ANSWER TO THE QUESTION NUMBER 1

টারনারি অপারেটর**:-**

টারনারি অপারেটর হল এমন একটি কন্ডিশনাল অপারেটর যেটার তিনটি অংশ থাকে। টরনারি অপারেটর ব্যবহার করে আমরা প্রোগ্রামকে ছোট করতে পারি। এবং এটি ব্যবহার করলে প্রোগ্রামের কন্ডিশনাল অংশটুকু এক লাইনে হয়ে যায়।
নিচে টানারি অপারেটর ব্যবহার করে একটি প্রোগ্রাম রচনা করা হলো

```
#include"stdio.h"
int main()
{
   int num1;
   int num2;
   scanf("%d",&num1);
   scanf("%d",&num1);
   int large=num1>num2?num1:num2;
   printf("the large is %d",large);
   return 0;
}
```

উক্ত প্রোগ্রামটিতেnum1>num2? এটি টারনারি অপারেটরের প্রথম অংশ num1এটি দ্বিতীয় অংশ num2এটি তৃতীয় অংশ যদি প্রথম অংশ সত্য হয় তাহলে large ভেরিয়েবলে মান হবে num1 আর মিথ্যা হলে num2 হবে।

লোকাল ভেরিয়েবল এবং গ্লোবাল ভেরিয়েবল**:**

নিচে লোকাল এবং গ্লোবাল ভেরিয়েবল এর মধ্যে পার্থক্য উল্লেখ করা হলো,
(১) লোকাল ভেরিয়েবল কে ফাংশনের ভিতরে ডিক্লেয়ার করা হয় কিন্তু গ্লোবাল ভেরিয়েবল কে সকল ফাংশনের বাইরে প্রোগ্রামের শুরুতে ডিক্লেয়ার করা হয়।

(২) লোকাল ভেরিয়েলের কার্যকারিতা সেই ফাংশন এর মধ্যে সীমাবদ্ধ যেখানে তাকে ডিক্লেয়ার করা হয়।
কিন্তু গ্লোবাল ভেরিয়েবল কে আমরা প্রোগ্রামের যেকোন ফাংশন এর মধ্যেই ব্যবহার করতে পারি কারণ গ্লোবাল ভেরিয়েবলের কার্যকারিতা সমগ্র প্রোগ্রামের মধ্যেই।

নিচে লোকাল এবং গ্লোবাল ভেরিয়েবল এর উদাহরণ দেওয়া হল

```
#include"stdio.h"
int x=10;
void fun_local()
{
   int i=20;
   printf("%d",i); // i is a local variable
}
int main()
{
```

```
    printf("%d ",x); // x is a global variable
    return 0;
}
```

এখানে আমরা ভেরিয়েবল x এর উপর যে কোন ফাংশন থেকেই অপারেশন চালাতে পারবো যেমন আমরা main ফাংশন থেকে একে প্রিন্ট করতে পারছি। অপরদিকে লোকাল ভেরিয়েবল i কে অন্য একটি ফাংশনে যেহেতু ডিক্লেয়ার করা হয়েছে তাই main ফাংশনে একে আমরা প্রিন্ট করতে পারছি না অথবা মেইন ফাংশন থেকে এর উপর কোন অপারেশনে চালাতে পারছি না main ফাংশনে যদি আমরা i কে প্রিন্ট করতে চাই তাহলে i কে যে ফাংশনের ডিক্লেয়ার করা হয়েছে তাকে main ফাংশনে কল করতে হবে অর্থাৎ লোকাল ভেরিয়েবলের এক্সেস সরাসরি অন্য কোন ফাংশন পাবে না।

ANSWER TO THE QUESTION NUMBER 2

নিচে এরে ব্যবহার না করে ফিবোনাক্কি নাম্বার বের করার অ্যালগরিদম লেখা হল,

```
ধাপ ১) শুরু
ধাপ ২) ভেরিয়েবল ডিক্লারেশন a, b,c,i,n;
ধাপ  ৩)ভেরিয়েবল ইনিশিয়ালাইজেশন  a=0,b=1
ধাপ ৪)scanf ফাংশন ব্যবহার করে nকে ইনপুট নিতে হবে কারণ n হলো ফিবোনাক্কি সিরিজ নাম্বার।
ধাপ ৫)লুকিং শুরু করার আগেই a এবং b কে প্রথমে  প্রিন্ট করে নিতে হবে।
ধাপ ৬)FOR i =3; i<=n;i++

      c=a+b;
      printf(" %d",c);
      a=b;
      b=c;
```

এখানে i কে ৩ থেকে শুরু করা হয়েছে কারণ ফিবোনাক্কি সিরিজের প্রথম দুটি সংখ্যা আমরা লুপিং শুরু করার আগে প্রিন্ট করে নিয়েছি।

এবং i কে এক এক করে যোগ করে n পর্যন্ত লুপটি চালানো হয়েছে।

নিচে ফিবোনাক্কি নাম্বার বের করার একটি সি প্রোগ্রাম লেখা হলো,

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

```
        a=b;
        b=c;
    }
    return 0;
}
```

ANSWER TO THE QUESTION NUMBER 3

পয়েন্টার হল এমন এক ধরনের ভেরিয়েবল যা অপর কোন ভেরিয়েবলের মেমোরি এড্রেস কে নিজের মধ্যে স্টোর করে রাখে।

Pass by value and Pass by reference

নিচে pass by value এবং pass by reference এর মধ্যকার পার্থক্য তুলে ধরা হলো,

১) main ফাংশন থেকে যখন কোন ভেরিয়েবলের value বা মান কে অপর কোনো ফাংশনে পাঠানো হয় তখন একে বলা হয় pass by value
অপরদিকে
main ফাংশন থেকে যখন কোন ভেরিয়েবলের মেমোরি অ্যাড্রেস কে অপর কোনো ফাংশনে পাঠানো হয় তখন একে বলা হয় pass by reference

২)pass by value ক্ষেত্রে ফাংশন প্যারামিটার হিসাবে ভেলু গ্রহণ করে কিন্তু pass by reference এর ক্ষেত্রে ফাংশন প্যারামিটার হিসাবে পয়েন্টার বা মেমোরি এড্রেস গ্রহণ করে
৩)pass by value এর ক্ষেত্রে কোন ফাংশন একটি ভ্যারিয়েবলের(যে ফাংশনে তাকে ডিক্লেয়ার করা হয়েছে সেটি বাদে)সম্পূর্ণ এক্সেস পায়না শুধুমাত্র তার মানটি নিয়ে কাজ করতে পারে। অপরদিকে pass by reference ক্ষেত্রে কোন ফাংশনকে বেরিয়েবলে এর সম্পূর্ণ এক্সেস দেওয়া হয়।

নিচে pass by Value and pass reference কে সি প্রোগ্রামিং এর মাধ্যমে বর্ণনা করা হলো ,
Pass by Value

```
#include<stdio.h>
void fun_sum(int a,int b,int c);
int main()
{
    int a=10,b=20;
    int c=0;
    fun_sum(a,b,c);
    printf("%d",c);
    return 0;
}
void fun_sum(int a,int b,int c)
{
    c=a+b;
}
```

এখানে আমরা main ফাংশন থেকে fun_sum ফাংশনে শুধুমাত্র a,b,c এর ভ্যালু কে পাঠিয়েছি এবং fun_sum ফাংশনের ভিতরে a+b এর মান c তে স্টোর করতে চেয়েছি কিন্তু সেটি সম্ভব হবে না আমরা যদি main ফাংশনে এসে c কে প্রিন্ট করতে চাই তাহলে দেখতে পারবো c এর মান তার পূর্বের মান 0 রয়েছে। কারণ fun_sum ফাংশন এর ওই একসেসটা নেই যে সে main ফাংশনের variable এর মান পরিবর্তন করবে কারন আমরা fun_sum ফাংশনে a,b,c ভেরিয়েবল গুলোর কপি পাঠিয়েছিলাম।

Pass by reference

```
#include<stdio.h>
void fun_sum(int* a,int* b,int* c);
int main()
{
   int a=10,b=20;
   int c=0;
   fun_sum(&a,&b,&c);
   printf("%d",c);
   return 0;
}
void fun_sum(int* a,int*  b,int*  c)
{
   *c= *a + *b;
}
```

এখানে আমরা main ফাংশন থেকে fun_sum ফাংশনে a,b,c এর   এড্রেস কে পাঠিয়েছি এবং fun_sum ফাংশনের ভিতরে a+b এর মান c তে স্টোর করতে চেয়েছি এবং সেটি সম্ভব হবে আমরা যদি main ফাংশনে এসে c কে প্রিন্ট করতে চাই তাহলে দেখতে পারবো c এর মান a+b এর মান 30 হয়েছে। কারণ fun_sum ফাংশন এর ওই একসেসটা আছ যে সে main ফাংশনের variable এর মান পরিবর্তন করবে কারন আমরা fun_sum ফাংশনে a,b,c ভেরিয়েবল গুলোর কপি না পাঠিয়ে তাদের মেমোরি এড্রেসগুলো কে পাঠিয়েছিলাম।

ANSWER TO THE QUESTION NUMBER 4

```
#include<stdio.h>
#include<string.h>
void sort(char a[],int l);
int main()
{
   char  arr[100];
   scanf("%s",arr);
   int len=strlen(arr);
   sort(arr,len);
   return 0;
}
void sort(char a[],int l)
{
   int frequen[26];
```

```
    int i;
    for(i=0; i<=25; i++)
    {
        frequen[i]=0;
    }
    for(i=0; i<=8; i++)
    {
        frequen[a[i]-97]++;
    }
    for(i=0; i<=25; i++)
    {
        if( frequen[i]>0)
        {
            for(int j=0; j<frequen[i]; j++)
            {
                printf("%c",i+97);
            }
        }
    }

}
```

ANSWER TO THE QUESTION NUMBER 5

malloc ব্যবহার করে একটি প্রোগ্রাম

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n;
    int i;
    int* ptr;
    scanf("%d",&n);
    ptr=(int*)  malloc(n*sizeof(int));
    if(ptr==NULL)
    {
        printf("soory");
    }
    else
    {
        for(i=0; i<n; i++)
        {
            scanf("%d",(ptr+i));
        }
        printf("\n");
        for(i=0; i<n; i++)
        {
```

```
            printf("%d ",*(ptr+i));
        }
    }
    free(ptr);
    return 0;
}
```

নিচে **malloc** এবং **calloc** মধ্যকার পার্থক্য দেখানো হলো

১)   malloc পূর্ণরূপ হলো memory allocation এবংcontiguous allocation

২) mallco এর Syntax হলো ptr(এখানে ptr হল একটি পয়েন্ট আর ভেরিয়েবল)=(cast-type*) malloc(n*byte-size); অপরদিকে callco এর Syntax হলো ptr(এখানে ptr হল একটি পয়েন্ট আর ভেরিয়েবল)=(cast-type*) malloc(n,byte-size);

৩) malloc ইনিসিয়ালিক গার্বেজ ভ্যালু জমা রাখে কিন্তু calloc ইনিশিয়ালি প্রত্যেকটি এলিমেন্টকে 0 করে দেয়।

৪) malloc একটি প্যারামিডের গ্রহণ করে এবং প্যারামিটারটি হল (মোট উপাদান সংখ্যা গুণন একটি ডাটা এর বাইট সাইজ) কিন্তু calloc দুইটি প্যারামিটার গ্রহণ করে প্যারামিটার দুটি হল i>মোট উপাদান সংখ্যা ii>একটি ডাটা এর বাইট সাইজ।

৫) malloc যে পরিমাণ জায়গা নির্ধারিত করে সেটিকে আলাদা আলাদা কোন ভাগে ভাগ করে না
কিন্তু যে পরিমাণ জায়গা নির্ধারণ করে সেটি তার ডাটা টাইপ অনুযায়ী আলাদা আলাদা ব্লকে ভাগ করে।

ANSWER TO THE QUESTION NUMBER 6

ফাংশন কি
ফাংশন হলো প্রোগ্রামের একটি স্বতন্ত্র অংশ যেটি আমাদের চাহিদা অনুযায়ী নির্দিষ্ট কোন কাজ করে দিতে। ফাংশন দুই প্রকার

ক)বিল্ড ইন ফাংশন
খ)ইউজার ডিফাইন্ড ফাংশন

ইউজার ডিফাইন্ড ফাংশন-এর প্রকারভে

১)আর্গুমেন্ট এবং রিটার্ন ভ্যালু সহ ফাংশন।

এ ধরনের ফাংশন আর্গুমেন্ট হিসাবে এক বা একাধিক প্যারামিটার ইউজারের কাছ থেকে নিয়ে থাকে এবং অপারেশন শেষে একটি রিটার্ন ভ্যালু যে ফাংশন থেকে তাকে কল করা হয় সেখানে পাঠায়।
উদাহরণ :-

```
#include"stdio.h"
int prime(int n);
int main()
{
    int num;
    scanf("%d",&num);
    int ans=prime(num);
    printf("ANS = %d",ans);
    return 0;
```

```
}
int prime(int n)
{
    for(int i=2; i<n; i++)
    {
        if(n%i==0)
        {
            return 0;
        }
    }
    return 1;
}
```

উপরের ফাংশনটিতে আমরা দেখতে পাচ্ছি ফাংশনটি রিটার্ন ভ্যালু হিসাবে Integer type ডাটা রিটার্ন করে এবং এটি একিট Integer type parameter গ্রহণ করে। ফাংশনটি তে যে সংখ্যাটি প্রদান করা হয় সেটি যদি মৌলিক সংখ্যা হয় তাহলে এটি মেইন ফাংশনে Integer type data 1 রিটার্ন করে অন্যথায় 0 রিটার্ন করে।

২) রিটার্ন ভ্যালু এবং আর্গুমেন্ট ছাড়া ফাংশন

এ ধরনের ফাংশন কোন প্যারামিটার গ্রহণ করেনা এবং যে ফাংশন থেকে তাকে কল করা হয় সেখানে কোন ডাটা রিটার্ন পাঠায় না। শুধু একটি নির্দিষ্ট কাজ করে চলে যায়। এ ধরনের ফাংশনের বলতে গেলে শুধু এর নামটাই থাকে।
উদাহরণ :-

```
#include"stdio.h"
void ola(void);
int main(void)
{
    for(int i=1; i<=5; i++)
    {
        ola();
    }
    return 0;
}
void ola(void)
{
    printf("HELLO\n");
    return 1;
}
```

উপরের ফাংশনটি ইউজারের কাছ থেকে কোন প্যারামিটার নেয় না এবং এর কোন রিটার্ন ভ্যালু। এর কাজ হল একে যতবার কল করা হবে ততবার সে HELLO প্রিন্ট করে চলে যাবে।

৩)রিটার্ন ভ্যালু নেই কিন্তু আর্গুমেন্ট আছে ।

এ ধরনের ফাংশনের কোন রিটার্ন ভ্যালু থাকেনা কিন্তু user এর কাছ থেকে আর্গুমেন্ট হিসাবে প্যারামিটার নেয় এবং উক্ত প্যারামিটার গুলো ব্যবহার করে একটি নির্দিষ্ট কাজ করে দেয়।
উদাহরণ :-

```
#include<stdio.h>
void fun_sum(int* a,int* b,int* c);
int main()
{
   int a=10,b=20;
   int c=0;
   fun_sum(&a,&b,&c);
   printf("%d",c);
   return 0;
}
void fun_sum(int* a,int*  b,int*  c)
{
   *c= *a + *b;
}
```

উপরের ফাংশনটি কোন ভ্যালু রিটার্ন করে না। ফাংশনটি ৩টি integer ভেরিয়েবল এর এড্রেস নেয় এবং এগুলো ব্যবহার করে সে নিজের কাজ পরিচালনা করে।

৪)Function with no arguments and a return value.
এ ধরনের ফাংশন কোন রকম আরগুমেন্ট গ্রহণ করে না কিন্তু যে ফাংশন থেকে তাকে কল করা হয় সেখানে একটি ভ্যালু রিটার্ন করে।

উদাহরণ :-

```
#include"stdio.h"
int fun(void);
int main(void)
{
   int ans=fun();
   printf("%d",ans);
   return 0;
}
int fun(void)
{
   int a=10;
   return a;
}
```

ফাংশন ব্যবহারের সুবিধা

১)ইউজার ডিফাইন্ড ফাংশন ব্যবহার করলে মেইন ফাংশন দেখতে সুন্দর হয়।

২) ইউজার ডিফাইন্ড ফাংশন ব্যবহার করলে একই কোড বারবার লিখতে হয় না যার ফলে কাজের গতি বাড়ে।

৩) ইউজর ডিফাইন্ড ফাংশন ব্যবহার করে আমরা একটি প্রোগ্রামকে বিভিন্ন অংশে ভাগ করতে পারি যার ফলে আমাদের কাজ অনেক সহজ হয়ে যায়।

ANSWER TO THE QUESTION NUMBER 7

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

    int n,m;
    scanf("%d %d",&n,&m);
    int a1[n][m];
    int a2[n][m];
    int a3[n][m];
    int i,j,k;
    int sum=0;
    printf("enter first matrix>>\n");
    for(i=0; i<n; i++)
    {
        printf("\n");
        for(j=0; j<m; j++)
        {
            scanf("%d",&a1[i][j]);
        }
    }
    printf("\n");
    printf("enter second matrix>>\n");
    for(i=0; i<n; i++)
    {
        printf("\n");
        for(j=0; j<m; j++)
        {
            scanf("%d",&a2[i][j]);
        }
    }
    printf("\n");
    for(i=0; i<n; i++)
    {
        printf("\n");
        for(j=0; j<m; j++)
        {
            a3[i][j]=a1[i][j]+a2[i][j];
        }
    }
    printf("the product matrix is >>\n");
    for(i=0; i<n; i++)
    {
        printf("\n");
        for(j=0; j<m; j++)
        {
            printf("%d ",a3[i][j]);
```

```
        }
    }
    return 0;
}
```

ANSWER TO THE QUESTION NUMBER 8

স্ট্রাকচার এর সংজ্ঞা **:** সি প্রোগ্রামিং এ স্ট্রাকচার হল একটি কাস্টম বা ইউজার ডিফাইন্ড ডাটা টাইপ। স্ট্রাকচার ব্যবহার করে বিভিন্ন টাইপের ডাটা একসাথে ব্যবহার করে একটি নতুন ডাটা টাইপ তৈরি করা যায়।

স্ট্রাকচারের ব্যবহার

ক) যখন আমাদের একাধিক কিন্তু ভিন্ন ভিন্ন টাইপের ডাটা নিয়ে কাজ করতে হয় তখন আমরা স্ট্রাকচার ব্যবহার করি। যেমন কোন স্কুলের শিক্ষার্থীদের নাম, রোল এবং মার্ক ইত্যাদি নিয়ে কাজ করতে হয়।

খ) Array এর সীমাবদ্ধতা দূর করতে এরেতে আমরা শুধুমাত্র একটি টাইপের ডাটা নিয়ে কাজ করতে পারি
কিন্তু স্ট্রাকচার আমাদের একাধিক টাইপের ডাটা নিয়ে কাজ করার স্বাধীনতা দেয়।

প) কোন প্রোগ্রামে স্ট্রাকচার ব্যবহার করলে প্রোগ্রামটি অনেক অরগানাইজড হয়।

স্ট্রাকচারের মেম্বার এক্সেস করা

নিয়ম-১**:-** ডট (.)অপারেটর ব্যবহার করে।
Syntax হলো : structure variable name. structure member
উদাহরণ **:**

```
#include"stdio.h"
struct student
{
    float mark;
    int roll;
    char name[50];
};
int main()
{
    struct student s1;
    printf("enter marks:");
    scanf("%f",&s1.mark);
    printf("marks: %.2f",s1.mark);
    return 0;
```

নিয়ম২**:-** পয়েন্টার ব্যবহার করে,

উদাহরণ:

```
#include"stdio.h"
struct student
{
   float mark;
   int roll;
   char name[50];
};
int main()
{
   struct student s1;
   struct student* sp;
   sp=&s1;
   printf("enter marks:");
   scanf("%f",&s1.mark);
   printf("marks: %.2f",sp->mark);
   return 0;
}
```

PART C:

```
#include"stdio.h"
struct student
{
   float mark;
   int roll;
   char name[50];
};
int main()
{
   struct student s1;
   printf("enter information>>\n");

   printf("enter name:");
   scanf("%s",s1.name);

   printf("enter roll number:");
   scanf("%d",&s1.roll);

   printf("enter marks:");
   scanf("%f",&s1.mark);
```

```
    printf("displaying information>>\n");
    printf("name: %s\n",s1.name);
    printf("roll number: %d\n",s1.roll);
    printf("marks: %.2f",s1.mark);
    return 0;
}
```

ANSWER TO THE QUESTION NUMBER 9

সি প্রোগ্রামিং এ পাঁচ রকমের ইরোর গুলো হল :

১) Syntax error. প্রোগ্রাম রচনার সময় আমরা যখন নির্দিষ্ট কিছু symbol, keyword এবং টোকেন লিখতে ভুলে যাই যেগুলো না লিখলে কোন স্টেটমেন্ট অসম্পূর্ণ থেকে যায়, তখন এ ধরনের ভুলকে বলা হয় syntax error। syntax ইরর গুলো সাধারণত প্রোগ্রাম লেখার সময়ই কম্পাইলার আমাদের বলে দিতে পারে।

উদাহরণ :

```
#include<stdio.h>
int main()
{
    int a=10;
    int b=5;
    int c=a/b;
    printf("%d",c)
    return 0;
}
```

প্রোগ্রামটিতে আমরা c এর মান প্রিন্ট করার সময় সেমিকোলন (;) দেইনি এটি একটি syntax error

২) Logical error. যখন আমারা ভুল ভাবে প্রোগ্রাম করে থাকি অর্থাৎ আমাদের যে কাজ করা দরকার আমরা সেটি না করে অন্য কোনো অপারেশন করে থাকি অথবা আমাদের প্রোগ্রামটির যুক্তি যদি ঠিক না থাকে তাহলে একে আমরা লজিক্যাল ইরোর বলি। যেমন, আমাদের এমন একটি প্রোগ্রাম করা দরকার যেখানে আমরা দুটি সংখ্যা ইনপুট নিয়ে এদের যোগফল আউটপুটে প্রদর্শন করব কিন্তু আমরা ভুলক্রমে যোগের বদলে যদি বিয়োগ বা অন্য কোন অপারেশন করে থাকি অন্য কোন এটি লজিক্যাল error হিসেবে গণ্য হবে। লজিক্যাল error গুলো সাধারণত কম্পিউটার বুঝতে পারে না এটি আমাদের নিজেদেরকে হ্যান্ডেল করতে হয়।

উদাহরণ :

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

```
{
    int a=10;
    int b=5;
    int c=a/b;
    printf("%d + %d = %d",a,b,c);
    return 0;
}
```

উক্ত প্রোগ্রামটিতে আমাদের উদ্দেশ্য হল a এবং b এর যোগফল আউটপুটে প্রদর্শন করা। কিন্তু আমরা ভুলক্রমে a এবং b এর মাঝে ভাগ অপারেশন টি করেছি যার কারণে এটি একটি লজিক্যাল এরর হয়ে গেছে।

৩) Runtime error . যখন কম্পিউটার প্রোগ্রামের মাধ্যমে আমরা এমন কিছু কাজ করতে চাই যেটি আমাদের কাছে অসঙ্কায়িত অর্থাৎ যার কোন উত্তর বা সমাধান আমাদের কাছে নেই এ ধরনের প্রোগ্রাম গুলো রান করার সময় কম্পিউটার runtime error দিয়ে থাকে

উদাহরণ :

```
#include<stdio.h>
int main()
{
    int a=10;
    int b=0;
    int c=a/b;
    printf("%d",c);
    return 0;
}
```

উক্ত প্রোগ্রামটি বিল্ড করে আমরা যখন রান করতে যাব তখন কম্পিউটার আমাদের কোন রকম ফল প্রদর্শন করবে না কারণ আমরা যদি খেয়াল করি তাহলে দেখব প্রোগ্রামটিতে একটি পূর্ণ সংখ্যাকে শুন্য ধারা ভাগ করার চেষ্টা করা হয়েছে এবং এটি অসঙ্কায়িত। এই প্রোগ্রামটিতে কম্পিউটার যে ধরনের error প্রদর্শন করবে সেটি হবে error

৪) Linker error. সি প্রোগ্রামিং করার সময় আমরা যখন প্রি ডিফাইন্ড ফাংশন গুলো ব্যবহার করার সময় কোন রকম ভুল করে থাকি এদের নাম লেখার সময় ভুল অন্য কোন ভুল হলে কম্পাইলার হেডার ফাইল এর সাথে ওই ফাংশন গুলোকে লিংক করতে পারেনা তখন এ ধরনের এররকে বলা হয় লিঙ্কার এরর।

উদাহরণ :

```
#include"stdio.h"
int main()
{
    printf("hello");
    Printf("world")
    return 0;
}
```

উক্ত ফাংশনটিতে world প্রিন্ট করার সময় printf ফাংশনের প্রথম অক্ষর p কে বড় হাতের অক্ষরে লেখা হয়েছে কম্পাইলার হেডার ফাইল এর সাথে একে লিংক করতে পারেনি।

৫) Semantic error:

উদাহরণ :

```
#include<stdio.h>
int main()
{
    int a=10;
    int b=5;
    int a+b=c;
    printf("%d + %d = %d",a,b,c);
    return 0;
}
```

উক্ত ফাংশনটিতে আমরা a এবং b এর মান যোগ করে
c তে স্টোর করতে চেয়েছি কিন্তু সি প্রোগ্রামিং এর ব্যাকরণগত নিয়ম অনুযায়ী সমান চিহ্ন (=) এর ডান পাশে যে ভ্যালু বা ভেরিয়েবল থাকে তার মান বাম পাশের ভেরিয়েবলে স্টোর হয়। কিন্তু এখানে আমরা বাম পাশের মানকে ডান পাশের ভেরিয়েবলে স্টোর করতে চেয়েছি যেটি একটি ভুল পন্থা। এ ধরনের ইররকেই বলা হয় semantic error

ANSWER TO THE QUESTION NUMBER 10

A)
C হলো একটি মিড লেভেল প্রোগ্রামিং ল্যাঙ্গুয়েজ।
বর্তমানে যত আধুনিক প্রোগ্রামিং ল্যাঙ্গুয়েজ রয়েছে যেমন C++,Java Python,Go এসব ভাষার মূল ভিত্তি হলো সি ল্যাঙ্গুয়েজ। এসব ভাষার বেশিরভাগ কনসেপ্ট গুলো C ভাষা থেকে নেওয়া যেমন variable,array,function,loop,condition আরো অনেক কিছু। সিস্টেম অ্যাপ্লিকেশন বা অপারেটিং সিস্টেমগুলো তৈরি করতে C ল্যাংগুয়েজ ব্যবহার করা হয়। C এর এরকম আরো অনেক গুণের কারণে একে বলা হয় "mother of all programming language "

B)
i) প্রোগ্রামিং ভাষায় লিখিত স্টেটমেন্টের সমষ্টিকে একত্রে সোর্স কোড বলে। কোর্স কোড কে কম্পাইল বা ইন্টারপ্রিন্ট করে এক্সিকিউটিবল ফাইলে পরিণত করা হয়।

ii) source code কে   যখন কম্পাইলার বা ইন্টারপ্রিন্টার translate করে মেশিন ল্যাঙ্গুয়েজ (0,1) -এ পরিণত করে তখন একে বলা হয় অবজেক্ট কোড।আর যে ফাইলে অবজেক্ট কোড থাকে তাকে বলা হয় অবজেক্ট ফাইল।

iii). exe file বলতে এক্সিকিউতটবল ফাইলকে বোঝানো হয়েছে।
executable Code CPU এর জন্য মেশিন কোড ইন্সট্রাকশন। যে ফাইলে এক্সিকিউটেবল কোড থাকে তাকে বলা হয় এক্সিকিউটেবল ফাইল ।
cpu সরাসরি এক্সিকিউটেবল ফাইলকে এক্সিকিউট করতে পারে এবং তদানুযায়ী কাজ করতে পারে।

C)
আমি যখন একটি c প্রোগ্রাম রান করি তখন আমার ডিভাইজে মোট তিনটি ফাইল তৈরি হয়।

i)source code file
ii)object code file
iii)executable file /.exe file

D)

যে নির্দিষ্ট কন্ডিশন এর ওপর ভিত্তি করে কোন রিকার্সিব ফাংশন নিজেই নিজেকে কল করা বন্ধ করে তাকে বলা হয় ঐ রিকার্সি ফাংশন এর base case বা terminate condition ।

Recursion এর **real life example** হলো,

আমি একটা ফাংশন আমার কাজ হল আমাকে কেউ কোশ্চেন পেপার দিয়ে কল করলে আমি সেটা সল্ভ করে তার কাছে রিটার্ন করি।
এখন phitron আমাকে ফাইনাল এক্সামের কোশ্চেন দিয়ে কল করেছে যেখানে এক থেকে দশ পর্যন্ত মোট দশটি কোশ্চেন রয়েছে অর্থাৎ এখানে আমার Base case হল 10 number question.
এখন আমি এক নাম্বার কোশ্চেন সলভ করব এবং যদি দেখি আমার base case ১০ নাম্বার কোশ্চেন কে আমি ছুঁতে পারিনি তখন আবার দুই নাম্বার কোশ্চেন দিয়ে আমি নিজে নিজেকে কল করব। এবং যখন আমি পুরো দশটি কোশ্চেন সলভ করব তখন নিজে নিজেকে কল করা বন্ধ করব। এভাবে পুরো দশটি কোশ্চেন সলভ করে, আমি phitron কে কোশ্চেন পেপার টি সমাধান রিটার্ন করব।

E)

মেমরি অপচয় রোধ করার জন্য আমরা large data type কে স্ট্রাকচারের প্রথমে ডিক্লেয়ার করি।
আমরা যদি একটি স্ট্রাকচারে Integer type data কে double data এর আগে ডিক্লেয়ার করি তাহলে স্ট্রাকচারটি যে পরিমাণ মেমোরি নিবে তার থেকে কম মেমোরি নিবে যদি আমরা double type data কে integer type data এর আগে ডিক্লেয়ার করি।