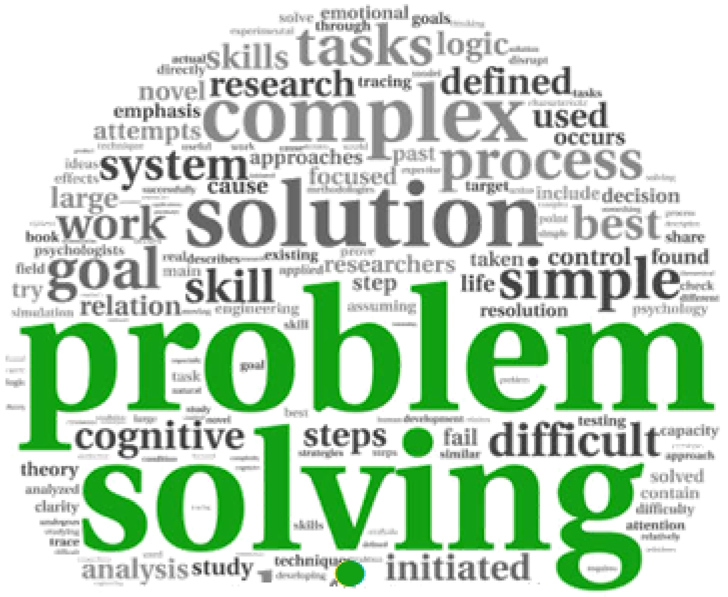

problem solving using computers

CSE 1051

S-18_2 PARAMETER PASSING TECHNIQUES

Objectives

To learn and appreciate the following concept:

- Pass by reference
- Returning multiple values from functions
- Nested Functions

Session outcome

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

- The overall ideology of parameter passing techniques

Functions-Overview:

Parameters/Arguments

Formal parameters

```
void dispNum( int n ) // function definition
   {
       printf(" The entered num=%d", n);
   }

int main(){   //calling program
    int no;
    printf("Enter a number \n");
    scanf("%d",&no);
   dispNum( no); //Function reference
return 0;
}
```

Actual parameters

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
```

Pass by Reference – Using Pointers:

```
void swap(int *x, int *y )
{
        int t=*x;
        *x=*y;
        *y=t;
}
```

Change is directly on the variable using the reference to the address.

When function is called:
address of a → x
address of b → y

Output:
After swap: a = 7 and b = 5

```
int main()
{
 int a=5,b=7;
 swap(&a, &b);
 printf("After swap: a=%d and b= %d",a,b);
 return 0; }
```

Pointers as functions arguments:

When we pass addresses to a function, the parameters receiving the addresses should be pointers.

```
int change (int *p)

{

 *p = *p + 10 ;

 return 0;

}
```

```
#include <stdio.h>

int main()

{

int x = 20;

 change(&x);

 printf("x after

change==%d",x);

 return 0;

}
```

Output :
X after change=30

Pointers as function arguments

- When the function change() is called, the address of the variable x, not its value, is passed into the function change().
- Inside change(), the variable p is declared as a pointer and therefore p is the address of the variable x. The statement
- *p=*p +10 adds 10 to the value stored at the address p. Since p represents the address of x, the value of x is changed from 20 to 30. therefore it prints 30.

Function that return multiple values-Using pointers

Using pass by reference we can write a function that return multiple values.

```
void fnOpr(int a, int b, int *sum, int *diff) {
   *sum = a + b;
   *diff = a -b;    }

   int main() {
   int x, y, s, d;
   printf("Enter two numbers: \n");
    scanf("%d %d",&x, &y);
    fnOpr(x, y, &s, &d);
    printf("Sum = %d & Diff =%d ", s, d);
    return 0; }
```

Output:

```
          x= 5 & y= 3
     Sum =8  & Diff = 2
```

Nesting of functions:

- C language allows nesting of functions by calling one function inside another function.
- Nesting of function does not mean that we can define an entire function inside another function. The following examples shows both valid and invalid function nesting in C language

```
// Wrong way of function nesting

void fun()
{
   printf("I am having Fun….");

    void sleep()
     {
         printf("I am having sleep");
      }
}
```

```
// Right way of function nesting

 void sleep()
{
         printf("I am having sleep");
 }

void fun()
{
   printf("I am having Fun….");
    sleep();
}
```

Nesting of Functions:

```
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;
}
```

Nesting of Functions:

```
void fnOpr(int a, int b, int *sum, int *diff)
{
  *sum = a + b;
  if (fnDiff(a,b))
    *diff = a -b;
  else
    *diff = b - a;
 }
 int main() {
 int x, y, s, d;
 printf(“Enter the values: \n“);
 scanf(“%d %d”, &x, &y);
 fnOpr(x, y, &s, &d);
 printf(“The results are, Sum =%d and Diff = %d“, s, d);
 return 0; }
```

```
int fnDiff(int p, int q) {
    if  (p>q)
      return(1);
    else
      return (0);}
```

```
Output:
          x= 3 & y= 5
          s =8  & d = 2
```

Go to posts/chat box for the link to the question **PQn. S18.2**

submit your solution in next 2 minutes

The session will resume in 3 minutes

Summary:

- Parameter passing techniques
    - pass by value
    - pass by reference
- Pointers as functions arguments
- Function returning Multiple values
- Nesting of functions