CS1010E Lecture 5

Value-Returning Functions

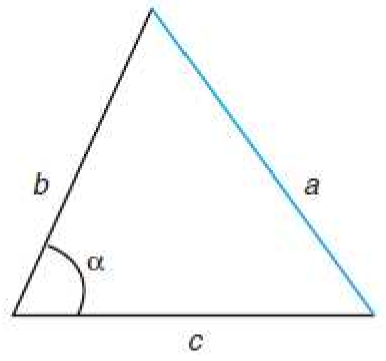
Henry Chia (hchia@comp.nus.edu.sg)

Semester 1 2016 / 2017

Lecture Outline

- Math library functions
- Function expression and evaluation
- User defined functions
- Pass-by-value
- Lexical scoping
- Boolean function
- Function interface comments

Math Functions



- Given the lengths of two sides of a triangle b and c, and the angle between them α, compute a

$$a^2 = b^2 + c^2 - 2bc\cos\alpha$$

or

$$a = \sqrt{b^2 + c^2 - 2bc\cos\alpha}$$

Math Library Functions

- To use C math library functions, include the preprocessor directive

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

- Refer to `http://www.acm.uiuc.edu/webmonkeys/book/c_guide` for a list of functions in `<math.h>`
  - Example: `cos` function declaration

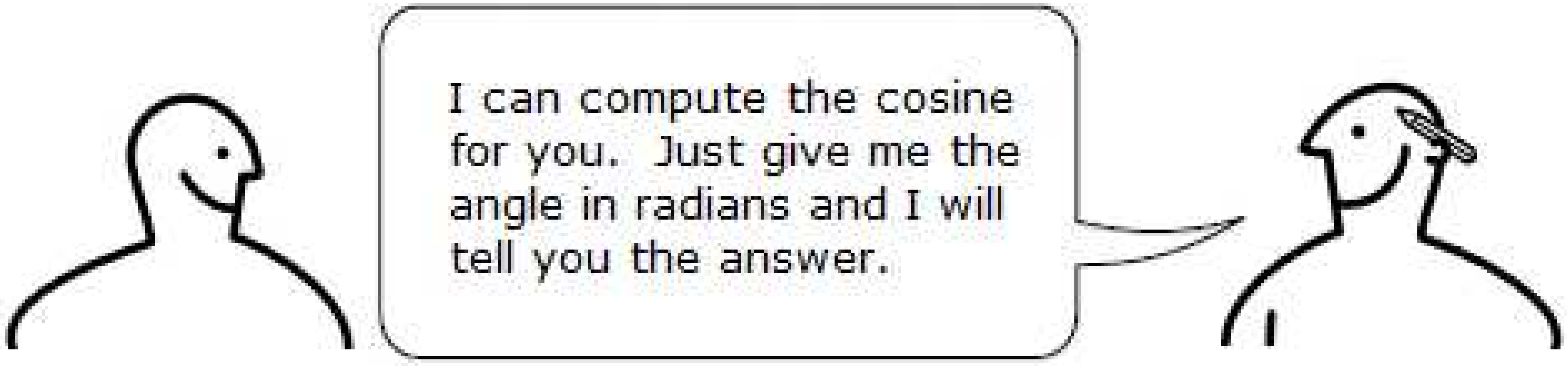
    ```
    double cos(double x);
    ```

  - `cos` function takes in `double` argument (say x) and produces another `double` value ($\cos(x)$)

Program: Side of a Triangle

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

#define PI 3.14159

int main(void) {
   double a, b, c, alpha;

   printf("Enter b, c and alpha(deg): ");
   scanf("%lf%lf%lf", &b, &c, &alpha);

   /* cos takes in angle in radians */
   a = sqrt((b * b) + (c * c) -
            (2 * b * c * cos(alpha * PI / 180)));

   printf("Length of side a is %f\n", a);

   return 0;
}
```

Functions – Motivation from Life


I can compute the cosine for you. Just give me the angle in radians and I will tell you the answer.

- Suppose you have a trustworthy friend whom you know can compute the cosine of an angle given in radians
- How to "call" him/her to find the cosine of say, π?
  - What do you say (or "pass") to your friend?
  - What does he/she give (or "return") back to you?
  - Do you need to know how he/she computes the answer?

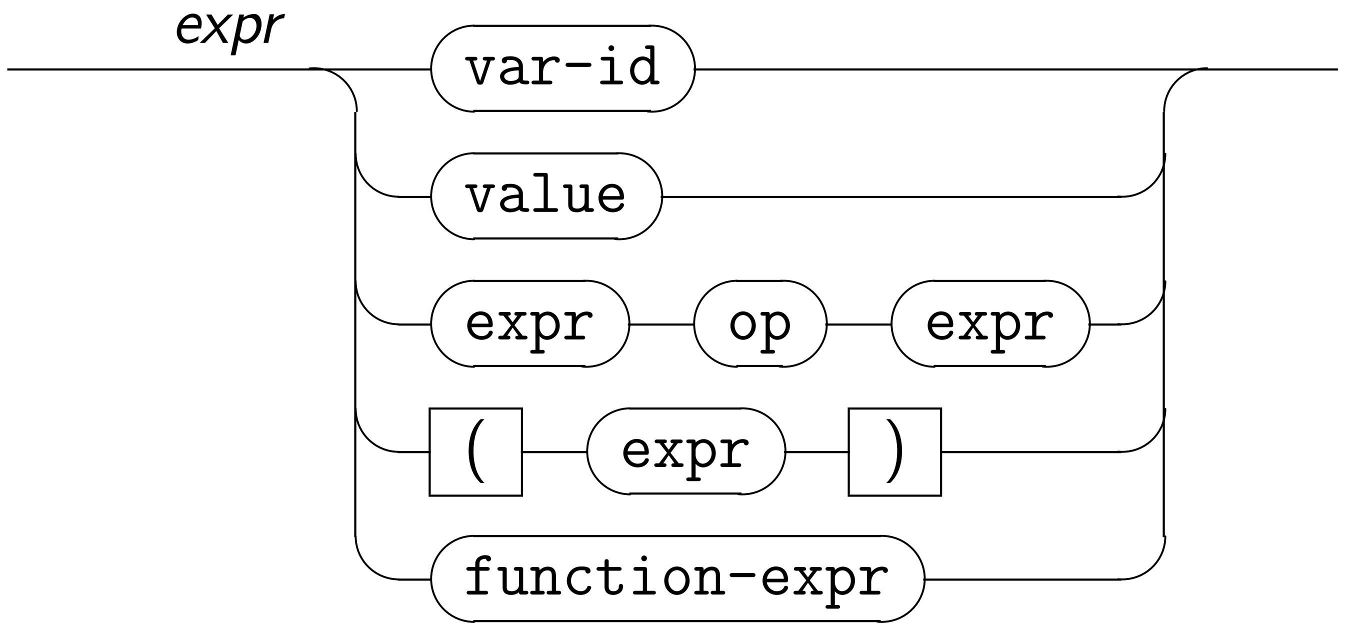
Function Evaluation

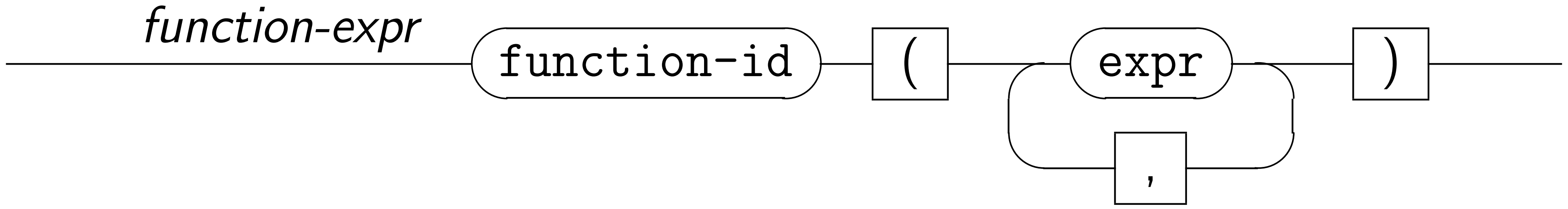
- Evaluate a function expression $\Rightarrow$ “call” the function

$$\text{argument value(s)} \rightarrow \boxed{\text{function}} \rightarrow \text{evaluated value}$$

- Suppose $\alpha = 90$, to evaluate `cos(alpha*PI/180)`
  1. Arguments are evaluated: `cos(1.570795)`
  2. Argument values `1.570795` passed to function `cos`
  3. `cos` function performs desired computation
  4. `cos` returns the value `0.0` as the function evaluation
- Only one function executes at any one time

Function Expression





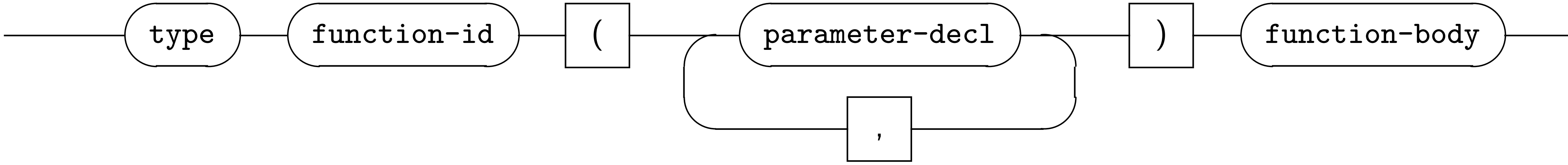
User-Defined Function

- Program template with user defined functions

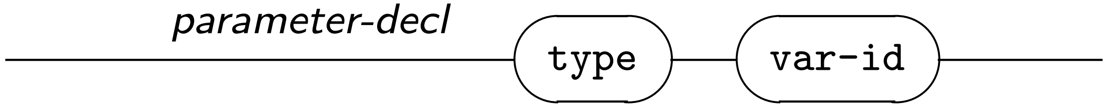
```
/* preprocessor directives */

/* function prototypes */

int main(void) {
    /* declarations */

    /* statements */

    return 0;
}

/* user defined function definitions */
```

User-Defined Function

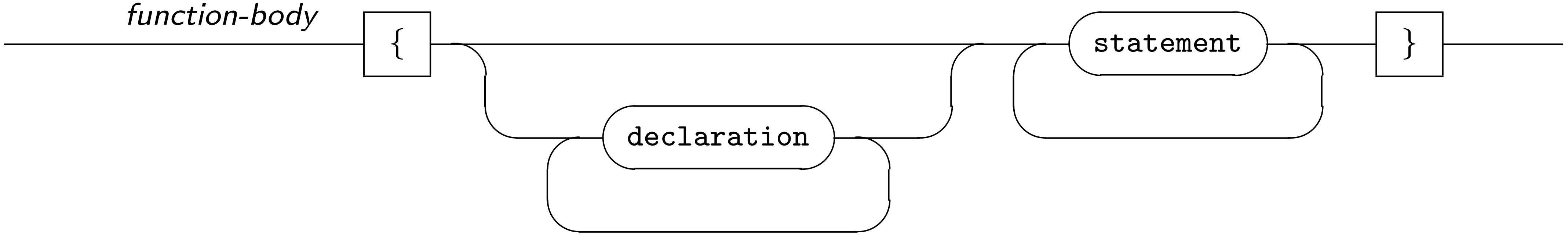
functionDefn



parameter-decl



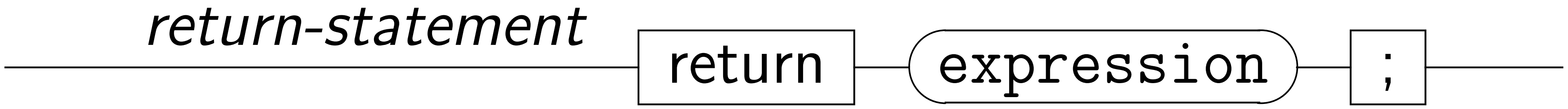
function-body



Function Definition

- *functionDefn*
  - `type`: type of value returned from the function
  - `function-id`: meaningful function identifier
  - `parameter-decl`: parameter (variable) declaration
- *parameter-decl*
  - Variables declared to store the values passed from the "caller"
  - One variable per declaration separated by commas
- *function-body*
  - Defines the function implementation
  - `return` statement included as the last logical statement in the function body

return Statement

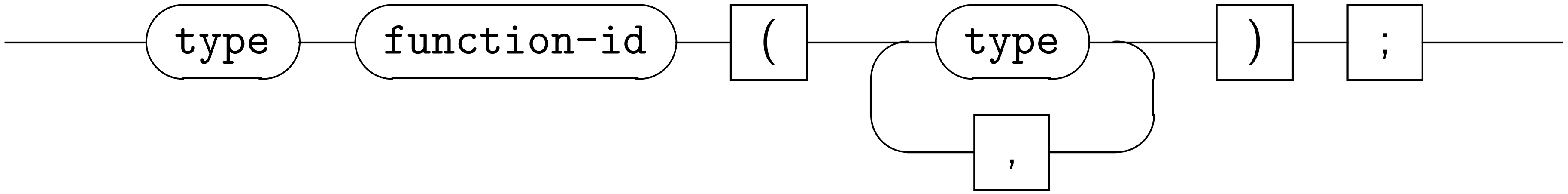


- Upon executing the return statement,
  1. the expression is evaluated to a value
  2. the function returns the value to the “caller” and terminates
  3. the “caller” function resumes execution with the value returned

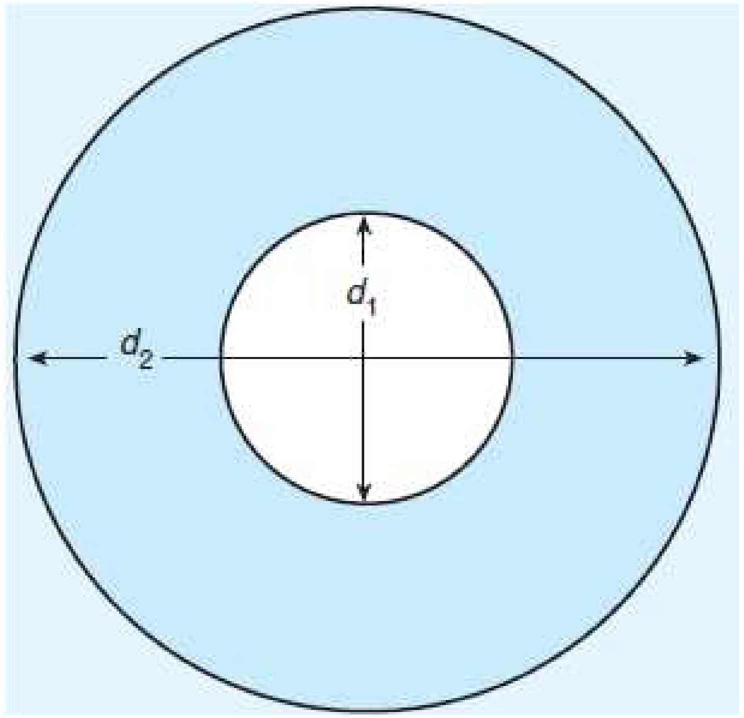
Function Prototype

- A function prototype declares the proper usage of the function in terms of function identifier, number/type of arguments and return values
- Hence, it is placed before the corresponding function calls and the definition of the function
- Almost the same as the corresponding function header, but need only specify the parameter type (parameter name is encouraged but not necessary)
- Keep in mind the semicolon at the end

functionPrototype



Example: Surface Area of a Flat Washer



- Given the diameters d_1 and d_2, find the surface area of the hollow disc
- E.g. $d_1 = 1.2$ units and $d_2 = 2.4$ units gives a surface area of 3.392917 square units

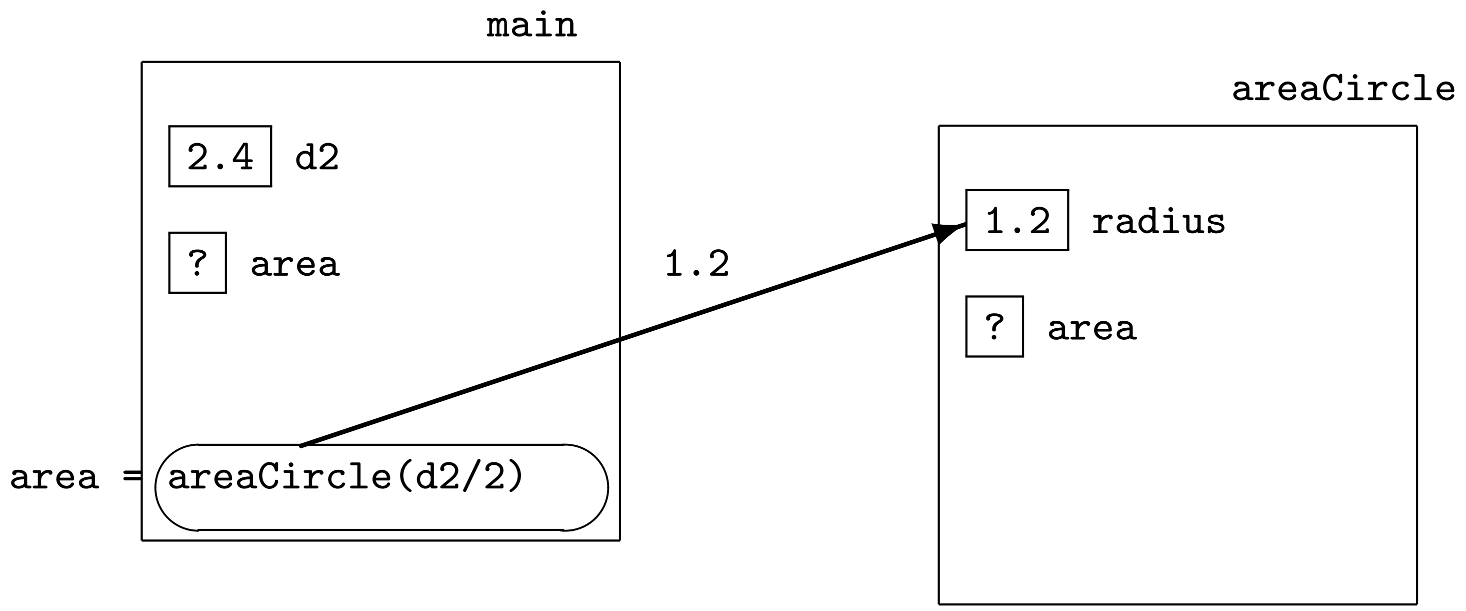
- Area of circle of radius r, $f_{area}(r) = \pi r^2$
- Surface area of the flat washer is $f_{area}(\frac{d_2}{2}) - f_{area}(\frac{d_1}{2})$
- In general, $f(x_1, x_2, \ldots x_n)$ is analogous to passing arguments $(x_1, x_2, \ldots, x_n)$ to the function f

Program: Surface Area of a Flat Washer

```c
#include <stdio.h>

#define PI 3.14159

double areaCircle(double);

int main(void) {
   double d2, d1, area;

   printf("Enter d2 and d1 diameters: ");
   scanf("%lf%lf", &d2, &d1);

   area = areaCircle(d2 / 2.0) - areaCircle(d1 / 2.0);

   printf("The surface area is %f\n", area);

   return 0;
}

double areaCircle(double radius) {
   double area;

   area = PI * radius * radius;

   return area;
}
```

Pass-by-Value

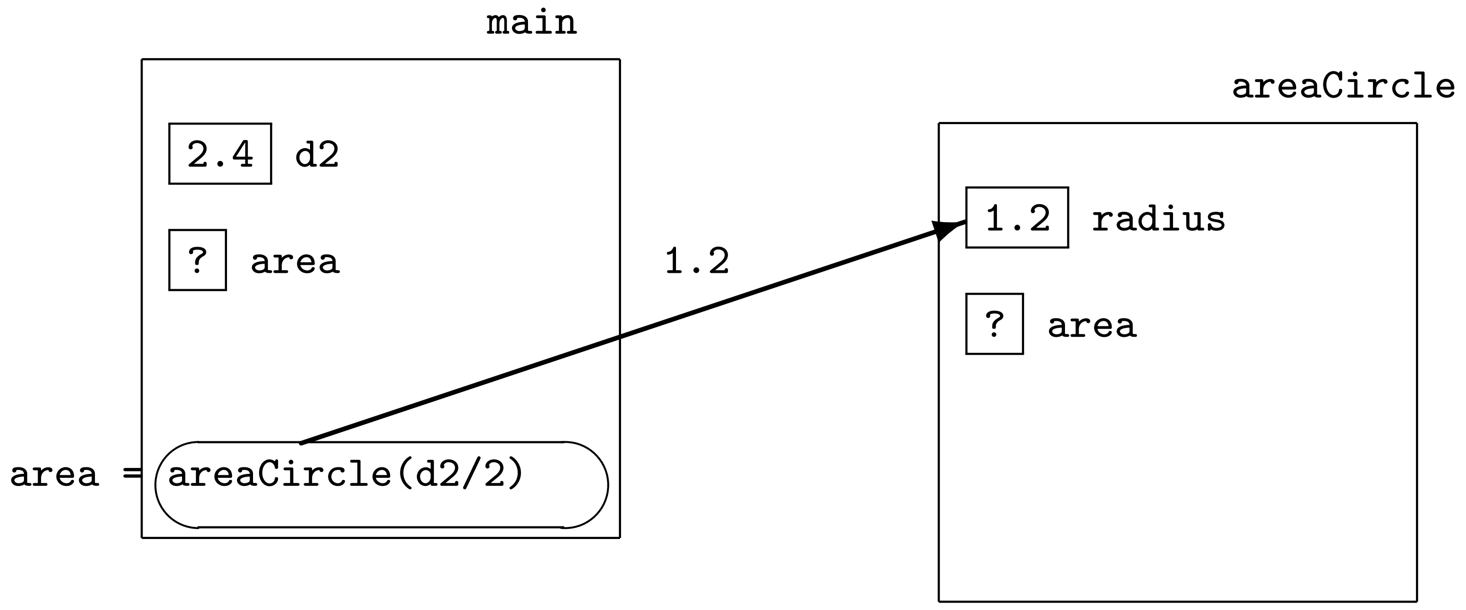
- During a function call, each argument expression is evaluated and the **value** passed to the function and stored in each input parameter



- In the example above, upon initiating the function call in `main`, `areaCircle` **activates** while execution in `main` **suspends**

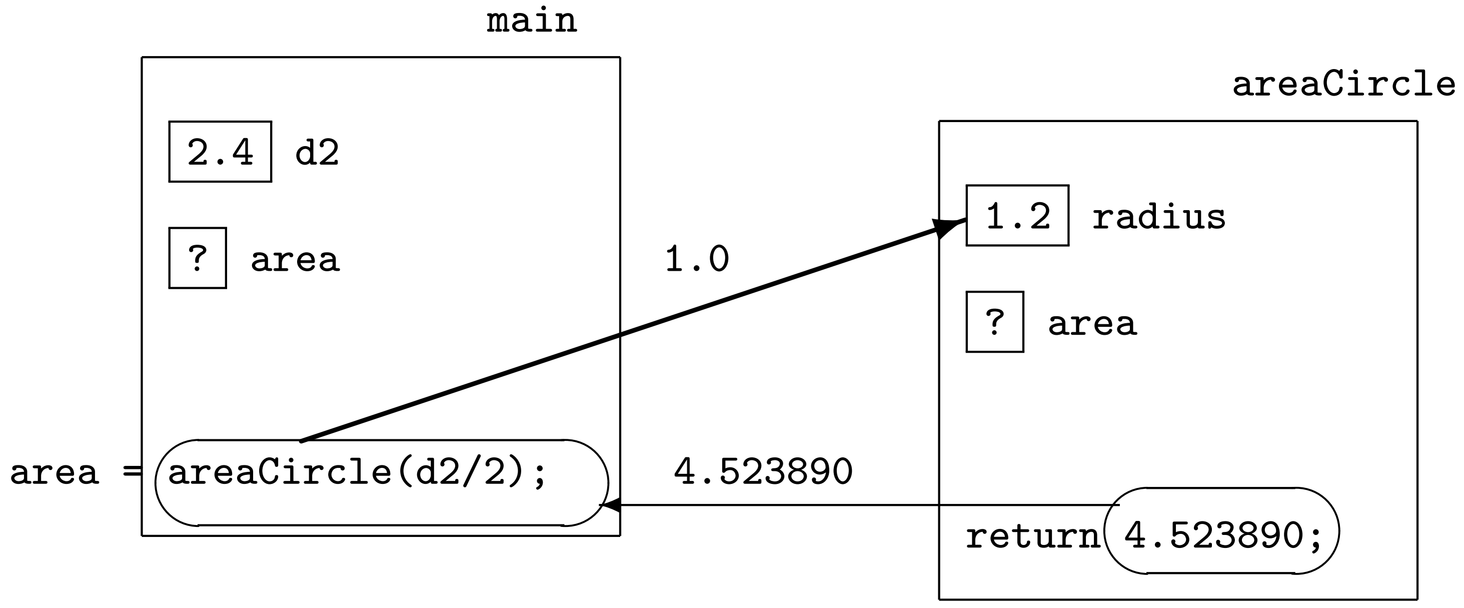
Lexical Scoping

- The scope of accessibility of a variable is within the function in which the variable is declared
- Variables within a function are **local** to that function
- Variables of the same name can co-exist across functions



Return Value

- Function terminates with a return **value** to the caller



- In the example above, `areaCircle` returns the value to the caller function and **terminates**; `main` then **resumes** execution

Program: Surface Area of a Flat Washer

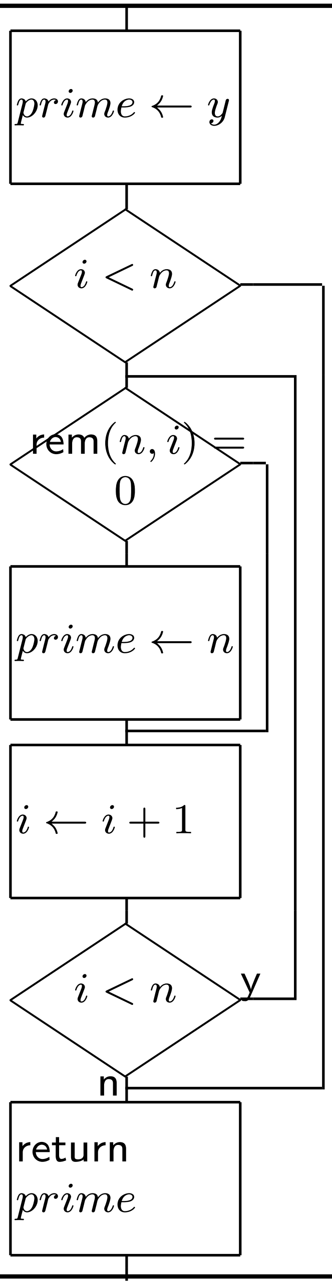
```
#include <stdio.h>
#define PI 3.14159
double areaCircle(double radius);
double surfaceArea(double dOut, double dIn);

int main(void) {
   double d2, d1, area;

   printf("Enter washer and hole diameters: ");
   scanf("%lf%lf", &d2, &d1);

   area = surfaceArea(d2,d1);
   printf("The surface area is %f\n", area);

   return 0;
}

double surfaceArea(double dOut, double dIn) {
   return areaCircle(dOut/2.0) - areaCircle(dIn/2.0);
}

double areaCircle(double radius) {
   return PI * radius * radius;
}
```

Boolean Functions

- *Boolean* functions — return `true` or `false` values
- Defined with `bool` return type with identifier names preferably starting with `is..`, e.g. `isPrime`
- Example: Primality testing
  - Given an integer $n(> 1)$, determine if n is prime
  - Loop through all divisors from 2 to $p - 1$
  - If there is a divisor that divides p, p is **not prime**
  - p is prime only when **all** divisors from 2 to $p - 1$ **do not** divide p

Program: Primality Testing

```
#include <stdio.h>
#include <stdbool.h>
bool isPrime(int n);

int main(void) {
   int n;

   printf("Enter a number: ");
   scanf("%d", &n);

   if (isPrime(n)) {
      printf("%d is prime\n", n);
   } else {
      printf("%d is not prime\n", n);
   }
   return 0;
}

bool isPrime(int n) {
   int i; bool prime=true;

   for (i = 2; i < n; i++) {
      if (n%i == 0) {
         prime = false;
      }
   }
   return prime;
}
```



Early Return

- The `isPrime` function can be defined to return early

```
bool isPrime(int n) {
   int i;

   for (i = 2; i < n; i++) {
      if (n%i == 0) {
         return false;
      }
   }
   return true;
}
```

- Preferred alternative that maintains single-entry-single-exit

```
bool isPrime(int n) {
   int i; bool prime=true;

   for (i = 2; i < n && prime; i++) {
      if (n%i == 0) {
         prime = false;
      }
   }
   return prime;
}
```

Function Interface Comments

- A function should contain a comment on
  - Pre-condition – condition that should hold true before the function is called
  - Post-condition – condition that must hold true after the function completes (can be the purpose of the function)
- Example:

```
/*
   isPrime function returns true if n is prime,
   false otherwise.

   Precondition: n > 1
*/
bool isPrime(int n) {
   ...
}
```

Lecture Summary

- When working with functions, ensure **type consistency**
  - Between the argument values and the input parameters

    ```
    double areaCircle(double radius); /* parameter type is double */
    ...
    area = areaCircle(1.23); /* argument type is double */
    ```

  - Between the type of return expression and the function return type

    ```
    double areaCircle(double radius) { /* return type is double */
       return PI * radius * radius;     /* expression type is double */
    }
    ```

  - Between the function return type and how it is used, e.g.

    ```
    printf("%f\n", areaCircle(radius));
    /* %f prints double-type return value */
    ```

- Appreciate the concepts of pass-by-value and lexical scoping