

CS19001: Programming and Data Structures Laboratory

Aritra Hazra;
CSE, IIT Kharagpur

<http://cse.iitkgp.ac.in/~aritrah/course/lab/PDS/Autumn2019/>

07-Sep-2019

Tutorial:
Functions

Tutorial:
Recursive
Functions

Table of Contents

1 Tutorial: Functions

2 Tutorial: Recursive Functions

CS19001:
Programming and
Data Structures
Laboratory

Aritra Hazra;
CSE, IIT
Kharagpur

Tutorial:
Functions

Tutorial:
Recursive
Functions

A mathematical function

$$\begin{aligned} &= 0 && \text{if } n = 0, \\ f(n) &= 2n - 1 && \text{if } n > 0, \\ &= -2n && \text{if } n < 0 \end{aligned}$$

- It would be nice to compute f such that

```
int main ()
{
    int n;
    while (1) {
        printf("Input n : "); scanf("%d",&n);
        printf("f(%d)=%d\n", n, f(n));
    }
}
```

And that is possible

```
int f ( int n )
{
    if (n == 0) return (0);
    else if (n > 0) return (2*n-1);
    else return (-2*n);
}

int main ()
{
    int n;
    while (1) {
        printf("Input n : "); scanf("%d",&n);
        printf("f(%d)=%d\n", n, f(n));
    }
}
```

Function Definition

```
return_type function_name (argument_list)
{
    function body
}
```

- Example

```
int gcd ( int a , int b )
{ int c;
  /* body */
  return c;
}

int main()
{ int x, y, z;
  /* body */
  z=gcd(x,y);
}
```

A more elaborate usage

```
int gcd ( int a , int b )
{
    int r;
    /* body */
    return r;
}

int main ()
{
    int i, j, s = 0;
    for (i=1; i<=20; ++i) {
        for (j=i; j<=20; ++j) {
            s += gcd(j,i);
        }
    }
    printf("The desired sum = %d\n", s);
}
```

Passing arguments by value

```
int gcd ( int a , int b )
{
    /*when called, a=j, b=i*/
    int r,...;
    /*any local assignment on a, b does not
    change i,j in main */

    ...../* body */
    return r;
}

int main ()
{
    .....
    s += gcd(j,i);
    .....
}
```

Passing arguments by value

```
int gcd ( int a , int b )
{
    int r,...; /* local variables like a,b*/
    /* r is not defined outside gcd() */

    ...../* body */
return r;
}
int main ()
{
    int r = 5; /* a different 'r' */
    .....
    s += gcd(j,i);
    .....
}
```


Passing arguments by value

```
int gcd ( int a , int b ) /*a=10,b=6*/
{
    int r,...;

    ...../* body */
return r; /*r=2*/
}
int main ()
{
    int r = 5;
    ...../*j=10,i=6,r= 5*/
    ...r=r+j.../*j=10,i=6,r=15*/
    s += gcd(j,i); /*s+=2*/
    ...../*j=10,i=6,r=15*/
}
```

Passing an array

When you pass an array, the computation effects the array passed by main

```
void bubble(int A[], int n)
{ /*no return type*/
    int c,d,temp;
    for (c=n-2; c>=0; --c){
        for (d=0; d<=c; ++d){
            if (A[d] > A[d+1]){
                temp = A[d];
                A[d] = A[d+1];
                A[d+1] = temp;
            }
        }
    }
}
```

We shall explain the reason later

Passing an array

```
int main()
{
    int array[100], n, k;
    printf("Enter no. of elements\n");
    scanf("%d", &n);
    printf("Enter %d integers\n", n);
    for (k = 0; k < n; k++)
        scanf("%d", &array[k]);
    bubble(array, n); /*array is passed*/
    printf("Sorted list\n");
    for ( k = 0 ; k < n ; k++ )
        printf("%d,", array[k]);
    printf("\n");
    return 0; /*sorting reflected on array[]*/
}
```

Table of Contents

1 Tutorial: Functions

2 Tutorial: Recursive Functions

CS19001:
Programming and
Data Structures
Laboratory

Aritra Hazra;
CSE, IIT
Kharagpur

Tutorial:
Functions

Tutorial:
Recursive
Functions

The well known Fibonacci function

$$\begin{aligned} &= 0 && \text{if } n = 0, \\ f(n) &= 1 && \text{if } n = 1, \\ &= f(n-1) + f(n-2) && \text{if } n \geq 2 \end{aligned}$$

- Similarly, many other well known functions can be defined *recursively*, i.e., in terms of itself

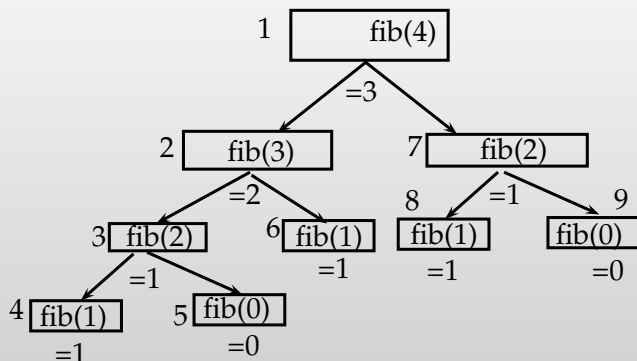
```
int fib ( int n )
{
    if (n == 0) return (0);
    if (n == 1) return (1);
    return (fib(n-1)+fib(n-2));
}
```

Why does this work

```
int fib ( int n )
{
    if (n == 0) return (0);
    if (n == 1) return (1);
    return (fib(n-1)+fib(n-2));
}
```

- Each call instance of fib works with its own copy of n
- The computer “remembers” every previous state of the problem. This information is “held” by the computer on the “activation stack” (i.e., inside of each function’s workspace).

The recursive call sequence

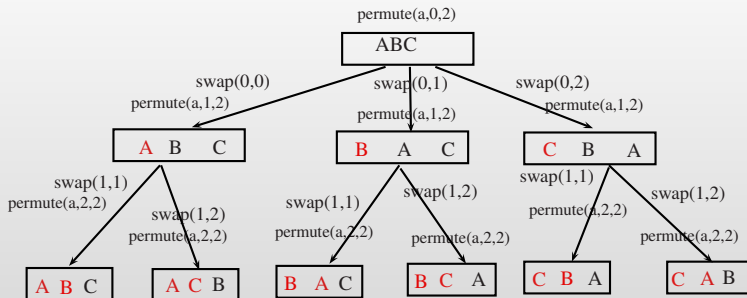


- The call sequence: 1,2,3,...
- The return sequence: 4,5,3,6,2,8,9,7,1

Recursive function for printing all permutations of a given string

```
void permute(char a[], int i, int n)
{ // i=current start index
    int j;
    if (i == n) printf("%s\n", a);
    else{
        for (j = i; j <= n; j++){
            swap(a[i], a[j]);
            permute(a, i+1, n);
            swap(a[i], a[j]); //backtrack
        }
    }
}
```


The recursive function call sequence



- Before each function call, position of a letter is fixed (marked in red) after swapping
- After any call returns, swapping is again performed to restore state
- Printing is done at leaf level

Thank You

CS19001:
Programming and
Data Structures
Laboratory

Aritra Hazra;
CSE, IIT
Kharagpur

Tutorial:
Functions

Tutorial:
Recursive
Functions