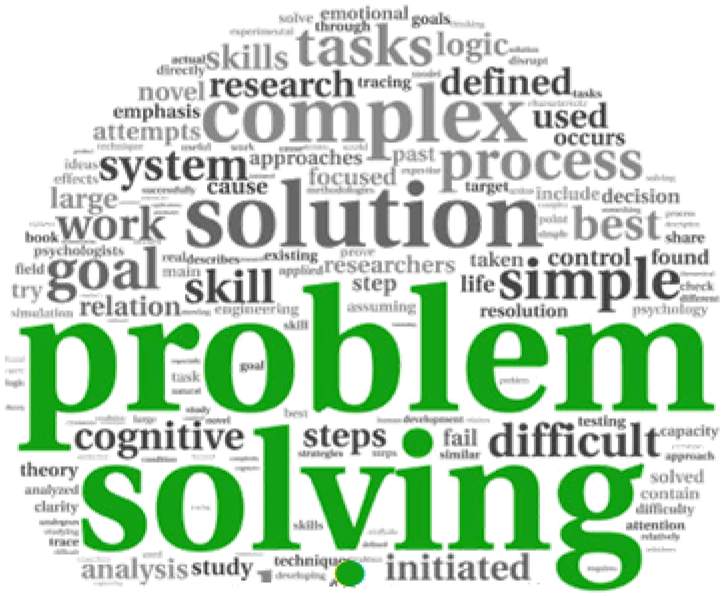


problem solving using computers

CSE 1051

S-17_1 MODULAR PROGRAMMING

Objectives:

To learn and appreciate the following concepts

- Modularization and importance of modularization
- Understand how to define and invoke a function
- Understand the flow of control in a program involving function call
- Function prototypes

Session outcome:

At the end of session one will be able to

- Understand modularization and function
- Write simple programs using functions
- Describe function prototypes

Programming Scenario . . .

In a large complex software development,

- Several Functionalities needs to be implemented
- Development needs to be done in a Team
- Lengthier code

Programming Scenario . . .

Lengthier programs

- Prone to errors
- tedious to locate and correct the errors

To overcome this

Programs broken into a number of smaller logical components, each of which serves a specific task.

Modularization

- Process of splitting the lengthier and complex programs into a number of smaller units is called **Modularization.**

- Programming with such an approach is called **Modular programming**

Advantages of modularization

- Reusability
- Readability
- Debugging is easier
- Build Library
- Manageability
- Develop in a Team
- Quality

Functions

- A **function** is a set of instructions to carryout a particular task.

- Using functions we can structure our programs in a **more modular** way.

Functions

- Standard functions

  (library functions or built in functions)

- User-defined functions

  (Written by the user/programmer)

General form of function definition

```
return_type  function_name(parameter_definition)
 {
     variable declaration;

     statement1;
     statement2;
              .
              .
              .
     return(value_computed);
 }
```

Defining a Function

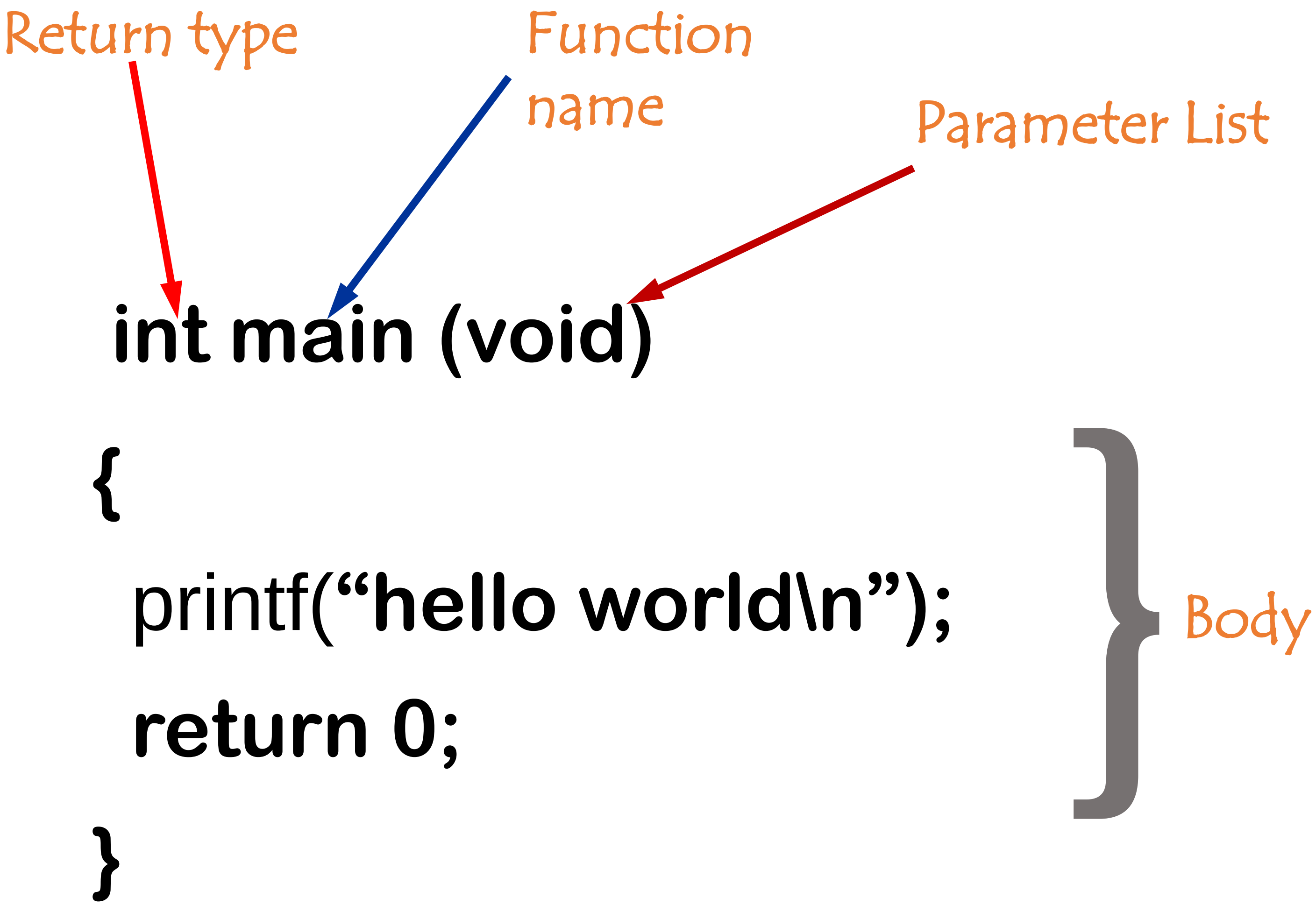
✓Name (function name)
  - You should give functions descriptive names
  - Same rules as variable names, generally

✓Return type
  - Data type of the value returned to the part of the program that activated (called) the function.

✓Parameter list (parameter_definition)
  - A list of variables that hold the values being passed to the function

✓Body
  - Statements enclosed in curly braces that perform the function's operations(tasks)

Understanding of main function



Return type

Function name

Parameter List

```
int main (void)
{
  printf("hello world\n");
  return 0;
}
```

Body

Function Definition and Call

// FUNCTION DEFINITION

Return type Function name Parameter List

```
void DisplayMessage(void)
{
        printf(“Hello from function DisplayMessage\n”);
}
int main()
{
        printf(“Hello from main \n”);
        DisplayMessage();   // FUNCTION CALL
        printf(“Back in function main again.\n”);
        return 0;
 }
```

Multiple Functions- An example

```
void First (void){          // FUNCTION DEFINITION
        printf("I am now inside function First\n");
}
 void Second (void){        // FUNCTION DEFINITION
        printf( "I am now inside function Second\n");
        First();            // FUNCTION CALL
        printf("Back to Second\n");
}
int main (){
        printf( "I am starting in function main\n");
        First ();           // FUNCTION CALL
         printf( "Back to main function \n");
        Second ();          // FUNCTION CALL
        printf( "Back to main function \n");
        return 0;
}
```

Arguments and parameters

- Both arguments and parameters are variables used in a **program** & **function.**

- Variables used in the *function reference or function call* are called as **arguments**. These are written within the parenthesis followed by the name of the function. They are also called actual parameters.

- Variables used in *function definition* are called **parameters**, They are also referred to as formal parameters.

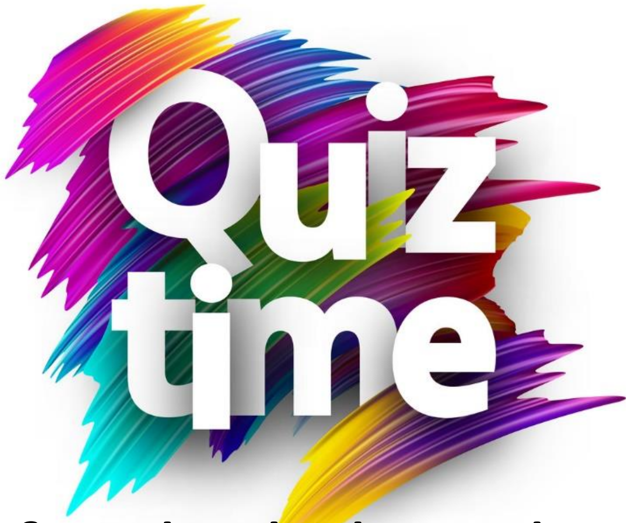
Functions

Formal parameters

```
void dispChar(int n, char c) {
     printf(" You have entered %d & %c“,n,c);
 }


int main(){   //calling program
     int no; char ch;
     printf(“Enter a number & a character: \n“);
     scanf(“%d %c”,&no,&ch);
     dispChar( no, ch); //Function reference
     return 0;
}
```

Actual parameters



Go to posts/chat box for the link to the question **PQn. S17.1**

submit your solution in next 2 minutes

The session will resume in 3 minutes

Function Prototypes

- Must be included for each function that will be defined, (required by Standards for C++ but optional for C) if not directly defined before main().
- In most cases it is recommended to include a function prototype in your program to avoid ambiguity.
- Identical to the function header, with semicolon (;) added at the end.
- Function prototype (declaration) includes
  - Function name
  - Parameters – what the function takes in and their type
  - Return type – data type function returns (default **int**)
- Parameter names are Optional.

Function Prototypes

- Function prototype provides the compiler the name and arguments of the functions and must appear before the function is used or defined.
- It is a model for a function that will appear later, somewhere in the program.
- General form of the function prototype:

fn_return_type fn_name(type par1, type par2, …, type parN);

- Example:

  int maximum(int, int, int);

  - Takes in 3 **int**s
  - Returns an **int**

Summary

- Modularization and importance of modularization
- Defining and invoking a function
- Flow of control of a program involving function call
- Function Prototypes