Functions

Introduction

- **Divide and conquer**
  - Construct a program from smaller pieces or components
    - These smaller pieces are called modules.
  - Each piece more manageable than the original program
  - Easier to debug and maintain.

Structured Programming

- For smaller problems it is easier to create a code which is easy to maintain and debug.
- For large complex problems this approach is Impractical as they contain smaller problems as subsets and one complex program is hard to understand and maintain.
- To effectively solve the complex problems we must employ the method of structured programming.

Known Facts

- C supports the use of library functions, which are used to carry out a number of predefined operations and task.
- Example:
    - » pow (a, I);
    - » Islower( );
    - » clrscr( );
    - » printf(“………….”);
- An important function which is present in all the programs:
    - » int main(void) { ……………….. }

Basics

- C allows programmers to define their OWN functions for carrying out specific task.
- The use of programmer-defined functions allows a large program to be broken down into smaller, self-contained components, each of which has some unique, identifiable purpose.
- The use of function avoids the need for redundant programming of same set of instructions.

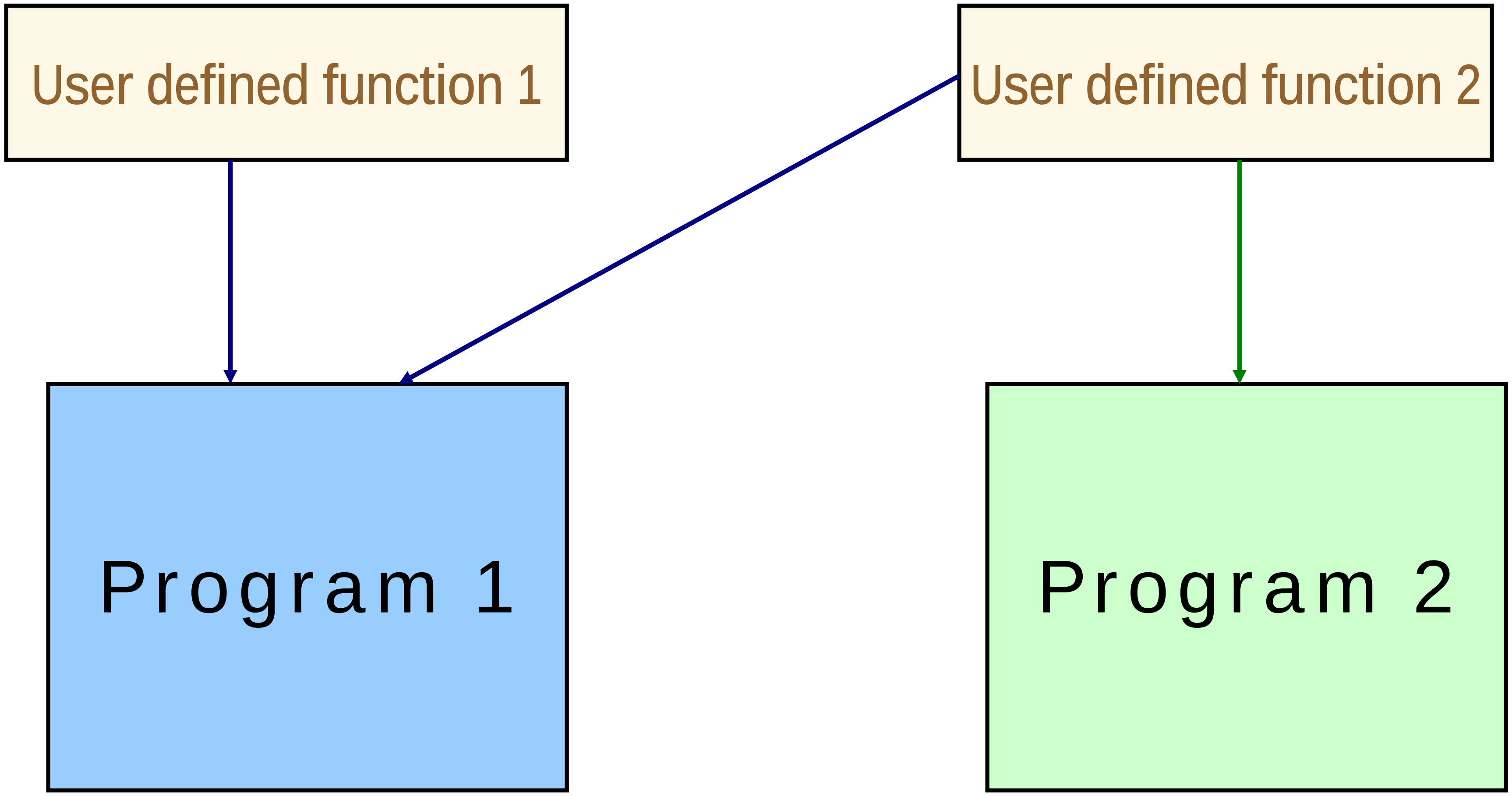
What is a function?

- A User-defined functions is a self-contained program segment that carries out well defined specific task.

- Such function once defined can be called any number of times from any program.

- It allows user to build his own customized library of frequently used routines which will avoid redundancy and enhance modularity of program.

Functional view



Classification

- Library Functions: Predefined functions which are meant for specific task. To use these functions in program one should include the appropriate header file.e.g. printf, scanf etc

- Programmer-defined Functions: Functions which are defined by user for a particular operation.

The main Function

- The main Function is the only function which does not need to be declared or called,as every C program must have a one and main is called by the operating system

Program to calculate sum of 3 numbers

```
#include<stdio.h>
int calsum (int , int , int );
main( )
{
    int a, b, c, sum;
    scanf(“%d %d %d”, &a, &b, &c);

    sum = calsum(a, b, c);
    printf(“\n\n Sum = %d”, sum);
}
int calsum (int x, int y, int z)
{ int d;
   d = x + y + z;
   return(d);
}
```

User defined function

Syntax

- The three important things regarding a functions :
  - A function can appear in a program in three ways, as a Declaration , Call and Definition.
  - A function has three attributes, the Return type, Identifier (or name) and Parameter List.
  - The Body of a function is represented by a code block (not optional) and has local declarations, expressions and a return type.

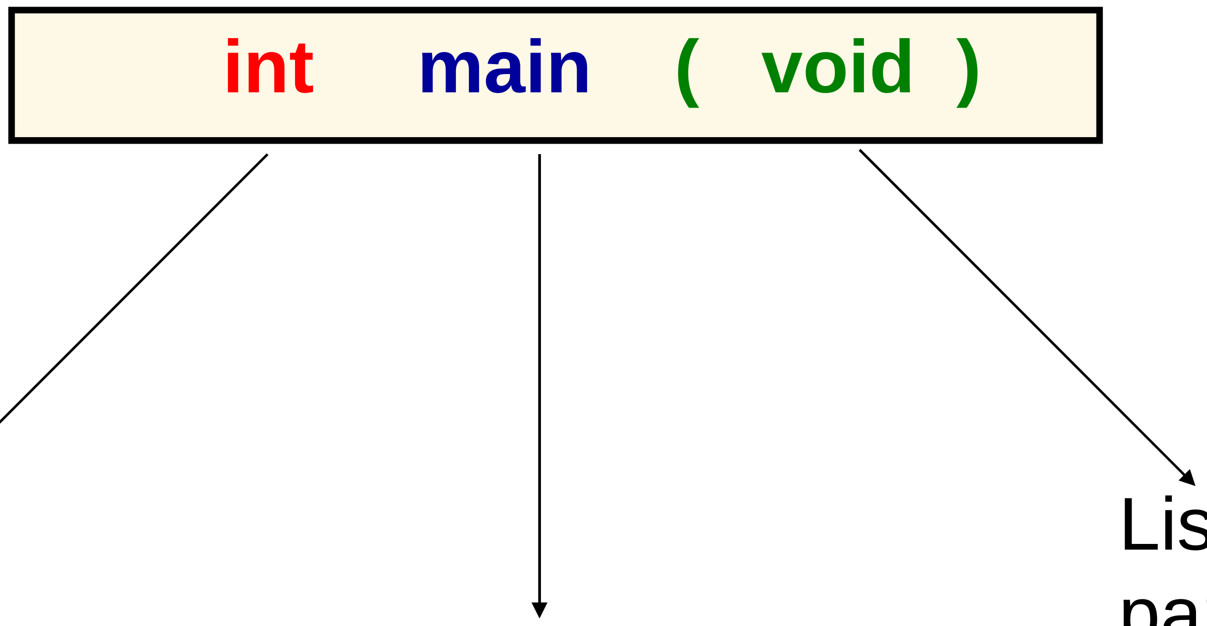
Declaration

- A function has to be declared in the program before it is called.
- Function declaration is called "PROTOTYPE" of the function and consist of three attributes :
  - » Return type
  - » Identifier (name)
  - » Parameter list.

- General syntax of prototype:

  return type name (parameter list) ;

Example



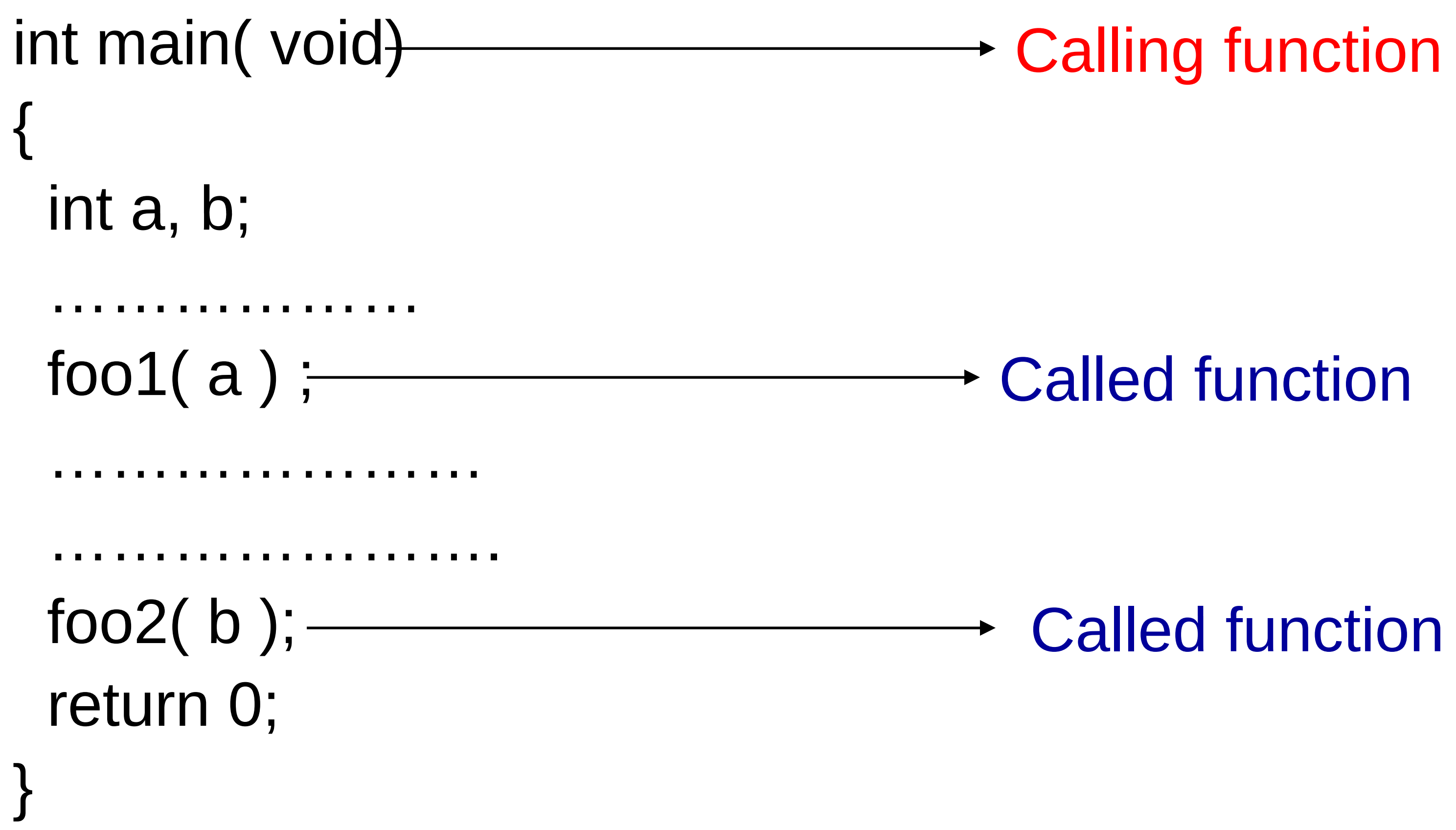
```
int    main    (  void  )
```

Return type of function. It will return integer after the execution.

Name of the function.

Follow the rules for identifiers.

List of paramete rs if there are none use keyword void.



```
int main( void)
{
  int a, b;
  ……………….
  foo1( a ) ;
  ………………….
  ………………….
  foo2( b );
  return 0;
}
```

Return type

- It is a way of sending data back to the “calling” function.
- Using “ return” keyword called function can send manipulated data back to the calling function.
- When no information or data is to be sent back to the calling function return type of called function is set to “void” which means nothing.
- Function can return integer, character, float or double, pointer.

Declaration Example

- int sort (void) ;
- int max3 (int , int , int);
- void error_message (void);
- char alpha (char , char);

Remember function declaration and prototype is one and the same thing.

Function call

- A call to a function is made when ever we want to execute the code of the function in program.
- Example

```
int main ( void )
{
    ……………..
    funct1(  ) ;
    return 0;
}
```

$\longrightarrow$ Call to function

Accessing a Function

- A function is called by specifying its name, followed by the actual arguments in parenthesis.
- Each actual argument must be of the same data type as its corresponding formal argument.
- Control returns to the point of call.

Example

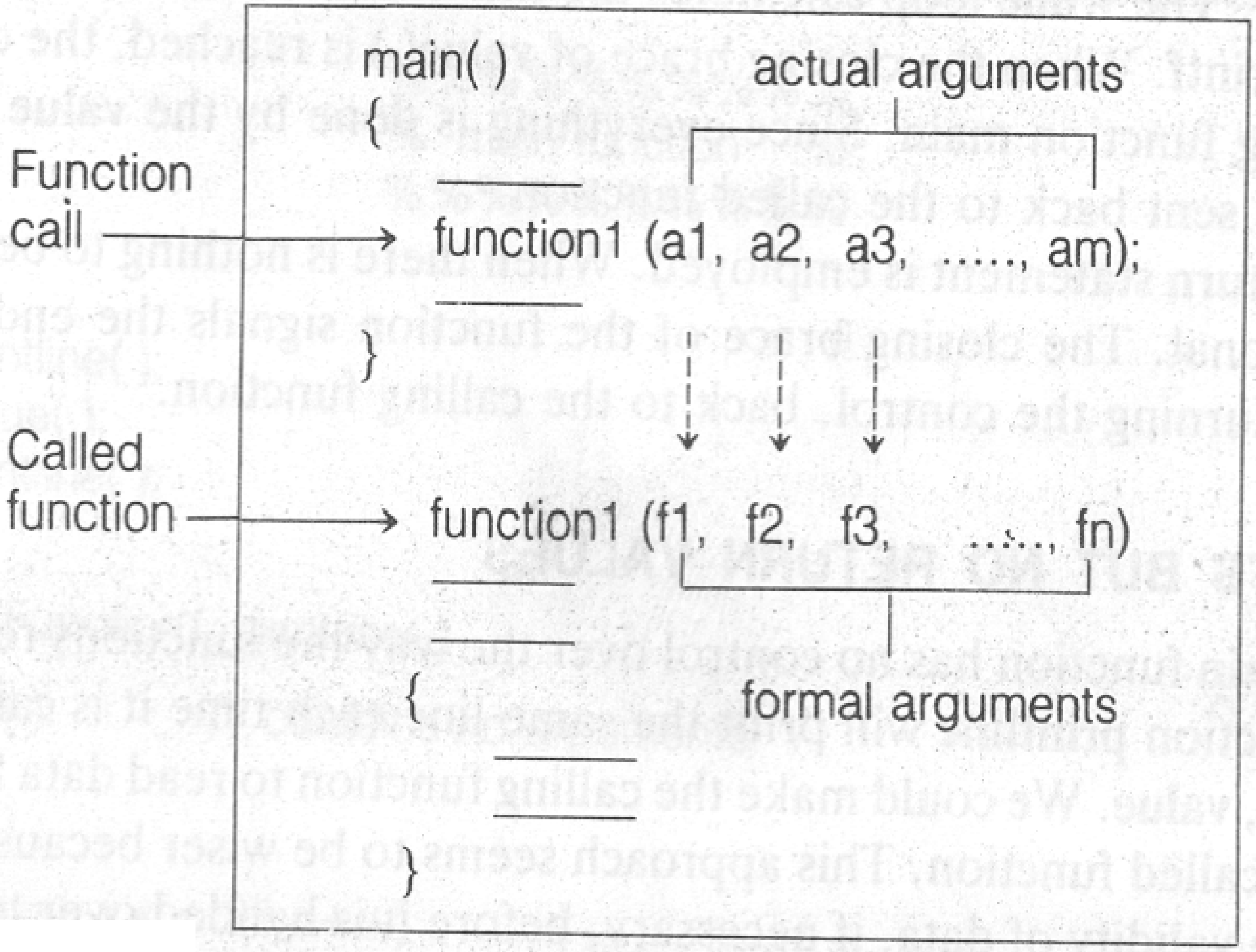
Program to calculate sum of three integer quantities

```
#include<stdio.h>
int calsum (int , int , int );
main( )
{
    int a, b, c, sum;
    scanf("%d %d %d", &a, &b, &c);

    sum = calsum(a, b, c);
    printf("\n\n Sum = %d", sum);
}
int calsum (int x, int y, int z)
{ int d;
   d = x + y + z;
   return(d);}
```

Actual arguments

Formal Arguments



Arguments matching between the function call and the called function

Function definition

- The Function Definition contains the same information as the declaration; return type, name and parameter list.
- The Function Definition also contains a block which contains the function’s code.

Syntax of definition

```
return_type identifier ( parameter_list )
                         {
                              local declarations;
                              local statements;
                              return return_value;
                         }
```

Example:

```
int  maximum( int x, int y)
          {     int z;
                 z=(x>=y) ? x : y ;
                return z;
          }
```

Example

```
/* Program to calculate sum of three integer quantities*/
# include<stdio.h>
int calsum (int , int , int );  ——→  Prototype / declaration
int main( )
{
    int a, b, c, sum;
    scanf("%d %d %d", &a, &b, &c);

      sum = calsum(a, b, c);  ——→  Call to function
     printf("\n\n Sum = %d", sum);  return 0;
}
int calsum (int x, int y, int z)  ——→  Definition
{ int d;
   d = x + y + z;
   return(d);}
```

Write a program to calculate greatest of two numbers by using function.

```
# include<stdio.h>
int check( int , int ) ;
int main(void)
{       int a,b,z;
        z= check( a, b) ;
        printf(“The greatest of two numbers =%d”,z);
}
int check( int a1, int b1)
{ int a;
  a = ( a1>= b1) ? a1 : b1 ;
  return a;   }
```

Write a program to convert a lowercase character to uppercase using a programmer defined function

```
# include<stdio.h>
char lower_to_upper( char ) ;    ← Function declaration
int main(void)
{         char lower, upper;
printf(“please enter a lowercase character:”);
scanf(“%c”,&lower);
          upper=lower_to_upper(lower) ;    ← Function call
printf(“The uppercase equivalent is %c”,upper);
}
char lower_to_upper(char c1)    ← Function definition
{ char c2;
  c2 = ( c1>=‘a’&&c1<=‘z’) ? (‘A’+c1-’a’) : c1 ;
  return c2;   }
```

Calculating Net Salary program

Problem:

Calculate the net salary if basic salary and loan amount as as input.

Steps involved :

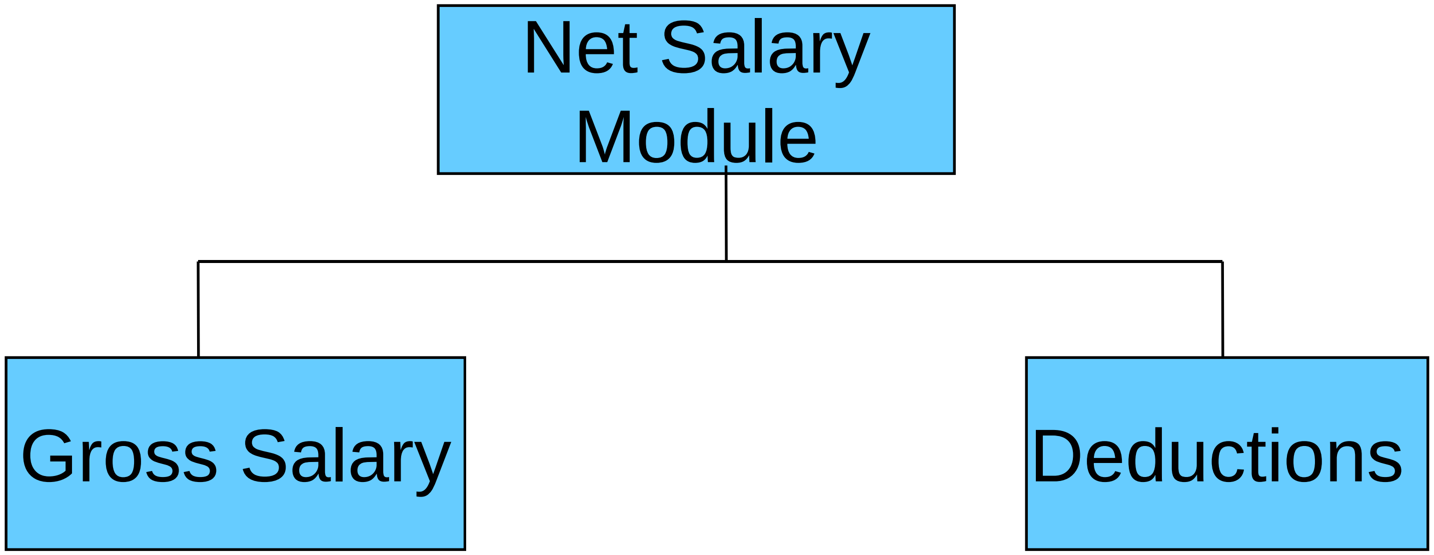
1. Calculate the gross salary
2. Calculate the Deductions.
3. Net salary = gross salary - deductions.
4. Print Net salary

Breaking into simpler Tasks:

Instead of finding all at one stretch and making the problem look complex, divide the work into smaller modules to find the gross salary and deductions separately and call those modules in your main module

Calculating Net Salary program

Breaking a bigger task into simpler tasks



Gross salary Function

```
float Gross_salary(int bas)
  {
        float da,hra,grs;
        da = 0.74 * bas;
        hra = 0.3 * bas;
        gros = bas + da + hra;
        return gross;
  }
```

Deductions Function

```
float Deductions (float gross)
{
  float ded, tax,loan;
  printf (“ enter the loan amount to be deducted/month”);
  scanf(“%f”,&loan);

    if (gross > 10000)
                 tax = 0.2 * gross;
    else
                 tax = 0.1 * gross;

    ded = tax + loan;

    return ded;
}
```

Net salary program

```
#include<stdio.h>
float Gross_salary(int);
float Deductions (float);
main( )
{
     int basic;
    float gross,deduct,net;
    printf (“ enter the basic salary of the person \n”);
   scanf(“%d”,&basic);
    gross = Gross_salary(basic);
    deduct = Deductions(gross);
    net = gross - deduct;
   printf(“ the net salary is %f”,net);
}
```

assignment

- Write a function double power(double x, int n) that will compute x^n ,the n^{th} power of x.

- Write a function, say int sum_n(int n, int m),which computes the sum of n integers starting with m^{th} integer i.e.

$$m+(m+1)+(m+2)+\ldots+(m+n-1)$$

```
double power2(double a, int b)
{
    if (b == 0) {
        // base case: anything to the 0 power is 1
        return 1.0;
    }
    if (a == 0.0) {
        // save us some time, 0 to any power other than 0 is 0
        return 0.0;
    }
    if (b < 0) {
        return 1 / (a * power2(a, -1 * b - 1));
    }
    return a * power2(a, b - 1);
}
```

Home assignment

Write a program to generate prime numbers for a given range by using function.

```
void prm(int min,int max)
{
 int num,i,count;
 printf("the prime numbers are\n");
 for(num = min;num<=max;num++)
{ count = 0;
 for(i=2;i<=num/2;i++) {
 if(num%i==0){
 count++; break; } }
 if(count==0 && num!= 1)
printf("\n%d ",num); } }
```

Types of Function Calls

- Call by value
  - Copy of argument passed to function
  - Changes in function do not affect original arguments
  - Use when function does not need to modify argument
    - Avoids accidental changes
- Call by reference **(will do in SDF –II)**
  - Passes original argument
  - Changes in function affect original arguments
  - Only used with trusted functions
- For now, we focus on call by value

Calculating area of a circle

```
include<stdio.h> /*The function calls are Call by
Value*/
#define pi 3.14
float area(float);
int main( )
{
float r, a;
printf(“Enter the radius\n”);
scanf(“%f”,&r);
a = area(r);
printf(“The area = %.2f”, a);
return 0;}
```

```
float area(float x)
{
return pi*x*x;
}
float perimeter(float y)
{
return 2.0*pi*y;
}
```

Calling a function multiple times

```
#include<stdio.h>
int square( int); /* function prototype */
int main()
{
int x;
printf("the squares of numbers from 1 to 10 are:\n");
 for(x=1 ;x <= 10; x++)
{
 y = square(x);   /*function call */
printf("the sqare of %d = %d\n",x, y); }
return 0;    }

/*function definition */
int square (int a)
   {
 int b;
   b = a * a;
return  b;
}
```