Functions:

If block of code that will perform only when it is called.

Syntax:

```
return type   function name ( para
{
    return;
}
```

No return → 3

Concept: 1+2

Type of functions: C-program → @ Hello-in

Syntax:

① void fun()
② int fun()
③ int fun(int a, int b)
④ int fun(int a)
⑤ char fun()
⑥ char fun(char c)

void fun() → void
int fun() → int
int fun(int a, int b) → int, int
int fun(int a) → int
char fun() → char
char fun(char c) → char

Dynamic Memory Allocation: Allocation → reserve memory for data

int a=10
int arr[5]={1,2,3,4,5}

reserve → a, 10
reserve → arr, 1 2 3 4 5

DMA

Keywords: ① malloc() ② calloc() ③ free()

① malloc() → it can allocate single block

② calloc() → Every values will reset. That's why memory for arrays

③ realloc() → previous memory for calloc & malloc to reallocate

free(arr) → realloc

malloc() → calloc() → realloc → free(arr)

Other different functions:

① return type with parameter
② return type with parameter
③ void return type with parameter
④ void return type with no parameter

① return type with parameter

```
int add(int a, int b){
    return a+b
}
```

② return type with no parameter

```
int add (){
    int a = 5, b=10;
    return a+b;
}
```

③ no return type with parameter

```
void add(int a, int b){
    int sum=a+b;
    printf("%d", sum)
}
int main(){
    int a,b;
    scanf("%d%d",&a,&b);
    add(a,b);
    return 0;
}
```

④ no return type with no parameter

```
void add(){
    int a=5, b=10;
    int sum=a+b;
    printf("%d", sum)
}
int main(){
    add();
    return 0;
}
```

Calculator using switch case:

```
#include <stdio.h>
int add(int a,int b);
int sub(int a,int b);
int mul(int a,int b);
int div(int a,int b);
int main() {
    int a,b,choice;
    float result;

printf("------CALCULATOR------\n");
printf("1. Addition");
printf("2. Subtraction");
printf("3. Multiplication");
printf("4. Division");
printf("Enter your choice: ");
scanf("%d",&choice);
printf("Enter the value of a and b: ");
scanf("%d%d",&a,&b);
switch(choice){
    case 1:
    result=add(a,b);
    break;
    case 2:
    result=sub(a,b);
    break;
    case 3:
    result=mul(a,b);
    break;
    case 4:
    if(b==0){
        result=divide(a,b);
        break;
    }
    printf("Expecting error");
    return 0;
    default:
    printf("Invalid choice\n");
    return 0;
}
printf("Result=%.2f",result);
return 0;
}
```

```
int add(int a,int b){
    return a+b;
}
int sub(int a,int b){
    return a-b;
}
int mul(int a,int b){
    return a*b;
}
float divide(int a,int b){
    return (float)a/b;
}
int main() {
    int a,b,choice;
    float result;

printf("------CALCULATOR------\n");
printf("1. Addition");
printf("2. Subtraction");
printf("3. Multiplication");
printf("4. Division");
printf("Enter your choice: ");
scanf("%d",&choice);
printf("Enter the value of a and b: ");
scanf("%d%d",&a,&b);
switch(choice){
    case 1:
    result=add(a,b);
    break;
    case 2:
    result=sub(a,b);
    break;
    case 3:
    result=mul(a,b);
    break;
    case 4:
    if(b==0){
        result=divide(a,b);
        break;
    }
    printf("Expecting error");
    return 0;
    default:
    printf("Invalid choice\n");
    return 0;
}
printf("Result=%.2f",result);
return 0;
}
```

Palindrome number:

using functions:

n=121

```
do{n=n
    while(n>0){
        rem = n%
        rev = rev*
        n=n/
    }
```

Recursion:

→ calling the function itself

Condition is not satisfied

Syntax:

```
return type fun_name(para
{
    ① base condition
    ② recursive condition
}
int fact(int n){
    if (n==0 || n==1)
        return n;
    return n*fact(n-1);
}
```

n=5

```
int fact (int n){
    if (n==0 || n==1) return 1;
    else
        return n*fact(n-1);
}
```

5*fact(4)

5*4*fact(3)

5*4*3*fact(2)

5*4*3*2*fact(1)

5*4*3*2*1

Types of Recursion:

① Direct Recursion
② Indirect Recursion
③ Tail Recursion → int fact(int n){ return n*fact(n-1); }
④ Non-Tail || Head Recursion → int fact(int n){ int x=n*fact(n-1); return x; }
⑤ Tree Recursion → int fibo(int n){ if (n==0 || n==1) return n; int x=n-1; return fibo(n-1)+fibo(n-2); }
⑥ Indirect Recursion → f() → g() → f()
⑦ Mutual Recursion → f() ⇄ g()

Any two out of (f)

0 1 1 2 3 5 8 13

int fibo(int n){
if (n==0 || n==1) return n;
int x=n-1;
return fibo(n-1)+fibo(n-2);
}

fibo(3)

Leaderboard:

Climbing stairs

n=5

min_step=1

max_step=2

Input: n=123

using Recursion you ha

1+2+3=