C Programming

Functions Handouts / Class Notes

Dr. Suresh Mudunuri, SRKR Engineering College

UNIT 5 Syllabus

UNIT-V (10 Hrs)	Functions: Designing, Structured Programs, Function in C, User Defined Functions, Inter Function Communication, Standard Functions, Passing Array to Functions, Passing Pointers to Functions, Recursion Text Input / Output: Files, Streams, Standard Library Input / Output Functions, Formatting Input / Output Functions, Character Input / Output Functions Binary Input / Output: Text versus Binary Streams, Standard Library, Functions for Files, Converting File Type.

Modularizing and Reusing of code through Functions

Calculation of area of Circle is separated into a separate module from Calculation of area of Ring and the same module can be reused for multiple times.

```
/* program to find area of a ring
*/
#include<stdio.h>
int   main()
{
    float a1,a2,a,r1,r2;
    printf("Enter the radius : ");
    scanf("%f",&r1);
    a1 = 3.14*r1*r1;
    printf("Enter the radius : ");
    scanf("%f",&r2);
    a2 = 3.14*r2*r2;
    a = a1- a2;
    printf("Area of Ring : %.3f\n",
a);
}
```

Repeated & Reusable blocks of code

```
/* program to find area of a ring */
#include<stdio.h>
float area();
int   main()
{
   float a1,a2,a;
   a1 = area();
   a2 = area();
   a = a1- a2;
   printf("Area of Ring : %.3f\n", a);
}
float area()
{
   float r;
   printf("Enter the radius : ");
   scanf("%f", &r);
   return (3.14*r*r);
}
```

Function Declaration

Function Calls

Function Definition

Categories of Functions

```
/* using different functions */
int main()
{
    float radius, area;
    printMyLine();
    printf("\n\tUsage of functions\n
");
    printYourLine('-',35);
    radius = readRadius();
    area = calcArea ( radius );
    printf("Area of Circle = %f", area
);
}
```

```
void  printMyLine()
{
      int i;
      for(i=1; i<=35;i++) printf("%c",  '-');
      printf("\n");
}
```

Function with No parameters and No return value

```
void printYourLine(char ch, int n)
{

       int i;
      for(i=1;  i<=n ;i++) printf("%c",  ch);
      printf("\n");
}
```

Function with parameters and No return value

```
float  readRadius()
{
      float r;
      printf("Enter the radius :  ");
      scanf("%f", &r);
      return  ( r );
}
```

Function with return value & No parameters

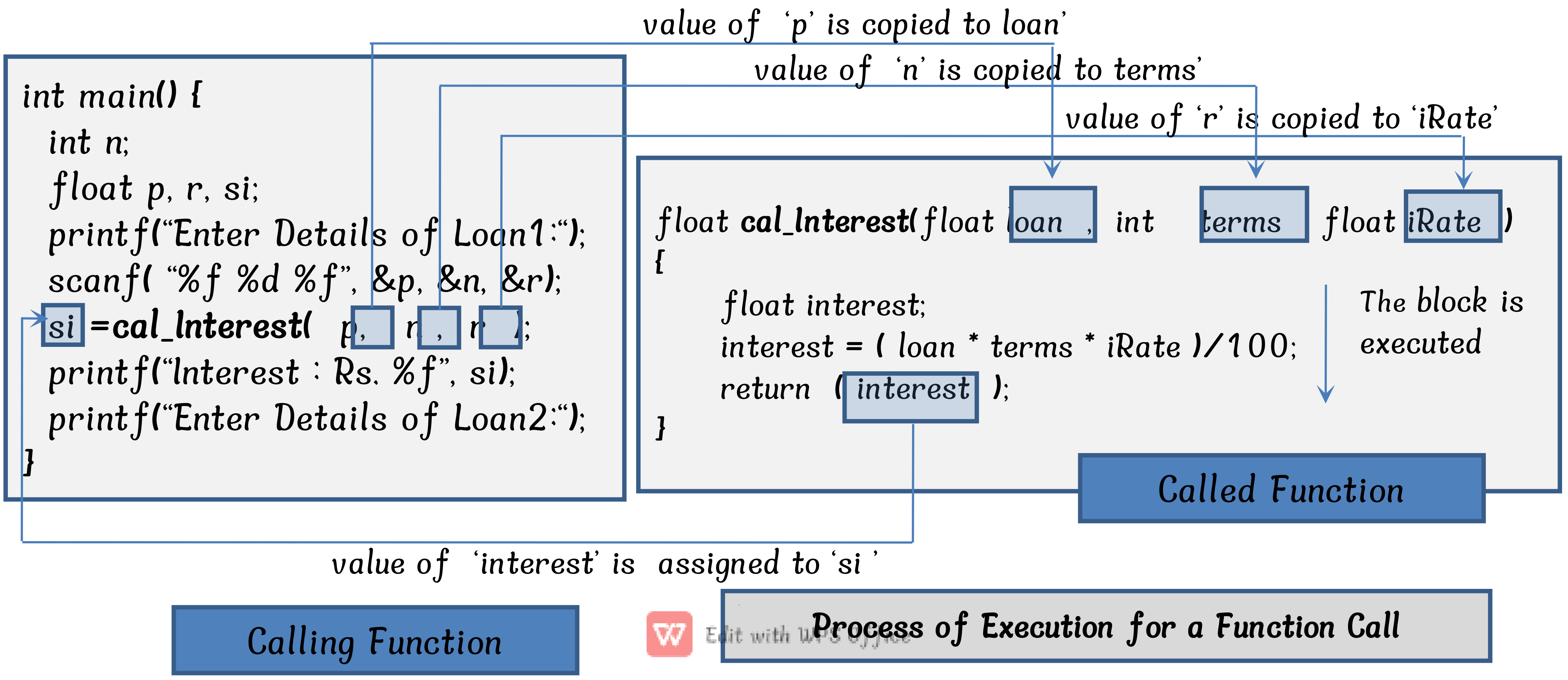
```
float  calcArea(float r)
{
      float  a;
      a = 3.14 * r * r ;
      return  ( a ) ;
}
```

Function with return value and parameters

Note: 'void' means "Containing nothing"

A **Function** is an **independent, reusable module** of statements, that specified by a name. This module (sub program) can be called by it's name to do a specific task. We can call the function, for any number of times and from anywhere in the program. The purpose of a function is to receive zero or more pieces of data, operate on them, and return at most one piece of data.

A **Called Function** receives control from a **Calling Function**. When the called function completes its task, it returns control to the calling function. It may or may not return a value to the caller. The function main() is called by the operating system; main() calls other functions. When main() is complete, control returns to the operating system.



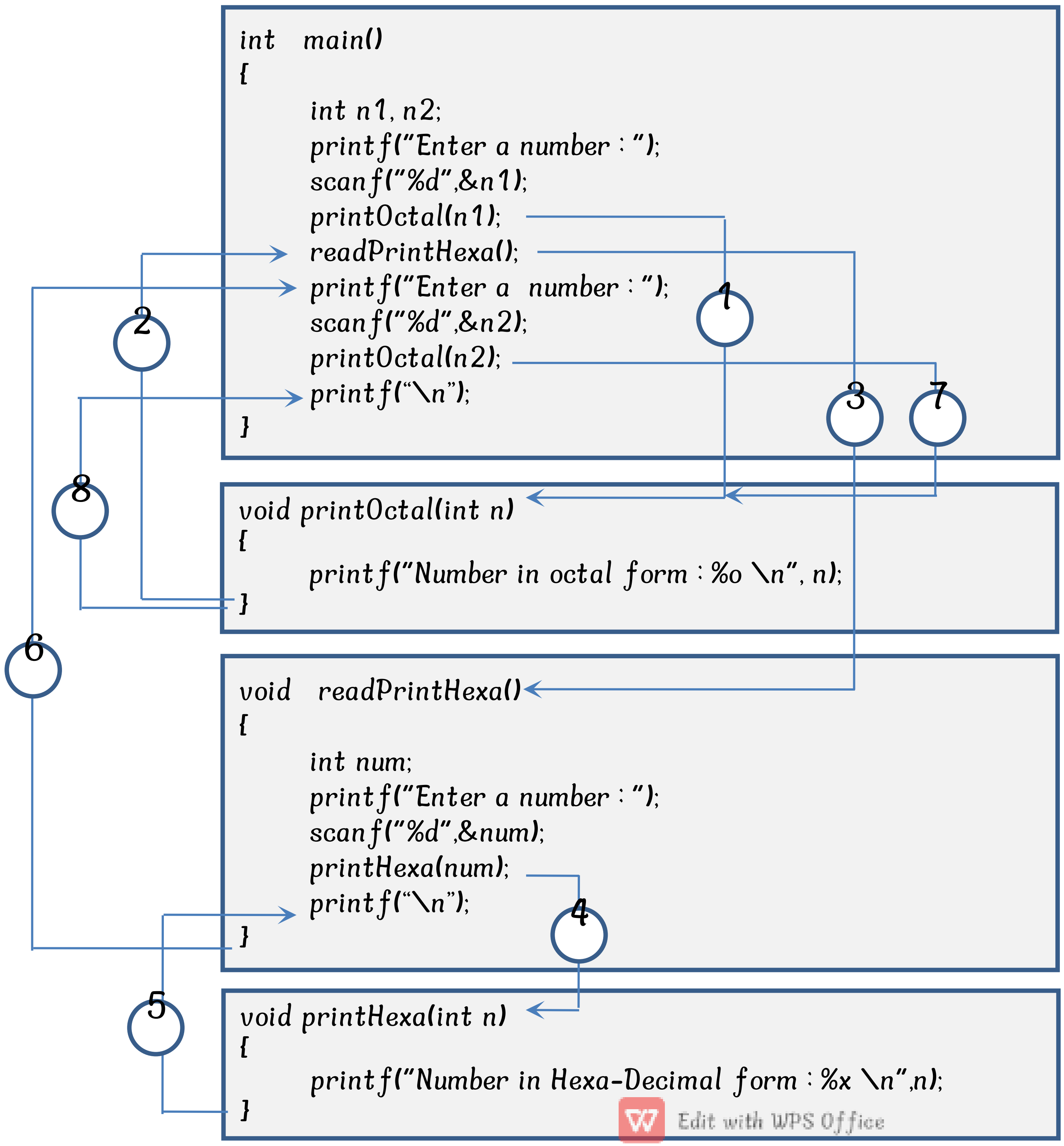
value of 'p' is copied to loan'
value of 'n' is copied to terms'
value of 'r' is copied to 'iRate'
The block is executed
value of 'interest' is assigned to 'si '
Called Function
Calling Function

```
int main() {
  int n;
  float p, r, si;
  printf("Enter Details of Loan1:");
  scanf( "%f %d %f", &p, &n, &r);
  si =cal_Interest(  p,  n ,  r  );
  printf("Interest : Rs. %f", si);
  printf("Enter Details of Loan2:");
}
```

```
float cal_Interest(float loan , int  terms  float iRate )
{
     float interest;
     interest = ( loan * terms * iRate )/100;
     return  ( interest  );
}
```

Process of Execution for a Function Call

Flow of Control in Multi-Function Program



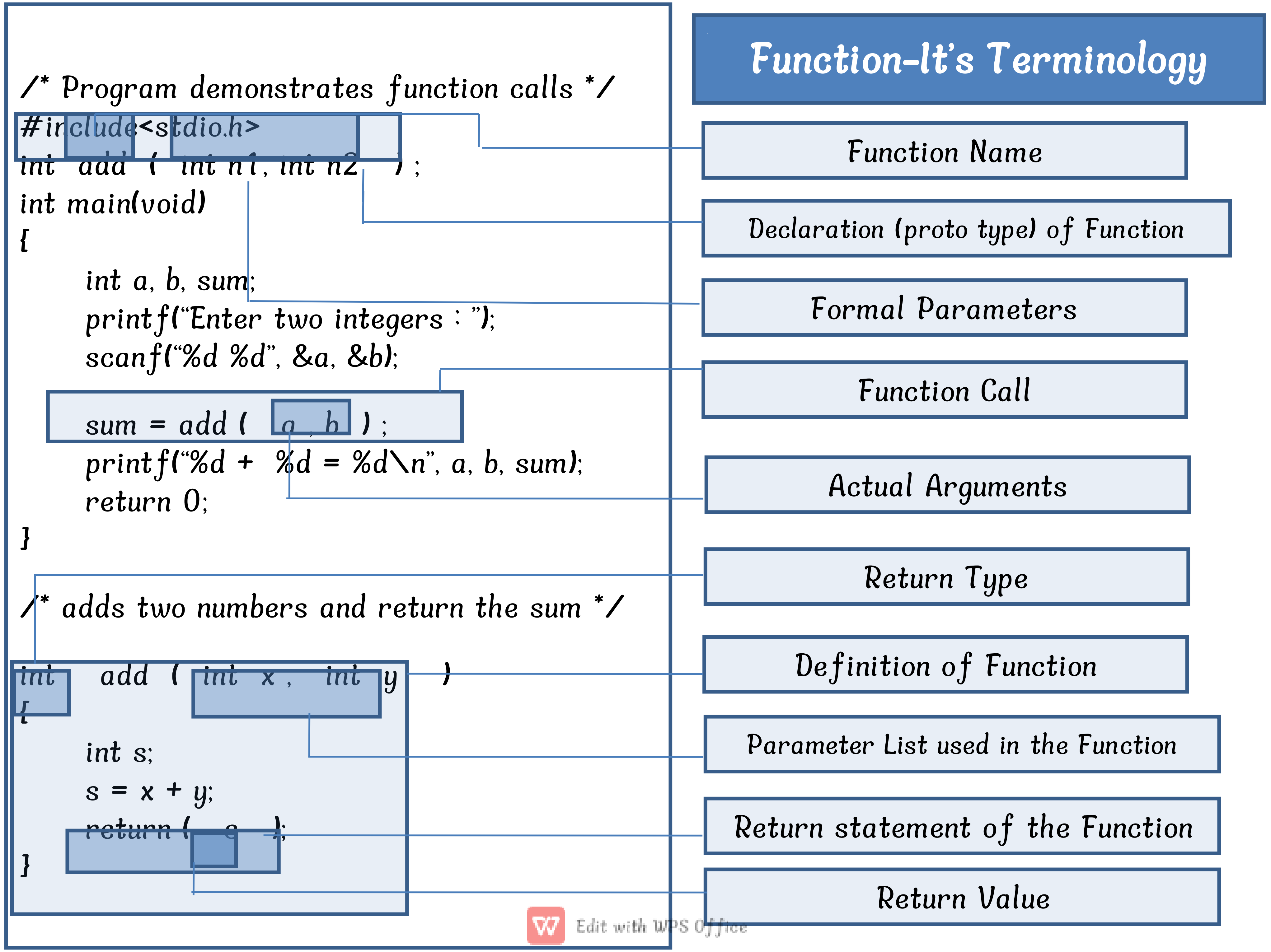
```
int   main()
{
        int n1, n2;
        printf("Enter a number : ");
        scanf("%d",&n1);
        printOctal(n1);
        readPrintHexa();
        printf("Enter a  number : ");
        scanf("%d",&n2);
        printOctal(n2);
        printf("\n");
}
```

```
void printOctal(int n)
{
        printf("Number in octal form : %o \n", n);
}
```

```
void   readPrintHexa()
{
        int num;
        printf("Enter a number : ");
        scanf("%d",&num);
        printHexa(num);
        printf("\n");
}
```

```
void printHexa(int n)
{
        printf("Number in Hexa-Decimal form : %x \n",n);
}
```

Function-It's Terminology



```
/* Program demonstrates function calls */
#include<stdio.h>
int  add  (  int n1, int n2   ) ;
int main(void)
{
      int a, b, sum;
      printf("Enter two integers : ");
      scanf("%d %d", &a, &b);

      sum = add (   a , b   ) ;
      printf("%d +  %d = %d\n", a, b, sum);
      return 0;
}

/* adds two numbers and return the sum */

int    add  (  int  x ,   int  y      )
{
      int s;
      s = x + y;
      return (   s    );
}
```

Storage Classes – Scope & Lifetime

```
#include<stdio.h>
float length, breadth;
int main()
{
    printf("Enter length, breadth : ");
    scanf("%f %f",&length,&breadth);
    area();
    perimeter();
    printf("\nEnter length, breadth: ");
    scanf("%f %f",&length,&breadth);
    area();
    perimeter();
}
```

External Global Variables

Scope: Visible across multiple functions
Lifetime: exists till the end of the program.

```
void area()
{
    static int num = 0;
    float a;
    num++;
    a = (length * breadth);
    printf("\nArea of Rectangle %d : %.2f", num, a);
}
```

Static Local Variables

Visible with in the function, created only once when function is called at first time and exists between function calls.

```
void perimeter()
{
    int no = 0;
    float p;
    no++;
    p = 2 *(length + breadth);
    printf("Perimeter of Rectangle %d: %.2f",no,p);
}
```

Automatic Local Variables

Scope : visible with in the function.
Lifetime: re-created for every function call and destroyed automatically when function is exited.

```
Enter length, breadth : 6   4
Area of Rectangle 1 : 24.00
Perimeter of Rectangle 1 : 20.00
Enter length, breadth : 8   5
Area of Rectangle 2 : 40.00
Perimeter of Rectangle 1 : 26.00
```

File1.c

```c
#include<stdio.h>
float length, breadth;

static float base, height;
int main()
{
   float peri;
   printf("Enter length, breadth : ");
   scanf("%f %f",&length,&breadth);
   rectangleArea();
   peri = rectanglePerimeter();
   printf("Perimeter of Rectangle : %f", peri);
   printf("\nEnter base , height: ");
   scanf("%f %f",&base,&height);
   triangleArea();
}
void rectangleArea() {
  float a;
  a = length * breadth;
  printf("\nArea of Rectangle : %.2f",  a);
}
void triangleArea() {
  float a;
  a = 0.5 * base * height ;
  printf("\nArea of Triangle : %.2f",  a);
}
```

File2.c

```c
extern float length, breadth ;
/*  extern base , height ;  --- error  */
float rectanglePerimeter()
{
   float p;
   p = 2 *(length + breadth);
   return ( p );
}
```

External Global Variables

Scope: Visible to all functions across all files in the project.
Lifetime: exists till the end of the program.

Static Global Variables

Scope: Visible to all functions with in the file only.
Lifetime: exists till the end of the program.

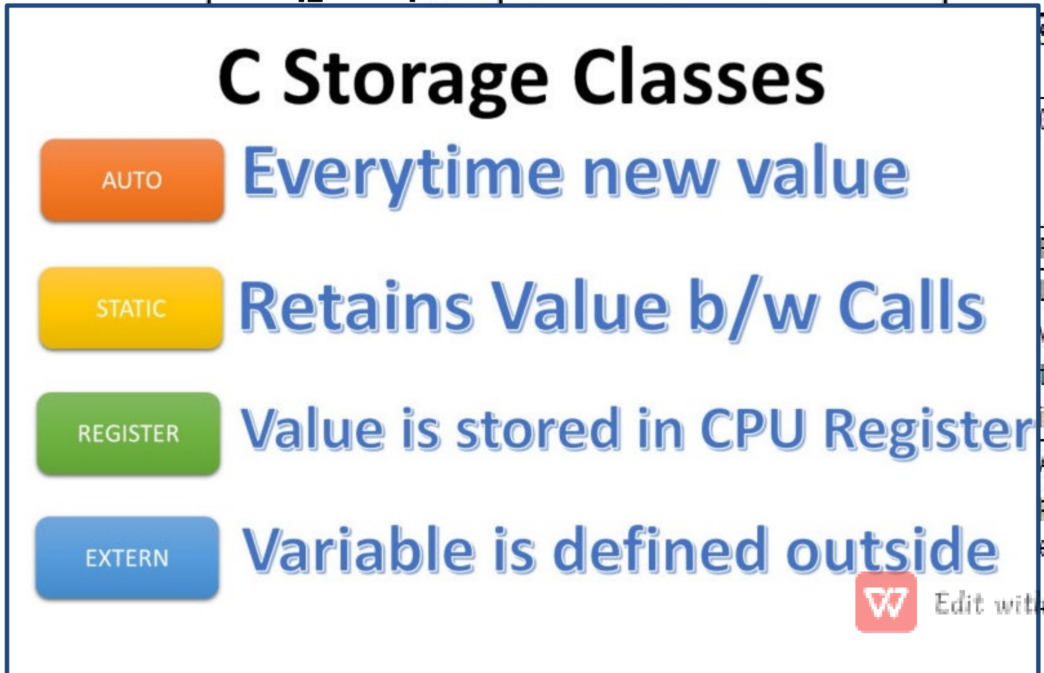
Register Variables

register int i;

Scope & Lifetime: Same as auto variable

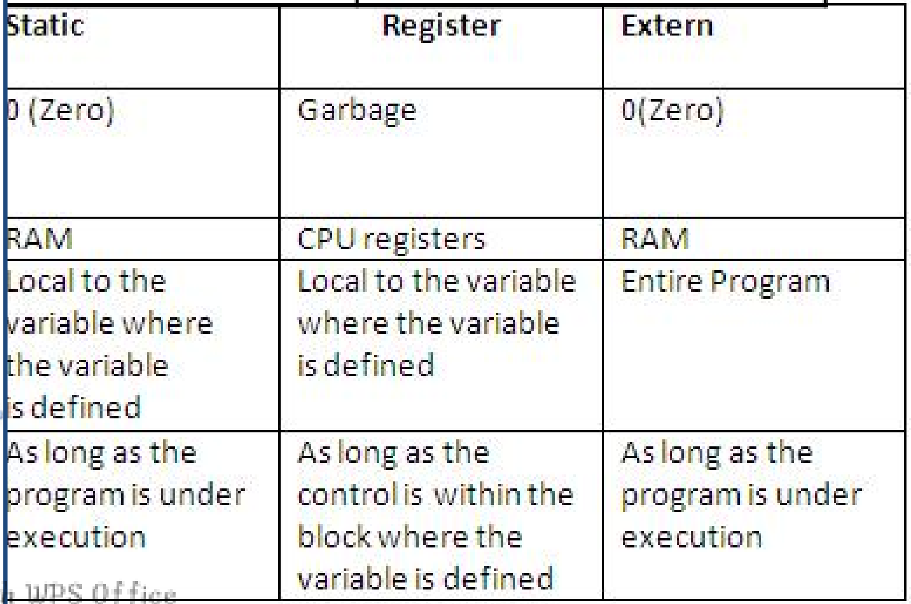
STORAGE CLASSES SUMMARY

Storage Class	Declaration Location	Scope (Visibility)	Lifetime (Alive)
auto	Inside a function/block	Within the function/block	Until the function/block completes
register	Inside a function/block	Within the function/block	Until the function/block completes
extern	Outside all functions	Entire file plus other files where the variable is declared as extern	Until the program terminates
static (local)	Inside a function/block	Within the function/block	Until the program terminates
static (global)	Outside all functions	Entire file in which it is declared	Until the program terminates



C Storage Classes

AUTO	Everytime new value
STATIC	Retains Value b/w Calls
REGISTER	Value is stored in CPU Register
EXTERN	Variable is defined outside

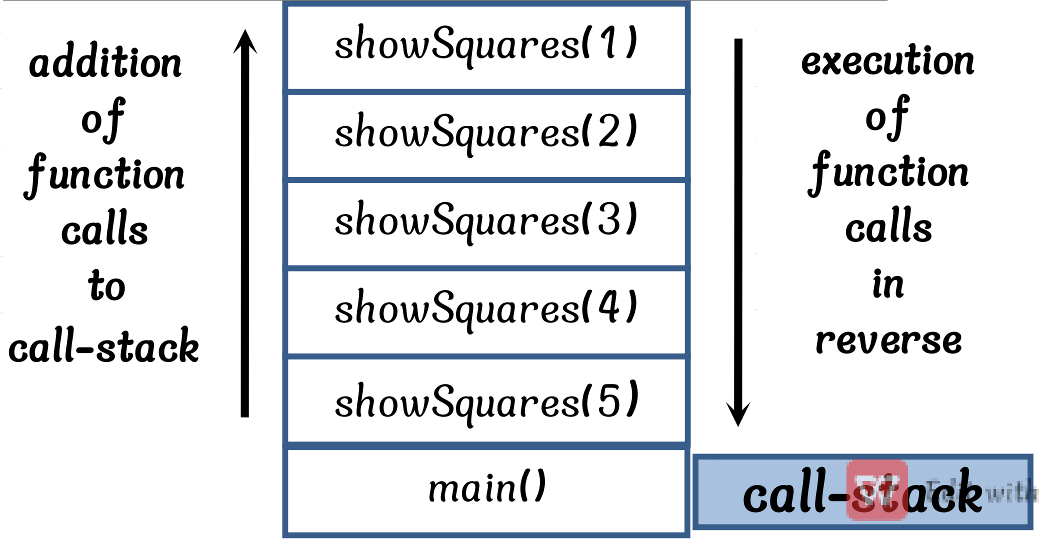


Static	Register	Extern
0 (Zero)	Garbage	0(Zero)
RAM	CPU registers	RAM
Local to the variable where the variable is defined	Local to the variable where the variable is defined	Entire Program
As long as the program is under execution	As long as the control is within the block where the variable is defined	As long as the program is under execution

```
#include<stdio.h>
void main()
{
    showSquares(5);
}
void showSquares(int n)
{
     if(n == 0)
        return;
     else
        showSquares(n-1);
     printf("%d  ", (n*n));
}
```

A function calling itself is Recursion

```
Output :  1   4   9   16   25
```



Preprocessor is a program that processes the source code before it passes through the compiler.

Preprocessor Directives

#define	- Define a macro substitution
#undef	- Undefines a macro
#ifdef	- Test for a macro definition
#ifndef	- Tests if a macro is not defined
#include	- Specifies the files to be included
#if	- Test a compile-time condition
#else	- Specifies what to do **#if** test fails
#elif	- Provides alternative test facility
#endif	- Specifies the end of **#if**
#pragma	- Specifies certain instructions
#error	- Stops compiling when error occurs

Predefined Macros

__DATE__ The current date in "MMM DD YYYY" format.

__TIME__ The current time as in "HH:MM:SS" format.

__FILE__ The current filename as a string literal.

__LINE__ The current line number as a decimal constant.

DEMO OF DIRECTIVES

https://fresh2refresh.com/c-programming/c-preprocessor-directives/