CS11001/CS11002
Programming and Data Structures (PDS) (Theory: 3-0-0)

Class Teacher: Pralay Mitra
Jayanta Mukhopadhyay
Soumya K Ghosh

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

PDS Class Test 1

- Date: August 25, 2016
- Time: 7pm to 8pm
- Marks: 20 (Weightage 50%)

Room	Sections	No of students
V1	Section 8 (All) Section 9 (AE,AG,BT,CE, CH,CS,CY,EC,EE,EX)	101 49
V2	Section 9 (Rest, if not allotted in V1) Section 10 (All)	50 98
V3	Section 11 (All)	98
V4	Section 12 (All)	94
F116	Section 13 (All)	95
F142	Section 14 (All)	96

Functions

Introduction

- Function
  - A self-contained program segment that carries out some specific, well-defined task.
- Some properties:
  - Every C program consists of one or more functions.
    - One of these functions must be called “main”.
    - Execution of the program always begins by carrying out the instructions in “main”.
  - A function will carry out its intended action whenever it is *called* or *invoked*.
  - In general, a function will process information that is passed to it from the calling portion of the program, and returns a single value.
    - Information is passed to the function via special identifiers called arguments or parameters.
    - The value is returned by the “return” statement.
  - Some function may not return anything.
    - Return data type specified as “void”.

Function Example

```
#include  <stdio.h>

int  factorial (int m)
{
     int i, temp=1;
     for (i=1; i<=m; i++)
              temp = temp * i;
     return (temp);
}
int main()
{
     int  n,fact;
     for  (n=1; n<=10; n++) {
              fact=factorial (n);
              printf (“%d! = %d \n”,n,fact);
     }
     return 0;
}
```

```
#include  <stdio.h>

int  factorial (int m)
{
     int i, temp=1;
     for (i=1; i<=m; i++)
              temp = temp * i;
     return (temp);
}
int main()
{
     int  n;
     for  (n=1; n<=10; n++)
              printf (“%d! = %d \n”,n,factorial (n));
     return 0;
}
```

Functions: Why?

- Functions
  - Modularize a program
  - All variables declared inside functions are local variables
    - Known only in function defined
  - Parameters
    - Communicate information between functions
    - They also become local variables.
- Benefits
  - Divide and conquer
    - Manageable program development
  - Software reusability
    - Use existing functions as building blocks for new programs
    - Abstraction - hide internal details (library functions)
  - Avoids code repetition

Defining a Function

- A function definition has two parts:
  - The first line.
  - The body of the function.

return-value-type *function-name* (*parameter-list*)
{
declarations and statements
}

Function: First Line

- The first line contains the return-value-type, the function name, and optionally a set of comma-separated arguments enclosed in parentheses.
  - Each argument has an associated type declaration.
  - The arguments are called formal arguments or formal parameters.
- Example:

```
int gcd (int A, int B)
```

- The argument data types can also be declared on the next line:

```
int gcd (A, B)
int A, B;
```

Function: Body

- The body of the function is actually a compound statement that defines the action to be taken by the function.

```
int  gcd  (int A, int B)
{
   int  temp;
   while ((B % A) != 0)  {
     temp = B % A;
     B = A;
     A = temp;
   }
   return (A);
}
```

BODY

Declarations and statements: function body (block)
- Variables can be declared inside blocks (can be nested)
- Function can not be defined inside another function

- Returning control
  - If nothing returned
    - `return;`
    - or, until reaches right brace
  - If something returned
    - `return` *expression*`;`

Function Not Returning Any Value

- Example: A function which only prints if a number if divisible by 7 or not.

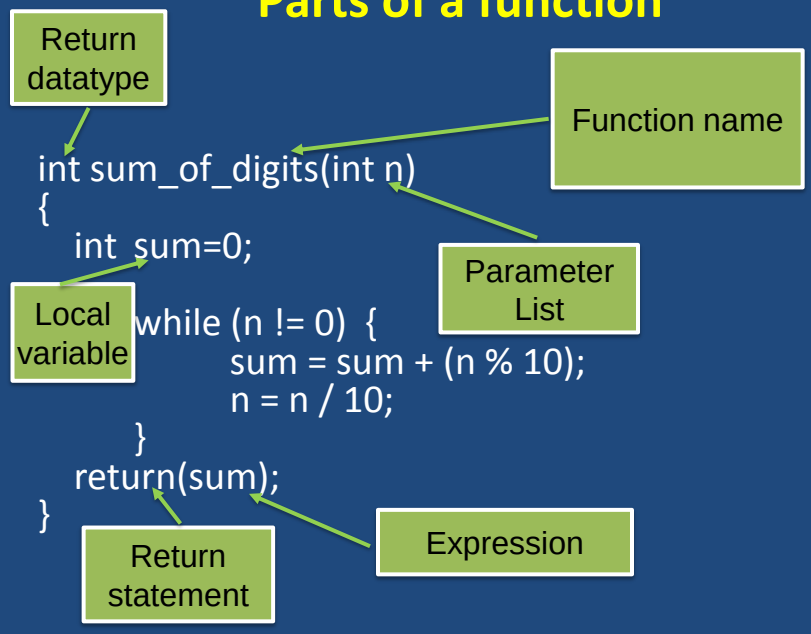
```
void  div7 (int n)
{
   if  ((n % 7) == 0)
       printf ("%d is divisible by 7", n);
   else
       printf ("%d is not divisible by 7", n);
   return;
}
```

OPTIONAL

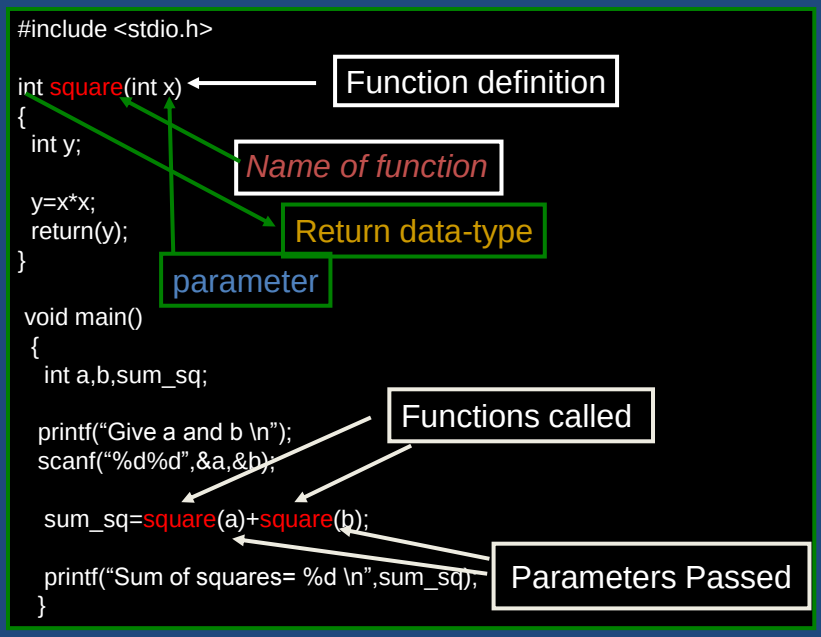
Function: Call

- When a function is called from some other function, the corresponding arguments in the function call are called actual arguments or actual parameters.
  - The formal and actual arguments must match in their data types.
- Point to note:
  - The identifiers used as formal arguments are "local".
    - Not recognized outside the function.
    - Names of formal and actual arguments may differ.

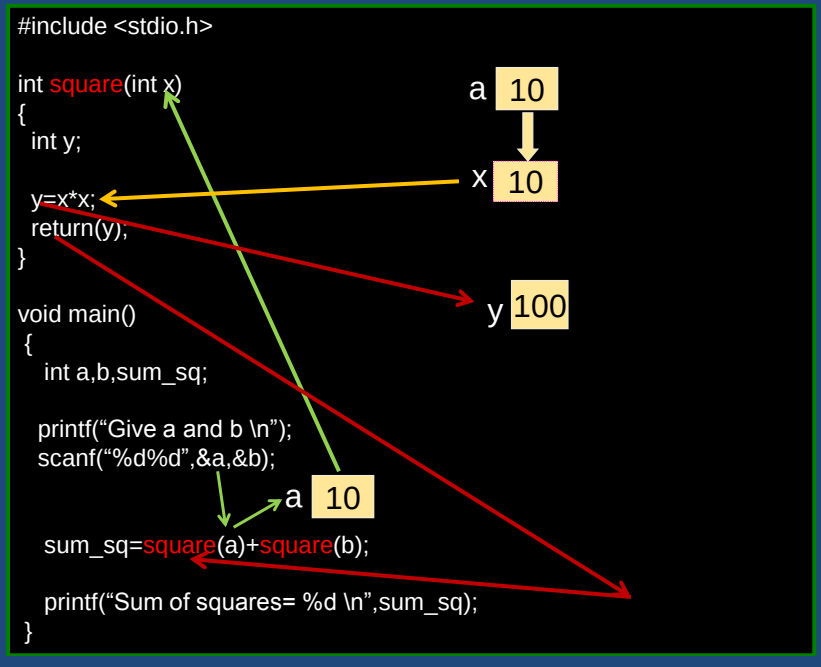
Parts of a function



Function: An Example



Invoking a function call : An Example



Function Prototypes

- Usually, a function is defined before it is called.
  - Easy for the compiler to identify function definitions in a single scan through the file.
- However, many programmers prefer a top-down approach, where the functions follow main().
  - Must be some way to tell the compiler.
  - Function prototypes are used for this purpose.
    - Only needed if function definition comes after use.

Function Prototype (Contd.)

- Function prototypes are usually written at the beginning of a program, ahead of any functions (including main()).
- Examples:

```
int  gcd (int A, int B);
void div7 (int number);
```

  - Note the semicolon at the end of the line.
  - The argument names can be different ~~(optional too)~~; but it is a good practice to use the same names as in the function definition.

Function Prototype: Examples

```
#include  <stdio.h>

int ncr (int n, int r);
int fact (int n);

int main()
{
     int i, m, n, sum=0;
     printf("Input m and n \n");
    scanf ("%d %d", &m, &n);

     for (i=1; i<=m; i+=2)
        sum = sum + ncr (n, i);

     printf ("Result: %d \n", sum);
     return 0;
}
```

Prototype declaration

Function prototype is optional if it is defined before use (call).

```
int  ncr (int n, int r)
{
     return (fact(n) / fact(r) / fact(n-r));
}

int  fact (int n)
{
     int  i, temp=1;
     for (i=1; i<=n; i++)
        temp *= I;
     return (temp);
}
```

Function definition

Function: Summary

```
#include  <stdio.h>

int  factorial (int m)
{
    int i, temp=1;
    for (i=1; i<=m; i++)
          temp = temp * i;
    return (temp);
}
```

Returned data-type

parameter

Function name

Local vars

Return statement

Self contained programme

```
int main()
{
    int  n;
    for  (n=1; n<=10; n++)
    printf ("%d! = %d \n", n, factorial (n) );
    return 0;
}
```

main() is a function

Calling a function

Functions: Some Facts

- A function cannot be defined within another function.
  - All function definitions must be disjoint.
- Nested function calls are allowed.
  - A calls B, B calls C, C calls D, etc.
  - The function called last will be the first to return.
- A function can also call itself, either directly or in a cycle.
  - A calls B, B calls C, C calls back A.
  - Called recursive call or recursion.

Functions: Some Facts

- A function can be declared within a function.
- The function declaration, call and definition must match with each other.
  - int gcd(int a, int b); // function declaration
  - gcd(a,b); //function call, a and b is int
  - int gcd(int a, int b) // function definition

```
{
        .....
}
```

Header Files

- Header files
  - contain function prototypes for library functions
  - <stdio.h>, <stdlib.h> , <math.h>, etc
  - Load with
    #include <filename>
  - #include <math.h>
- Custom header files
  - Create file with functions
  - Save as *filename.h*
  - Load in other files with #include "*filename.h*"
  - Reuse functions

Math Library Functions

- Math library functions
  - perform common mathematical calculations
  - `#include <math.h>`
  - `cc <prog.c> -lm`
- Format for calling functions
  `FunctionName(argument);`
    - If multiple arguments, use comma-separated list
  - `printf("%.2f",sqrt(900.0));`
    - Calls function `sqrt`, which returns the square root of its argument
    - All math functions return data type `double`
  - Arguments may be constants, variables, or expressions

Math Library Functions

double acos(double x)	-- Compute arc cosine of x.
double asin(double x)	-- Compute arc sine of x.
double atan(double x)	-- Compute arc tangent of x.
double atan2(double y, double x)	-- Compute arc tangent of y/x.
double ceil(double x)	-- Get smallest integral value that exceeds x.
double floor(double x)	-- Get largest integral value less than x.
double cos(double x)	-- Compute cosine of angle in radians.
double cosh(double x)	-- Compute the hyperbolic cosine of x.
double sin(double x)	-- Compute sine of angle in radians.
double sinh(double x)	-- Compute the hyperbolic sine of x.
double tan(double x)	-- Compute tangent of angle in radians.
double tanh(double x)	-- Compute the hyperbolic tangent of x.
double exp(double x)	-- Compute exponential of x
double fabs(double x)	-- Compute absolute value of x.
double log(double x)	-- Compute log(x).
double log10(double x)	-- Compute log to the base 10 of x.
double pow(double x, double y)	-- Compute x raised to the power y.
double sqrt(double x)	-- Compute the square root of x.

#define: Macro definition

- Preprocessor directive in the following form

  #define string1 string2

- Replaces the *string1* by *string2* wherever it occurs before compilation, e.g.

  #define PI 3.14

```
#include <stdio.h>
#define PI 3.14
main()
{
  float r=4.0,area;
  area=PI*r*r;
  return 0;
}
```

Compiler Preprocessing

```
#include <stdio.h>
int main()
{
  float r=4.0,area;
  area=3.14*r*r;
  return 0;
}
```

#define with argument

- It may be used with argument e.g.

```
#define sqr(x)  ((x)*(x))
```

Which one is faster to execute?

```
#include <stdio.h>
int sqr(int x)
{
  return (x*x);
}
int main()
{
 int y=5;
 printf("value=%d \n", sqr(y)+3);
 return 0;
}
```

```
#include <stdio.h>

int main()
{
 int y=5;
 printf("value=%d \n", ((y)*(y))+3);
 return 0;
}
```

```
#include <stdio.h>
#define sqr(x) ((x)*(x))

int main()
{
 int y=5;
 printf("value=%d \n", sqr(y)+3);
 return 0;
}
```

#define with arguments: A Caution

```
#define   sqr(x)   x*x
```

– How macro substitution will be carried out?

r = sqr(a) + sqr(30); ➔ r = a*a + 30*30;

r = sqr(a+b); ➔ r = a+b*a+b;

WRONG?

– The macro definition should have been written as:

```
#define  sqr(x)  (x)*(x)
```

r = (a+b)*(a+b);