিন্তু একটু সময় লাগে)।

এখন তোমাদের একটি সহজ প্রোগ্রাম লিখতে হবে। প্রথম  $n$ -সংখ্যক ধনাত্মক বেজোড় সংখ্যার যোগফল নির্ণয়ের প্রোগ্রাম।  $n$ -এর মান হবে ইনপুট, আর যোগফল হবে আউটপুট।

৬) আমাদের এবারকার প্রোগ্রামটি হবে তাপমাত্রাকে সেলসিয়াস (Celsius) থেকে ফারেনহাইটে (Farenheit) রূপান্তর করার প্রোগ্রাম।

সেলসিয়াসকে ফারেনহাইটে রূপান্তরের সূত্র হচ্ছে :  ${}^{\circ}\text{F} = ({}^{\circ}\text{C} \times 1.8) + 32$ ।

```
#include <stdio.h>
```

```

int main()
{
    double celsius, farenheit;
    printf("Enter the temperature in
celsius: ");
    scanf("%lf", &celsius);
    farenheit = 1.8 * celsius + 32;
    printf("Temperature in farenheit is:
%lf\n", farenheit);
    return 0;
}

```

প্রোগ্রাম : ৫.৭

এখন তোমাদের কাজ হচ্ছে ফারেনহাইট থেকে সেলসিয়াসে  
রূপান্তরের প্রোগ্রাম লেখা।

৭) এখন আমরা দুটি সংখ্যার গসাগু (GCD → Greatest Common Divisor বা HCF → Highest Common Factor)  
ও লসাগু (LCM → Least Common Multiple) নির্ণয় করার  
জন্য প্রোগ্রাম লিখব।

দুটি সংখ্যার গসাগু হচ্ছে যেসব সংখ্যা দিয়ে ওই দুটি সংখ্যা নিঃশেষে  
বিভাজ্য হয়, তাদের মধ্যে সবচেয়ে বড় সংখ্যা। তাহলে আমরা যেটি  
করব, দুটি সংখ্যা a ও b নেব। তারপর এদের মধ্যে যেটি ছোট, সেই  
মানটি আবার x ভেরিয়েবলে রাখব। গসাগু-এর মান x-এর চেয়ে বড়  
হওয়া সম্ভব নয় (5 ও 10-এর গসাগু-এর মান নিশ্চয়ই 5-এর চেয়ে  
বড় হবে না)। এখন a ও b, x দিয়ে নিঃশেষে বিভাজ্য হয় কি না ( $a \bmod x == 0$  এবং  $b \bmod x == 0$ ) সেটি পরীক্ষা করব। যদি হয়,  
তাহলে আমরা গসাগু পেয়ে গেছি। যদি a ও b উভয়েই নিঃশেষে  
বিভাজ্য না হয়, তখন x-এর মান এক কমিয়ে পরীক্ষা করব। যতক্ষণ  
না আমরা গসাগু পাচ্ছি x-এর মান কমাতেই থাকব। একসময় আমরা  
গসাগু পাবই, কারণ x-এর মান যখন 1 হবে, তখন তো x দিয়ে a ও

b দুটি সংখ্যাই নিঃশেষে বিভাজ্য। তোমরা কি প্রোগ্রামটি নিজে লেখার  
চেষ্টা করবে? না পারলে আমার কোড দেখো :

```

#include <stdio.h>
int main()
{
    int a, b, x, gcd;
    scanf("%d %d", &a, &b);
    if (a < b) {
        x = a;
    }
    else {
        x = b;
    }
    for(; x >= 1; x--) {
        if (a % x == 0 && b % x == 0) {
            gcd = x;
            break;
        }
    }
    printf("GCD is %d\n", gcd);
    return 0;
}

```

প্রোগ্রাম : ৫.৮

প্রোগ্রামে দেখো gcd পাওয়ার সঙ্গে সঙ্গে লুপ থেকে বের হয়ে যেতে  
হবে (আমি break ব্যবহার করেছি এই জন্য)। break ব্যবহার না  
করলে কী হবে সেটি পরীক্ষা করে দেখো।

তবে গসাগু বের করার জন্য আমি যে পদ্ধতি ব্যবহার করেছি সেটি খুব

সহজ হলেও ইফিশিয়েন্ট (efficient) নয়। যেমন, সংখ্যা দুটি খুব বড় হলে এবং সহমৌলিক (co-prime) হলে লুপটি কিন্তু অনেকবার ঘূরবে। কারণ সহমৌলিক হলে গসাগু হবে 1। তোমরা নিচয়ই জানো যে, দুটি সংখ্যার মধ্যে 1 ছাড়া আর কোনো সাধারণ উৎপাদক না থাকলে সংখ্যা দুটি সহমৌলিক।

গসাগু বের করার জন্য ইউক্লিডের একটি চমৎকার পদ্ধতি আছে। ইউক্লিড ভাগশেষ উপপাদ্যের (division algorithm) সাহায্যে গসাগু বের করার উপায় দেখিয়েছেন। এই পদ্ধতিতে খুব সহজে গসাগু বের করা যায় এবং প্রোগ্রামটিও বেশ ইফিশিয়েন্ট হয়। এর জন্য দুটি জিনিস জানা লাগবে :

a ও 0-এর গসাগু-এর মান a।

a ও b-এর গসাগু = b ও a % b-এর গসাগু।

তাহলে প্রোগ্রামে যেটি করতে হবে, একটি লুপের সাহায্যে a-এর মান b আর b-এর মান a%b বসিয়ে যেতে হবে, যতক্ষণ না b-এর মান শূন্য হয়। b-এর মান শূন্য হলেই বুঝে যাব যে গসাগু হচ্ছে a (এটা কিন্তু প্রোগ্রাম শুরুর সময় a-এর মান না, b-এর মান যখন শূন্য হবে সেই সময় a-এর মান)।

```
#include <stdio.h>
int main()
{
    int a, b, t, x, gcd;

    scanf("%d %d", &a, &b);
    if (a == 0) gcd = a;
    else if (b == 0) gcd = b;
    else {
        while (b != 0) {
            t = b;
            b = a % b;
            a = t;
        }
    }
}
```

b = a % b;

a = t;

}

gcd = a;

}

printf("GCD is %d\n", gcd);

return 0;

}

প্রোগ্রাম : ৫.৯

এই প্রোগ্রামটি আরও ইফিশিয়েন্ট করার চেষ্টা করো।

এবার লসাগু বের করার প্রোগ্রাম। তোমরা নিচয়ই স্কুলে শিখেছ, কীভাবে লসাগু বের করতে হয়। সেই পদ্ধতি অবলম্বন করে প্রোগ্রাম লিখে ফেলো। আর যারা সেই পদ্ধতি জানো না, তাদের জন্য একটি সূত্র বলে দিচ্ছি।

দুটি সংখ্যার লসাগু  $\times$  দুটি সংখ্যার গসাগু = সংখ্যা দুটির গুণফল।

আশা করি, লসাগু বের করার প্রোগ্রাম লিখতে আর সমস্যা হবে না।

## অধ্যায় ছয় : আবৰে

এতক্ষণে আমাদের প্রোগ্রামিং জ্ঞান-বুদ্ধি একটু বেড়েছে। চলো, এবাৰ তাহলে কিছু জনসেবামূলক কৰ্মকাণ্ড কৰা যাক। আমোৰা ক্লুলেৰ প্ৰিয় গণিত শিক্ষকেৰ জন্য পৱীক্ষাৰ ফলাফল বেৱে কৰাৰ প্ৰোগ্ৰাম লিখে দেব। ওই ক্লুলে প্ৰথম সাময়িক, দ্বিতীয় সাময়িক ও বাৰ্ষিক এই তিনটি পৱীক্ষাই 100 নম্বৰেৰ হয়। তাৰপৰ বাৰ্ষিক পৱীক্ষাৰ 50%, দ্বিতীয় সাময়িক পৱীক্ষাৰ 25% ও প্ৰথম সাময়িক পৱীক্ষাৰ 25% নিয়ে চূড়ান্ত ফলাফল প্ৰকাশ কৰা হয়। তাহলে আমাদেৱ প্ৰোগ্ৰামেৰ ইনপুট হচ্ছে ওই তিনটি পৱীক্ষাৰ নম্বৰ। আমাদেৱকে চূড়ান্ত ফলাফল দেখাতে হবে। এটি কোনো ব্যাপারই নয় :

```
#include <stdio.h>
int main()
{
    int ft_marks, st_marks, final_marks;
    double total_marks;
    ft_marks = 80;
    st_marks = 74;
    final_marks = 97;
    total_marks = ft_marks / 4.0 + st_marks
    / 4.0 + final_marks / 2.0;
    printf("%0.1f\n", total_marks);
    return 0;
}
```

প্ৰোগ্ৰাম : ৬.১

প্ৰোগ্ৰামটিৰ আউটপুট 87। (কিন্তু আমি যদি `total_marks = ft_marks / 4.0 + st_marks / 4.0 + final_marks / 2.0;` না লিখে এভাৱে লিখতাম `total_marks = ft_marks / 4 + st_marks / 4 + final_marks / 2;` তাহলে আউটপুট আসে 86।

কাৰণ কী? কম্পিউটাৰেৰ মাথা ঘাৰাপ, নাকি আমাৰ?)

আমোৰা কিন্তু আমাদেৱ প্ৰিয় শিক্ষকেৰ তেমন কোনো উপকাৰ কৰতে পাৰলাম না। কাৰণ তাৰ ক্লাসে মোট ছাত্ৰছাত্ৰীৰ সংখ্যা চল্লিশ। তাহলে স্যারকে চল্লিশবাৰ প্ৰোগ্ৰামটি চালাতে হবে। কিন্তু এটি তো কোনো কাজেৰ কথা হলো না। আমাদেৱ উচিত, সবাৰ চূড়ান্ত ফলাফল একটি প্ৰোগ্ৰামেৰ মাধ্যমে নিৰ্ণয় কৰা। তেমন কোনো কষ্টিন কাজ নয় এটি। আমোৰা এমন একটি প্ৰোগ্ৰাম লেখা শুৰু কৰে দিতে পাৰি :

```
#include<stdio.h>
int main()
{
    int ft_marks_1, st_marks_1,
    final_marks_1, ft_marks_2, st_marks_2,
    final_marks_2, ft_marks_3, st_marks_3,
    final_marks_3,
```

তোমোৰা নিচয়ই বুৰুতে পাৰছ, আমি কী কৰতে যাচ্ছি? বলো তো, এভাৱে প্ৰোগ্ৰামটি লিখতে গেলে মোট কয়টি ভেৱিয়েবলেৰ দৰকাৰ? 160টি। স্যারেৰ কষ্ট কমাতে গিয়ে আমাদেৱ কষ্ট এত বাঢ়ানোৰ কোনো মানে হয় না। কিন্তু এ ধৰনেৰ প্ৰোগ্ৰাম তো আমাদেৱ প্ৰায়ই লিখতে হবে। চিন্তা নেই! প্ৰায় সব প্ৰোগ্ৰামিং লাঙ্গুয়েজেই আবৰে (Array) নামে একটি চমৎকাৰ জিনিস আছে। এতে একই ধৰনেৰ অনেকগুলো ভেৱিয়েবল একসঙ্গে বাখা যায়। ভেৱিয়েবলেৰ যেমন নাম রাখি, আবৰেৰ বেলাতেও তেমন একটি নাম দিতে হয়। C-তেও আবৰে আছে।

ভেৱিয়েবলেৰ যেমন একটি ডাটা টাইপ থাকে, আবৰেও থাকে। আবৰেটি যে ডাটা টাইপেৰ হবে, তাতে কেবল সেই রকম ডাটাই বাখা যাবে। যেমন, `char` টাইপেৰ আবৰেতে কেবল `char` টাইপেৰ জিনিস থাকবে।

আবৰেতে কয়টি উপাদান থাকবে সেটি শুভতেই বলে দিতে হয়।

`int ara[10];` এভাৱে আমোৰা একটি আবৰে ডিক্ৰেয়াৰ কৰতে পাৰি,

যার নাম হচ্ছে ara, যেটিতে কেবল ইন্টিজার টাইপের ডাটা থাকবে আর এই অ্যারেতে মোট দশটি সংখ্যা রাখা যাবে। প্রথমটি হচ্ছে ara[0] (হাঁ, ara[1] না কিন্তু), দ্বিতীয়টি ara[1], তৃতীয়টি ara[2], এভাবে দশম সংখ্যাটি হচ্ছে ara[9]। অর্থাৎ, ara[i] হচ্ছে i+1 তম উপাদান।

এবারে চলো, অ্যারে নিয়ে একটু খেলাধুলা করা যাক। প্রতিটি প্রোগ্রাম কিন্তু অবশ্যই কম্পিউটারে চালিয়ে দেখবে।

```
#include <stdio.h>
int main()
{
    int ara[5] = {10, 20, 30, 40, 50};
    printf("First element: %d\n", ara[0]);
    printf("Third element: %d\n", ara[2]);
    return 0;
}
```

প্রোগ্রাম : ৬.২

আউটপুট ঠিকঠাক দেখতে পাচ্ছ?

আরেকটি প্রোগ্রাম :

```
#include <stdio.h>
int main()
{
    int ara[5] = {6, 7, 4, 6, 9};
    printf("%d\n", ara[-1]);
    printf("%d\n", ara[5]);
    printf("%d\n", ara[100]);
    return 0;
}
```

৯০

প্রোগ্রাম : ৬.৩

এটির জন্য কী আউটপুট আসা উচিত? আমি জানি না এবং এটি জানা সম্ভব নয়। যেকোনো ধরনের সংখ্যা আসতে পারে। এগুলোকে গারবেজ (garbage) বলে। কারণ আসলে তো ওই অ্যারেতে -1, 5, 100 এই ইনডেক্স বলতে কিছু নেই। অ্যারেটির দৈর্ঘ্যই হচ্ছে 5। সুতরাং ইনডেক্স হবে 0 থেকে 4।

এখন কোনো অ্যারের সব উপাদান যদি একসঙ্গে দেখাতে চাই, তাহলে উপায় কী? উপায় হচ্ছে প্রথম উপাদান (ara[0]), দ্বিতীয় উপাদান (ara[1]), তৃতীয় উপাদান (ara[2]) এভাবে একে একে সবগুলো প্রিন্ট করা। আর তার জন্য অবশ্যই আমরা লুপের সাহায্য নেব।

```
#include <stdio.h>
int main()
{
    int ara[10] = {10, 20, 30, 40, 50, 60,
    70, 80, 90, 100};
    int i;
    for(i = 0; i < 10; i++) {
        printf("%d th element is: %d\n",
        i+1, ara[i]);
    }
    return 0;
}
```

প্রোগ্রাম : ৬.৪

আর যদি শেষ উপাদান থেকে প্রথম উপাদান পর্যন্ত দেখাতে হতো? কোনো সমস্যা নেই, শুধু লুপে এ indexটি 9 থেকে 0 পর্যন্ত আনলেই চলবে। এখন তোমরা প্রোগ্রামটি লিখে ফেলো।

এবারে একটি ছোট সমস্যা। কোনো একটি অ্যারেতে দশটি উপাদান

৯১

আছে, সেগুলো বিপরীতক্রমে রাখতে হবে। অর্থাৎ দশম উপাদানটি হবে প্রথম উপাদান, প্রথমটি হবে দশম, দ্বিতীয়টি হবে নবম, নবমটি হবে দ্বিতীয়... এই রকম। তার জন্য আমরা যেটি করতে পারি, আরেকটি আরের সাহায্য নিতে পারি। দ্বিতীয় অ্যারেটিতে প্রথম আরের উপাদানগুলো বিপরীতক্রমে রাখবো। তারপর দ্বিতীয় অ্যারেটি প্রথম অ্যারেতে কপি করে ফেলব।

```
#include <stdio.h>
int main()
{
    int ara[] = {10, 20, 30, 40, 50, 60,
    70, 80, 90, 100};
    int ara2[10];
    int i, j;
    for(i = 0, j = 9; i < 10; i++, j--) {
        ara2[j] = ara[i];
    }
    for(i = 0; i < 10; i++) {
        ara[i] = ara2[i];
    }
    for(i = 0; i < 10; i++) {
        printf("%d\n", ara[i]);
    }
    return 0;
}
```

প্রোগ্রাম : ৬.৫

এখনে লক্ষ করো যে প্রথম অ্যারেটির ক্ষেত্রে আমি তৃতীয় বন্ধনীর ভেতর আরের উপাদানসংখ্যা বলে দিইনি, কারণ সি-এর কম্পাইলার দ্বিতীয় বন্ধনীর ভেতর সংখ্যাগুলো দেখেই বুঝে নিতে পারে যে ara-

তে দশটি উপাদান আছে। দ্বিতীয় আরের অর্থাৎ ara2-তে এখন কোনো কিছু নেই। তাই শুরুতেই বলে দিতে হবে যে তাতে কয়টি উপাদান থাকবে। তাহলে কম্পাইলার সেই অনুসারে কম্পিউটারের মেমোরির মধ্যে আরের জন্য জায়গা করে নেবে।

প্রোগ্রামটি ভালোভাবেই কাজ করছে। কিন্তু তোমরা একটু চিন্তাভাবনা করলেই বুঝতে পারবে যে দ্বিতীয় অ্যারেটি ব্যবহার করার কোনো দরকার ছিল না। আমরা একটি বহুল প্রচলিত পদ্ধতিতেই কাজটি করতে পারতাম।

```
int temp;
temp = ara[9];
ara[9] = ara[0];
ara[0] = temp;
প্রথম ও দশম উপাদান অদলবদল হয়ে গেল। তারপর
temp = ara[8];
ara[8] = ara[1];
ara[1] = temp;
দ্বিতীয় ও নবম উপাদান অদলবদল হয়ে গেল। তাহলে চলো,
প্রোগ্রামটি লিখে ফেলি :
```

```
#include <stdio.h>
int main()
{
    int ara[] = {10, 20, 30, 40, 50, 60,
    70, 80, 90, 100};
    int i, j, temp;
    for(i = 0, j = 9; i < 10; i++, j--) {
        temp = ara[j];
```

```

    ara[j] = ara[i];
    ara[i] = temp;
}
for(i = 0; i < 10; i++) {
    printf("%d\n", ara[i]);
}
return 0;
}

```

প্রোগ্রাম : ৬.৬

প্রোগ্রামটি চলাও। কী দেখলে? আউটপুট কি এরকম?

10  
20  
30  
40  
50  
60  
.70  
80  
90  
100

তার মানে কাজ হয়নি! আসলে আমি একটি ছোট্ট ভুল করেছি, সেটি তোমরা খুঁজে বের করো। এ ধরনের ভুলকে বলে বাগ (bug), তখন প্রোগ্রাম ঠিকমতো রান করে কিন্তু সঠিক আউটপুট দেয় না। আমার কোডে বাগ আছে, তোমরা ডিবাগ (debug) করো (মানে বাগটি বের করে ঠিক করো)।

এখন চলো, আমাদের আগের সমস্যায় ফিরে যাই। আমরা এখন প্রথম সামরিক পরীক্ষায় সবার গণিতের নম্বর একটি অ্যারেতে রাখব,

দ্বিতীয় সামরিক পরীক্ষার নম্বর আরেকটি অ্যারেতে, বার্ষিক পরীক্ষার নম্বরের জন্য আরও একটি এবং রেজাল্টের জন্যও একটি অ্যারে ব্যবহার করব।

```

int ft_marks[40], st_marks[40], final_marks[40];
double total_marks[40];

```

যার রোল নম্বর 1 তার নম্বরগুলো থাকবে অ্যারের প্রথম ঘরে (মানে index 0 হবে)। এখন বলো তো, total\_marks[34]-এ কার সর্বমোট নম্বর আছে? যার রোল নম্বর 35। তাহলে কারও রোল নম্বর n হলে তার সর্বমোট নম্বর হচ্ছে total\_marks[n-1]।

এখন প্রোগ্রামটি লিখে ফেলা যাক :

```

#include <stdio.h>
int main()
{
    int ft_marks[40] = {83, 86, 97, 95, 93,
    95, 86, 52, 49, 41, 42, 47, 90, 59, 63, 86,
    40, 46, 92, 56, 51, 48, 67, 49, 42, 90, 42,
    83, 47, 95, 69, 82, 82, 58, 69, 67, 53, 56,
    71, 62},
        st_marks[40] = {86, 97, 95, 93, 95, 86,
    52, 49, 41, 42, 47, 90, 59, 63, 86, 40, 46,
    92, 56, 51, 48, 67, 49, 42, 90, 42, 83, 47,
    95, 69, 82, 82, 58, 69, 67, 53, 56, 71, 62,
    49},
        final_marks[40] = {87, 64, 91, 43, 89,
    66, 58, 73, 99, 81, 100, 64, 55, 69, 85,
    81, 80, 67, 88, 71, 62, 78, 58, 66, 98, 75,
    86, 90, 80, 85, 100, 64, 55, 69, 85, 81,
    80, 67, 88, 71};

    int i;
    double total_marks[40];

```

```

for(i = 0; i < 40; i++) {
    total_marks[i] = ft_marks[i] / 4.0
    + st_marks[i] / 4.0 + final_marks[i] / 2.0;
}
for(i = 1; i <= 40; i++) {
    printf("Roll NO: %d\tTotal Marks:
%0.0lf\n", i, total_marks[i-1]);
}
return 0;
}

```

প্রোগ্রাম : ৬.৭

রান করে দেখো, কী সুন্দর আউটপুট! printf ফাংশনের ভেতরে দেখো এক .জায়গায় আমি \t লিখেছি, এতে ট্যাব (Tab) প্রিন্ট হবে (কি-বোর্ডের বাঁ দিকে দেখো)। রোল নং প্রিন্ট করার পরে একটি ট্যাব দিয়ে টোটাল মার্কস প্রিন্ট করলে দেখতে একটু ভালো লাগে এই জন্য \t ব্যবহার করেছি, এমনিতে কোনো দরকার নেই।

কিন্তু এত সুন্দর প্রোগ্রাম দেখে তোমার শিক্ষক কোথায় তোমাকে একটু চটপটি খাওয়াবেন, তা না, উল্টা আরেকটি আবদার করে বসলেন। কোন নম্বর কতজন পেয়েছে সেটি উনি দেখতে চান। মানে 50 কতজন পেল, 51 কতজন পেল ... এই রকম আরকি। বাকি অংশ পড়ার আগে প্রোগ্রামটি তোমরা নিজে লেখার চেষ্টা করো। এখন ইচ্ছা না করলে বইটি পড়া বন্ধ করে দাও এবং পরে কোনো এক সময় চেষ্টা করবে।

আশা করি, তোমাদের মধ্যে কেউ কেউ প্রোগ্রামটি লিখে ফেলেছ। যদি কমপক্ষে এক ঘণ্টা চেষ্টার পরেও লিখতে না পারো, তাহলে এখন আমরা সমাধানের চেষ্টা করতে পারি। শুরুতেই একটি ব্যাপার খেয়াল করো যে কেউ কিন্তু 50-এর নিচে নম্বর পায়নি। তাই 50 থেকে 100 পর্যন্ত কোন নম্বর কতজন পেল, সেটি বের করলেই চলবে। আমার যথায় প্রথমেই যে সমাধান আসছে, সেটি হলো total\_marks

আরেতে প্রথমে দেখব, কয়টি 50 আছে, তারপর আবার দেখব কয়টি 51 আছে ... এভাবে 100 পর্যন্ত দেখব। মানে 50 থেকে 100 পর্যন্ত সব সংখ্যার জন্য total\_marks আরেতে সংখ্যাগুলো চেক করব।

for(marks = 50; marks <= 100; marks++) { লুপের সাহায্যে প্রথমে marks-এর মান 50, তারপর 51, এভাবে এক এক করে বাড়াব 100 পর্যন্ত।

count = 0; ধরে নিছি শূন্য জন 'marks' নম্বর পেয়েছে। marks-এর সব কটি মানের জন্যই প্রথমে আমরা এই কাজটি করব।

এবাবে total\_marks আরেতে দেখব যে কোনো নম্বর যদি marks-এর সমান হয়, তবে count-এর মান এক বাড়িয়ে দেব। তাহলে কোনো একটি নম্বর (marks) যতবার আরেতে আছে, count-এর মান তত হবে।

```

for(i = 0; i < 40; i++) {
    if(total_marks[i] == marks) {
        count++;
    }
}

printf("Marks: %d Count: %d\n", marks, count);

```

তাহলে পুরো প্রোগ্রাম লিখে ফেলি :

```

#include <stdio.h>
int main()
{
    int marks, i, count;

```

```

int total_marks[] = {86, 78, 94, 68,
92, 78, 64, 62, 72, 61, 72, 66, 65, 65, 80,
72, 62, 68, 81, 62, 56, 68, 58, 56, 82, 70,
74, 78, 76, 84, 88, 73, 62, 66, 76, 70, 67,
65, 77, 63};
for(marks = 50; marks <= 100; marks++)
{
    count = 0;
    for(i = 0; i < 40; i++) {
        if(total_marks[i] == marks) {
            count++;
        }
    }
    printf("Marks: %d Count: %d\n",
marks, count);
}
return 0;
}

```

প্রোগ্রাম : ৬.৮

তেমন কঠিন কিছু নয়। নেস্টেড ফর লুপ ব্যবহার করে সহজ-সরল সমাধান করে ফেললাম। আচ্ছা বলো তো, if-এর ভেতর যে শর্তটি আমরা পরীক্ষা করছি (`total_marks[i] == marks`) এই কাজটি প্রোগ্রামে কতবার হয়? বাইরের লুপটি ঘুরবে 51 বার এবং প্রতিবারের জন্য ভেতরের লুপটি ঘুরবে 40 বার। তাহলে মোট  $51 \times 40 = 2040$  বার।

ওপরের প্রোগ্রামটি আমরা এখন একটু অন্যভাবে লেখার চেষ্টা করব। নিচের প্রোগ্রামটি চটপট টাইপ করে ফেলো এবং রান করো :

```
#include <stdio.h>
int main()
```

```

{
    int i;
    int total_marks[] = {86, 78, 94, 68,
92, 78, 64, 62, 72, 61, 72, 66, 65, 65, 80,
72, 62, 68, 81, 62, 56, 68, 58, 56, 82, 70,
74, 78, 76, 84, 88, 73, 62, 66, 76, 70, 67,
65, 77, 63};
    int marks_count[101];
    for(i = 0; i < 101; i++) {
        marks_count[i] = 0;
    }
    for(i = 0; i < 40; i++) {
        marks_count[total_marks[i]]++;
    }
    for(i = 50; i <= 100; i++) {
        printf("Marks: %d Count: %d\n", i,
marks_count[i]);
    }
}

```

প্রোগ্রাম : ৬.৯

এখানে আমি যেটি করেছি, একটি অতিরিক্ত অ্যারে ব্যবহার করেছি। `marks_count` একটি ইন্টিজার টাইপের অ্যারে এবং `marks_count[n]` দিয়ে আমরা বুঝব  $n$  সংখ্যাটি কতবার `total_marks`-এর মধ্যে আছে। নম্বর যেহেতু 0 থেকে 100-এর মধ্যে হতে পারে, তাই আমরা ওই অ্যারেতে মোট 101টি সংখ্যা রাখার ব্যবস্থা করলাম।

`int marks_count[101];`

শুরুতে যেহেতু কিছুই জানি না, তাই ধরে নিই, সব সংখ্যা শূন্য বার

আছে। তাই marks\_count অ্যারের সব ঘরে 0 বসিয়ে দিই :

```
for(i = 0; i < 101; i++) {  
    marks_count[i] = 0;  
}
```

এখন total\_marks অ্যারের প্রতিটি সংখ্যার জন্য marks\_count  
অ্যারের ওই ঘরের মান এক বাড়িয়ে দিই।

```
for(i = 0; i < 40; i++) {  
    marks_count[total_marks[i]]++;  
}
```

বুঝতে সমস্যা হচ্ছে নাকি? একটু চিন্তা করো।

যখন i-এর মান 0, তখন total\_marks[i] হচ্ছে total\_marks[0],  
অর্থাৎ 86। এখন আমাদের দরকার হচ্ছে marks\_count অ্যারের  
ওই ঘরটার (মানে marks\_count[86]) মান এক বাড়িয়ে দেওয়া।  
শুরুতে ছিল শূন্য, এখন হবে এক। আমরা কিন্তু সে কাজটিই করেছি  
marks\_count[total\_marks[i]]-এর মান এক  
বাড়িয়ে দিয়েছি marks\_count[total\_marks[i]]++;

আসলে ব্যাপারটি এইভাবেও লেখা যেত :

```
t_m = total_marks[i];  
marks_count[t_m]++;
```

এখনো যারা মাথা চুলকাছ তারা নিচের প্রোগ্রামটি কম্পিউটারে রান  
করাও।  
এখানে  
প্রতিবার  
marks\_count[total\_marks[i]]++;  
করার পরে  
marks\_count অ্যারেটি আমরা এক লাইনে প্রিন্ট করেছি।

```
#include <stdio.h>  
int main()  
{
```

```
int i, j;  
int total_marks[] = {6, 7, 4, 6, 9, 7,  
6, 2, 4, 3, 4, 1};  
int marks_count[11];  
for(i = 0; i < 11; i++) {  
    marks_count[i] = 0;  
}  
for(i = 0; i < 12; i++) {  
    marks_count[total_marks[i]]++;  
    for(j = 0; j <= 10; j++) {  
        printf("%d ", marks_count[j]);  
    }  
    printf("\n");  
}  
return 0;  
}
```

প্রোগ্রাম : ৬.১০

আশা করি, এখন আর বুঝতে সমস্যা হওয়ার কথা নয়। আর তুমি  
তোমার প্রোগ্রামের জন্য তোমার গণিত শিক্ষকের কাছ থেকে একদিন  
চটপটি দাবি করতেই পারো। তবে চটপটি পেতে হলে প্রোগ্রামটি  
এমনভাবে লিখতে হবে, যেন তোমার শিক্ষক সবার পরীক্ষার নম্বর  
নিজে ইনপুট হিসেবে দিতে পারেন। তুমি কি সেটি করতে পারবে?

মান তার চেয়ে দেশি সময় কিনি অন্যের দেখা পড়ে? একজন দেশক শারীরেন  
প্রোগ্রামিং জীবনে যাত্যা সময় নিজে দেওত দেবে, তার চেয়ে বেশি  
সময় আবেগ দেখা চোট পড়ে! অতি দেওত দেবার সবয় বেশাল  
যাবতে হলে, সেন সেটি পড়াও সুবিধাজনক হয়।

শাস্তি শুরু থেকে পড়ে এসেছ তারা ইতিমধ্যে অনেকবার ফাংশন  
খেয়াল করেছ যে `printf`, `scanf` ইত্যাদি, সেজন্স তোমরা ব্যবহার  
করছ সেগুলো একেকটি ফাংশন। আবার `main`ও একটি ফাংশন।  
আমরা এবার দেখব ফাংশন ব্যাপারটি আসলে কী, এর দরকারটাই বা  
কী। আব তারপর আমরা নিজেদের ফাংশন তৈরি করা শিখব।

ফাংশন ব্যবহার করা হয় কোনো একটি নির্দিষ্ট কাজ করার জন্য।  
যেমন `printf` ফাংশনটি দিয়ে আমরা মনিটরে আউটপুট দিই। আবার  
`scanf`, `getchar` এসব ফাংশন দিয়ে আমরা কিংবোর্ড থেকে ইনপুট  
নিই। এখন `printf` ফাংশনটি বে আবরা লিখলাম, কম্পিউটারের তে  
আর এটি বোমার কথা নহ। `printf` ফাংশনটি কী কাজ করবে,  
কীভাবে করবে সেটি আনলে বলে দেওয়া অছে `stdio.h` নামের  
একটি হেডার (header) কাইলের মধ্যে। এজন্যই আমরা আমাদের  
প্রোগ্রামগুলোতে (যেখানে `printf`, `scanf` ইত্যাদি ব্যবহার করেছি)  
ওই হেডার ফাইলটির কথা বলে দিই (`#include <stdio.h>`)।  
আবার স্ট্রিং-সংক্রান্ত ফাংশনগুলো ব্যবহার করলে `string.h`-এই  
হেডার ফাইলটির কথা বলে দিই। একন চিন্তা করো, `printf`  
ফাংশনের এই কোডটি যদি আমাদের নিজেদের লিখতে হতো, তাহলে  
ব্যাপারটি কী বিস্তৃক্রই ন হতো! এতকম অনেক ফাংশন আছ  
যেগুলোর ব্যবহার তোমরা আস্তে আস্তে জেনে হবে।

আছা, `main`-কেও তো আমি একটি ফাংশন কলেই কিনু এটি দিয়ে  
আমরা আবার কী করি? মি ল্যাঙ্গুজেজে এই ফাংশনটি নিয়েই আসল  
আমরা একটি প্রোগ্রাম চালাই। কম্পাইলার জন্মে `main` ফাংশন

যেখানে আছে, সেখান থেকেই কাজ শুরু করতে হবে। তাই একটি  
প্রোগ্রামে কেবল একটিই `main` ফাংশন থাকে।

এবারে দেখি, আমরা নিজেরা কীভাবে ফাংশন তৈরি করতে পারি।  
একটি ফাংশন যখন আমরা তৈরি করব, সেটির গঠন হবে মোটামুটি  
এই রকম:

```
return_type function_name (parameters)
{
    function_body
    return value
}
```

`return_type` : এখানে বলে দিতে হবে, ফাংশনটি কাজ শেষ করে  
বের হবার সময় কী ধরনের ডাটা রিটার্ন করবে। সেটি, `int`, `double`-  
এসব হতে পারে। আবার কিছু রিটার্ন করতে না চাইলে সেটি `void`  
হতে পারে। অর্থাৎ সে কিছুই রিটার্ন করবে না। এর মানে দাঁড়াচ্ছে,  
তুমি আসলে ফাংশনকে দিয়ে কোনো একটি কাজ করাবে, সেজন্য  
কাজ শেষে সে তোমাকে কী ধরনের ডাটা ফেরত দেবে সেটি বলে  
দিতে হবে।

`function_name` : এখানে আমাদের ফাংশনের নাম লিখতে হবে।  
ফাংশনের নাম হতে হবে অর্থপূর্ণ যাতে নাম দেখেই ধারণা করা যায়  
যে ফাংশনটি কী কাজ করবে। যেমন, কোনো সংখ্যার বর্গমূল নির্ণয়ের  
জন্য যদি আমরা একটি ফাংশন লিখি তাহলে সেটির নাম আমরা দিতে  
পারি `square_root` বা `sqr`। আমরা নিচয়ই সেটির নাম  
beautiful দিব না, যদিও কম্পাইলার তাতে কোনো আপত্তি করবে  
না।

`parameters` : এখানে ফাংশনটি কাজ করা  
আমরা দেব। যেমন স্ট্রিং-এর দৈর্ঘ্য নির্ণয়ের  
ফাংশনটি ব্যবহার করি, সেখানে কোন স্ট্রিং  
হবে সেটি বলে দিতে হয় (নইলে সেটি  
আবার বর্গমূল নির্ণয়ের জন্য ফাংশন লিখ

করতে হবে সেটি বলে দিতে হবে। প্যারামিটারের মাধ্যমে আমরা সেসব ডাটা ওই ফাংশনের কাছ পাঠাতে পারি। আবার কোনো কিছু পাঠাতে না চাইলে সেটি খালিও রাখতে পারি। যেমন, `getchar()` বা `main()` ফাংশন। একাধিক প্যারামিটার পাঠানোর সময় প্রতিটি প্যারামিটার কমা (,) দিয়ে আলাদা করতে হবে।

**function\_body :** ফাংশনটি কীভাবে কী কাজ করবে সেটি বড়তে বলে দিতে হবে। মানে কোড লিখতে হবে আরকি।

**return value :** ফাংশনটি কাজ শেষ করে, তাকে যে জায়গা থেকে কল করা হয়েছে সে জায়গায় ফিরে যায়। ফেরার সময় আমরা কোনো মান পাঠাতে পারি। যেমন `sqrt()` ফাংশনে আমরা চাই সে বর্গমূল বের করবে। তো বর্গমূলটি বের করে তো সেটি ফেরত পাঠাবার ব্যবস্থা রাখতে হবে? বর্গমূলটির মান যদি  $x$  হয়, তাহলে আমরা `return x;` স্টেটমেন্ট দিয়ে সেটির মান ফেরত পাঠাব।

```
int root = sqrt(25);
```

এখানে `sqrt` ফাংশন 25-এর বর্গমূল নির্ণয় করার পর বর্গমূলটি ফেরত পাঠাবে এবং সেটি `root` নামের একটি ইন্টিজার ভেরিয়েবলে জমা হবে।

একটি উদাহরণ দিই। তোমরা যারা ত্রিকোণমিতি পড়েছ, তারা নিচেরই `sin`, `cos`, `tan` ইত্যাদির সঙ্গে পরিচিত।  $\sin 30^\circ$ -এর মান হচ্ছে 0.5। এখানে `sin` কিন্তু আসলে একটি ফাংশন, যার প্যারামিটার হিসেবে আমরা কোণের মান দিচ্ছি। আর ফাংশনটি ওই কোণের `sine` (সংক্ষেপে `sin`)-এর মান রিটার্ন করছে।

এবার চলো, আর বকবক না করে প্রোগ্রামিং শুরু করে দিই। তারপর দেখি, কী করলে কী হয়।

```
#include <stdio.h>
int main()
{
    double a, b, c;
```

```
a = 2.5;
b = 2.5;
c = a + b;
printf("%lf\n", c);
return 0;
```

}

**প্রোগ্রাম :** ৭.১

প্রোগ্রামটি চালাও। আউটপুট কী? 5.000000।

এবার আমরা দুটি সংখ্যা যোগ করার জন্য একটি ফাংশন লিখে ফেলি। যোগের কাজটি আর `main` ফাংশনের ভেতরে করব না।

```
#include <stdio.h>
int add(int num1, int num2)
{
    double sum = num1 + num2;
    return sum;
}
int main()
{
    double a, b, c;
    a = b = 2.5;
    c = add(a, b);
    printf("%lf\n", c);

    return 0;
}
```

### প্রোগ্রাম : ৭.২

প্রোগ্রামটি চালাও। আউটপুট কী? 4.000000! ওহ, আমরা তো গাধার মতো একটি ভুল করেছি। num1 ও num2 তো আসলে int টাইপের হবে না, double টাইপের হবে। ওই দৃঢ়ি ভেরিয়েবল ইন্টিজার হিসেবে ডিক্লেয়ার করার কারণে 2.5 হয়ে গিয়েছে 2 (টাইপ কাস্টিংয়ের কথা মনে আছে তো?)। আমরা ভুল ঠিক করে ফেলি :

```
int add(double num1, double num2) {
    double sum = num1 + num2;
    return sum;
}
```

এবাবে প্রোগ্রামটি রান করলে আউটপুট কী? 5.000000। যাক, সমস্যার সমাধান হয়ে গেল! আচ্ছা, এবাবে আমরা a, b-এর মান একটু বদলাই।  $a = 2.8; b = 2.7;$  করে দিই। আউটপুট কত হবে? 5.500000? এটিই হওয়া উচিত ( $2.8 + 2.7 = 5.5$ ) কিন্তু প্রোগ্রামটি রান করে দেখো তো কত হয়? তুমি আউটপুট পাবে 5.000000। কারণ কী?

কারণ, আমাদের ফাংশনের রিটার্ন টাইপ int, যা কিনা একটি ইন্টিজার রিটার্ন করতে সক্ষম। num1 ও num2 যোগ করার পর sum-এর মধ্যে 5.5 ঠিকই থাকবে কিন্তু রিটার্ন করার সময় সেটি ইন্টিজারে বদলে যাবে। সুতরাং রিটার্ন টাইপ আমরা double করে দেব। এবাবে আমাদের প্রোগ্রাম ঠিকঠাক কাজ করবে :

```
#include <stdio.h>
double add(double n1, double n2)
{
    double sum = n1 + n2;
    return sum;
}
int main()
{
```

### double a, b, c;

$a = 2.8;$

$b = 2.7;$

$c = add(a, b);$

`printf("%lf\n", c);`

`return 0;`

}

### প্রোগ্রাম : ৭.৩

এখন আমরা একটি পরীক্ষা করব। add ফাংশনটি main ফাংশনের পরে লিখব :

```
#include <stdio.h>
int main()
{
    double a = 2.8, b = 2.7, c;
    c = add(a, b);
    printf("%lf\n", c);
    return 0;
}
```

double add(double n1, double n2)

{

double sum = n1 + n2;

return sum;

}

### প্রোগ্রাম : ৭.৪

এবাবে প্রোগ্রামটি রান করতে গেলে দেখবে, কম্পাইলার এরের দিচ্ছে :

"error: 'add' was not declared in this scope", অর্থাৎ, সে আর add ফাংশনটিকে চিনতে পারছে না। তবে চিন্তা নেই, এটিকে

চিনিয়ে দেওয়ার ব্যবস্থাও আছে। সেটি হচ্ছে main ফাংশনের আগে add ফাংশনের প্রোটোটাইপ (prototype) বলে দেওয়া :

double add(double n1, double n2);

প্রোটোটাইপে পুরা ফাংশনটি লিখতে হয় না। এর অংশগুলো হচ্ছে :  
return\_type function\_name (parameters);

সেমিকোলন দিতে ভুল করবে না কিন্তু। আর প্রোটোটাইপের প্যারামিটারে যে ভেরিয়েবল ব্যবহার করবে তার সঙ্গে মূল ফাংশনের ভেরিয়েবলের নাম এক রকম না হলে কোনো অসুবিধা নেই, তবে ডাটা টাইপ একই হতে হবে। এখন নিচের প্রোগ্রামটি ঠিকঠাক কাজ করবে :

```
#include <stdio.h>
double add(double x, double y);
int main()
{
    double a = 2.8, b = 2.7, c;
    c = add(a, b);
    printf("%lf\n", c);
    return 0;
}
double add(double n1, double n2)
{
    double sum = n1 + n2;
    return sum;
}
```

প্রোগ্রাম : ৭.৫

এবার আমরা আরও কিছু পরীক্ষা-নিরীক্ষা করব।

```
#include <stdio.h>
int test_function(int x)
{
    int y = x;
    x = 2 * y;
    return (x * y);
}
int main()
{
    int x = 10, y = 20, z = 30;
    z = test_function(x);
    printf("%d %d %d\n", x, y, z);
    return 0;
}
```

প্রোগ্রাম : ৭.৬

প্রোগ্রামটি না চালিয়ে শুধু কোড দেখে বলো তো আউটপুট কী হবে? আমাদের কোনো তাড়া নেই, তাই ধীরেসুস্থে চিন্তা করে বলো।

এবার কে কে আমার সঙ্গে একমত যে আউটপুট হবে :

20 10 200 (অর্থাৎ x = 20, y = 10, z = 200)?

কারণ x, y-এর মান তো test\_function-এর ভেতরে আমরা বদলে দিয়েছি। প্রথমে x-এর মান 10 যাচ্ছে প্যারামিটার হিসেবে, তারপরে সেই মানটি আমরা y-তে বসাচ্ছি। মানে y-এর মান এখন 10। তারপর x-এর মান বসাচ্ছি 2 \* y মানে 20। তারপর রিটার্ন করছি x \* y (যার মান, 20 \* 10 বা 200)। সুতরাং z-এর মান হবে 200।

এবার প্রোগ্রামটি চালাও, আউটপুট দেখবে : 10 20 200 (অর্থাৎ x = 10, y = 20, z = 200)। এমন হওয়ার কারণ কী? z-এর মান নিয়ে কোনো আপত্তি নেই, ফাংশনটি 200 রিটার্ন করে আর সেটি

আমরা z-এ বসিয়ে দিয়েছি। কথা হচ্ছে, x আর y-এর মান নিয়ে।  
আসলে test\_function-এর ভেতরে আমরা x, y-এর মান পরিবর্তন  
করায় main ফাংশনের x, y-এর কিছু আসে-যায় না। প্রত্যেক  
ফাংশনের ভেরিয়েবলগুলো আলাদা। একে বলে লোকাল ভেরিয়েবল  
(local variable)। আমরা main ফাংশনের x, y-এর মান প্রিন্ট  
করেছি test\_function ফাংশনের x, y-এর মান প্রিন্ট করিন। এক  
ফাংশনের লোকাল ভেরিয়েবলের অস্তিত্ব অন্য ফাংশনে থাকে না। তুমি  
এখন কিছু প্রোগ্রাম লিখে আরও পরীক্ষা-নিরীক্ষা করে দেখতে পারো।  
কী প্রোগ্রাম লিখবে সেটি তোমার ওপর ছেড়ে দিলাম।

আমরা যদি চাই, কোনো ভেরিয়েবলের অস্তিত্ব আমাদের প্রোগ্রামের  
সব ফাংশনের ভেতরে থাকতে হবে, তাহলে আমরা সেটি করতে পারি  
গ্লোবাল (global) ভেরিয়েবল ডিক্লেয়ার করার মাধ্যমে। আমরা  
প্রোগ্রামের শুরুতে কোনো ফাংশন বা ফাংশনের প্রোটোটাইপ লেখার  
আগে সেগুলো ডিক্লেয়ার করে দেব। যেমন :

```
#include <stdio.h>

double pi = 3.14;

void my_fnc() {
    pi = 3.1416; /* এখানে আমরা pi-এর মান একটু পরিবর্তন  
করে দিলাম */

    return; /* ফাংশনের রিটার্ন টাইপ void হলে এই return; না  
দিলেও কিন্তু চলে */
}

int main() {
    printf("%f\n", pi); /* এখানে pi-এর মান হবে 3.14 */
    my_fnc();

    printf("%f\n", pi); /* এখানে pi-এর মান হবে 3.1416 কারণ  
আমরা সেটি my_fnc ফাংশনে গিয়ে বদলে দিয়েছি। */
}
```

১১০

return 0;

}

আবার আমরা যদি my\_fnc ফাংশনের ভেতরে গিয়ে pi নামে একটি  
ভেরিয়েবল ডিক্লেয়ার করতাম (double pi;), তাহলে সেটি একটি  
লোকাল ভেরিয়েবল হতো এবং গ্লোবাল pi-এর মানের কোনো  
পরিবর্তন হতো না।

এতক্ষণ আমরা ফাংশনের প্যারামিটার হিসেবে কেবল ভেরিয়েবল  
ব্যবহার করেছি। এবারে আসো, আমরা ফাংশনের প্যারামিটার  
হিসেবে অ্যারে পাঠাই। আমরা একটি প্রোগ্রাম লিখব, যেটি কোনো  
একটি ইন্টিজার অ্যারে থেকে সবচেয়ে বড় সংখ্যাটি খুঁজে বের করবে।  
অ্যারে থেকে সর্বোচ্চ সংখ্যা খুঁজে বের করার কাজটি করার জন্য  
একটি ফাংশন লিখে ফেলি, কী বলো?

```
int find_max(int ara[], int n) { /* এখানে আমরা দুটি  
প্যারামিটার দিচ্ছি। প্রথমটা হচ্ছে একটি অ্যারে। আর তারপর একটি  
সংখ্যা, যেটি নির্দেশ করবে অ্যারেতে কয়টি সংখ্যা আছে। লক্ষ করো,  
প্যারামিটারে যখন অ্যারের কথাটি বলে দিচ্ছি, তখন সেখানে কয়টি  
উপাদান আছে সেটি না দিলেও চলে। যেমন, আমরা int ara[11] ও  
লিখতে পারতাম। */
```

```
int max = ara[0]; /* এখানে একটি ভেরিয়েবলে ধরে নিছি যে  
সবচেয়ে বড় সংখ্যাটি হচ্ছে অ্যারের প্রথম সংখ্যা। তারপরে আমরা  
অ্যারের বাকি উপাদানগুলোর সঙ্গে max-কে তুলনা করব আর যদি  
অ্যারের কোনো উপাদানের মান max-এর চেয়ে বড় হয় তখন সেই  
মানটি max-এ রেখে দেব। অর্থাৎ তখন আবার max হয়ে যাবে ওই  
অ্যারের সর্বোচ্চ সংখ্যা। */
```

int i;

for(i = 1; i < n; i++) {

if (ara[i] > max) {

max = ara[i]; /\* ara[i] যদি max-এর চেয়ে বড় হয়

১১১

তাহলে max-এ ara[i]-এর মানটি অ্যাসাইন করে দিচ্ছি। \*/

```
    }  
}  
  
return max; /* ফাংশন থেকে সর্বোচ্চ মানটি ফেরত পাঠাচ্ছি */  
}
```

এখন কথা হচ্ছে, এই ফাংশনকে আমরা কল করব কীভাবে? ভেরিয়েবলের জায়গায় তো এর নাম দিয়ে কল করতে হয়, কিন্তু অ্যারের বেলায় কী দেব? অ্যারের বেলাতেও শুধু নাম দিলেই চলবে। পুরো প্রোগ্রামটি এবারে রান করে দেখো :

```
#include <stdio.h>  
  
int find_max(int ara[], int n);  
  
int main()  
{  
  
    int ara[] = {-100, 0, 53, 22, 83, 23,  
89, -132, 201, 3, 85};  
  
    int n = 11;  
  
    int max = find_max(ara, n);  
  
    printf("%d\n", max);  
  
    return 0;  
}  
  
int find_max(int ara[], int n)  
{  
  
    int max = ara[0];  
    int i;  
  
    for(i = 1; i < n; i++) {  
        if (ara[i] > max) {  
            max = ara[i];  
        }  
    }  
}
```

```
    }  
}  
return max;
```

}

প্রোগ্রাম : ৭.৭

এখন তোমরা find\_min নামে আরেকটি ফাংশন লেখো, যার কাজ হবে সবচেয়ে ছোট সংখ্যাটি খুঁজে বের করা। find\_sum, find\_average এসব ফাংশনও লিখে ফেলতে পারো। আর তোমাদের নিচয়ই বলে দিতে হবে না, এসব ফাংশন কী কাজ করবে।

ফাংশনে ভেরিয়েবল পাস করা (pass, পাঠানো অর্থে) আর আরে পাস করার মধ্যে একটি গুরুত্বপূর্ণ পার্থক্য রয়েছে। আমরা ইতিমধ্যে দেখেছি যে ফাংশনের ভেতর ভেরিয়েবল পাস করলে ওই ফাংশনের ভেতরে সেটির আরেকটি কপি তৈরি হয়, সুতরাং সেখানে ওই ভেরিয়েবলের মান পরিবর্তন করলে মূল ফাংশন (যেখান থেকে ফাংশন কল করা হয়েছে) ভেরিয়েবলের মানের কোনো পরিবর্তন হয় না। তবে অ্যারের বেলায় ব্যাপারটি আলাদা। আগে আমরা একটি প্রোগ্রাম লিখে দেখি :

```
#include <stdio.h>  
  
void test_function(int ara[])  
{  
  
    ara[0] = 100;  
  
    return;  
}  
  
int main()  
{  
  
    int ara [] = {1, 2, 3, 4, 5};  
  
    printf("%d\n", ara[0]);  
  
    test_function(ara);  
}
```

```

        printf("%d\n", ara[0]);
    return 0;
}

```

প্রোগ্রাম : ৭.৮

এই প্রোগ্রামের আউটপুট কী হবে? প্রথম printf ফাংশনটি 1 প্রিন্ট করবে সেটি নিয়ে তো কোনো সন্দেহ নেই, কিন্তু দ্বিতীয় printf কী প্রিন্ট করবে? test\_function-এর ভেতর আমরা আরের প্রথম উপাদানের মান 100 অ্যাসাইন করেছি। এখন যদি সেটি মূল আরেকে পরিবর্তন করে, তবে ara[0]-এর মান হবে 100, আর পরিবর্তন না হলে মান হবে আগে যা ছিল তা-ই, মানে 1।

আমরা আউটপুট দেখব 100, কারণ আরেটির প্রথম উপাদানের মান পরিবর্তিত হয়েছে। অর্থাৎ, আমরা বুঝতে পারলাম, ফাংশনের ভেতরে আরে পাস করলে ওই আরের আলাদা কোনো কপি তৈরি হয় না। কারণ হচ্ছে, আমরা ফাংশনের ভেতর আরের নামটি কেবল পাঠাই, যেটি কিনা ওই আরেটি মেমোরির কোন জায়গায় আছে তার অ্যাড্রেস।

এখন তোমরা বৃত্তের ক্ষেত্রফল নির্ণয়ের জন্য একটি ফাংশন লিখে ফেলো। ক্ষেত্রফল বের করার সূত্রটি মনে আছে তো? মনে না থাকলে জ্যামিতি বই থেকে দেখে নাও।

## অধ্যায় আট : বাইনারি সার্চ

একটি সহজ খেলা দিয়ে শুরু করা যাক। এটি খেলতে দুজন দরকার। একজন মনে মনে একটি সংখ্যা ধরবে। আর দ্বিতীয়জন কিছু প্রশ্ন করে সেই সংখ্যাটি বের করবে। তবে 'তোমার সংখ্যাটি কত?' - এমন প্রশ্ন কিন্তু সরাসরি করা যাবে না। প্রশ্নটি হচ্ছে : সংখ্যাটি কি N (একটি সংখ্যা)-এর চেয়ে বড়, ছোট নাকি সমান?

আর সংখ্যাটি কিন্তু একটি নির্দিষ্ট সীমার মধ্যে হতে হবে (যেমন 1 থেকে 100, 10 থেকে 1000, -1000 থেকে 100000)। সীমা শব্দটা এখানে range বা ব্যবধি অর্থে ব্যবহার করা হয়েছে।

এখন ধরা যাক, প্রথমজন যে সংখ্যাটি ধরেছে সেটি 1 থেকে 1000-এর ভেতর একটি সংখ্যা। তাহলে কিন্তু সর্বোচ্চ এক হাজার বার সংখ্যাটি কি N-এর সমান?' প্রশ্নটি করে সেটি বের করে ফেলা যায়। (সংখ্যাটি কি 1? সংখ্যাটি কি 2? সংখ্যাটি কি 999?, সংখ্যাটি কি 1000?)। এভাবে প্রশ্ন করতে থাকলে সংখ্যাটি অবশ্যই বের হবে। তবে ভাগ্য খারাপ হলে এক হাজার বার ওই প্রশ্নটি করতে হবে।

কিন্তু আমাদের তো এত সময় নেই। ধরা যাক, 1 থেকে 1000-এর ভেতর ওই সংখ্যাটি হচ্ছে 50। তাহলে আমাদের প্রথম প্রশ্ন হবে :

- ১) সংখ্যাটি কি 500-এর চেয়ে বড়, ছোট নাকি সমান? ছোট।
  - ২) সংখ্যাটি কি 250-এর চেয়ে বড়, ছোট নাকি সমান? ছোট।
  - ৩) সংখ্যাটি কি 125-এর চেয়ে বড়, ছোট নাকি সমান? ছোট।
  - ৪) সংখ্যাটি কি 62-এর চেয়ে বড়, ছোট নাকি সমান? ছোট।
  - ৫) সংখ্যাটি কি 31-এর চেয়ে বড়, ছোট নাকি সমান? বড়।
  - ৬) সংখ্যাটি কি 46-এর চেয়ে বড়, ছোট নাকি সমান? বড়।
  - ৭) সংখ্যাটি কি 54-এর চেয়ে বড়, ছোট নাকি সমান? ছোট।
  - ৮) সংখ্যাটি কি 50-এর চেয়ে বড়, ছোট নাকি সমান? সমান।
- আমরা মাত্র আটটি প্রশ্ন করেই সংখ্যাটি পেয়ে গেছি!

তোমরা নিচয়ই পদ্ধতিটি বুঝে ফেলেছ? প্রতিবার প্রশ্ন করে সংখ্যাটি যে সীমার মধ্যে আছে তাকে অর্ধেক করে ফেলা হয়েছে। খেলা শুরুর সময় সীমাটি ছিল 1 থেকে 1000। তারপর সেটি হয়েছে 1 থেকে 500। তারপর 1 থেকে 250, 1 থেকে 125, 1 থেকে 62, 31 থেকে 62, 46 থেকে 62, 46 থেকে 54।

সংখ্যা বের করার এই পদ্ধতিকে বলে বাইনারি সার্চ। চলো আমরা তাহলে অ্যালগরিদমটি লেখার চেষ্টা করি:

**বাইনারি সার্চ (low, high, N):** (শুরুতে আমাদের তিনটি সংখ্যা জানতে হবে, সংখ্যাটি যে ব্যবধি বা রেঞ্জের মধ্যে আছে, তার নিম্নসীমা (low), উচ্চসীমা (high) এবং সেই সংখ্যা (N))

ধাপ 1:  $mid = (\text{low} + \text{high}) / 2$

ধাপ 2: যদি  $mid$  এবং  $N$ -এর মান সমান হয় তবে ধাপ 5-এ যাও।

ধাপ 3: যদি  $N$ ,  $mid$ -এর চেয়ে বড় হয়, তাহলে  $\text{low} = mid + 1$ .

ধাপ 1-এ যাও।

ধাপ 4: যদি  $N$ ,  $mid$ -এর চেয়ে ছোট হয়, তাহলে  $high = mid - 1$ .

ধাপ 1-এ যাও।

ধাপ 5: সংখ্যাটি পেয়ে গেছি ( $mid$ )।

এখন আমরা দেখব, একটি অ্যারে থেকে কীভাবে বাইনারি সার্চ করে কোনো সংখ্যা খুঁজে বের করতে হয়। অ্যারেতে কিন্তু সংখ্যাগুলো ছেট থেকে বড় কিংবা বড় থেকে ছোট ক্রমানুসারে থাকতে হবে। নইলে বাইনারি সার্চ ব্যবহার করা যাবে না। কারণটি কি কেউ বলতে পারো?

প্রথমে আমরা একটি ইন্টিজার অ্যারে নিই যেখানে সংখ্যাগুলো ছেট থেকে বড় ক্রমানুসারে সাজানো আছে।

```
int ara[] = {1, 4, 6, 8, 9, 11, 14, 15, 20, 25, 33, 83, 87, 97, 99, 100};
```

এখন বলো তো  $low$  আর  $high$ -এর মান কত হবে?  $low = 1$  এবং  $high = 100$ ? ঠিকই ধরেছ, কিন্তু এখানে একটু সমস্যা আছে।

আমরা এখানে সব সংখ্যার মধ্যে খুঁজব না, বরং অ্যারের ইনডেক্সের মধ্যে খুঁজব। আর অ্যারের ইনডেক্সগুলো ক্রমানুসারে থাকে বলেই অ্যারেতে বাইনারি সার্চ করা যায়। এখানে  $ara$ -এর সর্বনিম্ন ইনডেক্স হচ্ছে 0 এবং সর্বোচ্চ ইনডেক্স হচ্ছে 15। তাহলে আমরা দুটি ভেরিয়েবলের মান নির্দিষ্ট করে দিই -

$low\_idx = 0;$

$high\_idx = 15;$

যে সংখ্যাটি খুঁজব ধরা যাক সেটি হচ্ছে 97।

$num = 97;$

তোমাদের অনেকেই হয়তো ভাবছ,  $num$  সংখ্যাটি যদি  $ara$ -তে না থাকে, তখন কী হবে? সেটিও আমরা দেখব। সংখ্যাটি যদি খুঁজে পাওয়া না যায়, তাহলে সেটি জানিয়ে দেওয়ার ব্যবস্থা রাখতে হবে আমাদের প্রোগ্রামে।

আমাদের যেহেতু খোঁজার কাজটি বারবার করতে হবে, আমাদের একটি লুপ ব্যবহার করতে হবে। লুপের ভেতর আমরা খোঁজাখুঁজি করব আর সংখ্যাটি পেয়ে গেলে (কিংবা সংখ্যাটি নেই সেটি নিশ্চিত হলে) আমরা লুপ থেকে বের হয়ে যাব।

`while(1) {`

$mid\_idx = (low\_idx + high\_idx) / 2;$

`if(num == ara[mid_idx]) {`

`/* num যদি  $ara[mid\_idx]$ -এর সমান হয়, তবে  
সেটি আমরা পেয়ে গেছি */`

`break;`

`}`

`if(num < ara[mid_idx]) {`

`/* num যদি  $ara[mid\_idx]$ -এর ছোট হয়, তবে`

আমরা low\_indx থেকে mid\_indx - 1 সীমার মধ্যে খুঁজব। \*/

    high\_indx = mid\_indx - 1;

}

else {

    /\* num যদি ara[mid\_indx]-এর বড় হয়, তবে

আমরা mid\_indx + 1 থেকে high\_indx সীমার মধ্যে খুঁজব। \*/

    low\_indx = mid\_indx + 1;

}

}

বাইনারি সার্চের প্রোগ্রাম আমরা লিখে ফেললাম। খুবই সহজ-সরল প্রোগ্রাম। সংখ্যাটি খুঁজে না পাওয়া পর্যন্ত লুপটি চলতেই থাকবে, কারণ আমরা লিখেছি while(1) আর 1 সব সময় সত্য। কিন্তু সংখ্যাটি যদি ara-তে না থাকে, তাহলে লুপটি চলতেই থাকবে এবং আমাদের প্রোগ্রাম কখনো বন্ধ হবে না। সুতরাং একটা ব্যবস্থা করা দরকার। আচ্ছা, আমরা কীভাবে বুঝব যে সংখ্যাটি ara-তে নেই? তোমরা ইতিমধ্যে লক্ষ করেছ যে আমরা প্রতিবার সার্চের সীমাটা অর্ধেক করে ফেলি। এভাবে চলতে থাকলে একসময় ওই সীমার ভেতর একটি সংখ্যাই থাকবে। তখন low এবং high-এর মান সমান হবে। আর প্রতিবার যেহেতু হয় low-এর মান বাড়ছে না হয় high-এর মান কমছে, সুতরাং যেবার low আর high সমান হবে, তার পরের বার low-এর মান high-এর মানের চেয়ে বেশি হবে। তখন আমরা বুঝব যে সংখ্যাটি খুঁজে পাওয়া যায়নি। সুতরাং যতক্ষণ low <= high ততক্ষণ লুপটি চলবে। লুপ থেকে বের হয়ে যদি দেখি low > high, তখন বুঝব যে সংখ্যাটি খুঁজে পাওয়া যায়নি, আর না হলে বুঝব সংখ্যাটি খুঁজে পাওয়া গেছে এবং-এর মান ara[mid\_indx]।

তাহলে পুরো প্রোগ্রামটি এবারে লিখে ফেলা যাক :

```
#include <stdio.h>
```

```
int main()
{
    int ara[] = {1, 4, 6, 8, 9, 11, 14, 15,
20, 25, 33 83, 87, 97, 99, 100};
    int low_indx = 0;
    int high_indx = 15;
    int mid_indx;
    int num = 97;
    while (low_indx <= high_indx) {
        mid_indx = (low_indx + high_indx) /
2;
        if (num == ara[mid_indx]) {
            break;
        }
        if (num < ara[mid_indx]) {
            high_indx = mid_indx - 1;
        }
        else {
            low_indx = mid_indx + 1;
        }
    }
    if (low_indx > high_indx) {
        printf("%d is not in the array\n",
num);
    }
    else {
        printf("%d is found in the array.
It is the %d th element of the array.\n",
ara[mid_indx], mid_indx);
    }
}
```

## আব্যাস নথি : স্ট্রিং (string)

আমরা যারা string পদটির ধারণা করছি, তাম্বের অন্তর্ভুক্ত হওয়ার কোনো কাবণ নেই, যোজানিয়ে স্ট্রিং সেটিই সবচেয়ে উন্নতি কইকর বাণিজ্যিক লক্ষ্য। আবার যোগানের ঘৰে কোনো একটি জনৈ প্রাইভেট তাদের মাধ্যাত্মক ইয়াতো স্ট্রিং পিছতি পদটি ফুল এসেছে। আবার একটি উদাহরণের কোনো কাবণ নেই।

এক বা একাধিক character'র মিলে গঠিত হৈবি যা, যেখান কথায় স্ট্রিং হচ্ছে কাবেষ্টোর টাইপের আরেক কথে যোজানিয়ে পিছ ব্যবহার এতই বেশি যে কোনো কোনো স্থানে স্ট্রিংকে অন্তর্ভুক্ত কোটি ডাটা টাইপ হিসেবে ধৰা হয়। তবে সি-কে অন্তর্ভুক্ত টাইপের আরেক দিয়েই স্ট্রিংয়ের কাজ করব।

নিচের উদাহরণগুলো লক্ষ করো :

```
char country[11] = ("B", "a", "n", "g", "l",  
"a", "d", "e", "s", "t", "o", "d");
```

```
char country[] = ("B", "a", "n", "g", "l",  
"a", "d", "e", "s", "t", "o", "d");
```

```
char country[] = "Bangladesh";
```

```
char *country = "Bangladesh";
```

এভাবে আমরা স্ট্রিং ডিক্রেশাক করতে পারি। প্রাপ্তি ডিক্রেশন আসলে একই জিনিস। সবার শেষে একটি Null character (\0) দিলে কম্পাইলার বুঝতে পারে, এখানেই স্ট্রিংের শেষ। আবার তৃতীয় উদাহরণে আরেবে উপাদানগুলো আসলা করে দেখা যাবি, একসঙ্গে দেখা হয়েছে। এ ক্ষেত্ৰে কম্পাইলাৰ নিজেই Null character বিনিয়োগ কৰে। ততুর্থ উপাদানটি একটি অকৃত। এখানে যে জিনিসটা ব্যবহার কৰা হয়েছে তাৰ নাম পয়েন্টাৰ (pointer)। এ বইতে এককম জিনিস আমৰা মাত্ৰে ব্যবহূত কৰবো বিজ্ঞানিক আনোচনীয় ধাৰণা না।

এবাবে প্রোগ্ৰাম সোখাৰ আসল।

## অধ্যায় নং : স্ট্রিং (string)

তোমরা যারা string শব্দটির বাংলা অর্থ জানো, তাদের আতঙ্কিত হওয়ার কোনো কারণ নেই, প্রোগ্রামিংয়ে স্ট্রিং মোটেও দড়ি টানাটানির মতো কষ্টকর ব্যাপার নয়। আবার তোমাদের মধ্যে যারা একটু জ্ঞানী টাইপের, তাদের মাথায় হয়তো স্ট্রিং থিওরি শব্দটি চলে এসেছে। যাই হোক, উদ্বেগের কোনো কারণ নেই।

এক বা একাধিক character মিলে string তৈরি হয়। সোজা কথায় স্ট্রিং হচ্ছে ক্যারেক্টার টাইপের অ্যারে। তবে প্রোগ্রামিংয়ে এটির ব্যবহার এতই বেশি যে কোনো কোনো ল্যাঙ্গুয়েজে স্ট্রিংকে আলাদা একটি ডাটা টাইপ হিসেবে ধরা হয়। তবে সি-তে আমরা char টাইপের অ্যারে দিয়েই স্ট্রিংয়ের কাজ করব।

নিচের উদাহরণগুলো লক্ষ করো :

```
char country[11] = {'B', 'a', 'n', 'g',
                     'l', 'a', 'd', 'e', 's', 'h', '\0'};
char country[] = {'B', 'a', 'n', 'g', 'l',
                  'a', 'd', 'e', 's', 'h', '\0'};
char country[] = "Bangladesh";
char *country = "Bangladesh";
```

এভাবে আমরা স্ট্রিং ডিক্লেয়ার করতে পারি। চারটি ডিফল্যারেশন আসলে একই জিনিস। সবার শেষে একটি Null character ('\0') দিলে কম্পাইলার বুঝতে পারে, এখানেই স্ট্রিংয়ের শেষ। আবার তৃতীয় উদাহরণে অ্যারের উপাদানগুলো আলাদা করে লেখা হয়নি, একসঙ্গে লেখা হয়েছে। এ ক্ষেত্রে কম্পাইলার নিজেই Null character বসিয়ে নেবে। চতুর্থ উদাহরণটি একটু অস্তুত। এখানে যে জিনিসটা ব্যবহার করা হয়েছে তার নাম পয়েন্টার (pointer)। এ বইতে এরকম জিনিস আমরা মাঝে মাঝে ব্যবহার করলেও বিস্তারিত আলোচনায় যাব না।

এবারে প্রোগ্রাম লেখার পালা।

```

#include <stdio.h>
int main()
{
    char country[] = {'B', 'a', 'n', 'g',
    'l', 'a', 'd', 'e', 's', 'h', '\0'};
    printf("%s\n", country);
    return 0;
}

```

প্রোগ্রাম : ৯.১

এখনে লক্ষ করো যে printf-এর ভেতরে %s ব্যবহার করা হয়েছে স্ট্রিং প্রিন্ট করার জন্য। আর অ্যারেতে শেষের 'l'টা ব্যবহার না করলেও চলে আসলে। বর্তমানের কম্পাইলারগুলো এটি বুঝে নিতে পারে।

```

#include <stdio.h>
int main()
{
    char country[] = {'B', 'a', 'n', 'g',
    'l', 'a', 'd', 'e', 's', 'h', ' ', 'i',
    's', ' ', 'm', 'y', ' ', 'c', 'o', 'u',
    'n', 't', 'r', 'y'};
    printf("%s\n", country);
    return 0;
}

```

প্রোগ্রাম : ৯.২

প্রোগ্রামটি চালাও। তারপর নিচের প্রোগ্রামটি চালাও। আউটপুটে কি পার্থক্য দেখতে পাচ্ছ? পার্থক্যের কারণটা কী?

```

#include <stdio.h>
int main()

```

```

{
    char country[] = {'B', 'a', 'n', 'g',
    'l', 'a', 'd', 'e', 's', 'h', '\0', 'i',
    's', ' ', 'm', 'y', ' ', 'c', 'o', 'u',
    'n', 't', 'r', 'y'};
    printf("%s\n", country);
    return 0;
}

```

প্রোগ্রাম : ৯.৩

পার্থক্যটা কী, সেটি তোমরা প্রোগ্রাম দুটি কম্পিউটারে চালানেই বুঝবে। পার্থক্যের কারণ হচ্ছে দ্বিতীয় প্রোগ্রামে স্ট্রিংয়ের ভেতরে এক জায়গায় 'l' থাকায় কম্পাইলার ধরে নিচ্ছে, ওখানে স্ট্রিংটা শেষ হয়ে গেছে।

এবাবে আমরা একটি প্রোগ্রাম লিখব। একটি স্ট্রিংয়ের ভেতরের সব অক্ষরকে বড় হাতের অক্ষরে (অর্থাৎ capital letter বা uppercase character) রূপান্তর করা। তবে এর জন্য আমাদের একটি জিনিস জানতে হবে। প্রতিটি অক্ষরের বিপরীতে কম্পিউটার একটি সংখ্যার কোড ব্যবহার করে। সেই কোড অনুযায়ী, 'A'-এর মান হচ্ছে 65, 'B'-এর মান হচ্ছে 66, 'C'-এর মান হচ্ছে 67, এভাবে 'Z'-এর মান হচ্ছে 90। তেমনই 'a' হচ্ছে 97, 'b' হচ্ছে 98, এভাবে 'z' হচ্ছে 122। সুতরাং কোনো ক্যারেক্টার বড় হাতের কি না সেটি আমরা নির্ণয় করতে পারি এভাবে:

if(ch >= 'A' && ch <= 'Z') অথবা if(ch >= 65 && ch <= 90)।

তেমনই ছোট হাতের অক্ষরের জন্য : if(ch >= 'a' && ch <= 'z')  
অথবা if(ch >= 97 && ch <= 122)।

এখন কোনো ক্যারেক্টার যদি ছোট হাতের হয়, তাহলে তাকে বড় হাতের অক্ষরে রূপান্তর করার উপায় কী? উপায় খুব সহজ। একটি উদাহরণ দেখো :

```
char ch = 'c';
ch = 'A' + (ch - 'a');
```

এখানে যেটি হচ্ছে, প্রথমে ch থেকে 'a' বিয়োগ করা হচ্ছে মানে 'c' থেকে 'a' বিয়োগ (আসলে 99 থেকে 97 বিয়োগ হচ্ছে)। বিয়োগফল 2। এবারে 'A'-এর সঙ্গে যদি ওই 2 মোগ করে দিই তবে সেটি 'C' হয়ে যাবে!

এখন প্রোগ্রামটি লিখে ফেলা যাক :

```
#include <stdio.h>
int main()
{
    char country[] = {'B', 'a', 'n', 'g',
                      'l', 'a', 'd', 'e', 's', 'h'};
    int i, length;
    printf("%s\n", country);
    length = 10;
    for(i = 0; i < length; i++) {
        if(country[i] >= 97 && country[i]
        <= 122) {
            country[i] = 'A' + (country[i]
            - 'a');
        }
    }
    printf("%s\n", country);
    return 0;
}
```

প্রোগ্রাম : ৯.৮

এখন তোমরা uppercase থেকে lowercase-এ রূপান্তরের প্রোগ্রামটি লিখে ফেলো। তারপরে আবার বইটি পড়া শুরু করো।

এখানে লক্ষ করো যে স্ট্রিংয়ে (বা ক্যারেক্টারের অ্যারেতে) মোট কয়টি উপাদান আছে, সেটি আমি দেখেই লিখে ফেলেছি এবং সরাসরি বসিয়ে দিয়েছি `length = 10`।

এবার আমরা কোনো স্ট্রিংয়ের দৈর্ঘ্য মাপার জন্য একটি ফাংশন লিখব। এটি তেমন কঠিন কিছু নয়। একটি লুপের সাহায্যে স্ট্রিংয়ের প্রতিটি উপাদান পরীক্ষা করতে হবে এবং Null character ('\0') পেলে লুপ থেকে বের হয়ে যাবে অর্থাৎ, '\0' না পাওয়া পর্যন্ত লুপ চলতে থাকবে। আর লুপ যতবার চলবে স্ট্রিংয়ের দৈর্ঘ্যও তত হবে।

```
#include <stdio.h>
int string_length(char str[])
{
    int i, length = 0;
    for(i = 0; str[i] != '\0'; i++) {
        length++;
    }
    return length;
}
int main()
{
    char country[100];
    int length;
    while(1 == scanf("%s", country)) {
        length = string_length(country);
        printf("length: %d\n", length);
    }
    return 0;
}
```

প্রোগ্রাম : ৯.৫

ওপরের প্রোগ্রামটায় তোমরা দেখতে পাচ্ছ যে ইনপুট নেওয়ার জন্য scanf ফাংশন ব্যবহার করা হয়েছে এবং স্ট্রিং ইনপুট নেওয়ার জন্য %s ব্যবহৃত হয়েছে। scanf ফাংশনটি যতটি উপাদান ইনপুট হিসেবে নেয়, সেই সংখ্যাটি রিটার্ন করে। সাধারণত রিটার্ন ভ্যালুটি আমাদের দরকার হয় না, তাই scanf ব্যবহার করলেও আমরা ওই ভ্যালুটি রাখি না। যেমন দুটি ইন্টিজার ইনপুট নিতে গেলে আমরা লিখি : scanf("%d %d", &n1, &n2);। আমরা এটি চাইলে এভাবেও লিখতে পারতাম : value = scanf("%d %d", &n1, &n2);। তোমরা প্রিন্ট করলে দেখবে value-এর মান 2। while(1 == scanf("%s", country)) লাইনে যেটি ঘটছে তা হলো, যতক্ষণ একটি country-এর নাম scanf দিয়ে ইনপুট নেওয়া হচ্ছে, ফাংশনটি 1 রিটার্ন করছে, আর লুপের ভেতরের কন্ডিশন সত্য হচ্ছে (1 == 1), তাই লুপের কাজ চলতে থাকবে।

আরেকটি জিনিস খেয়াল করো যে country-এর আগে কোনো & চিহ্ন ব্যবহার করা হয়নি। তোমরা &country লিখে দেখো, প্রোগ্রামটি কী আচরণ করে। তবে %s ব্যবহারের একটি সমস্যা হচ্ছে, স্ট্রিংয়ে কোনো হোয়াইটস্পেস ক্যারেক্টার (যেমন : স্পেস, ট্যাব ইত্যাদি) থাকা যাবে না, এমন কিছু পেলে scanf ওই ক্যারেক্টার পর্যন্ত একটি স্ট্রিং ধরে নেয়। যেমন, ইনপুট যদি হয় this is তাহলে scanf প্রথমে this-কেই স্ট্রিং হিসেবে নেবে, তারপরে যদি আবার scanf ফাংশন কল করা হয়, তবে is-কে সে স্ট্রিং হিসেবে ইনপুট নিয়ে নেবে। এই সমস্যা এড়ানোর জন্য আমরা gets ফাংশন ব্যবহার করতে পারি। নিচের উদাহরণটি দেখো :

```
#include <stdio.h>
int main()
{
    char ara[100];
    while(NULL != gets(ara))
        printf("%s\n", ara);
```

126

```
}
```

```
return 0;
```

)  
প্রোগ্রাম : ৯.৬

এই প্রোগ্রামটিও চলতে থাকবে, যতক্ষণ না তুমি ctrl + z (মানে কি-বোর্ডে ctrl ও z একসঙ্গে) চাপো, লিনার্সের জন্য ctrl + d। ctrl + z বা ctrl + d দিলে gets ফাংশনটি NULL রিটার্ন করে। আরেকটি জিনিস লক্ষ করো যে আমি char ara[100]; ডিক্লেয়ার করে শুরুতেই বলে দিয়েছি, স্ট্রিংয়ের সর্বোচ্চ দৈর্ঘ্য হবে 100।

আরেকটি ব্যাপার। string\_length ফাংশনের ভেতরে আসলে দুটি ভেরিয়েবল ব্যবহার না করলেও চলে। আমরা ফাংশনটি এভাবেও লিখতে পারি :

```
int string_length(char str[])
{
    int i;
    for(i = 0; str[i] != '\0'; i++);
    return i;
}
```

এখন তোমাদের কাজ হবে string\_length ফাংশনটি for লুপ ব্যবহার না করে while লুপ ব্যবহার করে লেখা।

আমাদের পরবর্তী প্রোগ্রামের লক্ষ্য হবে দুটি স্ট্রিং জোড়া দেওয়া বা concatenate করা। যেমন একটি স্ট্রিং যদি হয় "bangla" এবং আরেকটি স্ট্রিং যদি হয় "desh" তবে দুটি জোড়া দিয়ে "bangladesh" বানাতে হবে।

প্রথমেই স্ট্রিংগুলো ডিক্লেয়ার করে নিই: char str1[] = "bangla", str2[] = "desh", str3[12];

আমাদের লক্ষ্য হচ্ছে str3-তে "bangladesh" রাখা। খুব সুবিধা হতো যদি আমরা এমন কিছু লিখতে পারতাম:

127

```
str3 = str1 + str2;
```

কিন্তু 'সি'-তে এভাবে দুটি অ্যারে বা স্ট্রিং যোগ করা যায় না। তাই একটি একটি করে str1-এর উপাদানগুলো str3-তে কপি করতে হবে, তারপর str2-এর উপাদানগুলো str3-তে কপি করতে হবে।

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    char str1[] = "bangla", str2[] =  
    "desh", str3[12];  
  
    int i, j, length1 = 6, length2 = 4;  
    for(i = 0, j = 0; i < length1; i++,  
        j++) {  
        str3[j] = str1[i];  
    }  
  
    for(i = 0, j = 0; i < length2; i++,  
        j++) {  
        str3[j] = str2[i];  
    }  
  
    str3[j] = '\0';  
    printf("%s\n", str3);  
    return 0;  
}
```

প্রোগ্রাম : ৯.৭

প্রোগ্রামটি চলাও। আউটপুট কী আসা উচিত? bangladesh। কিন্তু আউটপুট এসেছে desh। আসলে আমরা কিছু একটা ভুল করেছি। তোমাদের এখন সেই ভুলটি ঠিক করার চেষ্টা করা উচিত। অন্তত সিলিং মিনিট চেষ্টার পরও যদি ভুল বের করতে না পারো, তাহলে আমি বইটি পঢ়া শুরু করো।

```
for(i = 0, j = 0; i < length1; i++, j++) {  
    str3[j] = str1[i];  
}
```

এখানে আমরা শুরুতেই i-এর মান 0 করেছি কারণ i-কে আমরা str1-এর ইনডেক্স হিসেবে ব্যবহার করব। j-কে ব্যবহার করব str3-এর ইনডেক্স হিসেবে তাই j-এর মানও 0 করা হয়েছে। তারপর একে একে str1-এর উপাদানগুলো str3-তে কপি করাই এবং i ও j-এর মান 1 করে বাড়াচ্ছি (i++, j++)। লুপ শেষ হওয়ার পরে i ও j প্রত্যক্ষের মান হবে 6।

এখন পরের লুপে আমরা str2-কে str3-তে কপি করব। এখন str2-এর ইনডেক্স হিসেবে যদি i ব্যবহার করি, তাহলে তার মান লুপের শুরুতেই আবার 0 করে দিতে হবে। আমরা সেটি করেছি। কিন্তু ভুল করেছি সেই সঙ্গে j-এর মান 0 করে দিয়ে। j-এর মান 0 করলে তো str2-এর প্রথম (0তম) উপাদান str3-এর প্রথম (0তম) উপাদান হিসেবে কপি হবে, কিন্তু আমরা তো সেটি চাই না। আমরা চাই str2-এর প্রথম উপাদান হবে str3-এর সপ্তম উপাদান। তাহলে j-এর মান 0 করা যাবে না। তাই দ্বিতীয় লুপটি হবে এমন :

```
for(i = 0; i < length2; i++, j++) {  
    str3[j] = str2[i];  
}
```

আরেকটি ব্যাপার লক্ষ করো। দ্বিতীয় লুপ থেকে বের হবার পরে str3-এর শেষ ঘরে '\0' অ্যাসাইন করেছি (str3[j] = '\0';) যাতে স্ট্রিংটা যে ওখানেই শেষ, এটি কম্পাইলার বুবতে পারে।

আমাদের পরবর্তী প্রোগ্রাম হবে দুটি স্ট্রিংয়ের মধ্যে তুলনা করা। অর্থাৎ দুটি স্ট্রিংয়ের মধ্যে ছোট, বড়, সমান নির্ণয় করা। সংখ্যার ক্ষেত্রে যেমন >, <, >=, <=, == চিহ্ন ব্যবহার করে তুলনা করা যায়, স্ট্রিংয়ের ক্ষেত্রে সে ব্যবস্থা নেই। কিন্তু স্ট্রিংয়ের ক্ষেত্রে প্রায়ই আমাদের এই তুলনা করার দরকার পড়বে। যেমন ধরো, সর্টিংয়ের ক্ষেত্রে যেখানে ছোট থেকে বড় বা বড় থেকে ছোট ক্রমানুসারে

সাজাতে হবে (alphabetical ordering)।

স্ট্রিংয়ে ছোট-বড় আবার কী? বেশি কথা বলে ব্যাখ্যা না করে কিছু উদাহরণ দিই, তাহলেই বুঝতে পারবে।

'aaa'-এর চেয়ে 'aab' বড়। আবার 'ba' ও 'ca'-এর মধ্যে 'ca' বড়।

এই প্রোগ্রামে আমরা একটি ফাংশন লিখব স্ট্রিং\_কম্পারে() যেটির কাজ হবে দুটি স্ট্রিংয়ের মধ্যে তুলনা করে প্রথমটি দ্বিতীয়টির চেয়ে বড় হলে 1 রিটোর্ন করবে, ছোট হলে -1 আর দুটি সমান হলে 0 রিটোর্ন করবে।

ফাংশনের রিটোর্ন টাইপ হবে ইন্টিজার এবং প্যারামিটার হবে দুটি char টাইপের অ্যারে।

```
int string_compare(char a[], char b[])
{
}
```

আমাদের মূল কাজ হবে a-এর প্রথম উপাদানের সঙ্গে b-এর প্রথম উপাদান, a-এর দ্বিতীয় উপাদানের সঙ্গে b-এর দ্বিতীয় উপাদান এভাবে তুলনা করতে থাকা। যখনই a-এর কোনো উপাদান b-এর কোনো উপাদানের চেয়ে ছোট হবে, আমরা সঙ্গে সঙ্গে বলে দিতে পারি যে a, b-এর চেয়ে ছোট। সুতরাং -1 রিটোর্ন করে ফাংশন থেকে বের হয়ে আসব। একইভাবে যখনই a-এর কোনো উপাদান b-এর কোনো উপাদানের চেয়ে বড় হবে, সঙ্গে সঙ্গে 1 রিটোর্ন করে ফাংশন থেকে বের হয়ে আসব। কারণ, a, b-এর চেয়ে বড়। কিন্তু যদি সবগুলোই সমান হয়? তখন আমরা 0 রিটোর্ন করব। তাতে বুঝব যে স্ট্রিং দুটি সমান।

```
int string_compare(char a[], char b[])
{
    int i, j;
    for(i = 0; a[i] != '\0' && b[i] != '\0'; i++)
    {
```

```
        if(a[i] < b[i])
            return -1;
        }
        if(a[i] > b[i])
            return 1;
        }
        if(string_length(a) == string_length(b))
            return 0;
        if(string_length(a) < string_length(b))
        {
            return -1;
        }
        if(string_length(a) > string_length(b))
        {
            return 1;
        }
    }
```

স্ট্রিংয়ের বেসিক জিনিসগুলো নিয়ে আলোচনা করলাম। তবে মজার ব্যাপার হচ্ছে, সি ল্যাঙ্গুয়েজে একটি হেডার ফাইল আছে, যার নাম string.h এবং ওইখানে বেশিরভাগ স্ট্রিং-সংক্রান্ত কাজের জন্য ফাংশন তৈরি করে দেওয়া আছে (যেমন : strcmp, strlen, strcpy ইত্যাদি)। তোমাদের দিয়ে কাজগুলো আমি আবার করালাম বলে দৃঢ় পাওয়ার কোনো কারণ নেই, আমার ওপর রাগ করারও কিছু নেই। মৌলিক জিনিসগুলো শিখে রাখা সব সময়ই গুরুত্বপূর্ণ, যা তোমার প্রোগ্রামিং চিন্তাকে বিকশিত করবে।

এখন আমরা আরেকটি প্রোগ্রাম লিখব, যেটি ইনপুট হিসেবে একটি

স্ট্রিং নেবে (যেখানে অনেকগুলো শব্দ থাকবে)। এই স্ট্রিংয়ের সর্বোচ্চ দৈর্ঘ্য হবে 1000। শব্দগুলোর মাঝখানে এক বা একাধিক স্পেস থাকবে। আউটপুট হিসেবে প্রতিটি শব্দ আলাদা লাইনে প্রিন্ট করতে হবে। বিরামচিহ্নগুলো (punctuation) প্রিন্ট করা যাবে না এবং শব্দের প্রথম অক্ষর হবে বড় হাতের।

অনেক শর্ত দিয়ে ফেললাম। তবে প্রোগ্রামটি খুব কঠিন কিছু নয়।  
নিজে নিজে চেষ্টা করতে পারো।

আর না পারলে এখন চলো দেখি কীভাবে সমাধান করা যায়।

প্রথম কথা হচ্ছে, ইনপুট নেব কীভাবে? বুঝতেই পারছ যে ইনপুটে যেহেতু স্পেস থাকবে, `scanf("%s")` ব্যবহার করা যাবে না। তাই আমরা `gets()` ব্যবহার করব। তার পরের কথা হচ্ছে একটি শব্দে কোন কোন ক্যারেক্টর থাকতে পারে? যেহেতু বলা নেই, আমরা ধরে নিই 'a' থেকে 'z', 'A' থেকে 'Z' আর '0' থেকে '9' থাকবে।

তার পরের প্রশ্ন হচ্ছে, আমরা কখন বুঝব বা আমাদের প্রোগ্রামকে কীভাবে বোঝাব যে একটি শব্দ শুরু হয়েছে? এর জন্য আমরা একটি ভেরিয়েবল রাখতে পারি। ভেরিয়েবলের নাম যদি দিই `is_word_started`, তাহলে এর মান 0 হলে বুঝব শব্দ শুরু হয়ন।  
শব্দ শুরু হলে এর মান আমরা 1 করে দেব। আবার শব্দ শেষ হলে 0 করে দেব। যখন দেখব শব্দ শুরু হয়ে গেছে (`is_word_started`-এর মান 1) কিন্তু কোনো ক্যারেক্টরের মান 'a' – 'z' বা 'A' – 'Z', বা '0' – '9' এই রেঞ্জের মধ্যে নেই, তখনই বুঝব শব্দটি শেষ। তোমরা যদি এর আগে প্রোগ্রামটি চেষ্টা করার পরও লিখতে না পারো, এখন চেষ্টা করলে পারবে আশা করি। আমি এখন কোডটি লিখে দেব, তবে সেটি দেখার আগে অবশ্যই নিজে করার চেষ্টা করতে হবে।

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s[1002], word[100];
```

```
    int i, j, length, is_word_started;
    gets(s);
    length = strlen(s);
    is_word_started = 0;
    for (i = 0, j = 0; i < length; i++) {
        if (s[i] >= 'a' && s[i] <= 'z') {
            if (is_word_started == 0) {
                is_word_started = 1;
                word[j] = 'A' + s[i] - 'a';
                // first character is capital
                j++;
            }
            else {
                word[j] = s[i];
                j++;
            }
        }
        else if (s[i] >= 'A' && s[i] <=
'Z') {
            if (is_word_started == 0) {
                is_word_started = 1;
            }
            word[j] = s[i];
            j++;
        }
        else if (s[i] >= '0' && s[i] <=
'9') {
            if (is_word_started == 0) {
```

```

        is_word_started = 1;
    }
    word[j] = s[i];
    j++;
}
else {
    if (is_word_started == 1) {
        is_word_started = 0;
        word[j] = '\0';
        printf("%s\n", word);
        j = 0;
    }
}
return 0;
}

```

প্রোগ্রাম : ৯.৮

প্রোগ্রামটি বুঝতে কি একটু সমস্যা হচ্ছে? সে পরে দেখা যাবে। আগে প্রোগ্রামটি চটপট কম্পিউটারে টাইপ করে ফেলো, কম্পাইল ও রান করো। যারা লিনাক্স ব্যবহার করছ তারা gets() ব্যবহারের কারণে কম্পাইলার থেকে একটি সতর্ক সংকেত (warning) পেতে পারো, পাত্তা দিয়ে না।

ইনপুট হিসেবে যেকোনো কিছু লিখতে পারো। যেমন : This is a test। আউটপুট কী?

আউটপুট হচ্ছে এই রকম :

This

is

A

কী মুশকিল! test গেল কোথায়?

এখন তোমার কাজ হবে test-এর নিখেঁজ হওয়ার রহস্যটা তদন্ত করা। তারপর আমি প্রোগ্রামটি ব্যাখ্যা করব।

তোমরা দেখো প্রোগ্রামে আমি স্ট্রিংয়ের দৈর্ঘ্য নির্ণয়ের জন্য strlen ফাংশন ব্যবহার করেছি। আর-এর জন্য আমাকে string.h হেডার ফাইলটি include করতে হয়েছে। ইনপুট হিসেবে স্ট্রিংটা নিলাম s-এ। আর word রাখার জন্য একটি অ্যারে ডিক্লেয়ার করে রেখেছি। তারপর আমি i = 0 থেকে length পর্যন্ত একটি লুপ চালিয়েছি s-এর ভেতরের প্রতিটি ক্যারেক্টার পরীক্ষা করার জন্য।

if (s[i] >= 'a' && s[i] <= 'z') দিয়ে পরীক্ষা করলাম এটি ছোট হাতের অক্ষর নাকি। যদি ছোট হাতের অক্ষর হয়, তাহলে একটি শব্দের প্রথম অক্ষর কি না সেটি জানতে হবে। কারণ প্রথম অক্ষর হলে ওটাকে আবার বড় হাতের অক্ষরে রূপান্তর করতে হবে। সেই পরীক্ষাটা আমরা করেছি : if (is\_word\_started == 0) দিয়ে। এটি সত্য হওয়া মানে শব্দ শুরু হয়নি, এটিই প্রথম অক্ষর। তাই আমরা is\_word\_started-এর মান 1 করে দেব। আর word[j]-তে s[i]-এর বড় হাতের অক্ষরটা নেব। তারপর j-এর মান এক বাড়াতে হবে।

else if (s[i] >= 'A' && s[i] <= 'Z') এবং else if (s[i] >= '0' && s[i] <= '9') এই দুটি শর্তের ভেতরেই আমরা একই কাজ করি। s[i]-কে word[j]-তে কপি করি। তাই চাইলে দুটি শর্তকে একসঙ্গে এভাবেও লিখতে পারতাম :

else if ((s[i] >= 'A' && s[i] <= 'Z') || (s[i] >= '0' && s[i] <= '9'))

তার পরের else-এর ভেতরে ঢোকার মানে হচ্ছে আগের if ও else if-এর শর্তগুলো মিথ্যা হয়েছে। তাই s[i]-এর ভেতরে যেই ক্যারেক্টার আছে সেটি word-এ রাখা যাবে না। এবং যদি word ইতিমধ্যে শুরু হয়ে গিয়ে থাকে, তাহলে সেটি শেষ করতে হবে এবং wordটি প্রিন্ট করতে হবে। আর যদি word শুরু না হয়ে থাকে, তাহলে কিছু করার

দরকার নেই।

```
else {
    if (is_word_started == 1) {
        is_word_started = 0;
        word[j] = '\0';
        printf("%s\n", word);
        j = 0;
    }
}
```

তোমরা কি test-রহস্য সমাধান করতে পেরেছ? তোমরা চেষ্টা করতে থাকো আর আমি এখন প্রোগ্রামটি অন্যভাবে লিখব (এর সঙ্গে test-রহস্যের কোনো সম্পর্ক নেই, সেটি বলে রাখলাম)।

এখন আমি যেটি করব, প্রোগ্রামটি এমনভাবে লিখব, যাতে word অ্যারেটিই ব্যবহার করতে না হয়! একটু চিন্তা করে দেখো। আসলে তো এই অ্যারেটি নিয়ে আমরা কিছু করছি না প্রিন্ট করা ছাড়া। তাই এর আসলে কোনো দরকার নেই।

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s[1002], ch;
    int i, length, is_word_started;
    gets(s);
    length = strlen(s);
    is_word_started = 0;
    for (i = 0; i < length; i++) {
        if (s[i] >= 'a' && s[i] <= 'z')
```

```
        if (is_word_started == 0) {
            is_word_started = 1;
            ch = 'A' + s[i] - 'a';
            printf("%c", ch);
        }
        else {
            printf("%c", s[i]);
        }
    }
    else if ((s[i] >= 'A' && s[i] <=
'Z') || (s[i] >= '0' && s[i] <= '9')) {
        if (is_word_started == 0) {
            is_word_started = 1;
        }
        printf("%c", s[i]);
    }
    else {
        if (is_word_started == 1) {
            is_word_started = 0;
            printf("\n");
        }
    }
    printf("\n");
    return 0;
}
```

প্রোগ্রাম : ৯.৯

এখন প্রোগ্রামটি বুকতে চেষ্টা করো এবং বিভিন্ন ইনপুট দিয়ে পরীক্ষা  
করে দেখো। যেমন :

This is test number 9.9

স্ট্রিং-সংক্রান্ত সমস্যাগুলো দেখতে জটিল মনে হলেও আসলে সহজ।  
আর এ ধরনের সমস্যা সমাধানের যত চৰ্চা করবে দক্ষতা তত  
বাড়বে।

## অধ্যায় দশ : মৌলিক সংখ্যা

মৌলিক সংখ্যা (Prime Number) গণিতবিদদের কাছে যেমন প্রিয়,  
তেমনই প্রোগ্রামারদেরও অনেক প্রিয় একটি বিষয়। তোমাদের বিভিন্ন  
সময়ে এই মৌলিক সংখ্যাসংক্রান্ত নানা সমস্যার সমাধান করতে  
হবে। মৌলিক সংখ্যা জিনিসটি যে গুরুত্বপূর্ণ সেটি বোঝার আরেকটি  
উপায় হলো, এই বইতে বিষয়টির জন্য আমি একটি পৃথক অধ্যায়  
বরাদ্দ করেছি।

মৌলিক সংখ্যা হচ্ছে সেসব সংখ্যা, যারা 1-এর চেয়ে বড় পূর্ণসংখ্যা  
এবং সেটি কেবল 1 এবং ওই সংখ্যাটি দ্বারাই নিঃশেষে বিভাজ্য হবে।  
খুবই সহজ-সরল জিনিস। এখন কোনো সংখ্যা মৌলিক কি না সেটি  
বের করার জন্য একটি প্রোগ্রাম লিখে ফেলা যাক।

```
#include <stdio.h>
int is_prime(int n)
{
    int i;
    if (n < 2) {
        return 0;
    }
    for(i = 2; i < n; i++) {
        if(n % i == 0) {
            return 0;
        }
    }
    return 1;
}
int main()
{
    int n;
```

```

while(1) {
    printf("Please enter a number
    (enter 0 to exit): ");
    scanf("%d", &n);
    if(n == 0) {
        break;
    }
    if(1 == is_prime(n)) {
        printf("%d is a prime
number.\n", n);
    }
    else {
        printf("%d is not a prime
number.\n", n);
    }
}
return 0;
}

```

প্রোগ্রাম : ১০.১

মৌলিক সংখ্যা নির্ণয়ের জন্য আমরা একটি ফাংশন লিখেছি, যেটির প্যারামিটার হচ্ছে একটি ইন্টিজার নম্বর  $n$ । ফাংশনে আমরা  $n$ -কে 2 থেকে  $n-1$  পর্যন্ত সংখ্যাগুলো দিয়ে ভাগ করার চেষ্টা করেছি একটি লুপের সাহায্যে। যদি এর মধ্যে কোনো সংখ্যা দিয়ে  $n$  নিঃশেষে বিভাজ্য হয়, তাহলে আমরা সঙ্গে সঙ্গেই বলে দিতে পারি যে সেটি মৌলিক সংখ্যা নয়। সেক্ষেত্রে লুপ থেকে বের হয় ফাংশনটি 0 রিটুর্ন করে। আর যদি সব সংখ্যা দিয়ে ভাগ করার পরও দেখা যায় যে কোনো সংখ্যাই  $n$ -কে নিঃশেষে ভাগ করতে পারেনি, তখন আমরা এই সিদ্ধান্তে আসতে পারি যে  $n$  একটি মৌলিক সংখ্যা। আর তখন ফাংশন থেকে 1 রিটুর্ন করি। আমরা মৌলিক সংখ্যা নির্ণয় করা শিখে

### গোলাম!

আমি প্রোগ্রামটি লেখার সময় যে পথ অবলম্বন করেছি সেটি হচ্ছে শুরু সহজ-সরল পথ। প্রোগ্রামটিকে মোটেও ইফিশিয়েন্ট (efficient) বানানোর চেষ্টা করিনি। তোমরা শুরু সহজেই ব্যাপোরটি বৃক্ষতে পাবে। প্রোগ্রামে ইনপুট হিসেবে 2147483647 দাও। এটি যে মৌলিক সংখ্যা সেটি বের করতে বেশ সহজ লাগে। কারণ তখন 2147483647-কে 2 থেকে 2147483646 পর্যন্ত সব সংখ্যা নিয়ে ভাগ করার ব্যর্থ চেষ্টা করা হয়। প্রোগ্রামটিকে আরও ইফিশিয়েন্ট করতে হবে। একটি শুরু তোমাদের মাঝার এর মধ্যেই নিষ্ঠাই এসে গোছে। সেটি হচ্ছে 2 থেকে  $n-1$  পর্যন্ত সব সংখ্যা নিয়ে ভাগ করার চেষ্টা না করে 2 থেকে  $n/2$  পর্যন্ত সংখ্যাগুলো নিয়ে ভাগ করার চেষ্টা করলেই হয়। তাহলে প্রোগ্রামের গতি বিঃগু হয়ে যাবে।

এখন তোমরা আবেক্ষণি বিষয় লক্ষ করো। কোনো সংখ্যা যদি 2 নিয়ে নিঃশেষে বিভাজ্য না হয়, তাহলে সেটি অন্য কোন জোড় সংখ্যা নিয়ে নিঃশেষে বিভাজ্য হওয়ার প্রশংস্তি আসে না। তাই 2 বাদে অন্য জোড় সংখ্যাগুলো ( $4, 6, 8, \dots$ ) নিয়ে ভাগ করার চেষ্টা করাটা আসলে বোকামি। জোড় সংখ্যা নিয়ে বিভাজ্যতার পরীক্ষাটি আমরা ফাংশনের প্রক্রিয়েই করে নিতে পারি।

এখন আমাদের ফাংশনটির চেহুরা দাঁড়াবে এই রকম :

```

int is_prime(int n)
{
    int i;
    if (n < 2) {
        return 0;
    }
    if (n == 2) {
        return 1;
    }
}

```

```

if(n % 2 == 0) {
    return 0;
}

for(i = 3; i <= n / 2; i = i + 2) {
    if(n % i == 0) {
        return 0;
    }
}

return 1;
}

```

প্রথমে আমরা পরীক্ষা করেছি  $n$ -এর মান 2 কি না। যদি 2 হয়, তাহলে বলে দিয়েছি যে  $n$  মৌলিক সংখ্যা। তারপরে আমরা পরীক্ষা করেছি  $n$  জোড় সংখ্যা কি না। যদি জোড় হয়, তাহলে  $n$  মৌলিক সংখ্যা নয়, কেবল 2ই একমাত্র জোড় মৌলিক সংখ্যা, যেটির পরীক্ষা আমরা একেবারে শুরুতেই করে ফেলেছি। তারপর আমরা 3 থেকে  $n / 2$  পর্যন্ত সব বেজোড় সংখ্যা দিয়ে  $n$ -কে ভাগ করার চেষ্টা করেছি। এখন তোমরা বিভিন্ন ইনপুট দিয়ে প্রোগ্রামটি পরীক্ষা করে দেখো। 2147483647 দিয়ে পরীক্ষা করলে বুঝতে পারবে যে প্রোগ্রামের গতি আগের চেয়ে বেড়েছে কিন্তু তার পরও একটু সময় লাগছে। আমার কম্পিউটারে চার সেকেন্ডের মতো সময় লাগছে। কিন্তু এত সময় তো দেওয়া যাবে না।

তোমাদের, যদের গণিতিক বুদ্ধিশুद্ধি বেশি, তারা একটু চিন্তা করলেই প্রোগ্রামটির গতি বাড়ানোর একটি উপায় বের করে ফেলতে পারবে। সেটি হচ্ছে  $n$ -এর উৎপাদক বের করার জন্য আসলে  $n / 2$  পর্যন্ত সব সংখ্যা দিয়ে পরীক্ষা করার দরকার নেই।  $n$ -এর বর্গমূল পর্যন্ত পরীক্ষা করলেই হয়।  $n = p \times q$  হলে,  $p$  বা  $q$  যেকোনো একটি সংখ্যা অবশ্যই  $n$ -এর বর্গমূলের সমান বা তার ছোট হবে। বর্গমূল নির্ণয়ের জন্য আমরা math.h হেডার ফাইলের sqrt()

ফাংশনটি ব্যবহার করব।  
আমাদের প্রোগ্রামটি দাঁড়াচ্ছে এই রকম:

```

#include <stdio.h>
#include <math.h>
int is_prime(int n)
{
    int i, root;

    if(n == 2) {
        return 1;
    }

    if(n % 2 == 0) {
        return 0;
    }

    root = sqrt(n);
    for(i = 3; i <= root; i = i + 2) {
        if(n % i == 0) {
            return 0;
        }
    }

    return 1;
}

int main()
{
    int n, m;
    while(1) {
        printf("Please enter a number

```

```

    (enter 0 to exit): "
        scanf("%d", &n);
        if(n == 0) {
            break;
        }
        if(1 == is_prime(n)) {
            printf("%d is a prime
number.\n", n);
        }
        else {
            printf("%d is not a prime
number.\n", n);
        }
    }
    return 0;
}

```

প্রোগ্রাম : ১০.২

এখন তোমরা প্রোগ্রামটি চালিয়ে বিভিন্ন ইনপুট দিয়ে পরীক্ষা করে দেখো। একটি কথা বলে দিই। প্রোগ্রামটায় একটি বাগ আছে (মানে ভুল আছে)। সেটি খুঁজে বের করে ঠিক করে ফেলো।

প্রাইম নম্বর বের করতে পেরে তোমরা নিশ্চয়ই বেশ খুশি? কিন্তু আমাদের চেষ্টা এখানেই থেমে থাকবে না। আমরা এখন দেখব আরেকটি চমৎকার পদ্ধতি, গ্রিক গণিতবিদ ইরাতোস্থেনেস (Eratosthenes) আজ থেকে দুই হাজার বছরেরও আগে এই পদ্ধতি আবিষ্কার করেছিলেন। এজন্য এর নাম হচ্ছে সিভ অব ইরাতোস্থেনেস (Sieve of Eratosthenes)। পদ্ধতিটি ব্যাখ্যা করা যাক।

ধরো, আমরা 2 থেকে 40 পর্যন্ত সব মৌলিক সংখ্যা বের করব।

ওরুতে সব সংখ্যা লিখে ফেলি :

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,  
33, 34, 35, 36, 37, 38, 39, 40.

এখন দেখো, তালিকার প্রথম সংখ্যা হচ্ছে 2। এবারে 2-এর সব গুণিতক (2 বাদে, মানে 2-এর চেয়ে বড়গুলো আরকি) বাদ দিয়ে দাও। তাহলে থাকবে :

2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31,  
33, 35, 37, 39.

এখন তালিকার দ্বিতীয় সংখ্যা 3-এর সব গুণিতক বাদ দাও।

2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37.

এখন তালিকার তৃতীয় সংখ্যা 5-এর সব গুণিতক বাদ দাও।

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37.

পরবর্তী সংখ্যা হচ্ছে 7 কিন্তু সেটির গুণিতক খোঁজার চেষ্টা করা বৃথা। কারণ তালিকার সর্বোচ্চ সংখ্যা 37-এর বর্গমূল 7-এর চেয়ে ছোট। সুতরাং 7-এর যে গুণিতকগুলো তালিকায় ছিল সেগুলো ইতিমধ্যে তালিকা থেকে বাদ পড়েছে। কারণটি বুঝতে সমস্যা হচ্ছে? দেখো 7-এর গুণিতকগুলো ছিল 14, 21, 28, 35। 7-এর সঙ্গে যেসব সংখ্যা গুণ করে ওই গুণিতকগুলো পাওয়া যায়, সেগুলো সবই 7-এর চেয়ে ছোট সংখ্যা এবং তাদের গুণিতকগুলো আমরা ইতিমধ্যেই বাদ দিয়েছি।

এবারে ইমপ্লিমেন্ট করার পালা। আমরা তালিকা রাখার জন্য একটি আরে ব্যবহার করব। ধরা যাক, তার নাম হচ্ছে ara। আরেটি এমনভাবে তৈরি করতে হবে, যাতে কোনো একটি সংখ্যা n-এর অবস্থা (অর্থাৎ সেটি মৌলিক কি না) ara[n] দিয়ে প্রকাশ করা যায়। যদি ara[n]-এর মান 1 হয়, তাহলে n মৌলিক সংখ্যা আর ara[n]-এর মান 0 হলে n মৌলিক সংখ্যা নয়। ইমপ্লিমেন্টেশনের আগে অ্যালগরিদমটা লেখা যাক :

ধাপ ১ : ধরা যাক, অ্যারেতে  $n^{\text{th}}$  উপাদান আছে। শুরুতে অ্যারের সব উপাদানের মান 1 বসাই।

ধাপ ২ : অ্যারের প্রতিটি উপাদানের জন্য সেটির মান 1 কি না, তা পরীক্ষা করি। যদি 1 হয়, তাহলে তৃতীয় ধাপে যাই।

ধাপ ৩ : ওই সংখ্যাকে 2 থেকে m পর্যন্ত ক্রমিক সংখ্যাগুলো দিয়ে গুণ করি এবং গুণফল যত হবে, অ্যারের তত নম্বর উপাদানে শূন্য (0) বসাই। অর্থাৎ সেটি যে মৌলিক নয় তা চিহ্নিত করি। এখানে m-এর মান এমন হবে যেন ওই সংখ্যার সঙ্গে m-এর গুণফল n-এর চেয়ে ছেট বা সমান হয়।

এখন তোমরা কোডটি লেখার চেষ্টা করো। কমপক্ষে তিন ঘণ্টা নিজে চেষ্টা করার পর এবার আমার কোড দেখো।

```
#include <stdio.h>
#include <math.h>
const int size = 40;
int ara[size];
void print_ara()
{
    int i;
    for(i = 2; i < size; i++) {
        printf("%4d", ara[i]);
    }
    printf("\n");
    for(i = 2; i < size; i++) {
        printf("----");
    }
    printf("\n");
    for(i = 2; i < size; i++) {
        printf("%4d", i);
    }
}
```

১৪৬

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

void sieve()
{
    int i, j, root;
    for(i = 2; i < size; i++) {
        ara[i] = 1;
    }
    root = sqrt(size);
    print_ara();
    for(i = 2; i <= root; i++) {
        if(ara[i] == 1) {
            for(j = 2; i * j <= size; j++)
            {
                ara[i * j] = 0;
            }
            print_ara();
        }
    }
}
int is_prime(int n)
{
    int i;
    if(n < 2) {
        return 0;
    }
    return ara[n];
}
```

১৪৭

```

int main()
{
    int n, m;
    sieve();
    while(1) {
        printf("Please enter a number
        (enter 0 to exit): ");
        scanf("%d", &n);
        if(n == 0) {
            break;
        }
        if(n >= size) {
            printf("The number should be
            less than %d\n", size);
            continue;
        }
        if(1 == is_prime(n)) {
            printf("%d is a prime
            number.\n", n);
        }
        else {
            printf("%d is not a prime
            number.\n", n);
        }
    }
    return 0;
}

```

## প্রোগ্রাম : ১০.২

প্রতিবার অ্যারের অবশ্য বোকানোর জন্য আমি একটি ফাংশন ব্যবহার করেছি, print\_ara()। তোমরা দেখো এবাবে ইনপুট নেওয়ার আগেই আমরা sieve() ফাংশন কল করে আ্যারেটি তৈরি করে ফেলেছি। তারপর যতবারই ইনপুট নাও, কোনো চিন্তা নেই। ইনপুট যদি n হয় তাহলে ara[n]-এর মান পরীক্ষা করলেই চলে। মান যদি 1 হয় তাহলে n মৌলিক সংখ্যা। যদি 0 হয় তবে n মৌলিক সংখ্যা নয়। কত পর্যন্ত সংখ্যা হিসাব করতে চাও সেটি size-এ বসিয়ে দিলেই হবে। এখন এই প্রোগ্রামে গতি নিয়ে কোনো সমস্যা নেই। খুবই ফাস্ট (fast)। কিন্তু আর কোনো সমস্যা তোমাদের চোখে পড়ছে? তোমরা কি বুঝতে পারছ যে প্রোগ্রামটি অনেক বেশি মেমোরি খরচ করে? ধরো, আমরা যদি 100 কোটি পর্যন্ত সংখ্যা মৌলিক কি না সেটি বের করতে চাই, তাহলে তো আমাদের 100 কোটির একটি অ্যারে দরকার হবে। “সময় বাঁচাব, না মেমোরি” সমস্যায় প্রোগ্রামারদের প্রায়ই পড়তে হয়। আমাদের সমস্যার ক্ষেত্রে আমরা একটি মাঝামাঝি সমাধানে পৌঁছতে পারি। n-এর সর্বোচ্চ মান যত হবে তার বর্গমূলটিকে size-এর মান হিসেবে নিতে পারি। তোমাকে যদি বলা হয়, n-এর মান সর্বোচ্চ 100000000 (দশ কোটি) পর্যন্ত হতে পারে, তাহলে তুমি এর বর্গমূল অর্থাৎ 10000 পর্যন্ত সংখ্যাগুলোর জন্য sieve ফাংশন ব্যবহার করে মৌলিক সংখ্যাগুলো বের করবে। তারপর কী করবে? নাহু আর কিছু বলা যাবে না, তোমরাই চিন্তা করে ঠিক করো কী করবে।

আরেকটি কথা বলে দেওয়া দরকার। একটি ইন্টিজার কিন্তু চার বাইট জায়গা দখল করে, যেখানে একটি ক্যারেক্টার করে এক বাইট। সুতরাং ইন্টিজারের পরিবর্তে তোমরা ক্যারেক্টারের অ্যারে ব্যবহার করে মেমোরি খরচ চার ভাগের এক ভাগে নামিয়ে আনতে পারো। আমাদের তো আসলে ইন্টিজার অ্যারের দরকার নেই, কারণ অ্যারেতে কেবল দুই ধরনের মান থাকবে 0 বা 1। এটি ছাড়াও আমার লেখা sieve ফাংশনে আরও বেশ কিছু উপায় আছে ইফিশিয়েলি বাড়ানোর। এর মধ্যে একটি হচ্ছে গুণের বদলে যোগ করা। তোমরা সেটি করার চেষ্টা করো।

## অধ্যায় এগারো : আবারও অ্যারে

গণিত শিক্ষকের জন্য লেখা প্রোগ্রামটির কথা মনে আছে তো? সেই যে আমরা তিনটি অ্যারে ব্যবহার করে প্রোগ্রাম লিখেছিলাম ছাত্রছাত্রীদের গণিত পরীক্ষার মোট নম্বর বের করার জন্য। মনে না থাকলে প্রোগ্রামটি আবার দেখে নাও।

আমরা প্রথম সাময়িক পরীক্ষার নম্বর রেখেছিলাম একটি অ্যারেতে, দ্বিতীয় সাময়িক পরীক্ষার নম্বর আরেকটি অ্যারেতে, ফাইনাল পরীক্ষার নম্বর আরেকটি অ্যারেতে আর মোট নম্বর আরও একটি অ্যারেতে—মোট চারটি অ্যারে ব্যবহার করেছি। তো এখন যদি পুরো ক্লাসের সব ছাত্রছাত্রীর সব বিষয়ের সব পরীক্ষার জন্য একটি করে অ্যারে ব্যবহার করা খুবই ঝামেলার কাজ হয়ে যাবে। তাই মোটামুটি সব প্রোগ্রামিং ল্যাপ্টুপেজেই মাল্টি-ডাইমেনশনাল অ্যারের ব্যবস্থা আছে। এতক্ষণ আমরা যেসব অ্যারে ব্যবহার করেছি, তার সবগুলোই ছিল এক ডাইমেনশনের অ্যারে।

এখন ওই প্রোগ্রামটি আমরা একটু অন্যভাবে লিখব, একটিমাত্র অ্যারে ব্যবহার করে। আপাতত কষ্ট কমানোর জন্য ধরি ক্লাসে মোট 10জন ছাত্রছাত্রী আছে। নিচের ছকে তাদের নম্বরগুলো লিখলাম :

	Roll I: 1	Roll I: 2	Roll I: 3	Roll I: 4	Roll I: 5	Roll I: 6	Roll I: 7	Roll I: 8	Roll I: 9	Roll: 10
First terminal exam	80	70	92	78	58	83	85	66	99	81
Second terminal exam	75	67	55	10 0	91	84	79	61	90	97
Final exam	98	67	75	89	81	83	80	90	88	77
Total marks										

Total Marks সারির ঘরগুলো ফাঁকা, কারণ এগুলো এখনো হিসাব করিনি। প্রথম সাময়িক পরীক্ষার 25%, দ্বিতীয় সাময়িক পরীক্ষার 25% এবং ফাইনাল পরীক্ষার 50% নম্বর যোগ করে মোট নম্বর বের করতে হবে। এখন দেখো, আমাদের ছকে তাদের নম্বরগুলো রাখতে হয়েছে 4টি সারি (row) এবং 10টি কলাম। আমরা আগে যেই প্রোগ্রাম লিখেছিলাম, তাতে প্রথম রো-এর জন্য একটি অ্যারে, দ্বিতীয় রো-এর জন্য একটি, তৃতীয় রো-এর জন্য একটি এবং চতুর্থ রো-এর জন্য একটি অ্যারে ব্যবহার করেছিলাম। এখন ব্যবহার করব একটি 2-D অ্যারে (টু ডাইমেনশনাল অ্যারে)।

2-D অ্যারে ডিক্লেয়ার করার নিয়ম হচ্ছে : data type array name [number of rows][number of columns];

যেমন, ওপরের ছকটা যদি marks নামের একটি 2-D অ্যারেতে রাখতে চাই, তাহলে লিখতে হবে : int marks[4][10];

এখন, অ্যারের প্রথম রো হচ্ছে marks[0], দ্বিতীয় রো হচ্ছে marks[1], তৃতীয় রো হচ্ছে marks[2] এবং চতুর্থ রো হচ্ছে marks[3]। আবার marks[0][0] হচ্ছে প্রথম রো-এর প্রথম কলাম, marks[0][1] হচ্ছে প্রথম রো-এর দ্বিতীয় কলাম, marks[0][5] হচ্ছে প্রথম রো-এর ষষ্ঠি কলাম, marks[1][0] হচ্ছে দ্বিতীয় রো-এর প্রথম কলাম, marks[2][3] হচ্ছে তৃতীয় রো-এর চতুর্থ কলাম। আশা করি, বুঝতে পেরেছ।

এখন বলো তো, যার রোল নম্বর 10 তার দ্বিতীয় সাময়িক পরীক্ষার নম্বর কোন ঘরে আছে? আর marks[0][0] ঘরে কার এবং কোন পরীক্ষার নম্বর আছে?

marks[0][0]-এ থাকবে রোল 1-এর প্রথম সাময়িক পরীক্ষার নম্বর আর marks[1][9]-এ থাকবে রোল 10-এর দ্বিতীয় সাময়িক পরীক্ষার নম্বর। অ্যারেতে সংখ্যাগুলো এভাবে রাখতে পারি :

```
int marks[4][10] = {{80, 70, 92, 78, 58,
83, 85, 66, 99, 81}, {75, 67, 55, 100, 91,
84, 79, 61, 90, 97}, {98, 67, 75, 89, 81,
83, 80, 90, 88, 77}, {0, 0, 0, 0, 0, 0,
```

```
0, 0, 0));
```

এখানে ব্রেকল করেছ যে আমরা আসলে একটি অ্যারের ভেতর চারটি এক ডাইমেনশনের অ্যারে রেখেছি। marks[0]-তে আছে প্রথম সাময়িক পরীক্ষায সবার নম্বর, marks[1]-এ দ্বিতীয সাময়িক পরীক্ষার নম্বর, marks[2]-এ ফাইনাল পরীক্ষার নম্বর এবং marks[3]-তে মোট নম্বর (যেহেতু এখনো এটি হিসাব করিনি, তাই সব 0 লিখে দিলাম)।

এখন সম্পূর্ণ প্রোগ্রামটি তোমার কম্পিউটারে টাইপ করে কম্পাইল ও রান করো।

```
#include <stdio.h>
int main()
{
    int marks[4][10] = {{80, 70, 92, 78,
58, 83, 85, 66, 99, 81}, {75, 67, 55, 100,
91, 84, 79, 61, 90, 97}, {98, 67, 75, 89,
81, 83, 80, 90, 88, 77}, {0, 0, 0, 0, 0, 0,
0, 0, 0, 0}};
    int col;
    for(col = 0; col < 10; col++) {
        marks[3][col] = marks[0][col] / 4.0
        + marks[1][col] / 4.0 + marks[2][col] /
        2.0;

        printf("Roll NO: %d Total Marks:
%d\n", col + 1, marks[3][col]);
    }
    return 0;
}
```

প্রোগ্রাম : ১১.১

নম্বরগুলো আগে না লিখে যদি আমরা ব্যবহারকারীর কাছ থেকে ইনপুট হিসেবে নিতে চাইতাম, তাহলে কী করতে হতো?

```
int marks[4][10];
int i, j;

for (i = 0; i < 4; i++) {
    for (j = 0; j < 10; j++) {
        scanf("%d", &ara[i][j]);
    }
}
```

এভাবে নেক্স্টেড লুপের সাহায্যে আমরা ইনপুট নিতে পারি। প্রথম লুপটি ব্যবহার করা হয়েছে রো-এর জন্য আর দ্বিতীয লুপটি কলামের জন্য। যখন  $i = 0$ , অর্থাৎ, প্রথম রো-এর জন্য আমরা  $j = 0$  থেকে 9 পর্যন্ত সব ঘরের ইনপুট নিচ্ছি, তারপর আবার  $i = 1$  (দ্বিতীয রো)-এর জন্য  $j = 0$  থেকে 9 (প্রতি কলাম) পর্যন্ত সব ঘরের মান ইনপুট নেওয়া হচ্ছে।

এখন আমরা 1 থেকে 10 পর্যন্ত সংখ্যাগুলোর নামতা বের করার জন্য 2-D অ্যারে ব্যবহার করে একটি প্রোগ্রাম লিখব। একের নামতা হবে প্রথম রো-তে, দুইয়ের নামতা দ্বিতীয রো-তে...দশের নামতা দশম রো-তে থাকবে। তোমরা কি প্রোগ্রামটি নিজে নিজে লেখার চেষ্টা করবে? এক ঘণ্টার মধ্যেও যদি না হয়, তাহলে আমার কোডটি দেখো। প্রোগ্রামিং শেখার সময় অনেক সহজ প্রোগ্রাম লিখতেও প্রচুর সময় লাগে, তাতে হতাশ হওয়ার কিছু নেই।

```
#include <stdio.h>
int main()
{
    int namta[10][10];
    int row, col;
    for (row = 0; row < 10; row++) {
        for (col = 0; col < 10; col++) {
            // কোড এখানে লিখো
        }
    }
}
```

```

        namta[row][col] = (row + 1) *
    (col + 1);
}

for (row = 0; row < 10; row++) {
    for (col = 0; col < 10; col++) {
        printf("%d x %d = %d\n", (row +
1), (col + 1), namta[row][col]);
    }
    printf("\n");
}
return 0;
}

```

প্রোগ্রাম : ১১.২

সম্পূর্ণ আউটপুট স্ক্রিনে না-ও আটতে পারে, তাতে চিন্তার কিছু নেই।  
এখন তোমাদের জন্য একটি সমস্যা। ওপরের প্রোগ্রামটি পরিবর্তন করো, যাতে আউটপুট হিসেবে আমরা দেখতে পারি যে namta অ্যারেতে মোট কয়টি জোড় ও কয়টি বেজোড় সংখ্যা আছে। সেই সঙ্গে অ্যারেতে যতগুলো সংখ্যা আছে, তার যোগফলও বের করতে হবে। আশা করি, প্রোগ্রামটি ঠিকঠাক লিখতে পেরেছ। যদি কোনো সমস্যা হয়, তাহলে তুমি তোমার বন্ধুর সাহায্য নিতে পারো।

আচ্ছা, কেউ যদি বলে, সার্কভুক্ত সাতটি দেশের নাম একটি অ্যারেতে রাখতে, তাহলে কী করবে? একটি char type অ্যারেতে একটি দেশের নাম রাখা যায়। যেমন : char country[] = "Bangladesh";। তাহলে সাতটি দেশের নাম রাখার জন্য আমাদের একটি 2-D অ্যারে ব্যবহার করতে হবে। এই অ্যারেতে মোট কয়টি রো থাকবে? সাতটি। কলাম কয়টি থাকবে? জানি না। আসলে একেক দেশের নামের দৈর্ঘ্য তো একেক রকম। তাই আমরা একটি কাজ করতে পারি, কলামে 100টি ক্যারেক্টার রাখার ব্যবস্থা করতে পারি,

কারণ সাকের কোনো দেশের নামে তো 100টির বেশি ক্যারেক্টার  
নেই, কম আছে।

```

#include <stdio.h>
int main()
{
    char saarc[7][100] = {"Bangladesh",
"India", "Pakistan", "Sri Lanka", "Nepal",
"Bhutan", "Maldives"};
    int row;
    for (row = 0; row < 7; row++) {
        printf("%s\n", saarc[row]);
    }
    return 0;
}

```

প্রোগ্রাম : ১১.৩

প্রোগ্রামটি তোমার কম্পিউটারে লিখে রান করাও। এখন তোমরা বলো তো, saarc[3][3], saarc[0][5] ও saarc[5][0] এই তিনটি ঘরে কী কী ক্যারেক্টার আছে? একটু পরে একটি প্রোগ্রাম লিখব, তার সঙ্গে তোমার উত্তর মিলিয়ে নেবে।

আমরা যদি ওই অ্যারের প্রতিটি ক্যারেক্টার আলাদাভাবে প্রিন্ট করতে চাই, তাহলে প্রোগ্রামটি এভাবে লিখতে পারি :

```

#include <stdio.h>
#include <string.h>
int main()
{
    char saarc[7][100] = {"Bangladesh",
"India", "Pakistan", "Sri Lanka", "Nepal",
"Bhutan", "Maldives"};

```

```

int row, col, name_length;
for (row = 0; row < 7; row++) {
    name_length = strlen(saarc[row]);
    for (col = 0; col < name_length;
    col++) {
        printf("%c ", saarc[row][col]);
    }
    printf("\n");
}
return 0;
}

প্রোগ্রাম : ১১.৪

```

আবার যদি দেখতে চাই, কোন ঘরে কী আছে, তাহলে রো এবং  
কলাম নম্বরসহ প্রিন্ট করতে পারি।

```

#include <stdio.h>
#include <string.h>
int main()
{
    char saarc[7][100] = {"Bangladesh",
    "India", "Pakistan", "Sri Lanka", "Nepal",
    "Bhutan", "Maldives"};
    int row, col, name_length;
    for (row = 0; row < 7; row++) {
        name_length = strlen(saarc[row]);
        for (col = 0; col < name_length;
        col++) {
            printf("(%d, %d) = %c, ", row,
            col, saarc[row][col]);
        }
    }
}

```

```

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

```

প্রোগ্রাম : ১১.৫

এবারে নিচের ছকটি দেখো।

6	4	7	8	9
3	7	1	9	9
8	6	4	2	7
2	4	2	5	9
4	1	6	7	3

এখন 2-D অ্যারে ব্যবহার করে তোমাদের দুটি প্রোগ্রাম লিখতে হবে।  
প্রথম প্রোগ্রামটির কাজ হবে প্রতিটি রো-এর সংখ্যাগুলোর যোগফল  
বের করা আর দ্বিতীয় প্রোগ্রামের কাজ প্রতিটি কলামের সংখ্যাগুলোর  
যোগফল বের করা।

প্রথম প্রোগ্রামের আউটপুট হবে এই রকম :

Sum of row 1: 34

Sum of row 2: 29

Sum of row 3: 27

Sum of row 4: 22

Sum of row 5: 21

দ্বিতীয় প্রোগ্রামের আউটপুট হবে এই রকম :

Sum of column 1: 23

Sum of column 2: 22

Sum of column 3: 20

Sum of column 4: 31

Sum of column 5: 37

তোমাদের অনেকেরই দ্বিতীয় প্রোগ্রামটি লিখতে একটু সময় লাগতে পারে, কিন্তু হতাশার কোনো কারণ নেই। সময় লাগাই স্বাভাবিক। এবারে নিচের ছকটি দেখো। আগের ছকটির সঙ্গে এর কোথায় মেন একটু মিল রয়েছে।

6	3	8	2	4
4	7	6	4	1
7	1	4	2	6
8	9	2	5	7
9	9	7	9	3

আসলে আগের ছকের রো-গুলো নতুন ছকের কলাম, আর আগের ছকের কলামগুলো নতুন ছকের রো। যেমন, আগের ছকের প্রথম রো-টি ছিল : 6, 4, 7, 8, 9। আর এই ছকের প্রথম কলাম হচ্ছে : 6, 4, 7, 8, 9। একইভাবে আগের ছকের দ্বিতীয় রো নতুন ছকের দ্বিতীয় কলামের সঙ্গে মেলে। এখন আমরা একটি প্রোগ্রাম লিখব, যেটি একটি  $5 \times 5$  অ্যারেকে (অর্থাৎ 5 রো এবং 5 কলামবিশিষ্ট অ্যারে), আরেকটি  $5 \times 5$  অ্যারেতে এমনভাবে কপি করবে, যাতে প্রথম অ্যারের রো-গুলো হয় দ্বিতীয় অ্যারের কলাম আর প্রথম অ্যারের কলামগুলো হয় দ্বিতীয় অ্যারের রো। মানে ওপরের ছক দুটির মতো তাহলে দ্বিতীয় অ্যারের প্রথম রো যদি হয় : 1, 2, 3, 4, 5, বিষয়টি একটু জটিল মনে হচ্ছে? তাহলে কিছুক্ষণ বিশ্রাম নিয়ে তারপর নিচের প্রোগ্রামটি কম্পাইল করো, রান করো, আউটপুট দেখো এবং কীভাবে কাজ করছে বোঝার চেষ্টা করো।

```
#include <stdio.h>
#include <string.h>
int main()
{
    int ara1[5][5] = {{1, 2, 3, 4, 5}, {10,
20, 30, 40, 50}, {100, 200, 300, 400, 500},
{1000, 2000, 3000, 4000, 5000}, {10000,
20000, 30000, 40000, 50000}};
    int ara2[5][5];
    int r, c;
    printf("Content of first array (ara1):
\n");
    for (r = 0; r < 5; r++) {
        for(c = 0; c < 5; c++) {
            printf("%d ", ara1[r][c]);
        }
        printf("\n");
    }
    printf("\n");
    // now start copy
    for (r = 0; r < 5; r++) {
        for(c = 0; c < 5; c++) {
            ara2[c][r] = ara1[r][c];
        }
    }
    printf("Content of second array (ara2):
\n");
    for (r = 0; r < 5; r++) {
        for(c = 0; c < 5; c++) {

```

```

        printf("%d ", arr2[r][c]);
    }
    printf("\n");
}
return 0;
}

```

প্রোগ্রাম : ১১.৬

তোমরা যদি একক্ষণ 2-D অ্যারের সব উদাহরণ এবং যেসব প্রোগ্রাম নিজে লিখতে বলেছি, সেগুলো সব কম্পিউটারে রান করে থাকো এবং বুঝে থাকো (বুঝতে হলে অবশ্যই চিন্তা করতে হবে), তাহলে তোমাদের আর 2-D অ্যারে নিয়ে সমস্যা হওয়ার কথা নয়। অ্যারে কিন্তু 3-D, 4-D কিংবা আরও বেশি ডাইমেনশনের হতে পারে, তবে সেগুলো নিয়ে এই বইতে আর আলোচনা করব না।

## অধ্যায় বারো : বাইনারি সংখ্যা

আমরা তো দৈনন্দিন জীবনে নানা হিসাব-নিকাশের জন্য দশভিত্তিক (decimal) সংখ্যা পদ্ধতি ব্যবহার করি। কিন্তু কম্পিউটার ব্যবহার করে দুইভিত্তিক বা বাইনারি (binary) সংখ্যা পদ্ধতি। দশভিত্তিক সংখ্যা পদ্ধতিতে আছে মোট দশটি অঙ্ক ০, ১, ২, ৩, ৪, ৫, ৬, ৭, ৮, ৯ আর বাইনারিতে দুটি, ০ আর ১। আমরা এই অধ্যায়ে বাইনারি সংখ্যা পদ্ধতির কিছু মৌলিক জিনিস দেখব আর বাইনারি থেকে ডেসিম্বল এবং ডেসিম্বল থেকে বাইনারি সংখ্যায় রূপান্তর করা শিখব। ডেসিমালে আমরা গণনা করি এভাবে : ০, ১, ২, ৩, ৪, ৫, ৬, ৭, ৮, ৯, ১০, ১১, ১২,...১৯, ২০, ২১,..., ৯৮, ৯৯, ১০০, ১০১। দেখো, যখনই আমরা ডান দিকের ঘরে (এককের ঘরে) দশটি অঙ্ক ব্যবহার করে ফেলি, তখন তার বাঁয়ে দশকের ঘরের অঙ্কের মান এক বাড়াই (আর যদি না থাকে, তাহলে ১ বসাই বা ০-এর সঙ্গে ১ যোগ করি আরকি, কারণ ৯ আর ০৯ কিন্তু একই কথা, তাই ০৯-এর পরবর্তি সংখ্যা হচ্ছে ১০), আবার দশকের ঘরে ০ থেকে ৯ সব অঙ্ক ব্যবহার করে ফেলার পরে শতকের ঘরের অঙ্কের মান এক বাড়াই (আর যদি না থাকে তাহলে ১ বসাই বা ০-এর সঙ্গে ১ যোগ করি আর কি)। তেমনই বাইনারিতে আমরা গণনা করব এইভাবে : ০, ১, ১০, ১১, ১০০, ১০১, ১১০, ১১১, ১০০০, ১০০১, ১০১০, ১০১১। যেহেতু অঙ্ক মাত্র দুটি, তাই দুটি অঙ্কের ব্যবহার হয়ে গেলেই বাঁ দিকের ঘরে এক বসাতে হয় বা ০-এর সঙ্গে ১ যোগ করতে হয় (বাঁ দিকে তো আমরা ইচ্ছামতো শূন্য বসাতে পারি)।

বাইনারি সিস্টেমে অবশ্য আমরা এককের ঘর, দশকের ঘর, শতকের ঘর, সহস্রের ঘর না বলে বলব এককের ঘর, দুইয়ের ঘর, চারের ঘর, আটের ঘর। কেন বল তো? একটু চিন্তা করো।

ডেসিমলে যেমন 10 লিখতে দুটি অঙ্ক লাগে, 100 লিখতে তিনটি, 1000 লিখতে চারটি, তেমনই বাইনারিতে দুই লিখতে দুটি (10), চার লিখতে তিনটি (100), আট লিখতে চারটি (1000), মোলো লিখতে পাঁচটি (10000) অঙ্ক ব্যবহার করতে হয়। ডেসিমলে ডান দিকের প্রথম অঙ্ক ( $10^0 = 1$ ) হচ্ছে এককের ঘর, দ্বিতীয় অঙ্ক ( $10^1 =$

10) হচ্ছে দশকের ঘর, তৃতীয় অঙ্ক ( $10^2 = 100$ ) হচ্ছে শতকের ঘর, তেমনই বাইনারিতে ডান দিকের প্রথম অঙ্ক ( $2^0 = 1$ ) হচ্ছে একের ঘর, পরের অঙ্ক ( $2^1 = 2$ ) হচ্ছে দুইয়ের ঘর, তারপর ( $2^2 = 4$ ) হচ্ছে চারের ঘর, এই রকম।

দশভিত্তিক সংখ্যায় যেমন যোগ, বিয়োগ, গুণ, ভাগ করা যায়, তেমনই বাইনারিতে করা যায়। আসলে যোগ করতে পারলে কিন্তু বাকি কাজ করা কোনো ব্যাপার নয়। আবার বাইনারিতে ভগ্নাংশের ব্যাপার আছে, তবে আমি কেবল পূর্ণসংখ্যা নিয়েই আলোচনা করব।

যোগের ক্ষেত্রে মূল হিসাবগুলো হচ্ছে :

$$0 + 0 = 0,$$

$$0 + 1 = 1,$$

$$1 + 0 = 1,$$

$$1 + 1 = 101$$

ডেসিমলের মতোই হিসাব,  $1 + 1$  এর ক্ষেত্রে দেখো, দুইয়ের (10) শূন্য এল প্রথমে, হাতে থাকে এক, সেটি পরে লিখলাম।

$$101 + 101 = \text{কত?}$$

প্রথমে একের ঘরের যোগ,  $1 + 1 = 10$ । তাই যোগফলের একের ঘরে বসবে 0 আর হাতে থাকল 1 (carry)। এবাবে দুইয়ের ঘরে,  $0 + 0 = 0$ , এখন এই 0-এর সঙ্গে হাতের 1 যোগ করতে হবে। তাহলে যোগফলের দুইয়ের ঘরে 1। এবাবে চারের ঘরের যোগ করলে পাই,  $1 + 1 = 10$ । হাতে কিছু নেই (কোনো carry নেই)। সুতরাং চারের ঘরে বসবে 0 আর 1 বসবে আটের ঘরে। যোগফল : 1010। এবাবে বলো  $1011 + 1011 = \text{কত?}$  যোগ করে যদি দেখো যোগফল 10110 হয়নি, তাহলে তুমি যোগে কোথাও ভুল করেছ।

বিয়োগের ক্ষেত্রেও ডেসিমলের মতো হিসাব হবে।

$$0 - 0 = 0,$$

$$1 - 0 = 1,$$

$$1 - 1 = 0,$$

$$0 - 1 = 11$$

শেষেরটা খেয়াল করো,  $23 - 15$  করার সময় আমরা কী করি? তখন 3-এর বাঁয়ে একটি কাল্পনিক 1 ধরে নিই (বা 1 ধার করি), তারপর  $13 - 5 = 8$  লেখি। আর যেই একটি ধার করলাম, সেটি পরের ঘরে 1-এর সঙ্গে যোগ করে দিই। তেমনই বাইনারিতে  $0 - 1$  করতে গেলে 0-এর বাঁয়ে একটি এক ধরব, তখন সংখ্যাটি হবে 10 (দুই), এই দুই থেকে এক বাদ দিলে এক থাকবে। পরের ঘরে একটি এক যোগ করতে হবে (যে সংখ্যাটি বিয়োগ হচ্ছে তার সঙ্গে)।

$$110 - 101 = \text{কত?}$$

একের ঘরে 0 থেকে 1 বাদ দিলে থাকে 1, এখানে 1 ধার করতে হয়েছে। তাই 101-এর দুইয়ের ঘরে সেটি যোগ করে দেব। তাহলে দুইয়ের ঘরে  $1 - 1 = 0$ , চারের ঘরে  $1 - 1 = 0$ । তাই বিয়োগফল হবে : 001 বা 1। যোগ-বিয়োগ পারলে গুণ-ভাগ না পারার কারণ নেই। ডেসিমলের মতোই নিয়ম।

আবার কোনো ডেসিম্বল সংখ্যাকে আমরা নির্দিষ্ট অঙ্ক  $\times 10^{3\text{ই অঙ্গের অবস্থান}} - 1$ -এর যোগফল হিসেবে যেমন লিখতে পারি, বাইনারি সংখ্যাকেও নির্দিষ্ট অঙ্ক  $\times 2^{3\text{ই অঙ্গের অবস্থান}} - 1$ -এর যোগফল হিসেবে লেখা যায়। যেমন :  $1903 = 1 \times 10^3 + 9 \times 10^2 + 0 \times 10^1 + 3 \times 10^0$ ।

বাইনারি :  $10110 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$ । ইংরেজিতে একে বলে Exponential Expression।

এখন কোনো বাইনারি সংখ্যার মান যদি ডেসিমলে বের করতে চাই, তাহলে প্রথমে বাইনারি সংখ্যাটিকে এক্সপোনেনশিয়াল এক্সপ্রেশন আকারে লিখতে হবে। তারপর গুণফলগুলো ডেসিমলে হিসাব করতে হবে। নিচের উদাহরণটি দেখো :

$$\begin{aligned} 10110 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 1 \times \\ &16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 \\ &= 16 + 0 + 4 + 2 + 0 = 22 \end{aligned}$$

অর্থাৎ, বাইনারি 10110 = ডেসিমল 22। আমরা অনেকক্ষণ কোনো প্রোগ্রামিং করছি না, চলো বাইনারি সংখ্যার ডেসিমল মান বের করার একটি প্রোগ্রাম লিখে ফেলি। পদ্ধতি তো জানা হয়ে গেছে। এখন গুরুত্বপূর্ণ প্রশ্ন হচ্ছে, বাইনারি সংখ্যা রিড করব কী দিয়ে? আমরা স্ট্রিং ব্যবহার করতে পারি।

```
char binary[] = "10110";
```

```
int len = 5; // স্ট্রিংয়ের দৈর্ঘ্য 5।
```

```
int decimal = 0; // এখনো কোনো হিসাব করিনি, তাই ধরলাম  
ডেসিমল মান 0।
```

এবার আমরা একটি লুপের সাহায্যে বাইনারি সংখ্যার প্রতিটি অক্ষের সঙ্গে  $2^{\text{position}}$  অঙ্গের অবস্থান গুণ করে সেটি ডেসিমলের সঙ্গে যোগ করে দেব। প্রথম ক্যারেটার, অর্থাৎ `binary[0]`-তে তো '1' আছে, এর অবস্থান কত বলো তো? এর অবস্থান হচ্ছে 4। তারপরের অক্ষের বেলায় অবস্থানের মান এক কমবে, এভাবে একেবারে শেষের অক্ষের বেলায় অবস্থান হবে 0।

```
int position = 4;
```

```
int indx;
```

```
for(indx = 0; indx < len; indx++) {
```

```
    decimal = decimal + pow(2, position);
```

```
    position--;
```

```
}
```

লুপ থেকে বের হলে আমরা সম্পূর্ণ বাইনারি সংখ্যার ডেসিমল মান পেয়ে যাব। এখনে দেখো, আমি `pow` ফাংশন ব্যবহার করেছি। এটির কাজ বলা আছে `math.h` হেডার ফাইলে।  $a^b$ -এর মান বের করার জন্য `pow(a, b)` বলে দিলেই হয়। তাহলে আমাদের পুরো প্রোগ্রামটি দাঁড়াচ্ছে এই রকম :

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <math.h>
int main()
{
    char binary[65];
    int len, decimal, power, i;
    printf("Enter the binary number: ");
    scanf("%s", binary);
    decimal = 0;
    len = strlen(binary);
    power = len - 1;
    for(i = 0; i < len; i++) {
        decimal += pow(2, power);
        power--;
    }
    printf("Decimal value is %d\n",
    decimal);

    return 0;
}
```

প্রোগ্রাম : ১২.১

প্রোগ্রাম কম্পাইল করে রান করো। ইনপুট যদি 10110 দাও, তাহলে আউটপুট কত আসে? আউটপুট আসে 31, কিন্তু আউটপুট তো আসা উচিত 22। তাহলে আমরা কোথাও ভুল করেছি। তোমরা নিজে নিজে ভুলটি বের করার চেষ্টা করো।

আমাদের তো আসলে `pow(2, position)`-কে বাইনারি সংখ্যার ওই `position`-এর অক্ষটি দিয়ে গুণ করার কথা, সেটি আমরা করতে ভুলে গেছি। অর্থাৎ আমাদের লিখতে হবে :

```
decimal += binary[i] * pow(2, power);
```

একটি ব্যাপার খেয়াল করছ তো? 10110-এর একের ঘরের অঙ্কটি আমাদের অ্যারের শেষ ক্যারেক্টার, আর ঘোলোর ঘরের অঙ্কটি হচ্ছে অ্যারের প্রথম ক্যারেক্টার। অ্যারেতে সংখ্যাটি আছে এইভাবে : [1, '0', '1', '1', '0']। তাই `binary[0]`-এর সঙ্গে গুণ হবে `pow(2, 4)`, `binary[1]`-এর সঙ্গে গুণ হবে `pow(2, 3)`, ..., এভাবে `binary[4]`-এর সঙ্গে গুণ হবে `pow(2, 0)`। এখন প্রোগ্রামটি ঠিক করে নিয়ে তারপর চালাও। ইনপুট 10110-এর জন্য কী আউটপুট?

আমি তো আউটপুট দেখতে পাচ্ছি Decimal value is 1510। ভুলটি কোথায় হলো? সব তো ঠিকই করলাম। তোমরা আবার বিরক্ত হয়ে যাচ্ছ না তো? টেস্ট ক্রিকেট খেলার সময় যেমন ধৈর্যের প্রয়োজন, প্রোগ্রামিংও তেমনই ধৈর্যের খেলা।

ভুলটি যে decimal += binary[i] \* pow(2, power); স্টেটমেন্টে  
হয়েছে তাতে কোনো সন্দেহ নেই। কারণ আমরা এখানেই একটু  
পরিবর্তন করেছি। লক্ষ করো, binary[i]-এর মান হয় '0' বা '1'  
(মানে ক্যারেক্টার '0' বা ক্যারেক্টার '1')। এখন কম্পিউটার '0' বলতে  
বোঝে 48 আর '1' বলতে বোঝে 49। ঝামেলাটা এখানেই হয়েছে।  
এখন এই '0'-কে 0 আর '1'-কে 1 বোঝাব কীভাবে?

$$'0' - '0' = 48 - 48 = 0$$

$$'1' - '0' = 49 - 48 = 1$$

বুদ্ধিটা দারুণ না? আমরা `binary[i]` না লিখে (`binary[i] - '0'`)  
লিখলেই জামেলা শেষ। এবারে প্রোগ্রাম ঠিকঠাক কাজ করবে (যদি না  
তৃতীয় নতুন কোনো ভুল করে থাকো)।

এবার আমরা দেখব ডেসিমল থেকে বাইনারিতে রূপান্তর। একটি উদাহরণের সাহায্যে পদ্ধতিটা দেখাই। ধরো 95-কে বাইনারিতে রূপান্তর করতে হবে। এখন আমাদের বের করতে হবে  $n$ -এর সর্বোচ্চ মান, যেখানে  $2^n <= 95$ । দুইয়ের পাওয়ারগুলো হচ্ছে 1, 2, 4, 8, 16, 32, 64, 128, ...। এখানে আমরা দেখতে পাচ্ছি,  $64 < 95$  বা  $2^6 < 95$ । তাহলে  $n$ -এর মান 6। আব আমাদের বাইনারি সংখ্যাটি

হবে সাত অক্ষের (0 থেকে 6 মোট সাতটি অঙ্ক)। যেহেতু  $64 < 95$ ,  
 তাই এই সংখ্যাটি নেওয়া যায়। তাহলে চৌষট্টির ঘরে (বাঁ থেকে প্রথম  
 বা ডান থেকে সপ্তম) হবে 1 ( $1xxxxxx$ )। এখন  $n$ -এর মান 1  
 কমাই।  $64 + 2^5 = 64 + 32 = 96$ , যা কিনা 95-এর চেয়ে বড়।  
 তাই একে নেওয়া যাবে না। অতএব বত্তিশের ঘরে 0 বসাই  
 তাই একে নেওয়া যাবে না। আবার এক কমাই,  $n$ -এর মান  
 ( $10xxxxx$ )। এবারে  $n$ -এর মান আবার এক কমাই,  $n$ -এর মান  
 এখন 4।  $64 + 2^4 = 64 + 16 = 80 < 95$ । সুতরাং ঘোলোর ঘরে  
 হবে 1 ( $101xxxx$ )। এখন  $n$ -এর মান এক কমাই,  $n = 3$ ।  $80 +$   
 $2^3 = 80 + 8 = 88 < 95$ । তাই আটের ঘরেও 1 বসবে  
 ( $1011xxx$ )। এরপর একইভাবে,  $n = 2$ -এর জন্য  $88 + 2^2 = 88$   
 $+ 4 = 92 < 95$ । চারের ঘরেও 1 বসবে ( $10111xx$ )। তারপর  $n =$   
 $1$ ,  $92 + 2^1 = 92 + 2 = 94 < 95$ । দুইয়ের ঘরেও 1  
 ( $101111x$ )। এখন  $n = 0$ ,  $94 + 2^0 = 94 + 1 = 95$ । তাই  
 একের ঘরেও 1। সুতরাং, বাইনারি সংখ্যাটি হচ্ছে  $10111111$ ।  
 তোমরা এখন এই পদ্ধতিতে কোনো দশভিত্তিক সংখ্যাকে বাইনারিতে  
 রূপান্তর করার প্রোগ্রাম লিখে ফেলো এবং বিভিন্ন মান দিয়ে পরীক্ষা  
 করে দেখো।

এখন একই কাজ আমরা একটু অন্যভাবে করব। নিচের টেবিলটি  
দেখো:

	ভাগফল	ভাগশেষ
95/ 2	47	1
47/2	23	1
23/2	11	1
11/2	5	1
5/2	2	1
2/2	1	0
1 /2	0	1

এবার ভাগশেষ কলামের অঙ্গগুলো শেষ থেকে প্রথম ক্রমে জেখলেই  
আমরা বাইনারি নম্বরটা পেয়ে যাব : 1011111। আর ভাগের কাজটি  
আমরা ততক্ষণ করব যতক্ষণ না ভাগফল 0 পাচ্ছি।

এই পদ্ধতিতেও তোমরা ডেসিমল থেকে বাইনারি রূপান্তরের জন্য  
একটি কোড লিখে ফেলো। রূপান্তরের কোডটি main ফাংশনে না  
করে আলাদা একটি ফাংশনে করবে।

### অধ্যায় ত্রো : কিছু প্রোগ্রামিং-সমস্যা

এই অধ্যায়ে আমরা কয়েকটি সহজ সমস্যা দেখব ও সমাধানের চেষ্টা  
করব।

আমাদের প্রথম সমস্যা হচ্ছে, বিভিন্ন ধরনের আকৃতি তৈরি করা।  
নিচের ছবিগুলো দেখো।

cccccccc  
cccccc  
ccccc  
cccc  
ccc  
cc  
c  
cc  
ccc  
cccc  
ccccc  
cccccc  
ccccccc

cccccccccccc  
cccccccccc  
cccccccc  
cccccc  
cccc  
ccc  
c  
ccc  
cccc  
ccccc  
cccccc  
cccccccc

ছবি ১৩.১

ছবি ১৩.২

cccccccc  
ccccccc  
ccccc  
cccc  
ccc  
cc  
c  
cc  
ccc  
cccc  
ccccc  
cccccc  
ccccccc  
cccccccc

c c  
cc cc  
ccc ccc  
cccc cccc  
ccccc cccccc  
cccccc ccccccc  
ccccc cccccc  
cccccc ccccccc  
ccccccc cccccccc  
cccccccc cccccccc  
cccccccccc cccccccc  
cccccccccc cccccccc  
cccccccccccccccc

ছবি ১৩.৩

ছবি ১৩.৪

তোমাদের চারটি প্রোগ্রাম লিখতে হবে এই চার ধরনের আকৃতি তৈরি করার জন্য। কেবল printf ফাংশন ব্যবহার করলেই হবে না, লুপ (স্পেস ক্যারেক্টার) প্রিন্ট করে তোমরা প্রোগ্রামগুলো লিখে ফেলতে পারো। আরও খেলাধুলা করার ইচ্ছা হলে আরও নানান রকম আকৃতি তৈরির চেষ্টা করতে পারো।

প্যালিনড্রোম (palindrome) কী জিনিস, তোমরা জানো? কোনো শব্দকে উল্টাভাবে (মানে শেষ থেকে শুরু) লিখলে যদি সেটি আর নতুন শব্দটি একই রকম হয় তবে সেটি একটি প্যালিনড্রোম। যেমন: madam। এটিকে শেষ থেকে শুরু পর্যন্ত লিখলেও madam হবে। এখন একটি প্রোগ্রাম লিখব, যেটিতে কোনো শব্দ ইনপুট দিলে সেটি প্যালিনড্রোম কি না বলে দেবে।

এজন্য আমরা কী করতে পারি? প্রথমে শব্দটি স্ট্রিং হিসেবে একটি অ্যারেতে ইনপুট নেব। তারপর আরেকটি অ্যারেতে সেটি উল্টাভাবে রাখব। তারপর যদি দুটি একই স্ট্রিং হয়, তবে সেটি প্যালিনড্রোম। তাহলে প্রোগ্রামটি লিখে ফেলি :

```
#include <stdio.h>
#include <string.h>
int main()
{
    char word[80], reverse_word[80];
    int i, j, len;
    scanf("%s", word);
    len = strlen(word);
    for(i = 0, j = len - 1; i < len; i++, j--) {
        reverse_word[i] = word[j];
    }
    reverse_word[i] = '\0';
}
```

১৭০

```
printf("%s\n", reverse_word);
if (0 == strcmp(word, reverse_word)) {
    printf("%s is a palindrome.\n",
word);
}
else {
    printf("%s is not a palindrome.\n",
word);
}
return 0;
}
```

প্রোগ্রাম : ১৩.১

কী মজা! আমি প্রোগ্রামটি লিখে দিলাম। তবে আমি এখানে বেশ কিছু বোকামি করেছি, যার মধ্যে অন্যতম হচ্ছে একটি অতিরিক্ত অ্যারে ব্যবহার করা। সুতরাং তোমাদের এখন প্রোগ্রামটি এমনভাবে লিখতে হবে, যাতে কেবল একটি অ্যারে ব্যবহার করেই কাজ হয়। আর তখন strcmp ফাংশনটিও দরকার হবে না। প্রোগ্রামটি লিখতে সময় বেশি লাগতে পারে, লাগুক, অসুবিধা নেই। তবে লিখতে হবে ঠিকঠাক, এটিই হলো কথা।

তোমরা তো ফ্যাক্টরিয়াল (factorial) জিনিসটির সঙ্গে পরিচিত? এটি একটি গাণিতিক অপারেশন, যা কোনো ধনাত্মক পূর্ণসংখ্যার ক্ষেত্রে ব্যবহার করা যায়।  $n$  একটি ধনাত্মক পূর্ণ সংখ্যা হলে এর ফ্যাক্টরিয়ালকে প্রকাশ করা হয়  $n! = n * (n - 1) * (n - 2) * \dots * 3 * 2 * 1$ । যেমন,  $4! = 4 * 3 * 2 * 1 = 24$ । আবার  $6! = 6 * 5 * 4 * 3 * 2 * 1 = 720$ ।  $1! = 1$ ।  $0! = 1$  (০-এর ক্ষেত্রে ব্যতিক্রমিত লক্ষ করো, কিছু বিশেষ সূবিধার জন্য ০-এর ফ্যাক্টরিয়ালের মান 1 ধরা হয়)। এখন তোমরা কোনো ধনাত্মক পূর্ণসংখ্যার ফ্যাক্টরিয়াল বের করার প্রোগ্রামটি লিখে ফেলো। সহজ প্রোগ্রাম, একটি লুপ ব্যবহার করেই করা যায়। এখন বিভিন্ন সংখ্যা দিয়ে প্রোগ্রামটি টেস্ট করে দেখো, ফ্যাক্টরিয়াল ঠিকঠাক বের করতে

১৭১

পারে কি না। প্রোগ্রামে তুমি যদি ডাটা টাইপ int ব্যবহার করে থাকो তাহলে 12-এর চেয়ে বড় কোনো পূর্ণ সংখ্যার ফ্যাক্টরিয়ালের মান ঠিকমতো দেখাবে না (ক্যালকুলেটরে করে মিলিয়ে দেখতে পারো)। কারণ হচ্ছে 12-এর চেয়ে বড় কোনো পূর্ণ সংখ্যার জন্য সেই সংখ্যার ফ্যাক্টরিয়ালের মান রেঞ্জের বাইরে চলে যায়।

এখন তোমাদের একটি মজার সমস্যা সমাধান করতে হবে। কোনো পূর্ণসংখ্যা n (যেখানে  $1 < n < 100$ , মানে n-এর মান 2 থেকে 99 পর্যন্ত হতে পারে)-এর ফ্যাক্টরিয়ালকে মৌলিক সংখ্যার গুণফল হিসেবে প্রকাশ করলে কোন মৌলিক সংখ্যা কতবার আছে সেটি বের করতে হবে। যেমন, আমরা জানি,  $5! = 120 = 2 * 2 * 2 * 3 * 5$ । এখনে 2 আছে 3 বার, 3 আছে 1 বার আর 5 আছে 1 বার। তাই ইনপুট 5 হলে আউটপুট হবে :  $5! = (2, 3), (3, 1), (5, 1)$ । তোমরা কি একটি ব্যাপার বুঝতে পারছ যে শুরুতে n-এর ফ্যাক্টরিয়ালের মান বের করে তারপর মৌলিক উৎপাদকে ভাগ্নতে গেলে একটি ঝামেলা হয়ে যাবে? কারণ n-এর মান সর্বোচ্চ হতে পারে 99 আর ইন্টিজারে তো 12-এর চেয়ে বড় কোনো সংখ্যার ফ্যাক্টরিয়ালের মান রাখা যায় না। আসলে এই প্রোগ্রামের জন্য  $n!$ -এর মান বের করার কোনো দরকার নেই। শুধু একটু গাণিতিক যুক্তি-বুদ্ধি খাটোও। আর 2 থেকে 99 পর্যন্ত মৌলিক সংখ্যাগুলো একটি অ্যারেতে রেখে নাও। প্রোগ্রামটি ঠিকভাবে করতে তোমাদের অনেকেরই দু-তিন দিন সময় লেগে যেতে পারে, এতে হতাশ হওয়ার কিছু নেই।

এখন আমরা একটি প্রোগ্রাম লিখব। যার উদ্দেশ্য হবে কোনো অ্যারেতে কিছু সংখ্যা থাকলে সেগুলোকে ছোট থেকে বড় ক্রমে সাজানো। যেমন, কোনো অ্যারে যদি এমন হয় : int ara[] = {3, 1, 5, 2, 4}, তাহলে আমাদের প্রোগ্রাম সেই অ্যারের সংখ্যাগুলো এমনভাবে সাজাবে, যাতে  $ara[] = \{1, 2, 3, 4, 5\}$  হয়। প্রোগ্রামটি একটু পরে লিখব, তার আগে ঠিক করে নিন্ত যে সেটি কীভাবে কাজ করবে। তোমার কাছে পাঁচটি সংখ্যা আছে : 3, 1, 5, 2, 4। ছোট থেকে বড় ক্রমে সাজাতে হবে। তুমি প্রথমে কী করবে? প্রথমে সবচেয়ে ছোট সংখ্যাটি খুঁজে বের করে তাকে শুরুতে লিখবে : 1।

তখন বাকি থাকে চারটি সংখ্যা : 3, 5, 2, 4। এখন এই চারটির মধ্যে সবচেয়ে ছোট সংখ্যাটি 1-এর পরে লিখবে : 1, 2। বাকি রইল মধ্যে সবচেয়ে ছোট 3। তাই তুমি লিখবে : 1, 2, 3, 5, 4। এদের মধ্যে সবচেয়ে ছোট 3। তাই তুমি লিখবে : 1, 2, 3, 4। এই দুটি সংখ্যার মধ্যে সবচেয়ে ছোট 4। সেটি 3। এখন বাকি 5, 4। এই দুটি সংখ্যার মধ্যে সবচেয়ে ছোট 4। সেটি 3। এখন বাকি 5, 4। এখন বাকি একটি সংখ্যা, 5। তুমি 3-এর পরে লিখবে : 1, 2, 3, 4, 5। তোমার সাজানোর সেটি তুমি 4-এর পরে লিখবে। 1, 2, 3, 4, 5। তোমার সাজানোর সেটি তুমি 4-এর পরে লিখবে। বিভিন্ন উপায়ে এটি কাজ হয়ে গেল। একে সর্টিং (sorting) বলে। বিভিন্ন উপায়ে করলাম। এখন করা যায়। তবে আমরা একটি সহজ-সরল উপায়ে করলাম। এখন প্রোগ্রামটি লিখব কীভাবে?

প্রথমে একটি অ্যারেতে সংখ্যাগুলো রাখো : int ara1[] = {3, 1, 5, 2, 4};

এখন আরেকটি অ্যারে নাও : int ara2[5]; অ্যারেটি এখনে খালি। তাই একটি ভেরিয়েবলে ইনডেক্স 0 লিখে রাখো।

int index\_2 = 0;  
এখন একটি একটি করে ara2-তে সংখ্যাগুলো রাখতে হবে। তার জন্য একটি লুপ দরকার।

for(index\_2 = 0; index\_2 < 5; index\_2++) // মানে 0 থেকে 4 পর্যন্ত প্রতিটি ঘরে আমরা সংখ্যা বসাব।

এই লুপের ভেতরে আরেকটি লুপ দরকার, যেটি দিয়ে আমরা ara1-এর সবচেয়ে ছোট সংখ্যা খুঁজে বের করব।

minimum = 100000; // এমন একটি বড় সংখ্যা অ্যাসাইন করলাম, যেটি ara1-এর যেকোনো সংখ্যার চেয়ে বড়।

```
for (i = 0; i < 5; i++) {
    if (ara1[i] < minimum) {
        minimum = ara1[i];
    }
}
```

এখন ara1-এর ক্ষুদ্রতম সংখ্যাটি minimum-এ চলে এল। সেটি  
এখন ara2 তে রাখি :

```
ara2[index_2] = minimum;
```

সবশেষে ara2-এর সব সংখ্যা প্রিন্ট করে দেখব।

এবার চলো, পুরো প্রোগ্রামটি লিখে কম্পাইল ও রান করে দেখি  
আউটপুট কী আসে।

```
#include <stdio.h>

int main()
{
    int ara1[] = {3, 1, 5, 2, 4};
    int ara2[5];
    int i, minimum, index_2;
    for (index_2 = 0; index_2 < 5; index_2++) {
        minimum = 10000;
        for (i = 0; i < 5; i++) {
            if (ara1[i] < minimum) {
                minimum = ara1[i];
            }
        }
        ara2[index_2] = minimum;
    }
    for (i = 0; i < 5; i++) {
        printf("%d\n", ara2[i]);
    }
    return 0;
}
```

প্রোগ্রাম : ১১.২  
কী সুন্দর প্রোগ্রাম! আউটপুট কী? আউটপুটও খুব সুন্দর, একে একে  
পাঁচটি ।

1

1

1

1

1

কিন্তু আমরা তো এমন আউটপুট চাইনি। কোথাও গোলমাল হয়েছে।  
এখন আমার কোডে দেখো তো, কোনো ভুল বের করা যায় কি না।

একটি ঝামেলা হয়েছে। ভেতরের লুপে (যেখানে সবচেয়ে ছোট সংখ্যা  
বের করা হয়) কিন্তু সব সময়ই minimum-এর মান 1 আসবে,  
কারণ 1 হচ্ছে ওই পাঁচটির মধ্যে সবচেয়ে ছোট সংখ্যা। এজন্য দ্বিতীয়  
অ্যারেতে পাঁচটি সংখ্যাই 1 হয়ে যাচ্ছে। তাই আমরা যখন  
minimum বের করব, তখন অ্যারের যে ঘরে সবচেয়ে ছোট সংখ্যা  
পাব সে ঘরের মান একটি অনেক বড় সংখ্যা দিয়ে দেব। এজন্য  
একটি ভেরিয়েবল রাখি minimum\_index। আর লুপটি এখন এমন  
হবে :

```
minimum = 10000;
for (i = 0; i < 5; i++) {
    if (ara1[i] < minimum) {
        minimum = ara1[i];
        minimum_index = i;
    }
}
```

এখন minimum-এর মান আমরা পেয়ে গেছি এবং সেই সঙ্গে এটিও  
জানি যে এটি আসলে আছে ara1[minimum\_index] ঘরে।

```
ara1[minimum_index] = 10000;
```

তাহলে প্রোগ্রামটি ঠিক করে আবার চালাই :

```
#include <stdio.h>
int main()
{
    int ara1[] = {3, 1, 5, 2, 4};
    int ara2[5];
    int i, minimum, index_2, minimum_index;
    for (index_2 = 0; index_2 < 5;
index_2++) {
        minimum = 10000;
        for (i = 0; i < 5; i++) {
            if (ara1[i] < minimum) {
                minimum = ara1[i];
                minimum_index = i;
            }
        }
        ara1[minimum_index] = 10000;
        ara2[index_2] = minimum;
    }
    for (i = 0; i < 5; i++) {
        printf("%d\n", ara2[i]);
    }
    return 0;
}
```

প্রোগ্রাম : ১১.৮

এখন প্রোগ্রামটি আউটপুট ঠিকঠাক দেখাবে। আচ্ছা, সব কাজই তো আমি করে দিলাম। তোমাদের কাজটি কী? তোমাদের কাজ হবে প্রোগ্রামটি এমনভাবে লেখা, যাতে দ্বিতীয় অ্যারের প্রয়োজন না হয়। শুরুতে যে অ্যারেটি আছে, তার ভেতরেই সর্টিং করতে হবে। এজন্য সবচেয়ে ছোট সংখ্যাটি অ্যারের প্রথম ঘরে নিয়ে আসো আর যে ঘর থেকে সবচেয়ে ছোট সংখ্যা পেয়েছ সেখানে প্রথম ঘরের সংখ্যাটি রাখো। এখন তোমার অ্যারের প্রথম ঘরে আছে সবচেয়ে ছোট সংখ্যা। এবারে বাকি চারটি ঘরের মধ্যে সবচেয়ে ছোট সংখ্যাটি অ্যারের দ্বিতীয় ঘরে রাখো এবং যে ঘর থেকে ওই সংখ্যাটি পেয়েছ, সেখানে দ্বিতীয় ঘরের সংখ্যাটি রাখো। আর কিছু বলা যাবে না।

এখন, রোবট নিয়ে আমরা একটি প্রোগ্রাম লিখব। কোনো একটি  $N \times N$  গ্রিডে একটি রোবট আছে। শুরুতে তার একটি অবস্থান আছে। আমরা সেটিকে কিছু কমান্ড দেব, এক ঘর ডানে, বাঁয়ে, ওপরে ও নিচে যাওয়ার কমান্ড।

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)	(0, 7)	(0, 8)
(1, 0)		(1, 2)						
(2, 0)	(2, 1)	R (2, 2) 2)	(2, 3)					
(3, 0)		(3, 2)						
(4, 0)								
(5, 0)								
(6, 0)								
(7, 0)								
(8, 0)								(8, 8)

গ্রিডটি দেখো। ওপরের একেবারে বাঁ দিকের ঘর হচ্ছে (0, 0)। ওপরের একেবারে ডান দিকের ঘর হচ্ছে (0, 8)। নিচের একেবারে বাঁ দিকের ঘর হচ্ছে (8, 0)। নিচের একেবারে ডান দিকের ঘর হচ্ছে (8, 8)। ধরা যাক, এই মুহূর্তে রোবটটি আছে (2, 2) ঘরে। এক ঘর ওপরে যেতে বললে সে যাবে (1, 2) ঘরে। নিচে যেতে বললে যাবে (3, 2) ঘরে। ডানে আর বাঁয়ে যেতে বললে যথাক্রমে (2, 3) ও (2, 1) ঘরে যাবে। কমান্ডগুলো হচ্ছে U (up), D (down), L (left), R (right), S (stop)। এখন তোমাকে যদি শুরুর অবস্থান আর কমান্ডগুলো বলে দিই, তাহলে রোবটের শেষ অবস্থান (stop করার পর অবস্থান) বের করতে হবে।

তোমরা কি প্রোগ্রামটি নিজে লেখার জন্য কিছুক্ষণ চেষ্টা করবে?

তোমরা নিচয়ই বুঝতে পারছ যে একটি 2-D অ্যারে দরকার হবে এই প্রোগ্রামে। আসলে কিন্তু এখানে অ্যারের কোনোই দরকার নেই। এটি সাধারণ যোগ-বিয়োগের প্রোগ্রাম। মনে করি, শুরুর অবস্থান হচ্ছে (x, y)। এখন U কমান্ড দিলে এক ঘর ওপরে যাবে, তখন x-এর মান এক কমে যাবে, y-এর মানের কোনো পরিবর্তন হবে না। D কমান্ড দিলে এক ঘর নিচে যাবে, তখন x-এর মান এক বেড়ে যাবে, y-এর মানের কোনো পরিবর্তন হবে না। R কমান্ড দিলে y-এর মান এক বাঢ়বে, x-এর মান অপরিবর্তিত থাকবে। L কমান্ড দিলে y-এর মান এক কমবে, x-এর মান অপরিবর্তিত থাকবে। তাহলে আমাদের পুরো প্রোগ্রামটি দাঁড়াবে এই রকম :

```
#include <stdio.h>
int main()
{
    int x, y;
    char c;
    printf("Please enter the initial
position: ");
    scanf("%d %d", &x, &y);
```

```
while (1) {
    scanf("%c", &c);
    if (c == 'S') {
        break;
    }
    else if (c == 'U') {
        x--;
    }
    else if (c == 'D') {
        x++;
    }
    else if (c == 'R') {
        y++;
    }
    else if (c == 'L') {
        y--;
    }
}
printf("Final position of the robot is:
%d, %d\n", x, y);
return 0;
```

প্রোগ্রাম : ১১.৫

আউটপুট কী হবে সেটি নির্ভর করবে তোমার ইনপুটের ওপর। যেমন :

Please enter the initial position: 2 2

D

R

D

R

S

Final position of the robot is: 4, 4

বেশ সহজ-সরল প্রোগ্রাম। কিন্তু এখন যদি বলি যে গ্রিডে কিছু কিছু ঘরে যাওয়া নিষেধ এবং ওই ঘরগুলোতে যেতে বললে রোবটটি কিছুই করবে না (অর্থাৎ ওই কমান্ডকে উপেক্ষা করবে), তখন আমরা প্রোগ্রামটি কীভাবে লিখব? যেমন একটি উদাহরণ দিই। ধরা যাক, (0, 4) ঘরটি নিষিদ্ধ (blocked)। যদি রোবটের অবস্থান হয় (0, 3) ঘরে এবং তাকে 'R' কমান্ড দেওয়া হয়, তখন তার অবস্থানের কোনো পরিবর্তন হবে না। কারণ এক ঘর ডানে [মানে (0, 4) ঘরে] যাওয়া সম্ভব নয়।

এই সমস্যার সমাধান করতে যে প্রোগ্রামটি লিখতে হবে, তাতে কিন্তু একটি 2-D অ্যারে ব্যবহার করতে হবে। এই অ্যারের সাহায্যে আমরা বুঝব যে কোন ঘরে যাওয়া যাবে আর কোন ঘরে যাওয়া যাবে না। সেটি কীভাবে? খুবই সহজ। যেসব ঘরে যাওয়া যাবে অ্যারের ওই ঘরগুলোতে 1 আর যেসব ঘরে যাওয়া যাবে না সেগুলোতে 0 রাখব।

প্রথমে  $10 \times 10$  গ্রিডের জন্য একটি 2-D অ্যারে ডিক্রেয়ার করি :

```
int grid[10][10];
```

তারপর শুরুতে ধরে নিই সব ঘরে যাওয়া যাবে।

```
for (i = 0; i < 10; i++) {  
    for (j = 0; j < 10; j++) {  
        grid[i][j] = 1;  
    }  
}
```

এখন কোন ঘরগুলোতে যাওয়া যাবে না, তা ব্যবহারকারীর কাছ থেকে ইনপুট নিই:

```
printf("Please enter the number of  
blocked cells: ");
```

```
scanf("%d", &n);
```

```
printf("Now enter the cells: ");
```

```
for (i = 0; i < n; i++) {
```

```
    scanf("%d %d", &x, &y);
```

```
    grid[x][y] = 0;
```

```
}
```

এখন কোনো ঘরে যাওয়া যাবে কি না, সেটি বোঝার জন্য একটি শর্ত পরীক্ষা করলেই হবে।

```
if (grid[x][y] == 1) {
```

যদি সত্য হয়, তবে  $(x, y)$  ঘরে যাওয়া যাবে।

```
}
```

এখন তোমরা সম্পূর্ণ প্রোগ্রামটি নিজে নিজে লিখে ফেলো।

## অধ্যায় চৌদ্দ : শেষের শুরু

আমরা বইয়ের শেষ অধ্যায়ে চলে এসেছি। তোমরা যদি আগের অধ্যায়গুলো ঠিকমতো পড়ে থাকো, উদাহরণগুলো নিজে নিজে কম্পিউটারে চালিয়ে দেখে থাকো এবং যখনই আমি তোমাদের কোনো প্রোগ্রাম নিজে লিখতে বলেছি, সেগুলো নিজে লেখার চেষ্টা করে থাকো, তাহলে তোমাকে অভিনন্দন! তুমি প্রোগ্রামিং শেখার জন্য প্রস্তুত হয়ে গেছ। যদি বলতে পারতাম তুমি প্রোগ্রামিং শিখে ফেলেছ তাহলে তোমাদেরও ভালো লাগত, আমারও ভালো লাগত; কিন্তু মিথ্যা কথা বলে কী লাভ?

প্রোগ্রামিং হচ্ছে চৰ্চার বিষয়। মুখস্থ করে পরীক্ষায় অনেক ভালো রেজাল্ট করা যায়, এমনকি কলেজ-বিশ্ববিদ্যালয়ে প্রোগ্রামিং পরীক্ষাতেও মুখস্থ করে অনেকেই বেশ ভালো নম্বর পায়। তবে এই ভালো নম্বর পাওয়ার সঙ্গে ভালো প্রোগ্রামার হওয়ার আসলে কোনো সম্পর্ক নেই। প্রোগ্রামিং হচ্ছে একধরনের দক্ষতা (skill) এবং কেবল নিয়মিত অনুশীলনের মাধ্যমেই এই দক্ষতা অর্জন করা সম্ভব। এর জন্য ভালো ছাত্র হওয়ার দরকার নেই, জিনিয়াস হওয়ারও কোনো দরকার নেই। দরকার হচ্ছে প্রোগ্রামিংকে ভালোবাসা। যখন তুমি প্রোগ্রামিং করতে বসলে খাওয়াওয়ার কথা ভুলে যাবে, রাতে কোনো প্রোগ্রামিং-সমস্যা নিয়ে কাজ শুরু করলে আর কিছুক্ষণ পরে দেখবে বাইরে ভোরের আলো ফুটছে, কিংবা ভুলে বাথরুমের স্যান্ডেল পরে ঢাকে চলে যাবে, তখন বুরবে যে তুমি প্রোগ্রামার হয়ে যাচ্ছ।

এখন তোমার উচিত হবে বইটি আরেকবার পড়া এবং সঙ্গে সঙ্গে প্রোগ্রামগুলো আবার করা। তারপর তোমরা আরও বেশি সি শিখতে চাইলে সি-এর কোনো বই পড়তে পারো। তোমরা যদি প্রোগ্রামিং কটেস্টের ব্যাপারে উৎসাহী হও তাহলে সি প্লাস প্লাস (C++) শেখা শুরু করে দিতে পারো কোনো বই থেকে। আবার জাভা (Java), সি শার্প (C#), পিএইচপি (PHP) কিংবা পাইথন (Python) শিখতে পারো। কেনোটি শিখতেই তেমন ঝামেলা পোহাতে হবে না কারণ তুমি প্রোগ্রামিংয়ের মৌলিক জিনিসগুলো এতক্ষণে আয়ত্তে এনে স্লেছ। বই ও ওয়েবসাইটের তালিকা আমি পরিশিষ্ট অংশে লিখেছি।

একজন দক্ষ প্রোগ্রামার হতে গোলো যে জিনিসগুলো লাগবে তা হচ্ছে :

- ১) এক বা একাধিক প্রোগ্রামিং ল্যাঙ্গুয়েজে ভালো দখল,
- ২) ভালো একটি IDE ব্যবহারের দক্ষতা,
- ৩) প্রোগ্রামিংয়ের মৌলিক বিষয়গুলো সম্পর্কে স্বচ্ছ ধারণা,
- ৪) গণিত ও যুক্তিতে দক্ষতা,
- ৫) অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিংয়ে (OOP : Object Oriented Programming) দক্ষতা,
- ৬) ডাটা স্ট্রাকচার ও অ্যালগরিদমের মৌলিক জ্ঞান ও তা প্রয়োগের দক্ষতা,
- ৭) যোগাযোগে দক্ষতা (Communication Skills),
- ৮) ইন্টারনেট ঘেঁটে অল্প সময়ে কোনো সমস্যার সমাধান বের করা বা দ্রুত কোনো নতুন বিষয় শিখে নেওয়ার দক্ষতা,
- ৯) একটি সমস্যার পেছনে লেগে থাকার মানসিকতা,
- ১০) প্রোগ্রামিংয়ের প্রতি ভালোবাসা।

তোমাদের প্রোগ্রামিং জীবন আনন্দময় হোক, তোমাদের নিজের জীবন আনন্দময় হোক, তোমাদের কারণে তোমাদের আশপাশের মানুষের জীবন আনন্দময় হোক। সবাইকে শুভেচ্ছা।

## পরিষিষ্ট শূন্য : অ্যালগরিদম ও ফ্লোচার্ট

যখন কোনো সমস্যা আমরা প্রোগ্রামিং করে সমাধান করব, বা কোনো সমস্যা সমাধানের জন্য প্রোগ্রাম লিখব, তখন সমস্যাটি দেখেই হট হবে। কারণ আয়ই এমন হয় যে আমাদের সমাধানটি সঠিক কিন্তু জিনিসটির সমাধান করতে হবে সেটি ঠিকভাবে বুঝতে পারিনি। তাই প্রথম শর্ত হচ্ছে সমস্যাটি ভালোভাবে বোঝা। তারপরের কাজ হচ্ছে সমস্যা সমাধানের ধাপগুলো চিহ্নিত করা। ধরা যাক, আমাদের একটি প্রোগ্রাম লিখতে হবে যেখানে কোনো সংখ্যা জোড় না বেজোড় সেটি বের করতে হবে। তাহলে এই সমস্যা সমাধানের ধাপগুলো কেমন হবে?

- 1) প্রথমে ব্যবহারকারীর কাছ থেকে সংখ্যাটি ইনপুট নিতে হবে।
- 2) তারপর সংখ্যাটিকে ২ দিয়ে ভাগ করে ভাগশেষ বের করতে হবে।
- 3) ভাগশেষ যদি শূন্য হয়, তবে আউটপুট দিতে হবে সংখ্যাটি জোড়।
- 4) ভাগশেষ যদি এক হয়, তাহলে আউটপুট দিতে হবে সংখ্যাটি বেজোড়।

আবার ধরা যাক, আমাদের বের করতে হবে ১ থেকে ১০০ পর্যন্ত সমস্ত সংখ্যার যোগফল। তাহলে এই সমস্যা সমাধানের ধাপগুলো কী হবে? আর হ্যাঁ, আমরা এখানে সরাসরি যোগফল বের করার সূত্রটি ব্যবহার করব না।

- 1) প্রথমে আমরা ধরি যোগফলের মান শূন্য। এর জন্য আমরা একটি ভেরিয়েবল ব্যবহার করতে পারি,  $sum = 0$ ;
- 2) এখন একটা সংখ্যা ধরি  $k$  যার মান এক। অর্থাৎ  $k = 1$ ;
- 3) যোগফলকে  $k$  বাড়িয়ে দিই। মানে, যোগফলের বর্তমান যে মান রয়েছে, তার সঙ্গে  $k$  যোগ করি।  $sum = sum + k$ ;
- 4) এবারে  $k$ -এর মান ১ বাড়াই। অর্থাৎ,  $k = k + 1$ ;

৫)  $k$ -এর মান কি ১০০ থেকে বড় হয়ে গেছে? যদি না হয়, তাহলে ৩ নম্বর ধাপে যাই। আর যদি হয়, তাহলে আমাদের আর কিছু করার নেই,  $k$ -এর মান ১ থেকে ১০০ পর্যন্ত সব সংখ্যার ক্ষেত্রে যোগফল sum-এ যোগ করা হয়ে গেছে।

৬) sum-এর মান আউটপুটে দেখাই।  
ওপরের ধাপগুলো দিয়ে কি সমস্যার সমাধান সঠিকভাবে করা গেল? উত্তরটি তুমি নিজেই খুঁজে বের করো। আর সেজন্য খাতা-কলম নিয়ে বসো এবং ওপরের পদ্ধতি অনুসরণ করে ১ থেকে ১০ পর্যন্ত সংখ্যাগুলোর যোগফল বের করা যায় কি না, দেখো।

আমরা এভাবে সমস্যা সমাধানের যে ধাপগুলো লিখি, সেটিকে বলে অ্যালগরিদম। তুমি জেনে খুশি হবে যে এই মধ্যে তুমি অ্যালগরিদম লেখা শিখে গিয়েছ! তবে একটি কথা বলে নেওয়া ভালো, অ্যালগরিদম বিভিন্নভাবে লেখা যায়। তুমি যদি পরীক্ষার হলে অ্যালগরিদম লিখতে বসো, তাহলে তোমার পাঠ্যবইয়ের অ্যালগরিদম লেখার পদ্ধতি অনুসরণ করলেই হবে। আর পরীক্ষার হল ছাড়া যেকোনো জায়গায়, জীবনের যেকোনো সময়, তোমরা আমার দেখানো পদ্ধতি নিশ্চিতে অনুসরণ করতে পারো। এখন তোমার নিজে নিজে একটি কাজ করতে হবে। ১ থেকে ১০০ পর্যন্ত সব জোড় সংখ্যার যোগফল বের করার অ্যালগরিদম লিখে ফেলতে হবে।

ওধূ কি প্রোগ্রামিং সমস্যার সমাধানের জন্যই অ্যালগরিদম? না, আসলে বাস্তব জীবনের অনেক সমস্যা, যেখানে সমাধানের কিছু ধাপ আছে, সেখানেই অ্যালগরিদম ব্যবহার করা যায়। যেমন ধো, তোমার টেবিল ল্যাম্পটি জ্বলছে না। তুমি তখন কীভাবে সেটি ঠিক করার চেষ্টা করবে?

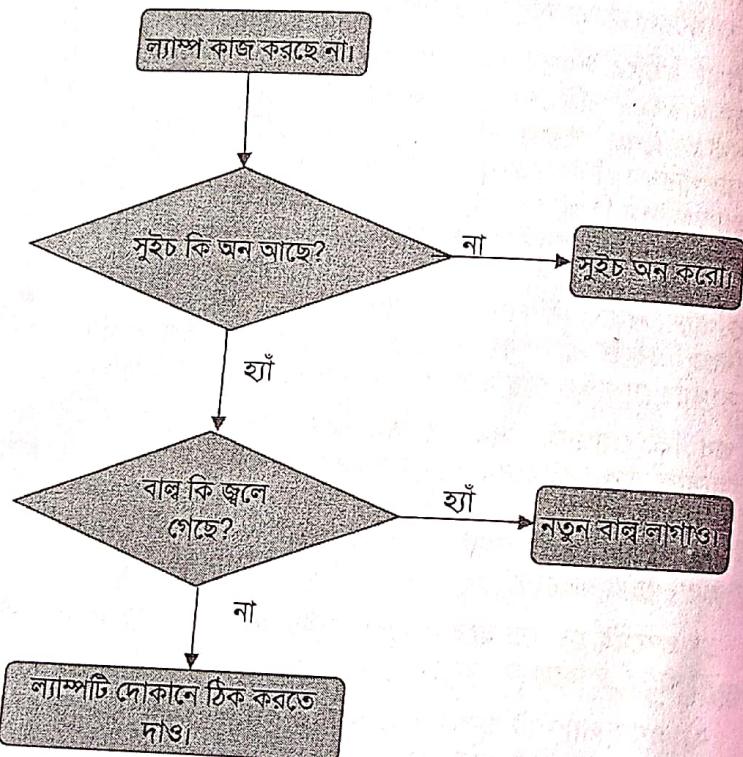
- ১) প্রথমেই দেখতে হবে ল্যাম্পের সুইচ অন করা আছে কি না। যদি না থাকে, তাহলে অন করো।
- ২) এখন ল্যাম্পটি জ্বলছে? উত্তর যদি হ্যাঁ হয় তাহলে ৬ নম্বর ধাপে যাও, আর না হলে পরের ধাপে যাও।
- ৩) ল্যাম্প থেকে বাল্বটি খুলে দেখো যে বাল্বটি কি জ্বলে বা পুড়ে গেছে? যদি হ্যাঁ হয় তাহলে পরের ধাপে যাও, না হলে ৫ নম্বর ধাপে যাও।
- ৪) ল্যাম্পে নতুন বাল্ব এনে লাগাও। এবারে কী ল্যাম্পটি জ্বলছে? উত্তর হ্যাঁ হলে ৬ নম্বর ধাপে যাও। না হলে পরের ধাপে যাও।

৫) এই ধাপে আসা মানে তোমার সুইচ ও বাল্ব ঠিক আছে।  
ল্যাম্পটিতে সমস্যা। তাই ল্যাম্পটি দোকানে ঠিক করতে দাও।

৬) ল্যাম্পটি এখন কাজ করছে। সমস্যার সমাধান হয়ে গেল।

এখানে আমরা ল্যাম্প ঠিক করার অ্যালগরিদম লিখে ফেললাম।

এই ব্যাপারটিকে একটি ছবির সাহায্যেও বোঝানো যায়। ল্যাম্প ঠিক করার অ্যালগরিদমটি নিচের ছবি দিয়ে আমরা প্রকাশ করতে পারি।

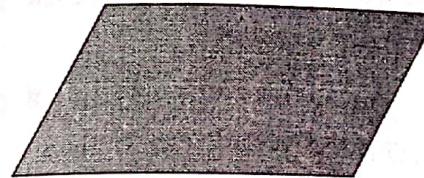


এই ছবিটিকে বলে ফ্লো-চার্ট (flow chart)। প্রতিটি অ্যালগরিদমকে  
আমরা ফ্লোচার্টের মাধ্যমে প্রকাশ করতে পারি। অ্যালগরিদমের লেখার  
চেয়ে ফ্লোচার্ট তৈরির সুবিধা হলো যে সেটি আকারে ছোট বা সংক্ষিপ্ত  
হয়, তাই দেখে সহজেই বোঝা যায়।

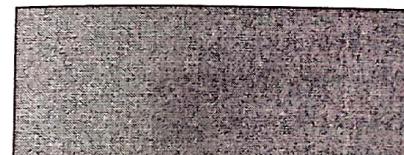
ফ্লোচার্ট তৈরির কিছু নিয়মকানুন আছে। কারণ একই চিহ্ন তো একেক  
জন একেক অর্থে ব্যবহার করলে হবে না। তাই আসো আমরা  
চিহ্নগুলোর সঙ্গে পরিচিত হই। সব চিহ্নের সাথে এখন পরিচয়  
করাবো না, কেবল যেগুলো না হলেই নয়, সেগুলোর সঙ্গে পরিচিত  
হবো।



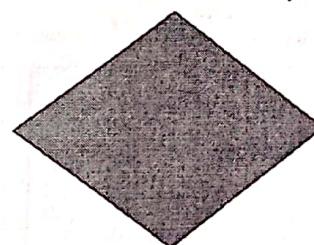
শুরু / শেষ (উপরের যেকোনো একটা চিহ্ন)



ইনপুট বা আউটপুট



প্রসেস (মানে কোনো স্টেটমেন্ট-জাতীয় কাজ, যেমন  $sum = sum + x;$  বা  $n = 0;$ )

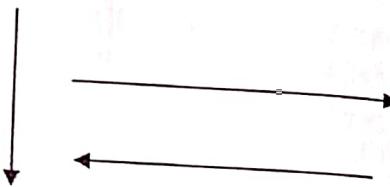


কন্ডিশন বা শর্ত পরীক্ষা



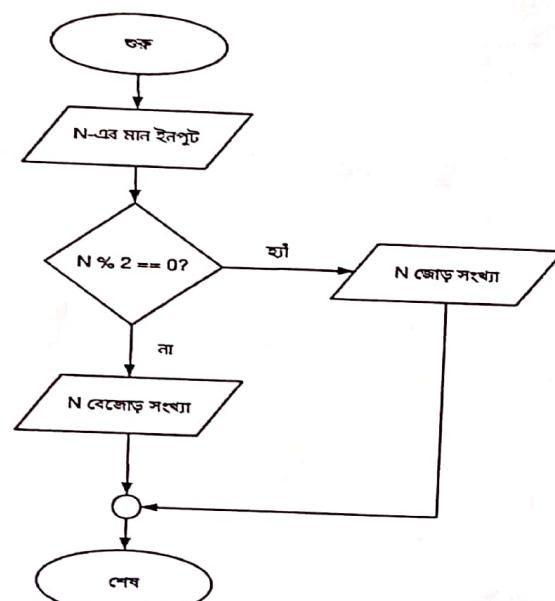
কানেক্টর (একাধিক ফ্লো একসঙ্গে এসে যুক্ত হয়)

এ ছাড়া বিভিন্ন অ্যারো (তির) চিহ্ন দিয়ে দিক নির্দেশ করা হয়।  
যেমন:

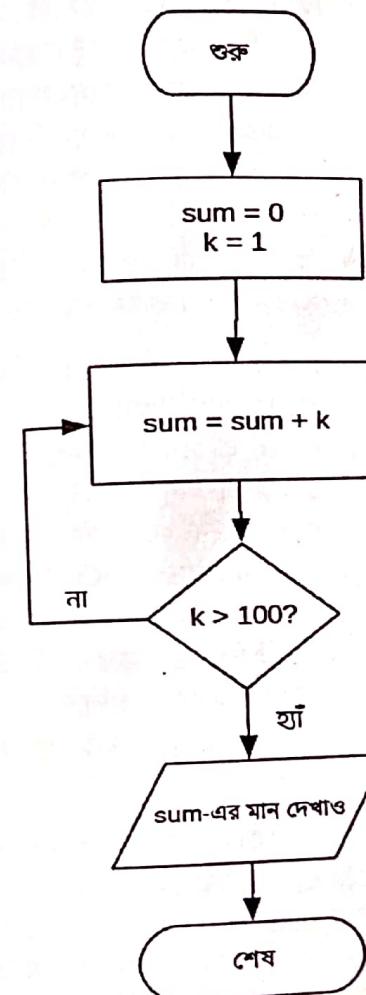


বিভিন্ন প্রোগ্রামের ফ্লোচার্ট আঁকার এগুলো একেবারে বেসিক চিহ্ন।  
পরবর্তী সময়ে তোমরা যখন প্রোগ্রামিং নিয়ে আরও লেখাপড়া করবে,  
আরও কিছু চিহ্ন শিখবে। প্রোগ্রামিং সমস্যার সমাধান ছাড়াও বিভিন্ন  
সিস্টেম ডিজাইনে ফ্লোচার্ট ব্যবহার করা হয়।

তো আমরা যেই চিহ্নগুলো শিখলাম, সেগুলো দিয়ে জোড়-বেজোড়  
সংখ্যা বের করার ফ্লোচার্ট এঁকে ফেলি।



এখন আমরা 1 থেকে 100 পর্যন্ত সংখ্যাগুলোর যোগফল নির্ণয়ের  
ফ্লোচার্ট আঁকব। এই ফ্লোচার্টে দেখবে শুরু এবং শেষ, এই দুই  
জায়গায় আমি একটু অন্যরকম চিহ্ন ব্যবহার করেছি, এই চিহ্নগুলোও  
সঠিক।



আশা করি, তুমি ফ্লোচার্টের তৈরির নিয়মটি বুঝতে পেরেছ।

## পরিশিষ্ট এক : প্রোগ্রামিং প্রতিযোগিতা

প্রোগ্রামিং প্রতিযোগিতা হচ্ছে প্রোগ্রামারদের মধ্যে লড়াই। এর মানে কিন্তু এই নয় যে প্রোগ্রামাররা একে অপরের সঙ্গে মারামারি করবে আর শেষ পর্যন্ত যে ঢিকে থাকবে সে-ই বিজয়ী। আসলে প্রোগ্রামিং প্রতিযোগিতা হচ্ছে একটি পরীক্ষার মতো, যেখানে প্রত্যেককে একটি নির্দিষ্ট সময়ে নির্দিষ্টসংখ্যক প্রোগ্রামিং সমস্যার সমাধান করতে দেওয়া হবে। যে সবচেয়ে বেশি সমস্যার নির্তুল সমাধান করবে সে বিজয়ী হবে। আর দুজন যদি সমানসংখ্যক সমস্যার সমাধান করে, তাহলে তাদের মধ্যে যে কম সময়ে করেছে সে বিজয়ী। তবে স্কুল-কলেজের পরীক্ষার সঙ্গে এর পার্থক্য হচ্ছে, এখানে বই থেকে সরাসরি প্রশ্ন করা হয় না। তাই মুখস্থ করার কোনো সুযোগ নেই। বিচারকেরা অনেক সময় নিয়ে প্রোগ্রামিং প্রতিযোগিতার সমস্যা তৈরি করেন। এর মধ্যে সহজ সমস্যাও থাকে আবার খুব কঠিন সমস্যাও থাকে।

স্কুল-কলেজের ছাত্রাত্মীদের জন্য সবচেয়ে বড় প্রতিযোগিতা হচ্ছে আইওআই (IOI : International Olympiad in Informatics)। ১৯৮৯ সাল থেকে প্রতিবছর এ প্রতিযোগিতা অনুষ্ঠিত হচ্ছে। একেক বছর একেক দেশে প্রতিযোগিতা অনুষ্ঠিত হয়। এ প্রতিযোগিতায় এখন পর্যন্ত আমাদের সেরা অর্জন হচ্ছে ২০০৯ সালে আবির্মল ইসলামের রোপ্য পদক (সিলভার মেডেল)। IOI-তে অংশগ্রহণ করার জন্য বাংলাদেশ দল গঠনের কাজটি করা হয় দুই ধাপে। প্রথমে বিভাগীয় ইনফরমেটিকস অলিম্পিয়াড। তারপর বিভাগীয় পর্যায়ের বিজয়ীদের নিয়ে জাতীয় ইনফরমেটিকস অলিম্পিয়াড অনুষ্ঠিত হয়। জাতীয় অলিম্পিয়াডের বিজয়ীদের মধ্য থেকেই দলের সদস্য বাছাই করা হয়।

বিশ্ববিদ্যালয় পর্যায়ের ছাত্রাত্মীদের জন্য সবচেয়ে বড় প্রোগ্রামিং প্রতিযোগিতা হচ্ছে এসএম আইসিপিসি (ACM ICPC : ACM International Collegiate Programming Contest)। এর জন্য দল বাছাই অনেকটা বিশ্বকাপ ফুটবলের মতো হয়। প্রতি মহাদেশ থেকে প্রতিযোগিতার মাধ্যমে দল নির্বাচন করা হয়। একটি দলে তিনজন সদস্য এবং একজন প্রশিক্ষক থাকেন। মজার ব্যাপার

হচ্ছে, দলের সদস্যদের কিন্তু কম্পিউটার-বিজ্ঞানের শিক্ষার্থী হতে হবে, এমন কোনো কথা নেই। যেকোনো বিভাগের শিক্ষার্থী এই প্রতিযোগিতায় অংশগ্রহণ করতে পারে। বাংলাদেশের প্রতিযোগীরা ঢাকায় ICPC Regional Contest-এ অংশগ্রহণ করে। এ ছাড়া ভারত ও আশপাশের দেশের ICPC Regional Contest-এও বাংলাদেশের প্রতিযোগীদের অংশগ্রহণের সুযোগ রয়েছে এবং প্রায়ই আমাদের দেশের কয়েকটি দল ওইসব প্রতিযোগিতায় অংশগ্রহণ করে। ICPC Regional Contest-এ বিজয়ী দলগুলো সুযোগ পায় চূড়ান্ত পর্বে (ICPC World Finals) অংশগ্রহণ করার। ১৯৯৮ চূড়ান্ত পর্বে বিজয়ী দলগুলো সুযোগ পায় সালের পর থেকে প্রতিবছরই বাংলাদেশ থেকে কমপক্ষে একটি দল চূড়ান্ত পর্বে অংশগ্রহণের যোগ্যতা লাভ করে যা আমাদের দেশের প্রোগ্রামারদের কৃতিত্বের পরিচয় বহন করে। তোমরা এ বিষয়ে উইকিপিডিয়াতে [আরও তথ্য পাবে এই লিঙ্কে:](http://en.wikipedia.org/wiki/ACM_ICPC_Dhaka_Site) [http://en.wikipedia.org/wiki/ACM\\_ICPC\\_Dhaka\\_Site](http://en.wikipedia.org/wiki/ACM_ICPC_Dhaka_Site)।

এ ছাড়া ইন্টারনেটে অনুষ্ঠিত হয় আরও নানা ধরনের প্রোগ্রামিং প্রতিযোগিতা যেখানে স্কুল-কলেজ-বিশ্ববিদ্যালয়ের ছাত্র, শিক্ষক ও পেশাজীবীরা অংশগ্রহণ করতে পারেন। এদের মধ্যে গুরুত্বপূর্ণ তিনটি হচ্ছে Google Code Jam (<http://code.google.com/codejam>), Topcoder (<http://www.topcoder.com/tc>) এবং Codechef (<http://www.codechef.com/>)। এই প্রতিযোগিতাগুলো অত্যন্ত কঠিন, তাই এতে অংশগ্রহণের জন্য পর্যাপ্ত দক্ষতা থাকতে হবে। তবে এসব প্রতিযোগিতায় কিন্তু বাংলাদেশের প্রোগ্রামাররা বেশ ভালো অবস্থানে রয়েছে।

প্রোগ্রামিং প্রতিযোগিতায় ভালো করতে হলে প্রোগ্রামিংয়ে দক্ষতার পাশাপাশি অ্যালগরিদম ও গণিতে বিশেষভাবে দক্ষ হতে হয়। জ্যামিতি, কম্বিনেটরিকস, সংখ্যাতত্ত্ব ইত্যাদির যথেষ্ট জ্ঞানের পাশাপাশি সমস্যা সমাধানের দক্ষতা অর্জন করতে হয়। এজন্য লেখাপড়ার পাশাপাশি নিয়মিত প্রোগ্রামিং সমস্যা সমাধানের কোনো বিকল্প নেই। আর বিভিন্ন ওয়েবসাইটে নিয়মিত প্রোগ্রামিং প্রতিযোগিতার আয়োজন করা হয় যেখানে তুমি অংশগ্রহণ করতে পারো ইন্টারনেটের মাধ্যমে।

## পরিশিষ্ট দুই : প্রোগ্রামিং ক্যারিয়ার

গণিত যেমন কেবল গণিতবিদেরাই ব্যবহার করেন না, বরং বিজ্ঞানের সব শাখায় রয়েছে এর ব্যবহার, তেমনই প্রোগ্রামিংও কিন্তু কেবল কম্পিউটার-বিজ্ঞানী বা কম্পিউটার ইঞ্জিনিয়ারদের জন্য নয়। বিশ্ববিদ্যালয়ে পড়তে গেলে বিজ্ঞান ও প্রকৌশলের সব বিভাগের শিক্ষার্থীদের জন্য প্রোগ্রামিং জানাটা খুব গুরুত্বপূর্ণ।

পেশা হিসেবে প্রোগ্রামিংয়ের আলাদা একটি গুরুত্ব আছে আমাদের জন্য। যেহেতু বিভিন্ন ধরনের প্রতিযোগিতার মাধ্যমে নিজেকে মেলে ধরার অনেক সুযোগ এখানে রয়েছে, তাই বাংলাদেশ থেকে লেখাপড়া করে সরাসরিই বিশ্বের নামকরা সফটওয়্যার নির্মাতা প্রতিষ্ঠান যেমন—গুগল, মাইক্রোসফট, ফেসবুক ইত্যাদিতে কাজ করার সুযোগ তৈরি হয়েছে। প্রতিবছরই বাংলাদেশ থেকে কয়েকজন প্রোগ্রামার নিজের মেধা ও জ্ঞানকে কাজে লাগিয়ে এই সুযোগের সম্বন্ধে ব্যবহার করছেন। অনেক ক্ষেত্রেই ওইসব প্রতিষ্ঠানে কাজ করার জন্য আবেদন করার প্রয়োজন হয় না, তারা নিজে থেকেই বিভিন্ন দেশের সেরা প্রোগ্রামারদের খুঁজে বের করে।

বিশ্বিখ্যাত সব প্রতিষ্ঠানে কাজ করা ছাড়াও প্রোগ্রামারদের জন্য আরেকটি সুবিধা হচ্ছে Telecommuting। অর্থাৎ কোনো অফিসে না গিয়ে কাজ করার সুযোগ। উন্নত বিশ্বের অনেক কোম্পানি তাদের নিজ দেশে প্রোগ্রামারদের দুষ্প্রাপ্যতার কারণে উন্নয়নশীল অর্থনীতির দেশের প্রোগ্রামারদের কাজের সুযোগ দেয়, আর সে ক্ষেত্রে নিজ দেশে বসেই কাজ করা যায়। কারণ ওই কোম্পানিগুলো জানে যে আমাদের মতো দেশের অর্থনীতি উন্নয়নশীল হলেও প্রোগ্রামাররা মোটেও অদক্ষ নন, বরং বিশ্বানের প্রোগ্রামার। বাংলাদেশের বেশ কিছু প্রোগ্রামার এখন বাংলাদেশে বসেই ইন্টারনেটের মাধ্যমে কাজ করছেন আমেরিকা, কানাডা ও ইউরোপের বিভিন্ন দেশের সফটওয়্যার কোম্পানিতে।

আরেকটি মজার ব্যাপার হচ্ছে, কেউ যদি ধরাৰ্ধা চাকরি করতে না চায়, তাহলে তার জন্য ফ্রিল্যান্স প্রোগ্রামিংয়ের সুযোগ রয়েছে।

ইন্টারনেটে অনেক ওয়েবসাইট আছে যেখানে ছেট-মার্কারি-বড় বিভিন্ন ধরনের সফটওয়্যারের প্রজেক্ট থাকে যেগুলোতে বিড (bid) করে কাজ করা যায়। বাংলাদেশে এখন শত শত প্রোগ্রামার ফ্রিল্যান্স করে কাজ করা যায়। এর জন্য কেবল কম্পিউটার ও ইন্টারনেট প্রোগ্রামিংয়ের সঙ্গে জড়িত। এর জন্য কেবল কম্পিউটার ও ইন্টারনেট সংযোগ থাকলেই চলবে। ফ্রিল্যান্স কাজ করার জন্য বিপুল ধৈর্যের প্রয়োজন। আর ইংরেজি ভাষায় যোগাযোগের দক্ষতা থাকতে হয়। প্রয়োজন। আমি মনে করি, ছাত্রাবস্থায় এ ধরনের কাজ না তবে ব্যক্তিগতভাবে আমি মনে করি, ছাত্রাবস্থায় এ ধরনের কাজ না করাই ভালো। কারণ ছাত্রজীবনে লেখাপড়া করার ও মৌলিক বিষয়গুলো আয়ত্ত করার যে সময় ও সুযোগ মেলে, জীবনের পরবর্তী বিষয়গুলো আয়ত্ত করার যে সময় ও সুযোগ পাওয়া যায় না। তাই তোমাদের প্রতি পর্যায়ে কখনোই সেই সুযোগ পাওয়া যায় না। তাই তোমাদের প্রতি আমার পরামর্শ থাকবে যে ছাত্রজীবনে অর্থ উপর্যুক্তের দিকে মনোযোগ না দিয়ে প্রচুর লেখাপড়া এবং সঙ্গে সঙ্গে নানা ধরনের সামাজিক ও সাংস্কৃতিক কর্মকাণ্ডে জড়িত থাকার চেষ্টা করবে, যেগুলো তোমার ভালো লাগে।

সবশেষ কথা হচ্ছে, প্রোগ্রামিং এমন একটি কাজ, যেখানে সব সময়ই তোমার নিজেকে উন্নত করার সুযোগ আছে। তাই লেখাপড়া করার মানসিকতা থাকতে হবে, পড়তে হবে নানা বইপত্র, ঘোঁটতে হবে ইন্টারনেট। নিজে কোনো সমস্যায় পড়লে প্রথমেই ইন্টারনেট মেঁটে ইন্টারনেট। নিজে কোনো সমস্যাটির সমাধান ইতিমধ্যে কেউ করে রেখেছে কি না। দেখবে যে সমস্যাটির সমাধান ইতিমধ্যে কেউ করে রেখেছে কি না। দেখবে যে সমস্যাটির সমাধান ইতিমধ্যে কেউ করে রেখেছে কি না। তবে প্রোগ্রামিং শেখার সময় কিন্তু সমাধানের সমাধান দিয়ে রাখে। তবে প্রোগ্রামিং শেখার সময় কিন্তু সমাধানের সমাধান দিয়ে রাখে। জন্য ইন্টারনেট ঘোঁটবে না, নিজে চেষ্টা করবে।

### পরিশিষ্ট তিনি : বই ও ওয়েবসাইটের তালিকা

তুমি যদি ইতিমধ্যে এই বইটি পড়ে ফেলো এবং এবারে ভালোভাবে সি শিখতে চাও, তবে Herbert Schildt-এর *Teach Yourself C* বইটি পড়তে পারো। আবার Brian Kernighan ও Dennis Ritchie-এর লেখা *The C Programming Language* বইটিও পড়তে পারো। লেখকদের একজন, Dennis Ritchie, সি ল্যাঙ্গুয়েজ ডিজাইন করেছেন। আর কেউ যদি তোমার কাছে জানতে চায় শুরুতে সি শিখতে হলে কোন ইংরেজি বইটি ভালো, তাহলে Stephen G. Kochan-এর *Programming in C* বইটির কথা বলে দেবে। এটি সি শেখার জন্য চমৎকার ও সহজ একটি বই। Schaums Outlines সিরিজের *Programming with C* বইটিও ভালো। বইতে প্রচুর উদাহরণ আর অনুশীলনী আছে।

সি শেখার পরে তুমি সি প্লাস প্লাস বা জাভা শিখতে পারো। সি প্লাস প্লাস শেখার জন্য ভালো বই হচ্ছে *Teach Yourself C++* (লেখক Herbert Schildt) আর জাভার জন্য *Java How to Program* (লেখক: Paul Deitel and Harvey Deitel)। তারপর অন্য ল্যাঙ্গুয়েজ শিখতে গেলে আর বই কেনার দরকার নেই। ইন্টারনেটে প্রচুর টিউটোরিয়াল আছে। সেগুলো পড়ে শিখে ফেলবে।

তুমি যদি কম্পিউটার-বিজ্ঞানে পড়তে চাও, কিংবা প্রোগ্রামিং কনটেক্টে ভালো করতে চাও, তাহলে তোমার Discrete Mathematics ভালো করে শিখতে হবে। এর জন্য Kenneth H. Rosen-এর *Discrete Mathematics* বইটি খুব ভালো। আগাগোড়া পড়ে ফেলবে। সঙ্গে সঙ্গে অনুশীলনীর সমস্যাগুলো সমাধানের চেষ্টা করবে। Discrete Mathematics শেখার পরে শিখতে হবে অ্যালগরিদম। অ্যালগরিদম শেখার শুরু আছে কিন্তু শেষ নেই। আর শুরু করার জন্য তোমরা পড়তে পারো *Introduction to Algorithms* (লেখক Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein) এটি অ্যালগরিদমের মৌলিক বিষয়গুলো শেখার জন্য আমার দেখা সবচেয়ে ভালো বই।

আর তোমাদের জন্য প্রোগ্রামিং, সফটওয়্যার তৈরি থেকে শুরু করে তথ্যপ্রযুক্তির নানান বিষয়ের ওপর বাংলায় নিয়মিত অনলাইন কোর্সের আয়োজন করছে দ্বিমিক কম্পিউটিং স্কুল। ওয়েবসাইটের ঠিকানা হচ্ছে <http://dimikcomputing.com>।

প্রোগ্রামিং প্রতিযোগিতার জন্য কিছু লিংক :

- <http://projecteuler.net/> এখানে অনেক মজার সমস্যা আছে, যেগুলোর বেশিরভাগই প্রোগ্রাম লিখে সমাধান করতে হয়। এখানে প্রোগ্রাম জমা দেওয়া লাগে না, কেবল প্রোগ্রাম দিয়ে বের করা উত্তরটা জমা দিতে হয়।
- <http://www.spoj.pl> এখানেও অনেক ভালো সমস্যা আছে। সমাধান করে প্রোগ্রাম জমা দিলে প্রোগ্রাম সঠিক হয়েছি কি না তা জানা যায়। এই ওয়েবসাইটের একটি বৈশিষ্ট্য হচ্ছে সি, সি প্লাস প্লাস, জাভা, পার্ল, পাইথন, রুবি, পিএইচপি ইত্যাদি ব্যবহার করে প্রোগ্রাম লেখা যায়।
- <http://uva.onlinejudge.org> এই সাইটে নিয়মিত অনলাইন প্রোগ্রামিং প্রতিযোগিতার আয়োজন করা হয়। এ ছাড়াও অনুশীলনের জন্য প্রচুর সমস্যা দেওয়া আছে। নতুন প্রোগ্রামারদের জন্য এটি বেশ ভালো জায়গা।
- <http://ace.delos.com/usacogate> এটি যদিও আমেরিকার ইনফরমেটিক্স অলিম্পিয়াড ট্রেনিং প্রোগ্রাম, কিন্তু সাইটে যেকোনো দেশের প্রোগ্রামাররাই রেজিস্ট্রেশন করে অনুশীলন করতে পারে। তোমরা যারা প্রোগ্রামিং প্রতিযোগিতায় ভালো করতে চাও, তাদের অবশ্যই এখানে অনুশীলন করা উচিত।
- <http://www.topcoder.com/tc> এখানেও নিয়মিত অনলাইন প্রোগ্রামিং প্রতিযোগিতা অনুষ্ঠিত হয়। এখানে ভালো ফলাফল করলে আবার টাকাও দেয় (কী আনন্দ!)। এ ছাড়া এখানে অনেক ভালো টিউটোরিয়াল ও আর্টিকেল