Programming Fundamental

Week 12 –

Procedure and Function in the C Programming Language

Dr. Maria Irmina Prasetiyowati, S.Kom,. M.T.
Alethea Suryadibrata, S.Kom., M.Eng.
Putri Sanggabuana Setiawan, S.Kom, M.T.I.
Januar Wahjudi, S.Kom., M.Sc.
Drs Slamet Aji Pamungkas, M.Eng
Kursehi Falgenti S.Kom., M.Kom.

Weekly Learning Outcomes for Subjects (Sub-CPMK):

Sub-CPMK 0614: Students are able to create simple programs with elements of selection control, repetition control, functions, or procedures, as well as implementing arrays and pointers in the C programming language (C6).

Review

1. For
2. While…
3. Do…While…
4. Break
5. Continue

Outline

1. Introduction
2. Standard Library
3. Programmer-Defined Functions
4. Recursion

Introduction

- The best way to develop and maintain a large program is to construct it from smaller pieces or **modules**
- This technique is called **divide and conquer**
- More manageable

WHY ?

- Modularity
- Readibility
- Code Reusability

Program Modules in C

- Modules in C are called **functions**
  - C Standard Library
    - Prepackaged functions
    - Provides a rich collection of functions for performing common mathematical calculations, string manipulations, character manipulations, input/output, and many other useful operations
  - Programmer-defined functions
    - Programmer can write functions to define specific tasks that may be used at many points in a program

Standard Library Headers

- Each standard library has a corresponding header containing the function prototypes for all the functions in that library and definitions of various data types and constants needed by those functions

Standard Library Header	Explanation
<stdio.h>	Standard input / output library functions
<math.h>	Math library functions
<string.h>	String-processing functions
<time.h>	Time and date manipulation functions
<stdlib.h>	Conversions of numbers to text and text to numbers, memory allocation, random numbers, and other utility functions
<ctype.h>	Functions that test characters for certain properties Functions that can be used to convert lowercase letters to uppercase letters and vice versa

Programmer-defined Functions

- Function Definitions

```
return_value_type function_name(parameter_list)
{
        statement;
}
```

- The **function_name** is any valid identifier
- The **return_value_type** is the data type of the result returned to the caller
- The **parameter_list** is a comma-separated list that specifies the parameters received by the function when it is called
- Together, the **return_value_type**, **function_name,** and **parameter_list** are sometimes referred to as the **function header**
- The statement within braces form the **function body**

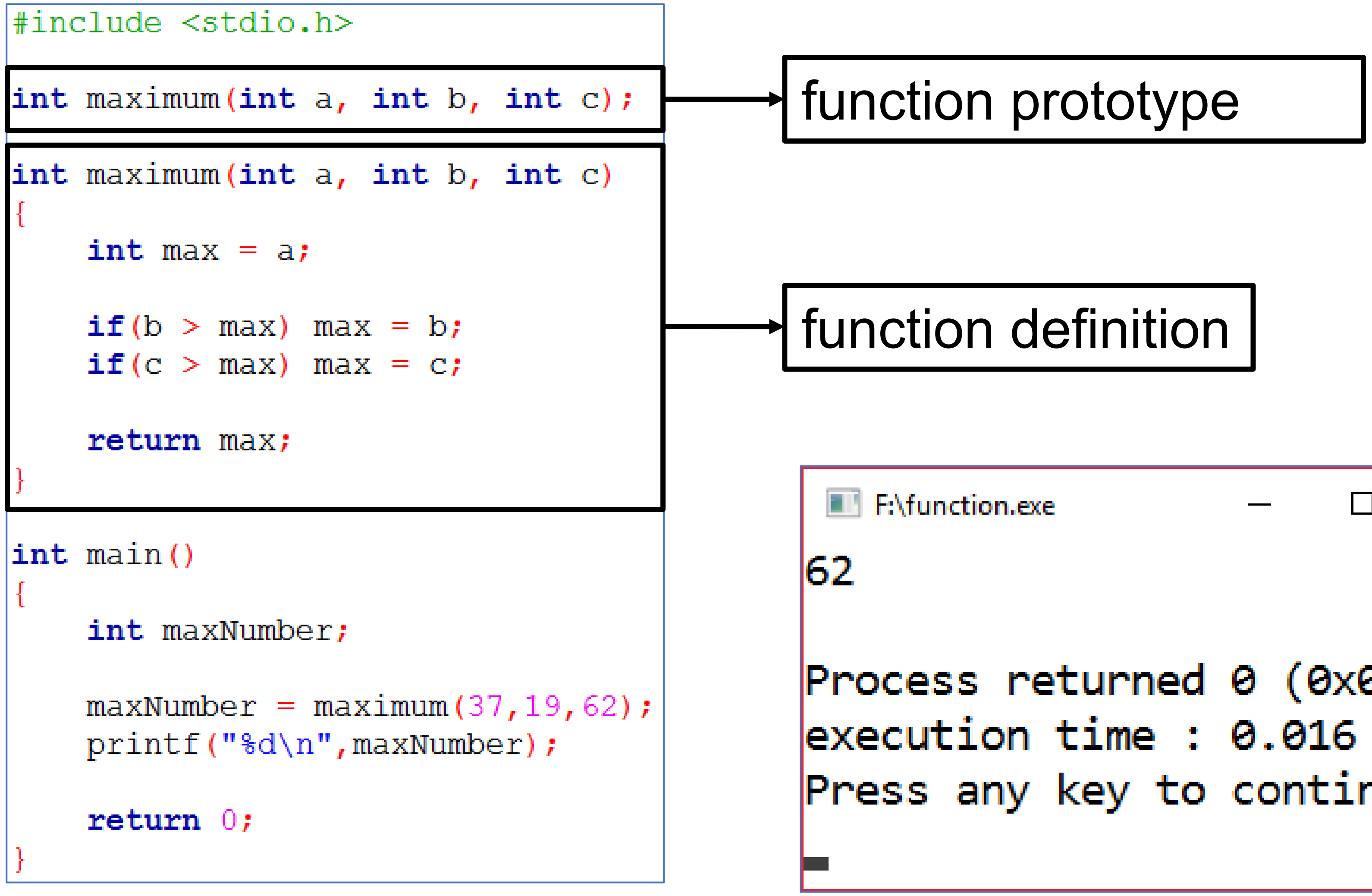
Programmer-defined Functions

- Function Prototypes

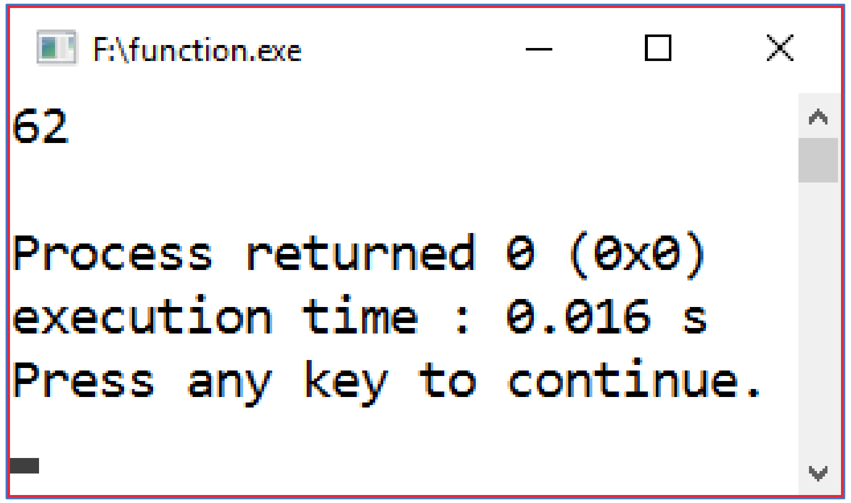
```
return_value_type function_name(parameter_list);
```

- A function prototype tells the compiler
  - The type of data returned by the function
  - The number of parameters the function expects to receive
  - The types of the parameters
  - The order in which these parameters are expected
- The compiler uses function prototypes to validate function calls
- A function call that **does not match** the function prototype is a **syntax error**
- An error is also generated if the function prototype and the function definition disagree

Programmer-defined Functions



```
#include <stdio.h>

int maximum(int a, int b, int c);

int maximum(int a, int b, int c)
{
    int max = a;

    if(b > max) max = b;
    if(c > max) max = c;

    return max;
}

int main()
{
    int maxNumber;

    maxNumber = maximum(37,19,62);
    printf("%d\n",maxNumber);

    return 0;
}
```



F:\function.exe

```
62

Process returned 0 (0x0)
execution time : 0.016 s
Press any key to continue.
```

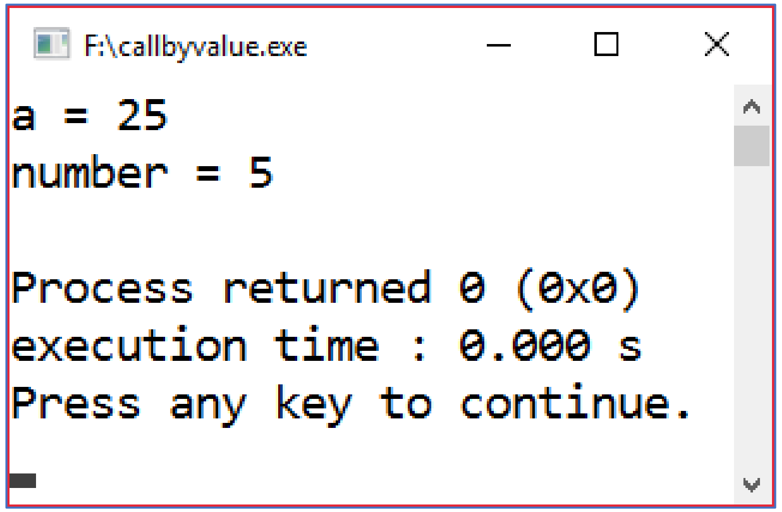
Calling Functions

- Call-by-Value
  - When arguments are passed by value, a copy of the argument’s value is made and passed to the called functions
  - Changes to the copy do not affect an original variable’s value in the caller
  - In C, all calls are by value

Calling Functions

- Call-by-Value

```
#include <stdio.h>

void square(int a)
{
    a *= a;
    printf("a = %d\n",a);
}

int main()
{
    int number = 5;

    square(number);
    printf("number = %d\n",number);

    return 0;
}
```



F:\callbyvalue.exe

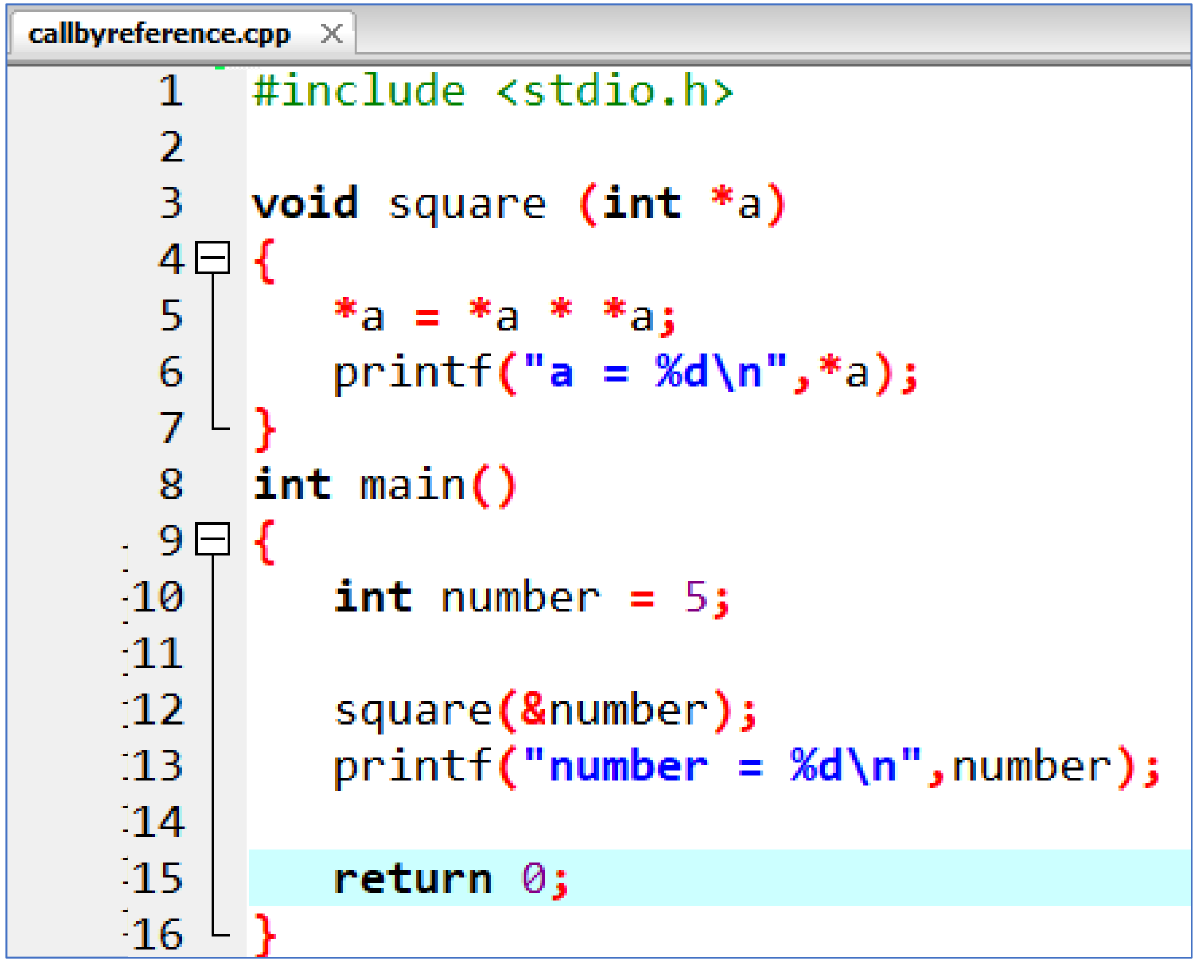
```
a = 25
number = 5

Process returned 0 (0x0)
execution time : 0.000 s
Press any key to continue.
```

Calling Functions

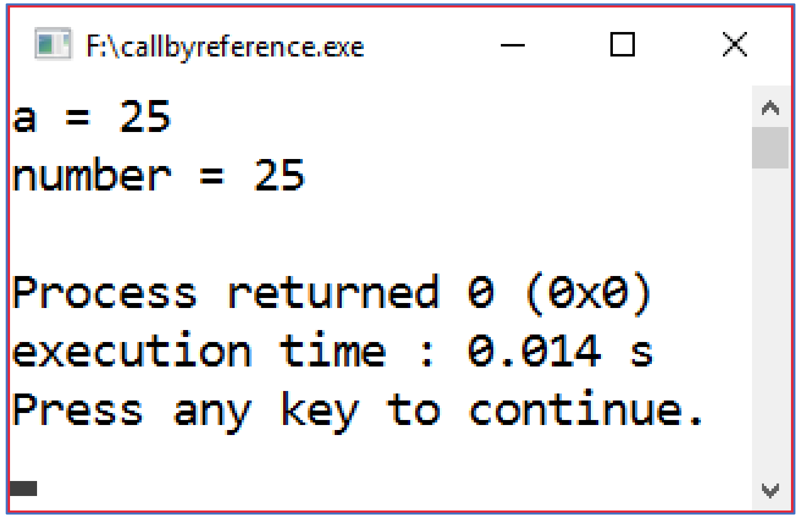
- Call-by-Reference
  - When an argument is passed by reference, the caller allows the called function to modify the original variable’s value

Calling Functions

- Call-by-Reference


callbyreference.cpp

```c
#include <stdio.h>

void square (int *a)
{
    *a = *a * *a;
    printf("a = %d\n",*a);
}
int main()
{
    int number = 5;

    square(&number);
    printf("number = %d\n",number);

    return 0;
}
```


F:\callbyreference.exe

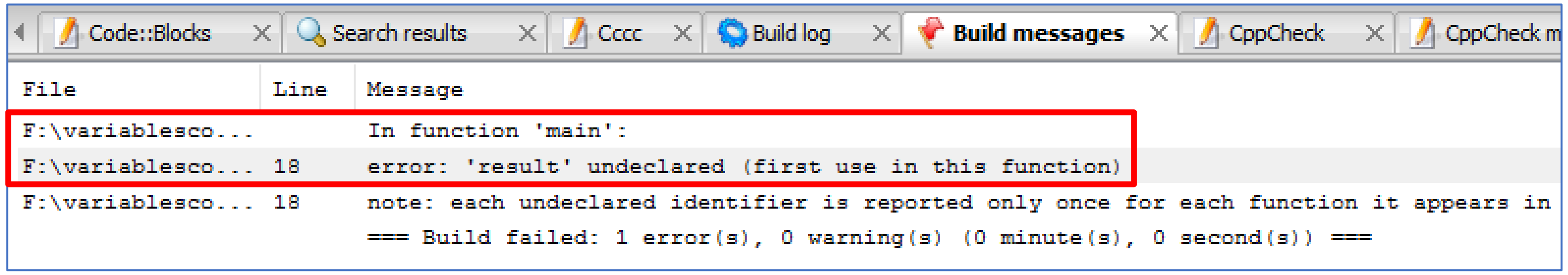
```
a = 25
number = 25

Process returned 0 (0x0)
execution time : 0.014 s
Press any key to continue.
```

Variable Scope

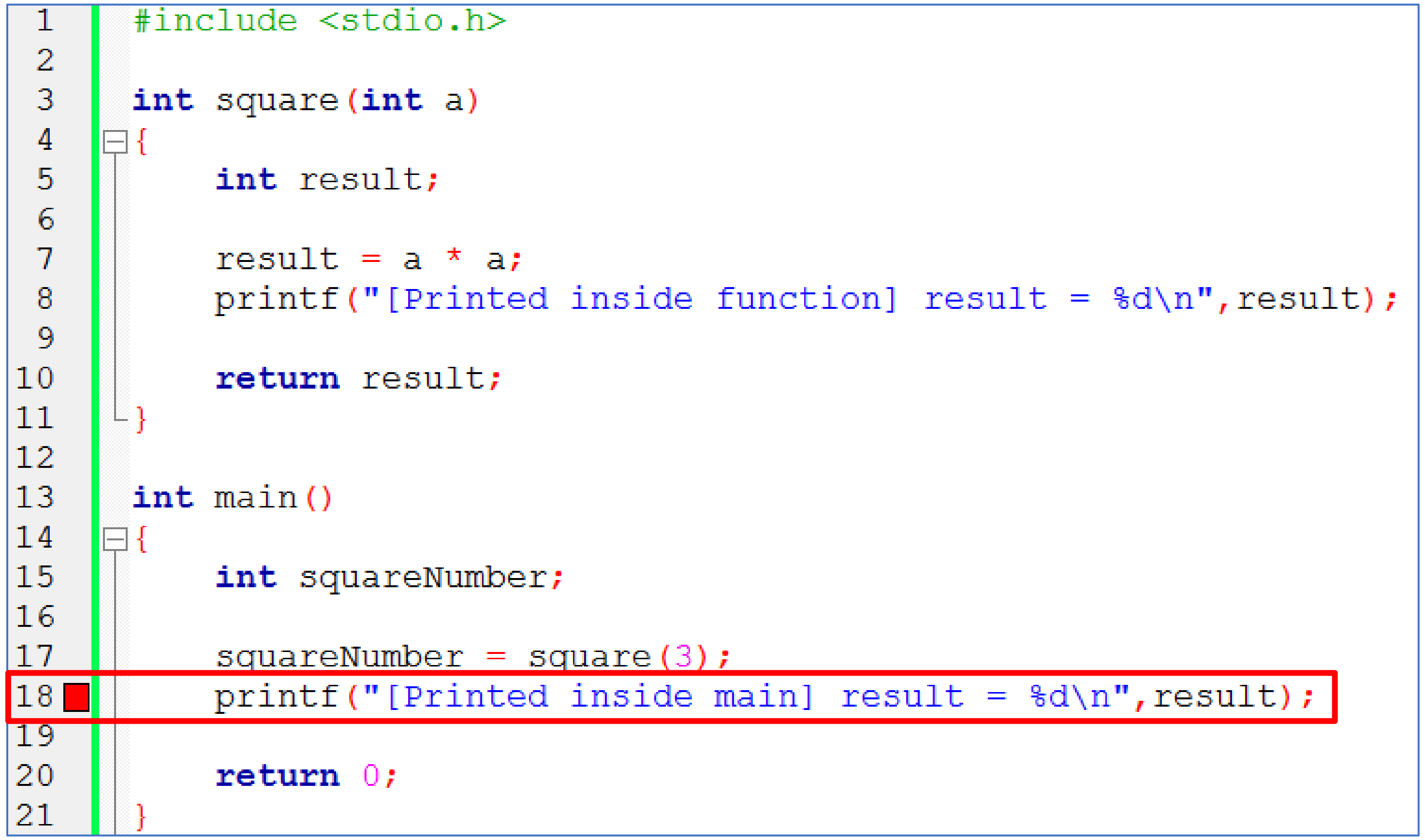
- Local Variables
  - All variables defined in function definitions are local variables
  - Known only in the function in which they are defined
  - A function’s parameters are also local variables of that function

Variable Scope

- Local Variables



```
File               Line   Message
F:\variablesco...         In function 'main':
F:\variablesco...  18     error: 'result' undeclared (first use in this function)
F:\variablesco...  18     note: each undeclared identifier is reported only once for each function it appears in
                          === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```



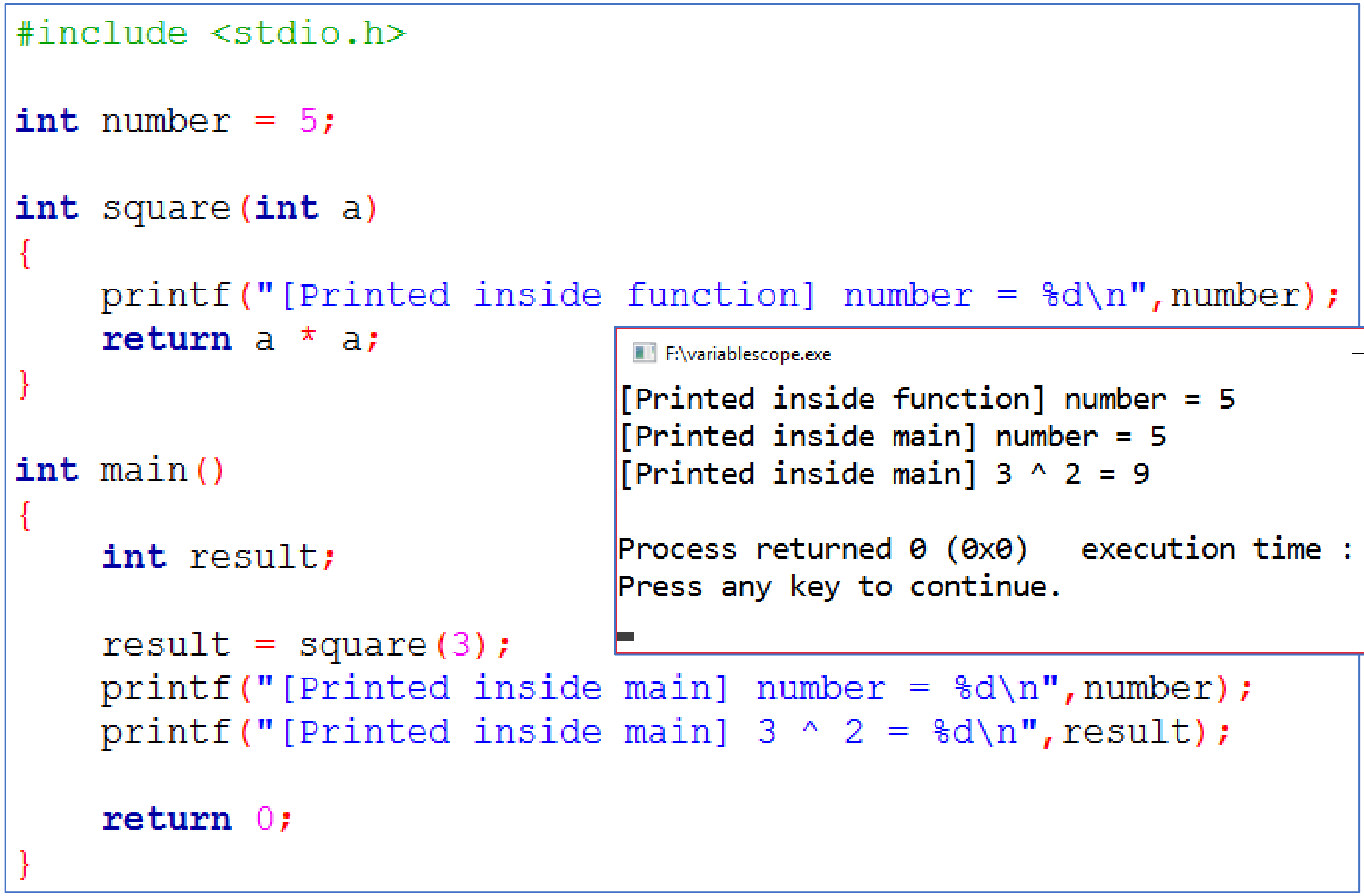
```c
#include <stdio.h>

int square(int a)
{
    int result;

    result = a * a;
    printf("[Printed inside function] result = %d\n",result);

    return result;
}

int main()
{
    int squareNumber;

    squareNumber = square(3);
    printf("[Printed inside main] result = %d\n",result);

    return 0;
}
```

Variable Scope

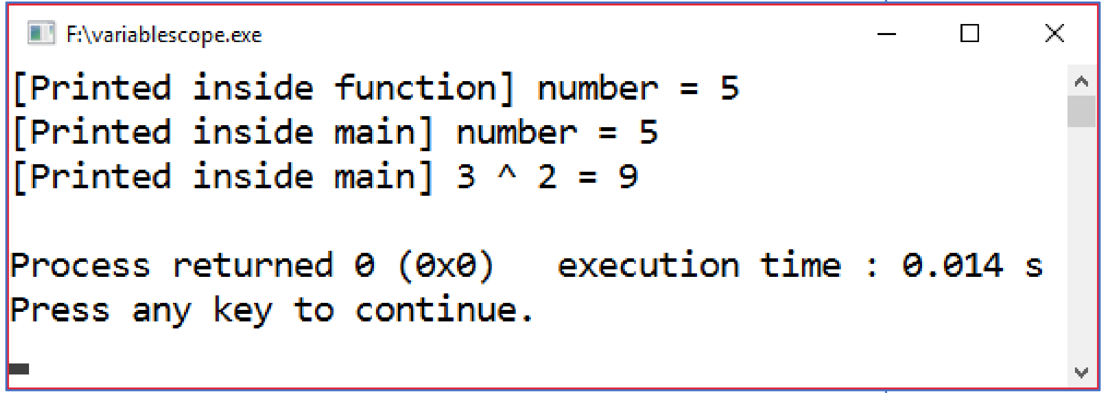
- Global Variables
  - Created by placing variable declarations **outside** any function definition
  - Retain their values throughout the execution of the program
  - Can be referenced by any function that follows their declarations or definitions in the file

Variable Scope

- Global Variables



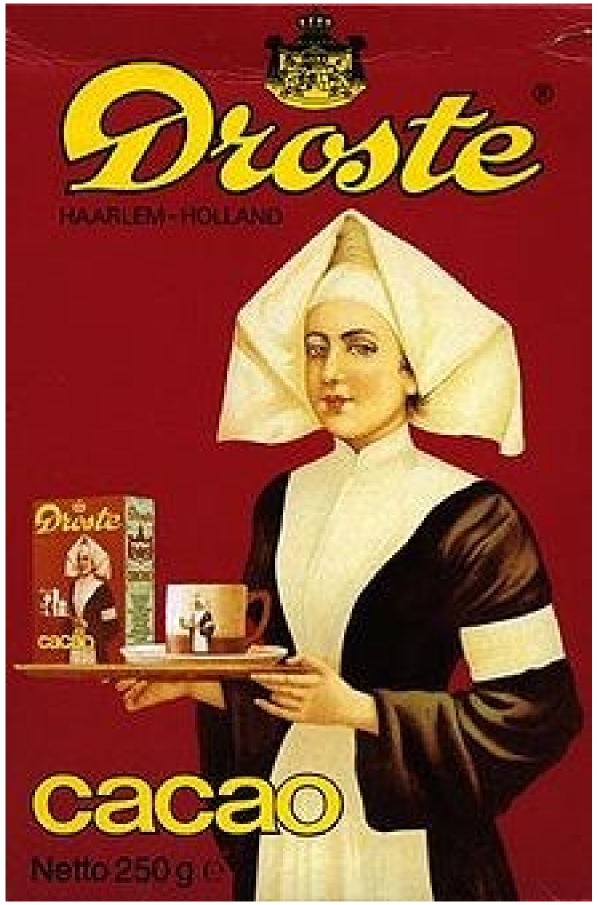
```c
#include <stdio.h>

int number = 5;

int square(int a)
{
    printf("[Printed inside function] number = %d\n",number);
    return a * a;
}

int main()
{
    int result;

    result = square(3);
    printf("[Printed inside main] number = %d\n",number);
    printf("[Printed inside main] 3 ^ 2 = %d\n",result);

    return 0;
}
```



F:\variablescope.exe

```
[Printed inside function] number = 5
[Printed inside main] number = 5
[Printed inside main] 3 ^ 2 = 9

Process returned 0 (0x0)   execution time : 0.014 s
Press any key to continue.
```

Recursion



- A **recursive function** is a function that calls itself

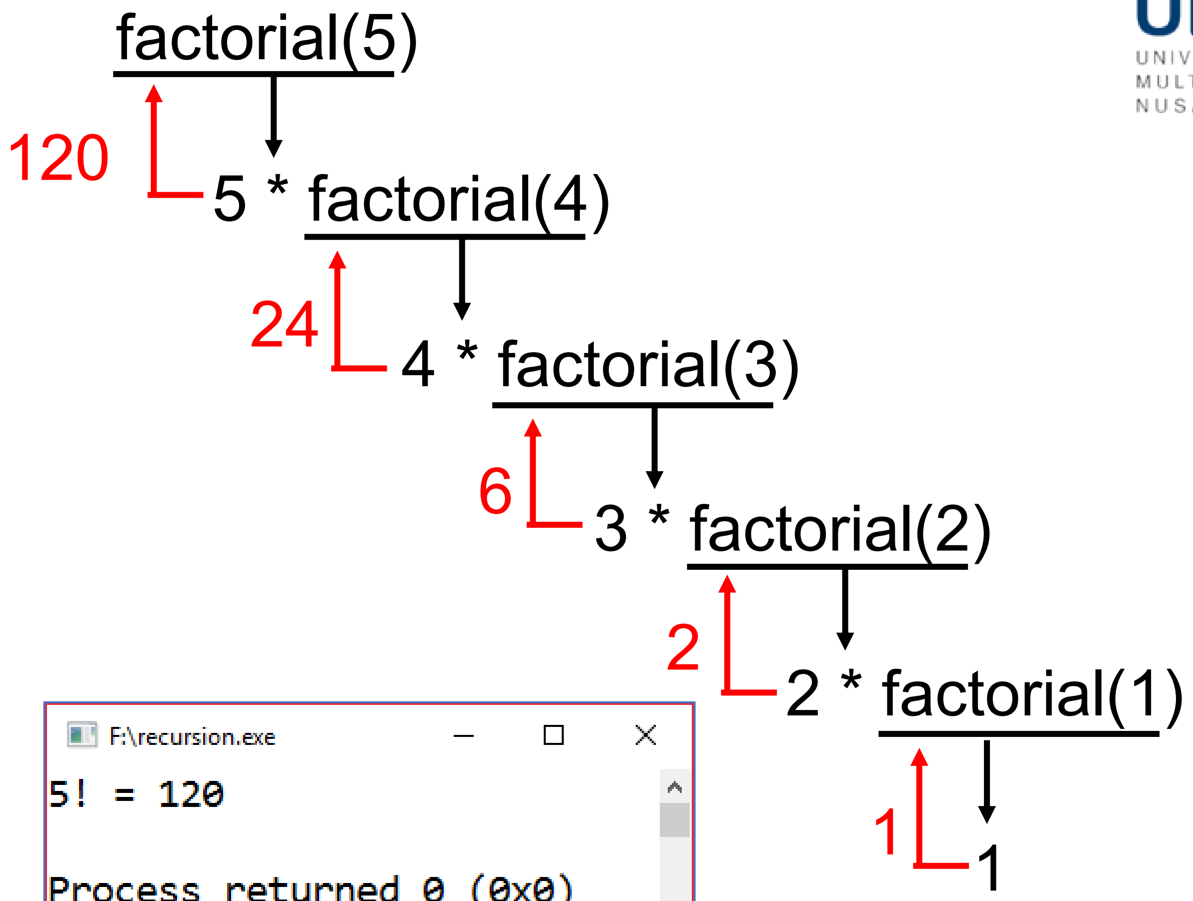
$$\begin{tikzcd}
\boxed{\text{A recursive function}} \arrow[d] \arrow[dr] & \\
& \boxed{\text{base case}} \\
& \boxed{\text{recursion step}}
\end{tikzcd}$$

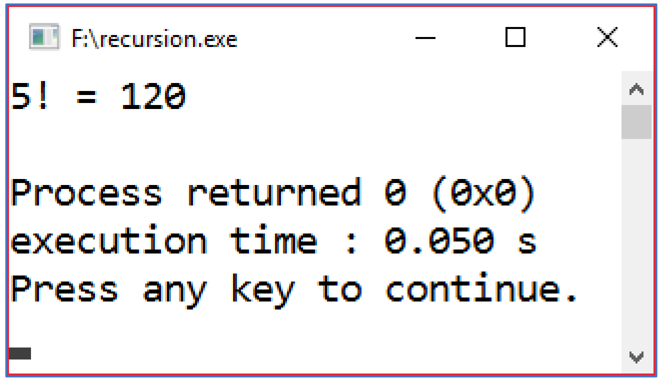
Recursion

```c
#include <stdio.h>

int factorial(int n)
{
    if(n == 1)
        return 1;
    else
        return n * factorial(n-1);
}

int main()
{
    printf("5! = %d\n",factorial(5));

    return 0;
}
```

base case

recursion step





```
5! = 120

Process returned 0 (0x0)
execution time : 0.050 s
Press any key to continue.
```

Practice 0

- Find the error in the following program segment and explain how the error can be corrected.

```
void f(int a)
{
     int a;
     printf("%d",a);
}
```

Practice 1

- Find the error in the following program segment and explain how the error can be corrected.

```
float sum(float x, float y)
{
    float result;
    result = x + y;
}
```

Practice 2

- Find the error in the following program segment and explain how the error can be corrected.

```
int sum(int n)
{
    if(n == 0)
    {
        return 0;
    }
    else
    {
        n + sum(n-1);
    }
}
```

Practice 3

- Find the error in the following program segment and explain how the error can be corrected.

```
void product()
{
    int p, q, r, result;
    printf("Enter 3 integers: ");
    scanf("%d %d %d",&p,&q,&r);
    result = p * q * r;
    printf("Result = %d\n",result);
    return result;
}
```

Practice 4

- Write a function **convertCase** that converts a lowercase alphabetic character to an uppercase letter and vice versa.

Statement printf(“%c”,convertCase(‘A’)); should print ‘a’

Statement printf(“%c”,convertCase(‘a’)); should print ‘A’

Practice 5

- Write a function **power(base, exponent)** that returns the value of $base^{exponent}$. For example, power(2, 5) = 2^5 = 32. Assume that exponent is a positive, nonzero integer, and base is an integer. Do not use any math library functions.

Practice 6

- Write a recursive function **fibonacci(n)** that calculates the nth Fibonacci number.

  fibonacci(n) = fibonacci(n-2) + Fibonacci(n-1)

  fibonacci(0) = 0

  Fibonacci(1) = 1

NEXT WEEK’S OUTLINE

1. Pointer
2. Pointer Operators
3. Array Declaration
4. Array Access

REFERENCES

- Hanly, Jeri R. and Koffman, Elliot B., 2013, Problem Solving and Program Design in C, Seventh Edition, Pearson Education, Inc.
- Deitel, Paul and Deitel, Harvey, 2016, C How to Program, Eighth Edition, Pearson Education, Inc.

Visi

Menjadi Program Studi Strata Satu Informatika **unggulan** yang menghasilkan lulusan **berwawasan internasional** yang **kompeten** di bidang Ilmu Komputer (*Computer Science*), **berjiwa wirausaha** dan **berbudi pekerti luhur**.

INFORMATIKA
UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

Misi

1. Menyelenggarakan pembelajaran dengan teknologi dan kurikulum terbaik serta didukung tenaga pengajar profesional.
2. Melaksanakan kegiatan penelitian di bidang Informatika untuk memajukan ilmu dan teknologi Informatika.
3. Melaksanakan kegiatan pengabdian kepada masyarakat berbasis ilmu dan teknologi Informatika dalam rangka mengamalkan ilmu dan teknologi Informatika.