Mahidol University

ITCS113
Fundamentals of Programming

Lecture 9 - Function

Instructor: Asst. Prof. Dr. Akara Supratak

Contact: akara.sup@mahidol.edu

Agenda

- What is a function?
- Define a function
- Function call
- Function call - Pass by Value
- Function Prototype
- C Standard Library

What is a function?

Why do we need functions?

- They allow us to conceive of our program as a sequence of sub-steps
  - Easy to understand the code
- They allow us to reuse code instead of rewriting it
  - Also easier to change the behaviour of the function
- Functions allow us to test small parts of our program in isolation from the rest
- Functions allow us to keep our variable namespace clean
  - Local variables only "live" as long as the function does

How to Design a Good Function?

- Typically, the criteria used to judge whether the functions are good depend on the goal of the application (i.e., application specific)
  - There is no right or wrong answer as long as the program can achieve its desired tasks
- Reusable
  - Receives input values (i.e., input arguments)
  - Return output values
- Perform one thing

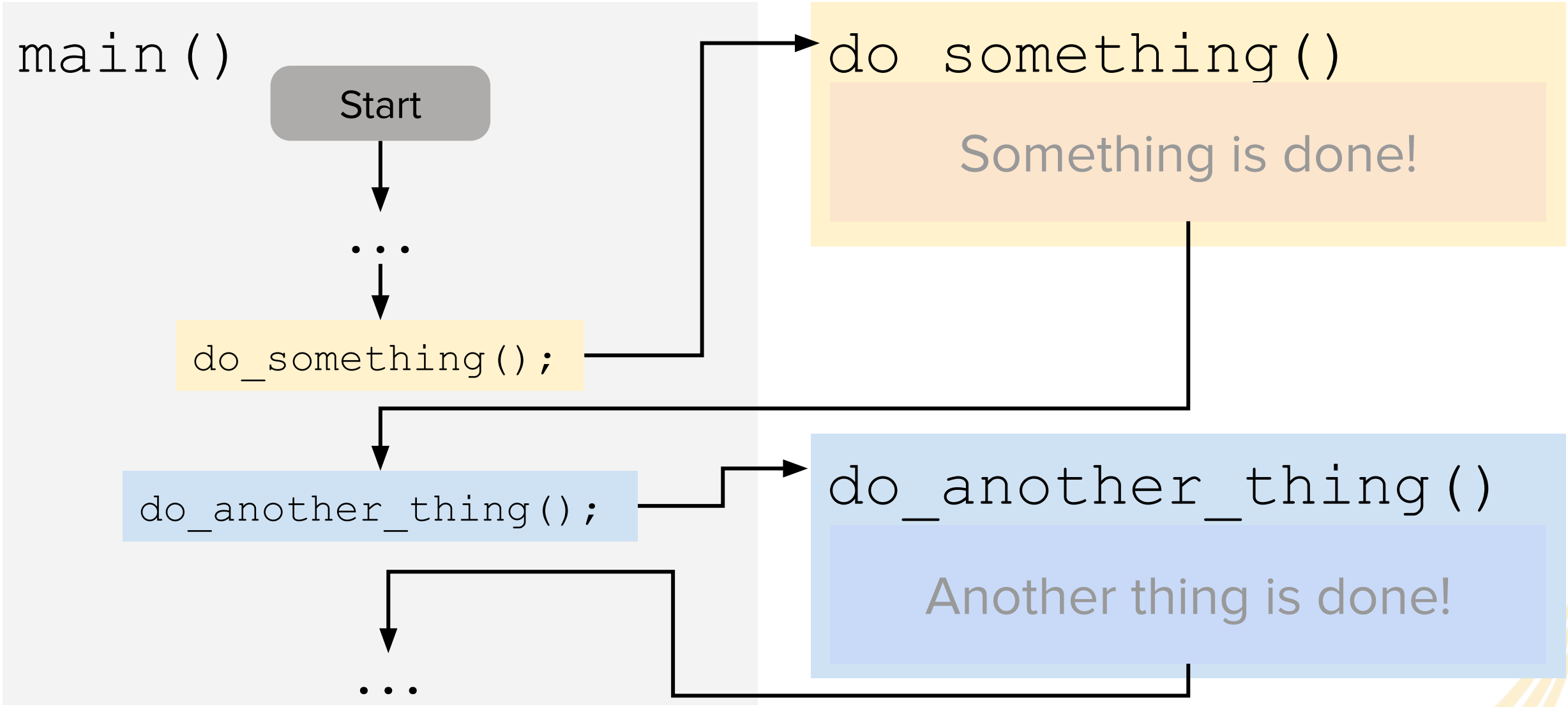
When we use functions,

- We do not care HOW a function does its task
- We just need to know WHAT it does
- What are the inputs and outputs of a function?

Examples:

- `printf(...)` writes outputs
- `scanf(...)` reads inputs

When we use functions



There are two types of functions

1. Built-in functions from C standard library
   - No need to implement
   - Just add `#include <library_name>`
     - `#include <stdio.h>`
     - `#include <math.h>`
2. Your own-defined functions
   - know WHAT it does
   - define INPUT and OUTPUT
   - decide HOW to do it (step by step)

Use Case of Functions

```
int main()
{
  int result,result2,result3,result4,result5;
  int a1=5,b1=5,a2=6,b2=6,a3=7,b3=7,a4=8,b4=8,a5=9,b5=9;

  result  = a1+b1;
  result2 = a2+b2;
  result3 = a3+b3;
  result4 = a4+b4;
  result5 = a5+b5;

  printf("the result is %d\n\n",result);
  printf("the result is %d\n\n",result2);
  printf("the result is %d\n\n",result3);
  printf("the result is %d\n\n",result4);
  printf("the result is %d\n\n",result5);

  return 0;
}
```

What if I would like to change something?
the printing format from "the result is %d"
to "the result: %d"?

Define a function

Define a function

```
return_dtype func_name(dtype1 param1, dtype2 param2)
{
    statement1;  // may define new params
    statement2;  // may use params
    …
    return return_value
}
```

Define a function: Function header

- Identifies the data **type** of the **return value**
- Provides the function with a **name**
- Specifies a list of **parameters/arguments** in order, and type of values expected by the function

Function Header

```
return_dtype func_name(dtype1 param1, dtype2 param2)
{
    statement1;  // may define new params
    statement2;  // may use params
    …
    return return_value;
}
```

Define a function: Function header (cont.)

- **func_name**: the name of the function
- **`dtype param`**: A list of input parameters or arguments that specifies the number, order, and type of input values expected by the function
  Example `int add(int a, int b) {...}`
  - This function receives **two** input arguments of type integer.
  - The first integer value is stored in **`a`**, the second is stored in **`b`**.
- It is possible that we do not have any arguments
  Example `void say_hello_world() {...}`

```
return_dtype func_name(dtype1 param1, dtype2 param2)
{
    …
    return return_value;
}
```

Define a function: Function header (cont.)

- **`return_dtype`** is the data type of **`return_value`**
  - `int, float, char, …`

```
return_dtype func_name(dtype1 param1, dtype2 param2)
{
    …
    return return_value;
}
```

- If this function does *not* return any output, use **`void`** in **`return_dtype`**

  Example `void say_hello_world() {...}`

Define a function: Function body

- A set of **statements** that operate on the passed parameters/arguments
- Define HOW to do a task (step by step)
- Returns *one* value

```
return_dtype func_name(dtype1 param1, dtype2 param2)
{
    statement1;  // may define new params
    statement2;  // may use params
    …
    return return_value;
}
```

Function Body

Define a function: Function body (cont.)

- **`return return_value;`**
  a return statement that specifies the *output* to be returned, and make the function call *ends*
- Data type of the **`return_value`** must match with the **`return_dtype`**
  Example
  - `return 0;`
  - `return (a+b);`
  - `return result;`
- If **`void`** is used in **`return_dtype`**, we can omit this statement.

  Example

```
void say_hello_world() {
    printf("Hello World!");
    /* No return */
}
```

Define a function

When we write a function, we need to …

- know WHAT it does
- define INPUT and OUTPUT
- decide HOW to do it (step by step)

Example:

```
float get_circle_area(float r)
{
    float r_square, area;
    r_square = r * r;
    area = PI * r_square;
    return area;
}
```

Define a function

When we write a function, we need to …

- know WHAT it does
- define INPUT and OUTPUT
- decide HOW to do it (step by step)

Example:

Function name

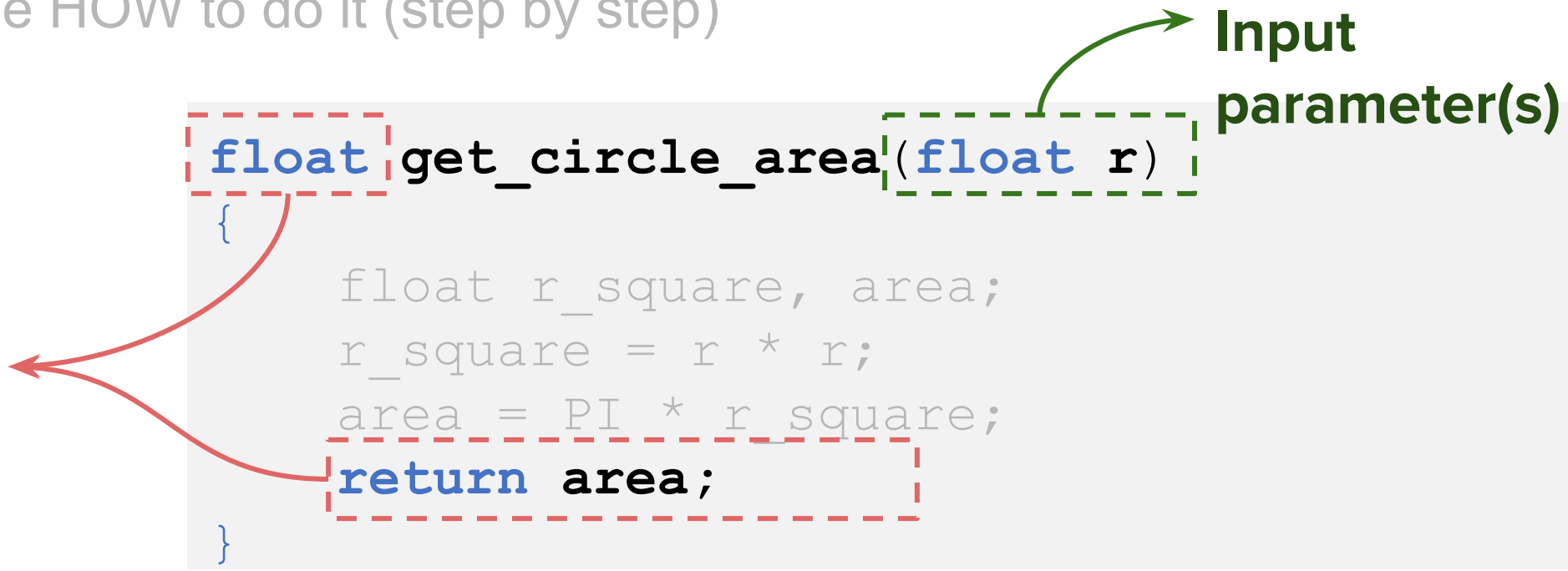
```
float get_circle_area(float r)
{
    float r_square, area;
    r_square = r * r;
    area = PI * r_square;
    return area;
}
```

Define a function

When we write a function, we need to …

- know WHAT it does
- define INPUT and OUTPUT
- decide HOW to do it (step by step)

Example:



Define a function

When we write a function, we need to …

- know WHAT it does
- define INPUT and OUTPUT
- decide HOW to do it (step by step)

Example:

```
float get_circle_area(float r)
{
    float r_square, area;
    r_square = r * r;
    area = PI * r_square;
    return area;
}
```

Function Call

Simple Function in C program

Writing a function definition **before** `main()`

```
#include <stdio.h>

void say_hello_world(){
    printf("Hello World!\n");
}

int main(){
    say_hello_world();
    printf("===========\n");
     say_hello_world();
    return 0;
}
```

Function definition

Function call(s)
(i.e., invocation)

Flow control of a function

```
int main()
```

Simple Function in C program

Writing a function definition **before** `main()`

```c
#include <stdio.h>

void say_hello_world(){
    printf("Hello World!\n");
}

int main(){
    say_hello_world();
    printf("===========\n");
     say_hello_world();
    return 0;
}
```

Output

```
Hello World!
===========
Hello World!
```

Function Call - Pass by Value

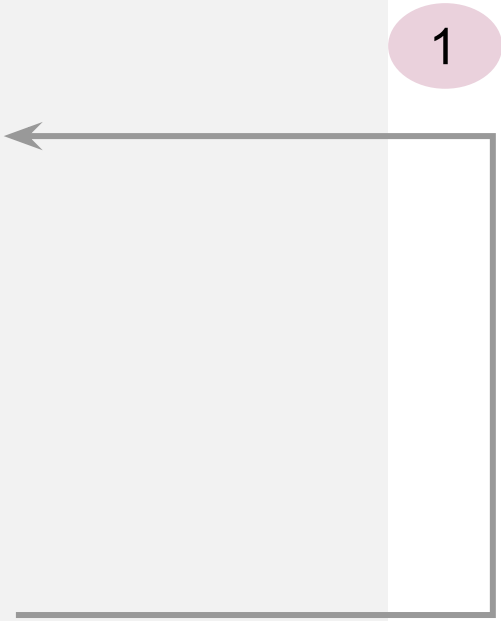
Pass by Value

So far, when we call a function, it receives copies of the values of input arguments

```
#include <stdio.h>

float get_circle_area(float r) {
    float r_square, area;
    r_square = r * r;
    area = 3.14 * r_square;
    return area;
}

int main() {
    float area1, area2;
    area1 = get_circle_area(3);
    area2 = get_circle_area(7);
    return 0;
}
```



1 When **called**, **3** is copied and stored in the variable **r**

Pass by Value

So far, when we call a function, it receives copies of the values of input arguments

```c
#include <stdio.h>

float get_circle_area(float r) {
    float r_square, area;
    r_square = r * r;
    area = 3.14 * r_square;
    return area;
}

int main() {
    float area1, area2;
    area1 = get_circle_area(3);
    area2 = get_circle_area(7);
    return 0;
}
```

1 When **called**, **3** is copied and stored in the variable **r**

2 At the **end** of function, **r** is **discarded**

Pass by Value

So far, when we call a function, it receives copies of the values of input arguments

```
#include <stdio.h>

float get_circle_area(float r) {
    float r_square, area;
    r_square = r * r;
    area = 3.14 * r_square;
    return area;
}

int main() {
    float area1, area2;
    area1 = get_circle_area(3);
    area2 = get_circle_area(7);
    return 0;
}
```

When **called**, **7** is copied and stored in the variable **`r`**

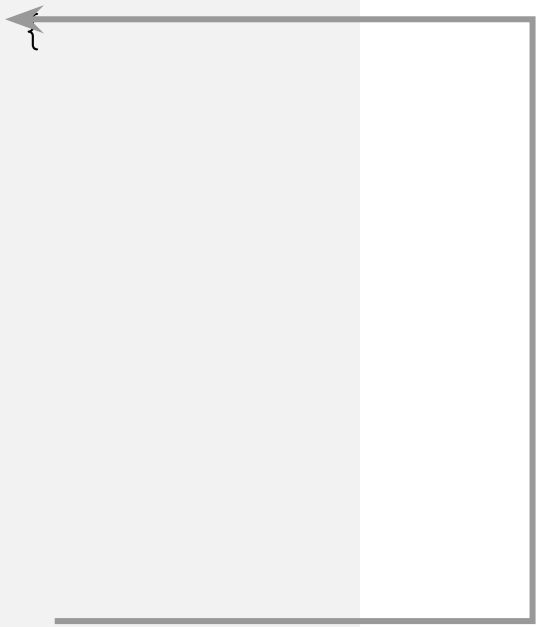
At the **end** of function, **`r`** is **discarded**

Pass by Value

```
#include <stdio.h>

float get_circle_area(float r) {
    float r_square, area;
    r_square = r * r;
    area = 3.14 * r_square;
    return area;
}

int main() {
    float r1 = 4.2, r2 = 6;
    float area1, area2;
    area1 = get_circle_area(r1);
    area2 = get_circle_area(r2);
    return 0;
}
```



When **called**, **`r1`** is copied and stored in the variable **`r`**

At the **end** of function, **`r`** is **discarded**

Pass by Value

```
#include <stdio.h>

float get_circle_area(float r) {
    float r_square, area;
    r_square = r * r;
    area = 3.14 * r_square;
    return area;
}

int main() {
    float r1 = 4.2, r2 = 6;
    float area1, area2;
    area1 = get_circle_area(r1);
    area2 = get_circle_area(r2);
    return 0;
}
```

When **called**, **`r2`** is copied and stored in the variable **`r`**

At the **end** of function, **`r`** is **discarded**

Pass by Value

```
#include <stdio.h>

float get_circle_area(float r) {
    float r_square, area;
    r_square = r * r;
    area = 3.14 * r_square;
    r = 0.7;
    return area;
}

int main() {
    float r1 = 4.2, r2 = 6;
    float area1, area2;
    area1 = get_circle_area(r1);
    area2 = get_circle_area(r2);
    return 0;
}
```

Any **changes** made to `r` in the `get_circle_area()` function **does not affect** the `r1` in the `main()` function

Variable r is discarded at the end of the function

Before and After call function, the values of `r1` and `r2` in the main() function are the same

`r1 = 4.2`
`r2 = 6`

Question: What is the output?

```
#include <stdio.h>

int find(int a, int b){
    if (a > b)
        return a;
    return b;
}

int main(){
    int a=5, b=1, c=-20, m, n;
    m = find(a, b);
    n = find(m, c);
    printf("%d", n);
    return 0;
}
```

Question: What is the output?

```
#include <stdio.h>

int abs(int v){
    if (v < 0) return -v;
    return v;
}
int find(int a, int b){
    if (abs(a) > abs(b))
        return a;
    return b;
}
int main(){
    int a=5, b=1, c=-20, m, n;
    m = find(a, b);
    n = find(m, c);
    printf("%d", n);
    return 0;
}
```

Function Prototype

Function Prototype

- Declaration statement for a function (similar to define a variable)
- Specify function name, parameters and return type as the same we define a function header

```
return_dtype func_name(dtype1 param1,...);
```

Example

```
int get_prod(int x, int y);
float get_area_rectangle(float h, float w);
void say_hello_world();
```

Function Prototype

Without Function Prototype

```
#include <stdio.h>

float get_circle_area(float r){
    float r_square, area;
    r_square = r * r;
    area = 3.14 * r_square;
    return area;
}

int main(){
    float area1, area2;
    area1 = get_circle_area(3);
    area2 = get_circle_area(7);
    return 0;
}
```

Function Prototype

With Function Prototype

```
#include <stdio.h>

float get_circle_area(float r);

int main(){
    float area1, area2;
    area1 = get_circle_area(3);
    area2 = get_circle_area(7);
    return 0;
}

float get_circle_area(float r){
    float r_square, area;
    r_square = r * r;
    area = 3.14 * r_square;
    return area;
}
```

Without Function Prototype

```
#include <stdio.h>

float get_circle_area(float r){
    float r_square, area;
    r_square = r * r;
    area = 3.14 * r_square;
    return area;
}

int main(){
    float area1, area2;
    area1 = get_circle_area(3);
    area2 = get_circle_area(7);
    return 0;
}
```

Example: Function Prototype

```
#include<stdio.h>

void say_hello_world();

int main() {
    say_hello_world();
    return 0;
}

void say_hello_world(){
    printf("Hello World!");
}
```

Function prototype

Function call

Function definition

Question 4

What is the function prototype of the following functions?

```
int get_sum (int x, int y) {
    int sum = x + y;
    return sum;
}
```

```
float get_circle_area(float r) {
    return PI * r * r;
}
```

```
int get_two_plus_two() {
    int sum = 2 + 2;
    return sum;
}
```

```
void print_hello() {
    printf("Hello\n");
}
```

C Standard Library

C Standard Library

- You can look up the library online
  - e.g., https://www.tutorialspoint.com/c_standard_library/index.htm
- What you need to know is WHAT it does (don't need to know HOW except you are writing your own function)
- They typically provide function prototypes to tell you how to use the functions
  - The function name and description
  - The number, order and types of the input arguments
  - The type of returned value (if any)

C Standard Library

```
#include <math.h>
```

Ref: https://www.tutorialspoint.com/c_standard_library/math_h.htm

Library Functions

Following are the functions defined in the header math.h −

Sr.No.	Function & Description
1	double acos(double x) Returns the arc cosine of x in radians.
2	double asin(double x) Returns the arc sine of x in radians.
3	double atan(double x) Returns the arc tangent of x in radians.
4	double atan2(double y, double x) Returns the arc tangent in radians of y/x based on the signs of both values to determine the correct quadrant.

Example - `pow` function

https://www.tutorialspoint.com/c_standard_library/c_function_pow.htm

Description

The C library function **double pow(double x, double y)** returns **x** raised to the power of **y** i.e. x^y.

Declaration

Following is the declaration for pow() function.

```
double pow(double x, double y)
```

Parameters

- **x** − This is the floating point base value.
- **y** − This is the floating point power value.

Return Value

This function returns the result of raising **x** to the power **y**.

Example - `pow` function

```
#include <stdio.h>
#include <math.h>

int main() {
    int a, b;
    printf("Input base and power values: ");
    scanf("%d %d", &a, &b);
    double result = pow(a, b);
    return 0;
}
```

- In this example, we can pass `a` and `b` which are integers as the input arguments to the function `pow` that accepts two `double`'s.
- You CAN do casting from:

  `int` ➔ `float` ➔ `double`

  as you **WILL NOT LOSE** the precision

Example - `pow` function

```
#include <stdio.h>
#include <math.h>

int main() {
    int a, b;
    printf("Input base and power values: ");
    scanf("%d %d", &a, &b);
    double result = pow(a, b);
    // int result = pow(a, b);  ->  SHOULD NOT do this
    return 0;
}
```

- However, you SHOULD NOT do casting from:

  `double` ➜ `float` ➜ `int`

  as you **WILL LOSE** the precision.

Lab Exercises