Basis and Practice in Programming

Chapter 9: Functions

Prof. Tamer ABUHMED
College of Software

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

Lecture Objectives

- Introduce functions
- Introduce functions with an array argument
- Introduce functions with arguments as implicit outputs
- Introduce functions that can modify the values of their arguments
- Keywords

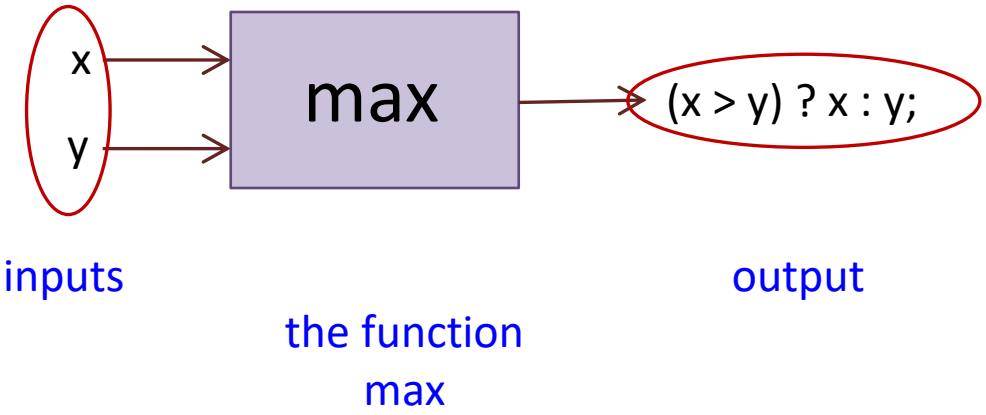
Functions

- What is a function?
  - It is a self-contained unit of program code designed to accomplish a particular task.
- Examples
  - printf(), scanf(), sizeof(), rand(), srand(), getchar(), putchar(), pow()
- Can we create our own functions?
  - Yes, Sure!
- Why do we need functions?
  - Make you code organized
  - Make your code reusable
  - Make your code easily debuggable
  - Make your code shorter by simply calling a function several times
    - Avoid repetition

Functions – contd.

- Again what is a function?
  - It is a black box that has a name, input(s), output(s) or without, and type
    - A function can be without inputs or outputs or none of them
  - **Function type**
    - It can be any variable type, void, or a pointer to a variable

- Example
  - int max(int x, int y);



Functions – contd.

- Example 1: A void function

```
# include <stdio.h>

void main(){
     int ii;
     char name[20];

     printf("Please, enter your name: ");
     scanf("%s", name);

     for(ii = 0; ii < 20; ii++)
①      putchar('-');

     printf("\n");

     printf("your name is %s\n", name);

     for(ii = 0; ii < 10; ii++)
②        putchar('-');

     printf("\n");

}
```

Output

```
Please, enter your name: Huey
--------------------
Your name is Huey
--------------------
```

```
// a void function
# include <stdio.h>

// prototype (function declaration)
void dash();

void main(){

     char name[20];

     printf("Please, enter your name: ");
     scanf("%s", name);

     dash();      // function call
     printf("your name is %s\n", name);
     dash();      // function call
}

void dash()      // function definition
{
    int ii = 0;
    for(ii = 0; ii < 20; ii++)
       putchar('-');

    printf("\n");
}
```

Functions – contd.

- Example 1 – What is new?
  - **Function declaration**
    - Tells the compiler the function's name (example: dash)
    - Tells what type of function is dash()
    - What arguments does it take and what outputs does it return, if any.
    - **Example:** void dash();
      - The function's name is "dash"
      - It is of type void (does not have outputs), and does not have inputs
  - **Function call**
    - Causes the function to be executed.
  - **Function definition**
    - Specifies what the function does.

```
// a void function
# include <stdio.h>

// prototype (function declaration)
void dash();

void main(){

     char name[20];

     printf("Please, enter your name: ");
     scanf("%s", name);

     dash();      // function call
     printf("your name is %s\n", name);
     dash();      // function call
}

void dash()      // function definition
{
    int ii = 0;
    for(ii = 0; ii < 10; ii++)
       putchar('-');

    printf("\n");
}
```

Functions – contd.

- Example 2: A function with integer output and input
  - Find the sum of integers from 1 up to an integer input by the user.

```
# include <stdio.h>

// prototype (function declaration)
int sum(int);       // same as  - int sum(int a); -

void main(){
    int limit = 0, out = 0;

    printf("Enter an integer: ");
    scanf("%d", &limit);

    out = sum(limit);     // function call
    printf("The sum from 1 to %d = %d\n", limit, out);

}

int sum(int count){      // function definition
    int ii = 0, result = 0;

    for(ii = 1; ii <= count; ii++)
        result += ii;

    return result;
}
```

$$\text{result} = \sum_{ii=1}^{\text{count}} ii$$

Output

```
Enter an integer: 5
The sum from 1 to 5 = 15
```

Functions – contd.

- Example 2: What is new?
  - Arguments are placed between the parentheses
    - int sum(int count);
      - The function has one argument of type integer (count)
      - The function returns an integer (function's type) → we must use the return statement
    - void main(void)
      - The function does not have any argument.
      - It does not return any value
  - Types of arguments
    - **Formal argument**
      - A variable at the definition of the function such as: int sum(int count){ … }
      - count is a formal argument
    - **Actual argument**
      - The variable or value passed to the function at the calling time:
        out = sum(limit); out = sum(5);
      - limit and 5 are actual arguments

Functions – contd.

- Example 2: What is new?
  - The following function calls use *actual arguments*
    - out = sum(rand());
    - out = sum(17 – x);
    - out = sum(pow(x, 2));
    - out = sum(rand() % 11);
    - etc.

Functions – contd.

- Example 3: A function with multiple arguments

```
# include <stdio.h>
# include <math.h>
# define N 10
// prototype (function declaration)
double POLY(double a, int b);

void main(){
    double r = 0., out = 0.;

    printf("Enter a double number: ");
    scanf("%lf", &r);

    out = POLY(r, N);     // function call
    printf("The result = %.3lf\n", out);

}

double POLY(double r, int n){      // function definition
    int ii = 0;
    double result = 0;

    for(ii = 1; ii <= N; ii++)
        result += pow(r, ii);

    return result;
}
```

$$\text{result} = \sum_{ii=1}^{N} r^{ii}$$

Output

```
Enter a double number: 0.5
The result = 0.999
```

Functions – contd.

- Example 4: A function with string argument

```
# include <stdio.h>

// prototype (function declaration)
int str_length(char a[]);

void main(){
    char name[30];
    int len = 0;

    printf("Please enter your name: ");
    scanf("%s", name);

    len = str_length(name);

    printf("Your name is %s of length %d\n", name, len);
}

int str_length(char a[])
{
    int count = 0;
    while(a[count] != '\0')
          count++;

    return count;
}
```

Output

```
Please enter your name: Freeman
Your name is Freeman of 7.
```

A Function with an Array as Argument

- Example 1
  - Generate a binary sequence and count the number of ones in it
    - The argument "int a[]" is used as input and output

```
// generate a binary sequence and find the
// probability of ones
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
# define N 20

int binary(int a[], int n); // (function declar.)

void main()
{
    int count = 0, bin[N];
    count = binary(bin, N); // function call

    printf("Probability of ones is %0.3lf\n",
            (double)count / N);
}
```

```
// function definition
int binary(int seq[], int n)
{
    int ii = 0, counter = 0;
    srand(time(0));

    for(ii = 0; ii < n; ii++){
        seq[ii] = rand() % 2;
        if(seq[ii] == 1)
            counter++;
    }

    return counter;
}
```

Output (N = 20)

```
Probability of ones is 0.300
```

Output (N = 20000)

```
Probability of ones is 0.497
```

A Function with an Array as Argument – contd.

- Example 2: Vector operations

```c
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
# define N 20

void init(int a[], int n);
void randarray(int a[], int n);
void printarray(int vec1[], int vec2[], int len);
void power(int in[], int out[], int size);

void main()
{
     int seq1[N], seq2[N];

     init(seq1, N);
     init(seq2, N);

     printarray(seq1, seq2, N);

     randarray(seq1, N);
     power(seq1, seq2, N);

     printarray(seq1, seq2, N);

}
```

```c
void init(int in[], int K){
    int ii = 0;
    for(ii = 0; ii < K; ii++)
        in[ii] = 0;
}

void randarray(int a[], int n){
    int ii = 0;

    srand(time(0));
    for(ii = 0; ii < n; ii++)
        a[ii] = rand() % 21;
}

void printarray(int vec1[], int vec2[], int len){
    int ii = 0;
    for(ii = 0; ii < len; ii++)
        printf("%d\t%d\n", vec1[ii], vec2[ii]);

    printf("\n");
}

void power(int in[], int out[], int n){
    int ii = 0;

    for(ii = 0; ii < n; ii++)
        out[ii] = in[ii] * in[ii];
}
```

A Function with an Array as Argument – contd.

- Example 2: Vector operations – contd.
  - void init(int a[], int n)
    - The function is void; no return value
    - The argument **a[]** is an array
      - The array is passed by address
      - Therefore, we can modify its value
    - The argument **n** is a single-variable
      - The variable is passed by value
      - We can't modify its value

```
      ...

# define N 20

void init(int a[], int n);
void randarray(int a[], int n);
void printarray(int vec1[], int vec2[], int len);
void power(int in[], int out[], int size);

void main()
{
    ...

}
```

```
void init(int in[], int K){
    int ii = 0;
    for(ii = 0; ii < K; ii++)
        in[ii] = 0;    // initiate all elements to 0
}

void randarray(int a[], int n){
    int ii = 0;

    srand(time(0));
    for(ii = 0; ii < n; ii++)
        a[ii] = rand() % 21;
}

void printarray(int vec1[], int vec2[], int len){
    int ii = 0;
    for(ii = 0; ii < len; ii++)
        printf("%d\t%d\n", vec1[ii], vec2[ii]);

    printf("\n");
}

void power(int in[], int out[], int n){
    int ii = 0;

    for(ii = 0; ii < n; ii++)
        out[ii] = in[ii] * in[ii];
}
```

A Function with an Array as Argument – contd.

- Example 2: Vector operations – contd.
  - void randarray(int a[], int n)
    - The argument **a[]** is an array
      - The array is passed by address
      - Therefore, we can modify its value
    - The argument **n** is a single-variable
      - The variable is passed by value
      - We can't modify its value

```
      ...

# define N 20

void init(int a[], int n);
void randarray(int a[], int n);
void printarray(int vec1[], int vec2[], int len);
void power(int in[], int out[], int size);

void main()
{
    ...

}
```

```
void init(int in[], int K){
    int ii = 0;
    for(ii = 0; ii < K; ii++)
        in[ii] = 0;
}

void randarray(int a[], int n){
    int ii = 0;

    srand(time(0));
    for(ii = 0; ii < n; ii++)
        a[ii] = rand() % 21; // generate random num.
}

void printarray(int vec1[], int vec2[], int len){
    int ii = 0;
    for(ii = 0; ii < len; ii++)
        printf("%d\t%d\n", vec1[ii], vec2[ii]);

    printf("\n");
}

void power(int in[], int out[], int n){
    int ii = 0;

    for(ii = 0; ii < n; ii++)
        out[ii] = in[ii] * in[ii];
}
```

A Function with an Array as Argument – contd.

- Example 2: Vector operations – contd.
  - void printarray(int a[], int b[], int len)
    - **a[]** and **b[]** are arrays
      - The array is passed by address
      - Therefore, we can modify their value
      - However, this func. doesn't modify them
    - The argument **len** is a single-variable
      - The variable is passed by value
      - We can't modify its value

```
      ...

# define N 20

void init(int a[], int n);
void randarray(int a[], int n);
void printarray(int vec1[], int vec2[], int len);
void power(int in[], int out[], int size);

void main()
{
     ...

}
```

```
void init(int in[], int K){
    int ii = 0;
    for(ii = 0; ii < K; ii++)
        in[ii] = 0;
}

void randarray(int a[], int n){
    int ii = 0;

    srand(time(0));
    for(ii = 0; ii < n; ii++)
        a[ii] = rand() % 21; // generate random num.
}
// print the elements of two arrays of the screen
void printarray(int vec1[], int vec2[], int len){
    int ii = 0;
    for(ii = 0; ii < len; ii++)
        printf("%d\t%d\n", vec1[ii], vec2[ii]);

    printf("\n");
}

void power(int in[], int out[], int n){
    int ii = 0;

    for(ii = 0; ii < n; ii++)
        out[ii] = in[ii] * in[ii];
}
```

A Function with an Array as Argument – contd.

- Example 2: Vector operations – contd.
  - void power(int in[], int out[], int len)
    - **in[]** and **out[]** are arrays
      - The array is passed by address
      - Therefore, we can modify their value
      - However, this func. doesn't modify in[]
    - The argument **size** is a single-variable
      - The variable is passed by value
      - We can't modify its value

```
      ...

# define N 20

void init(int a[], int n);
void randarray(int a[], int n);
void printarray(int vec1[], int vec2[], int len);
void power(int in[], int out[], int size);

void main()
{
    ...

}
```

```
void init(int in[], int K){
    int ii = 0;
    for(ii = 0; ii < K; ii++)
        in[ii] = 0;
}

void randarray(int a[], int n){
    int ii = 0;

    srand(time(0));
    for(ii = 0; ii < n; ii++)
        a[ii] = rand() % 21; // generate random num.
}

void printarray(int vec1[], int vec2[], int len){
    int ii = 0;
    for(ii = 0; ii < len; ii++)
        printf("%d\t%d\n", vec1[ii], vec2[ii]);

    printf("\n");
}
// out[ii] = in[ii]^2
void power(int in[], int out[], int n){
    int ii = 0;

    for(ii = 0; ii < n; ii++)
        out[ii] = in[ii] * in[ii];
}
```

How Can a Function Modify its Argument?
A little on pointers

- scanf() revisited ☺
  - Example
    - char name[10];
    - int size;

    - scanf("%s", name);
      - name is an address, so the string name can be modified by scanf()
    - scanf("%d", size);
      - WRONG because scanf() needs the address of size to modify its value
      - So, scanf("%d", &size); is the correct:
        - » remember that &size is the address where the variable size is stored in memory

  - **The question is**, how to create a function that can modify the value of a single-variable argument?
    - This can be done using pointers (addresses)

How Can a Function Modify its Argument?
A little on pointers – contd.

- Let's try to
  - Find num^n (num to power n)

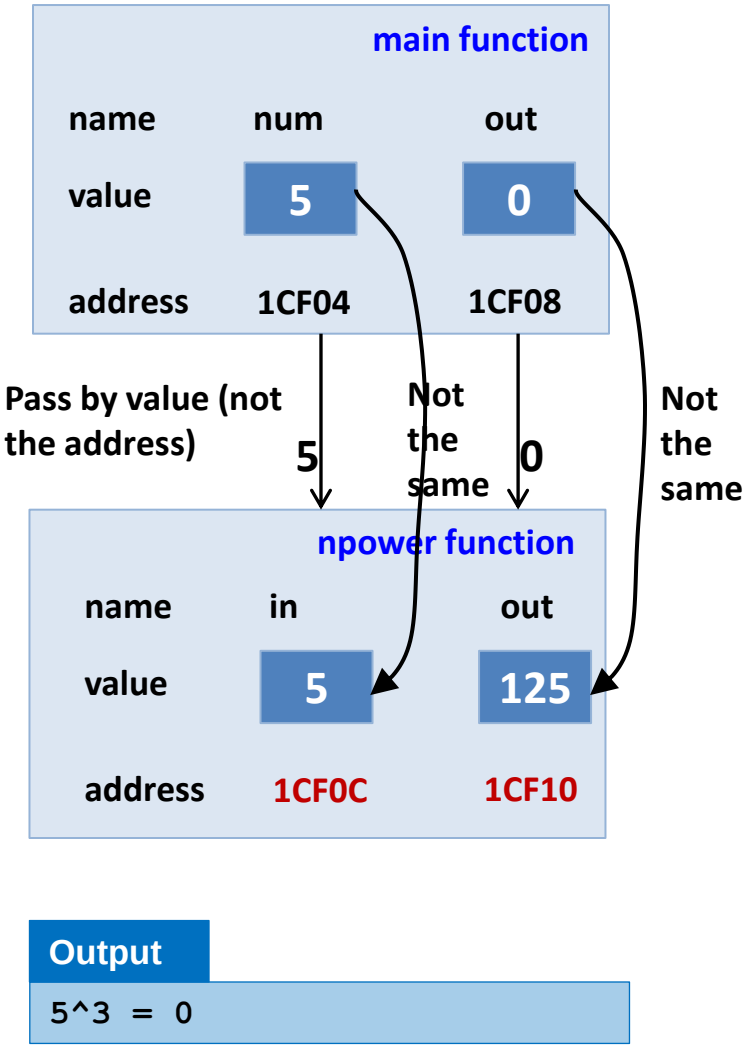
```
# include <stdio.h>
# define N 3

// function declaration
void npower(int in, int  out, int n);

void main()
{
    int num = 5, out = 0;
    npower(num, out, N);    // function call

    printf("%d^%d = %d\n", num, N, out);
}

// function definition
void npower(int in, int out, int n)
{
    int ii;
    (out) = 1;
    for(ii = 0; ii < n; ii++)
       (out) = (out) * in;
}
```



Output

```
5^3 = 0
```

How Can a Function Modify its Argument?
A little on pointers – contd.

- Let's try, again, to
  - Find num^n (num to power n)

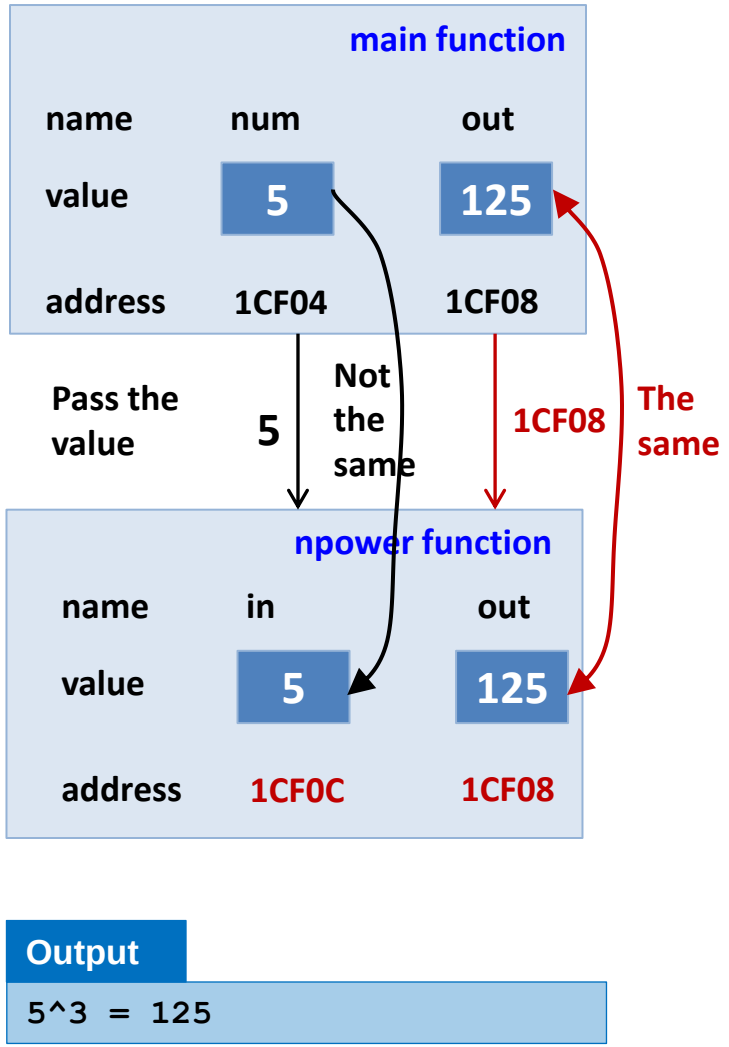
```
# include <stdio.h>
# define N 3

// function declaration
void npower(int in, int  * out, int n);

void main()
{
    int num = 5, out = 0;
    npower(num, &out, N);    // function call

    printf("%d^%d = %d\n", num, N, out);
}

// function definition
void npower(int in, int * out, int n)
{
    int ii;
    *(out) = 1;
    for(ii = 0; ii < n; ii++)
       *(out) = *(out) * in;
}
```



Output

```
5^3 = 125
```

Lecture Keywords

- Keywords
  - Function declaration (prototype)
  - Function call
  - Function definition
  - Actual argument
  - Formal argument
  - The return statement
  - Passing by value
  - Passing by address

Lecture Summary

- Introduced the functions in C language
- Introduced functions with an array argument
- Introduced functions with arguments as implicit outputs
- Introduced functions that can modify the values of their arguments