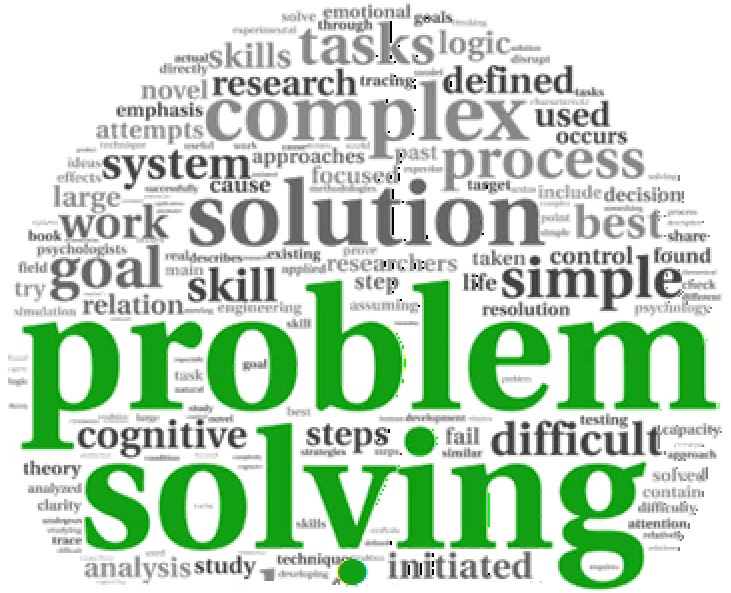


problem solving using computers

CSE 1051

S-18_1 PARAMETER PASSING TECHNIQUES

Objectives:

To learn and appreciate the following concepts

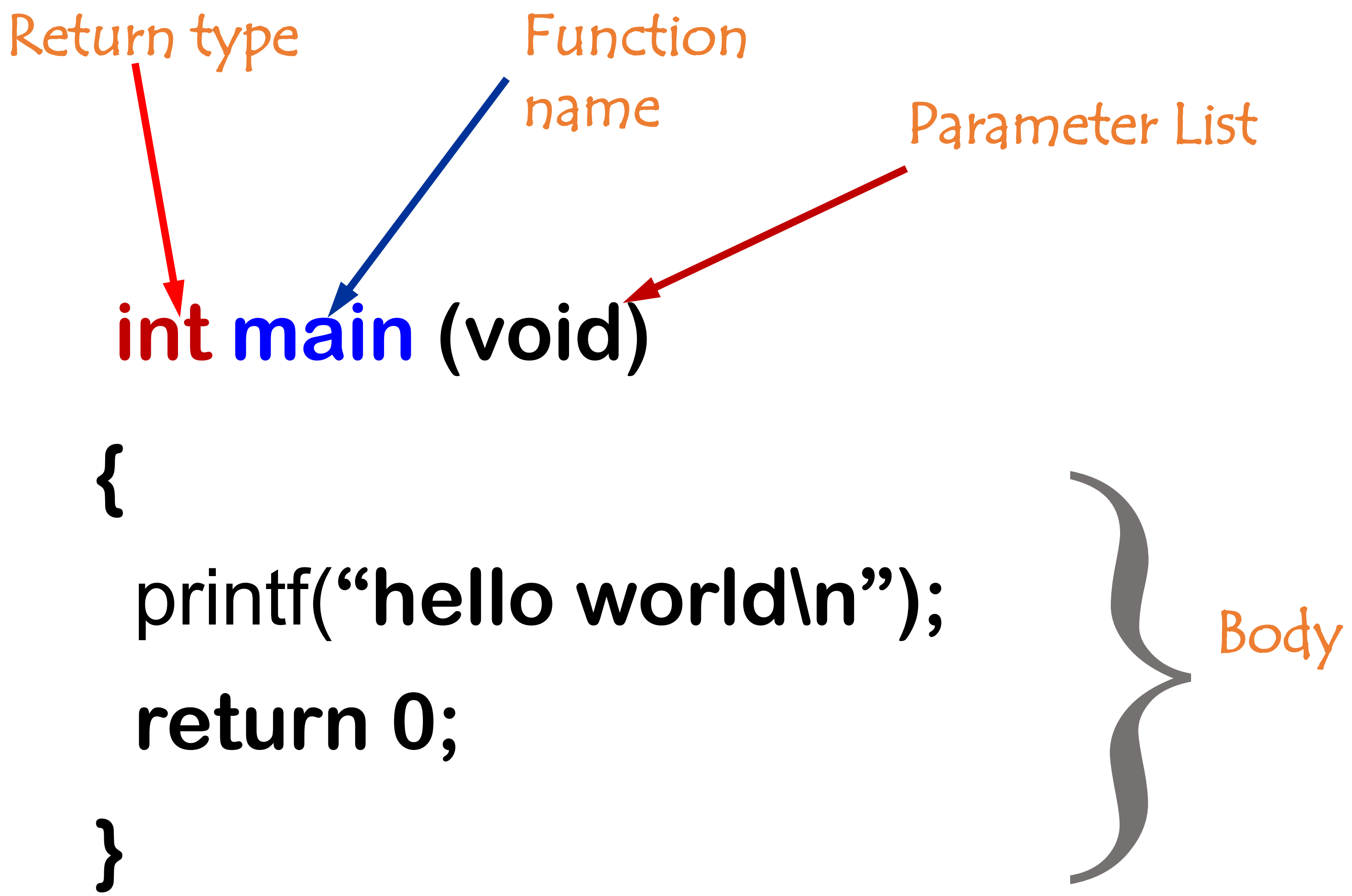
- Scope of variables
- Write C functions
- Invoking functions
- Write programs using functions
- Parameter passing techniques
- Pass by value

Session outcome

At the end of session one will be able to understand:

- The overall ideology using functions
- parameter passing techniques

Understanding a function – a recap



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

Functions – a recap

```
// FUNCTION DEFINITION
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 CALL
      return 0;
}
```

Formal parameters

Actual parameters

Scope of Variables

- A scope is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed.

- There are two types of variables in C
  1) **local** variables
  2) **global** variables

Local Variables

- Variables that are declared inside a function are called local variables.
- They can be used only by statements that are inside that function.
  - ✓ In the following example all the variables a, b, and c are local to main() function.

```
#include <stdio.h>
 int main () {
   /* local variable declaration */
    int a, b, c;
    a = 10;  b = 20; c = a + b;
    printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
    return 0;
 }
```

Global Variables

- Global variables are defined outside a function, usually on top of the program.
- Global variables hold their values throughout the lifetime of the program and they can be accessed inside any of the functions defined for the program.

```c
#include <stdio.h>
int g; /* global variable declaration */
int main () {
  int a, b; /* local variable declaration */
  a = 10; b = 20; g = a + b;
  printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
  return 0;
 }
```

Functions- Categories

Categorization based on the arguments and return values

1. Functions with no arguments and no return values.
2. Functions with arguments and no return values.
3. Functions with arguments and one return value.
4. Functions with no arguments but return a value.
5. Functions that return multiple values

   (will see later with parameter passing techniques).

Function with No Arguments/parameters & No return values

```
void dispPattern(void); // prototype

int main(){
     printf(“fn to display a line of stars\n”);
     dispPattern();
     return 0;
}

void dispPattern(void ){
     int i;
     for (i=1;i<=20 ; i++)
          printf( “*”);
}
```

Function with No Arguments but A return value

```
int readNum(void); // prototype

int main(){
     int c;
     printf("Enter a number \n");
     c=readNum();
     printf("The number read is %d",c);
     return 0;
}
int readNum(){
     int z;
     scanf("%d",&z);
     return(z);
}
```

Fn with Arguments/parameters & No return values

```
void dispPattern(char ch); // prototype

int main(){
     printf("fn to display a line of patterns\n");
     dispPattern('#');
     dispPattern('*');
     dispPattern('@');
             return 0;
}
void dispPattern(char ch ){
     int i;
          for (i=1;i<=20 ; i++)
               printf("%c",ch);
     }
```

Function with Arguments/parameters & One return value

```
int main(){
        int a,b,c;
        printf("\nEnter numbers to be added\n");
        scanf("%d %d",&a,&b);
        c=fnAdd(a,b);
        printf("Sum is %d ", c);
        return 0;
}
int     fnAdd(int x, int y ){
        int z;
        z=x+y
        return(z);
}
```

Problems:

Write appropriate functions to

1. Find the factorial of a number ‘n’.
2. Reverse a number ‘n’.
3. Check whether the number ‘n’ is a palindrome.
4. Generate the Fibonacci series for given limit ‘n’.
5. Check whether the number ‘n’ is prime.
6. Generate the prime series using the function written for prime check, for a given limit.

Factorial of a given number *n*

```
long  factFn(int); //prototype

int main() {
     int n;
     long  int f;

    printf("Enter a number :");
    scanf("%d",&n);
    f =factFn(n);
    printf("Fact= %ld",f);

    return 0;
}
```

```
//function definition
long int factFn(int num) {
 int i;
    long int fact=1;

    //factorial computation
    for (i=1; i<=num; i++)
          fact=fact * i;

    // return the result
    return (fact);
}
```

Reversing a given number *n*

```
int Reverse(int); //prototype

int main()
{
  int n,r;
  printf("Enter a number : \n");
  scanf("%d", &n);

 r=  Reverse(n);
 printf(" reversed no=%d",r)

 return 0;
}
```

```
int Reverse(int num)
{
    int rev=0;
    int digit;

    while(num!=0)
    {
       digit = num % 10;
       rev = (10 * rev) + digit;
       num = num/10;
    }
    return (rev);
}
```

Check whether given number is prime or not

```
int IsPrime(int); //prototype

int main() {
  int n;

  printf("Enter a number : ");
  scanf("%d",&n);
  if (IsPrime(n))
     Printf("%d is a prime no",n);
  else
     printf("%d is not a prime no",n);
  return 0;
}
```

```
//prime check
int IsPrime(int num){
    int p=1;
    for(int j=2;j<=num/2;j++)
    {
        if(num%j==0)
        {
            p=0;
            break;
        }
   }
   return p;
}
```

First *n* Fibonacci number generation

```
void fibFn(int); //prototype

int main() {
   int n;
   printf("Enter the limit”);
   scanf(“%d”,&n);
   fibFn(n); //function call
   return 0;
}
```

```
void fibFn(int lim) { //fib generation
    int  i, first, sec, next;
    if (lim<=0)
      printf("limit should be +ve.\n“);
    else
    {
      printf("\nFibonacci nos\n“);
      first = 0, sec = 1;
      for (i=1; i<=lim; i++) {
         printf(“%d”, first)
         next = first + sec;
         first = sec;
         sec = next;
      }
    }
}
```

Functions- points to note

1. The parameter list must be separated by commas.

   `dispChar( int n, char c);`

2. The parameter names do not need to be the same in the prototype declaration and the function definition.
3. The types must match the types of parameters in the function definition, in number and order.

```
void dispChar(int n, char c); //proto-type
void dispChar(int num, char ch){
  printf(" You have entered %d &%c", num,ch);
     }
```

4. Use of parameter names in the declaration(prototype) is optional but parameter type is a must.

```
void dispChar(int , char); //proto-type
```

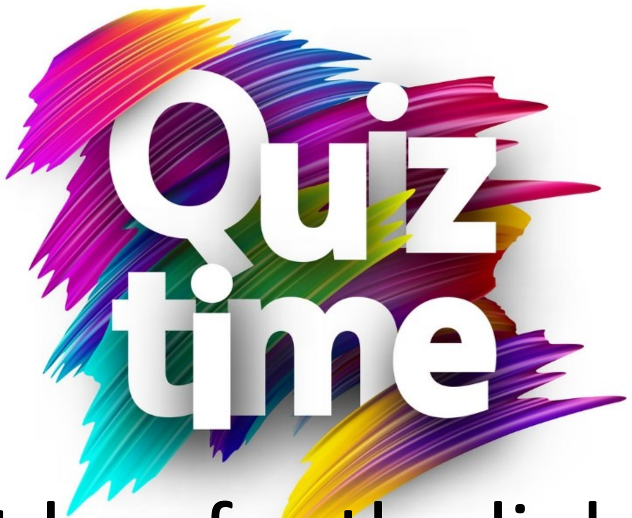
Functions- Parameter Passing

- Pass by value (call by value)
- Pass by reference (call by reference)

Pass by value:

```
                void swap(int x, int y )
                {
                        int t=x;
                        x=y;
                        y=t;
                        printf("In fn: x= %d and y=%d ",x,y);
                }
int main()
{
        int a=5,b=7;
        swap(a, b);
        printf("After swap:  a= %d and b= %d",a,b);
        return 0;
}
```

Output:
In fn: x = 7 & y = 5
After swap: a = 5 & b = 7



Go to posts/chat box for the link to the question

submit your solution in next 2 minutes

The session will resume in 3 minutes

Summary

- Scope of variable
- Write C Functions and how to invoke them
- Simple programs using functions
- Parameter Passing
- Pass by Value