Day_3 C-Programming

(RECAP OF PREVIOUS DAY)

Sub Programming, function, types of functions, passing parameters to functions: call by value Recursion: Definition, Types of recursive functions, Tower of Hanoi problem, scope of variable, local and global variables, Nesting of Scope Storage classes: Auto, Register, Static and Extern,

Modular programming: Dividing the program in to sub programs (modules/function) to achieve the given task is modular approach. More generic functions definition gives the ability to re-use the functions, such as built-in library functions.

Function (Modules): A function is a block of statements to perform a specific task. Functions are the subprograms. There are two types of functions in C programming.

- **Library Functions (Pre-Defined Functions/ Built-in Functions)**
- **User Defined Functions (Programmer Defined Functions)**

Library Functions: The functions that are defined in C library are known as library functions. Examples are- *printf, scanf, getch, strlen.*

User Defined Functions: The functions that are defined by the programmer are known as user defined function.

There are following three components associated with functions:

- Function Declaration (Function Prototype)
- Function Definition
- Function Call

Function Declaration:

Syntax: *return-type function-name(parameter-list);*

Function Definition:

```
Syntax: return-type function-name(parameter-list)
        {
        Statements;
        }
```

Function Call:

Syntax: *function-name(parameter-list);*

Advantages of using multiple functions:

(i) Functions reduce the length and complexity of programs.

(ii) Functions are useful when the problem is very complex.

(iii) It is easy to find the errors in functions.

(iv) Functions can be reused in programs.

Actual Parameters (Arguments): The parameters that are passed from the calling function to the called function are known as actual parameters.

Formal Parameters (Arguments): The parameters that hold the value of actual parameters in function definition are known as formal parameters.

Types of functions: There are following types of functions.

(i). *Function with arguments and with return value*
(ii). *Function with arguments and without return value*
(iii).*Function without argument and with return value*
(iv).*Function without argument and with return value*

(i) Function with arguments and with return value

```
#include<stdio.h>
#include<conio.h>
int sum(int, int);
void main()
{
int a,b,s;
printf(“Eneter two Numbers”);
scanf(“%d%d”,&a,&b);
s=sum(a,b);
printf(“Addition=%d”,s);
getch();
}
int sum(int x, int y)
{
int z;
z=x+y;
return z;
}
```

(ii) Function with arguments and without return value

```
#include<stdio.h>
#include<conio.h>
void sum(int, int);
void main()
{
int a,b;
printf(“Eneter two Numbers”);
scanf(“%d%d”,&a,&b);
sum(a,b);
getch();
}
void sum(int x, int y)
{
int z;
z=x+y;
printf(“Addition=%d”,z);
}
```

(iii) Function without argument and with return value

```
#include<stdio.h>
#include<conio.h>
int sum();
void main()
{
int s;
s=sum();
printf(“Addition=%d”,s);
getch();
}
int sum()
{
int x,y,z;
printf(“Enter Two Numbers”);
scanf(“%d%d”,&x,&y);
z=x+y;
return z;
}
```

(iv) Function without argument and with return value

```
#include<stdio.h>
#include<conio.h>
void sum();
void main()
{
sum();
getch();
}
void sum()
{
int x,y,z;
printf(“Enter Two Numbers”);
scanf(“%d%d”,&x,&y);
z=x+y;
printf(“Addition=%d”,z);
}
```

✓ **Write a program to calculate the factorial of a number using function**

```
#include<stdio.h>
#include<conio.h>
int fact(int);
void main()
{
int n,f;
printf(“Enter a Number”);
scanf(“%d”,&n);
f=fact(n);
printf(“Factorial=%d”,f);
getch();
}
int fact(int num)
{
```

```
int i,x=1;
for(i=1;i<=num;i++)
x=x*i;
return x;
}
```

Types of Function Call (Parameter passing Mechanism or Parameter Passing Methods):

There are two types of function call

- **(i) Call by value (Pass by value)**
- **(ii) Call by reference (Pass by reference)**

Call by value (Pass by value): In this method the value of the variable is passed from the calling function to the called function.

```
#include<stdio.h>
#include<conio.h>
void swap(int, int);
void main()
{
int a=10,b=20;
swap(a,b);
printf("The value of Actual Arguments: a=%d, b=%d",a,b);
getch();
}

void swap(int x, int y)
{
int t;
t=x;
x=y;
y=t;
printf("The value of Formal Arguments: x=%d, y=%d",x,y);
}
```

Call by Reference (Pass by Reference): In this method the address of the variable is passed from the calling function to the called function.

```
#include<stdio.h>
#include<conio.h>
void swap(int *, int *);
void main()
{
int a=10,b=20;
swap(&a,&b);
printf("The value of Actual Arguments: a=%d, b=%d",a,b);
getch();
}
void swap(int *x, int *y)
{
int t;
t=*x;
*x=*y;
*y=t;
printf("The value of Formal Arguments: x=%d, y=%d",*x,*y);
}
```

Difference between Call by Value and Call by Reference

Call by Value	Call by Reference

1. In this method, the value of variable is passed from the calling function to the called function.
2. Any change in formal parameters will not reflect in actual parameters.
3. This method is slow.
Example:
#include<stdio.h>
#include<conio.h>
void swap(int,int);
void main()
{
int a,b;
printf("Enter two Number");
scanf("%d%d",&a,&b);
swap(a,b);
printf("Value of actual parameters=%d,%d",a,b);
getch();
}
void swap(int x,int y)
{
int t;
t=x;
x=y;
y=t;
printf("Value of formal parameters=%d,%d",x,y);
}
	1. In this method, the address of variable is passed from the calling function to the called function.
	2. Any change in formal parameters will reflect in actual parameters.
	3. This method is fast.
	Example:
	#include<stdio.h>
	#include<conio.h>
	void swap(int*,int*);
	void main()
	{
	int a,b;
	printf("Enter two Number");
	scanf("%d%d",&a,&b);
	swap(&a,&b);
	printf("Value of actual parameters=%d,%d",a,b);
	getch();
	}
	void swap(int *x,int *y)
	{
	int t;
	t=*x;
	*x=*y;
	*y=t;
	printf("Value of formal parameters=%d,%d",*x,*y);
	}

Recursion

Definition

Recursion is a process where a function calls itself directly or indirectly to solve a larger problem by breaking it into smaller sub-problems.

Types of Recursive Functions

1. **Direct Recursion: A function directly calls itself.**

Example:

```
#include <stdio.h>

void printNumbers(int n) {
  if (n > 0) {
        printf("%d\n", n);
        printNumbers(n - 1); // Function calls itself
    }
}

int main() {
    printNumbers(5);
    return 0;
}
```

2. **Indirect Recursion:** A function calls another function, which in turn calls the first function.

Example:

```
#include <stdio.h>

void functionA(int n);
void functionB(int n);

void functionA(int n) {
    if (n > 0) {
        printf("%d\n", n);
        functionB(n - 1);
    }
}

void functionB(int n) {
    if (n > 0) {
        printf("%d\n", n);
        functionA(n - 1);
    }
}

int main() {
    functionA(5);
    return 0;
}
```

✓ **Write a program to calculate the factorial of a number using recursion**

```
#include<stdio.h>
#include<conio.h>
int fact(int);
void main()
{
int n,f;
printf(“Enter a Number”);
scanf(“%d”,&n);
f=fact(n);
printf(“Factorial=%d”,f);
getch();
}
int fact(int x)
{
if(x==0)
return (1);
else
return (x*fact(x-1));
}
```

✓ **Write a program to display the Fibonacci series using recursion.**

```
#include<stdio.h>
#include<conio.h>
int rec(int);
void main()
{
int i,n;
printf(“Enter the Limit”);
scanf(“%d”,&n);
for(i=0;i<n;i++)
{
printf(“%d\t”,rec(i));
}
getch();
}
int rec(int x)
{
if(x==0)
return (0);
else if(x==1)
return (1);
else
return (rec(x-1)+rec(x-2));
}
```

Tower of Hanoi Problem

The Tower of Hanoi is a classic problem that demonstrates recursion. The problem involves moving disks from one rod to another, following these rules:

1. Only one disk can be moved at a time.
2. A larger disk cannot be placed on top of a smaller disk.
3. Disks can only be moved between three rods.

Solution Using Recursion:

```
#include <stdio.h>

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
  if (n == 1) {
    printf("Move disk 1 from %c to %c\n", from_rod, to_rod);
    return;
  }
```

```
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    printf("Move disk %d from %c to %c\n", n, from_rod, to_rod);
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}

int main() {
    int n = 3; // Number of disks
    towerOfHanoi(n, 'A', 'C', 'B'); // A, B, and C are the rod names
    return 0;
}
```

Output for n = 3:

```
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

---

2. Scope of Variables

Local Variables

- Variables declared inside a function are local to that function.
- They can only be accessed within the function where they are defined.

Example:

```
#include <stdio.h>

void test() {
    int localVar = 10;
    printf("Local variable: %d\n", localVar);
}

int main() {
    test();
    // printf("%d", localVar); // Error: localVar is not accessible here
    return 0;
}
```

Global Variables

- Variables declared outside all functions are global.
- They can be accessed by any function in the program.

Example:

```
#include <stdio.h>

int globalVar = 20; // Global variable

void test() {
    printf("Global variable: %d\n", globalVar);
}

int main() {
    test();
    printf("Global variable in main: %d\n", globalVar);
    return 0;
}
```

Nesting of Scope

- Inner blocks can access variables of outer blocks.
- Inner variables with the same name as outer variables will shadow the outer variables**.**

Example:

```
#include <stdio.h>

int main() {
    int x = 10;

    {
        int x = 20; // Shadows the outer x
        printf("Inner block x: %d\n", x);
    }

    printf("Outer block x: %d\n", x);
    return 0;
}
```

Output:

```
Inner block x: 20
Outer block x: 10
```

---

3. Storage Classes

Storage classes in C define the scope, lifetime, and visibility of variables.

In C, variables have three important properties:

1. **Scope:** Where the variable is accessible in the program.
2. **Lifetime:** How long the variable retains its value in memory.
3. **Visibility:** Which part of the program can see (use) the variable.

Types of Storage Classes

1. **Auto**
    - **Default storage class for local variables.**
    - **Scope: Local to the block in which it is defined (default for all local variables).**
    - **Lifetime: Limited to the execution of the block where it is defined.**
    - **Visibility: Not visible outside the block**

Example:

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

```
void test() {
    auto int a = 10; // Auto is implicit
    printf("Auto variable: %d\n", a);
}

int main() {
    test();
    return 0;
}
```

2. **Register**
    - **Scope: Local to the block in which it is defined.**
    - **Lifetime: Limited to the execution of the block where it is defined.**
    - **Visibility: Not visible outside the block.**
    - **Special Feature: Stored in CPU registers (if available) for faster access. Address-of operator (&) cannot be used.**

Example:

```
#include <stdio.h>

int main() {
    register int x = 10;
    printf("Register variable: %d\n", x);
    return 0;
}
```

3. **Static**
    - **Retains its value between function calls.**
    - **Scope: Local to the block in which it is defined.**
    - **Lifetime: Retains its value between function calls (entire program execution).**
    - **Visibility: Not visible outside the block (local static variable).**

Example:

```
#include <stdio.h>

void test() {
    static int count = 0;
    count++;
    printf("Static variable count: %d\n", count);
}

int main() {
    test();
    test();
    test();
    return 0;
}
```

Output:

```
Static variable count: 1
Static variable count: 2
Static variable count: 3
```

4. **Extern**
    - **Used to declare a global variable in another file.**
    - **Scope: Global across all files where it is declared.**
    - **Lifetime: Entire program.**
    - **Visibility: Visible in all files that declare it using the extern keyword.**

Example (Two files):

File1.c:

```
#include <stdio.h>

extern int globalVar;

void display() {
   printf("Extern variable: %d\n", globalVar);
}
```

File2.c:

```
#include <stdio.h>

int globalVar = 50;

void display();

int main() {
   display();
   return 0;
}
```

Practice Problems

1. **Write a program to calculate the factorial of a number using recursion.**
2. **Implement the Tower of Hanoi problem for n disks.**
3. **Demonstrate the use of local and global variables in a program.**
4. **Write a program to demonstrate the difference between auto and static storage classes.**
5. **Implement a program to calculate the Fibonacci series using recursion.**
6. **Demonstrate the use of extern variables across two files.**
7. **Write a program to calculate the sum of digits of a number using recursion.**
8. **Demonstrate the use of register variables in a program.**
9. **Implement nested scopes and show variable shadowing.**
10. **Write a program to reverse a string using recursion.**
11. **Write a program to find out G.C.D. of two numbers using recursion.**