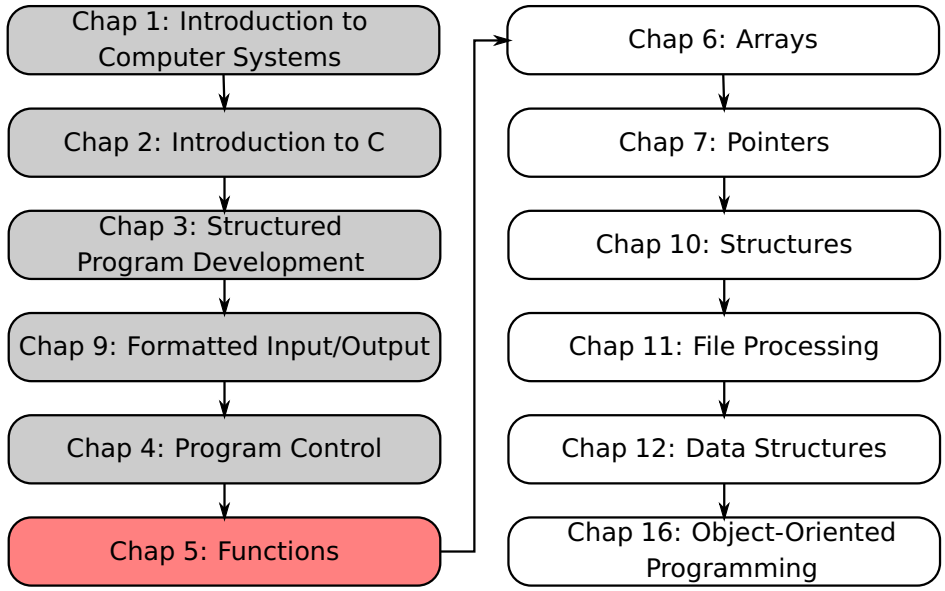
Computer Programming 143 – Lecture 10
Functions I

Electrical and Electronic Engineering Department
University of Stellenbosch

Corné van Daalen
Thinus Booysen
Willie Smit
Willem Jordaan

Module Overview



Lecture Overview

5.1 Introduction

This chapter introduces

- Construction of a program from smaller pieces or components
  - These smaller pieces are called modules or functions
- Each piece more manageable than the original program
- Reduces the duplication of code in a program
- Enabling reuse of code across multiple programs
- Improving readability of a program

5.2 Introduction to Functions I

What are functions?

A function is a piece of code or a module that

- Has been “packaged” as a unit
- Usually serve a single function / task
- Performs a task and then returns control to the caller
  - After the function has perfomed the task, the program will continue execution from the point after the call
- Can be executed several times and called from different places during a single execution of a program

5.2 Introduction to Functions II

Functions we have used:

From `stdio.h`

- Print text on the screen

```
printf( "The value of x is %d\n", x );
```

- Read input from user

```
scanf( "%d", &x );
```

From `math.h`

- Calculate one number raised to the power of a second number

```
x = pow( 5, 3 );
```

`main()`

- The main function in every C program

5.2 Introduction to Functions III

The components of a function

- A body of code to be executed
- Arguments that are passed to the function (input/data)
- A value that is returned (output/result)

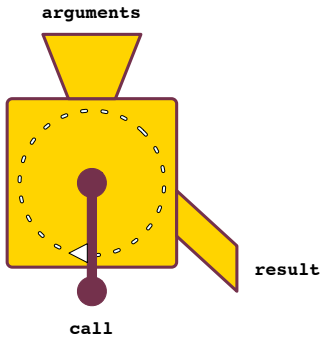
5.2 Introduction to Functions IV

How does a program execute the code in a function?

- Function calls

```
x = cos( 1.15 );
```

  - Provide function name and arguments
  - Function performs operations or manipulations
  - Function returns a result



5.3 Math Library Functions I

Math library functions

- perform common mathematical calculations
- `#include <math.h>`

Format for calling functions

```
functionName( argument1, argument2, ... );
```

- If multiple arguments, use comma-separated list
- Arguments may be constants, variables or expressions
- Example:

```
double x;
x = sqrt( 1000.0 );
```

  - Calls function sqrt, which returns the square root of its argument
  - All math functions return data type `double`

5.3 Math Library Functions II

Commonly used math library functions:

Function	Description	Example
`sqrt( x )`	square root of x	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
`exp( x )`	exponential function e^x	exp(1.0) is 2.718282 exp(2.0) is 7.389056
`log( x )`	natural logarithm of x (base e)	log(2.718282) is 1.0 log(7.389056) is 2.0
`log10( x )`	logarithm of x (base 10)	log10(1.0) is 0.0 log10(10.0) is 1.0 log10(100.0) is 2.0
`fabs( x )`	absolute value of x	fabs(5.0) is 5.0 fabs(0.0) is 0.0 fabs(-5.0) is 5.0

5.3 Math Library Functions III

Function	Description	Example
`ceil( x )`	rounds x to the smallest integer greater than x	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
`floor( x )`	rounds x to the largest integer less than x	floor(9.2) is 9.0 floor(-9.8) is -10.0
`pow( x, y )`	x raised to power y (x^y)	pow(2, 7) is 128.0 pow(9, .5) is 3.0
`fmod( x, y )`	remainder of x/y as a floating point number	fmod(13.657, 2.333) is 1.992
`sin( x )`	trigonometric sine of x (x in radians)	sin(0.0) is 0.0
`cos( x )`	trigonometric cosine of x (x in radians)	cos(0.0) is 1.0
`tan( x )`	trigonometric tangent of x (x in radians)	tan(0.0) is 0.0

5.4 Benefits of Functions

Benefits of functions

- Divide and conquer
  - Manageable program development
- Software reusability
  - Use existing functions as building blocks for new programs
  - Abstraction - hide internal details (library functions)
- Avoid code repetition

5.5 Function Definitions I

Function definition format

return_value_type function_name(argument_list) {
declarations;
statements;
return control;
}

- Function_name: any valid identifier
- Return_value_type: data type of the result
  - **void** – indicates that the function returns nothing
- Argument_list: comma-separated list, declares arguments
  - A type must be listed explicitly for each argument

5.5 Function Definitions II

Function definition format (cont...)

return_value_type function_name(argument_list)
{
declarations;
statements;
return control;
}

- Definitions and statements: function body (block)
  - Variables can be declared inside blocks (can be nested)
  - Functions cannot be defined inside other functions
- Returning control
  - If nothing returned (**void**)
    - **return;**
    - or, until reaches right brace (})
  - If something returned
    - **return expression;**

5.5 Function Definitions - Example I

Example

```
int maximum( int x, int y, int z ); //function prototype

int main() {
   int num1, num2, num3; // three integers
   setbuf(stdout, 0);

   printf( "Enter three integers: \n" );
   scanf( "%d", &num2 );
   scanf( "%d", &num1 );
   scanf( "%d", &num3 );

   // num1, num2 and num3 are arguments to the maximum function call
   printf( "Maximum is: %d\n", maximum( num1, num2, num3 ) );
   return 0;
}
```


5.5 Function Definitions - Example II

Function body:

```
/* Function maximum definition */
int maximum( int x, int y, int z ){

   int max = x;      // assume x is largest

   if ( y > max ) {  // if y > max, then max = y
      max = y;
   }
   if ( z > max ) {  // if z > max, then max = z
      max = z;
   }

   return max;       // max is largest value
}
```

5.5 Function Definitions - Example III

Output:

```
Enter three integers:
22
85
17
Maximum is: 85
Enter three integers:
85
22
17
Maximum is: 85
Enter three integers:
22
17
85
Maximum is: 85
```


5.6 Function Prototypes I

Function prototype

- Function name
- Arguments – parameters that the function receives
- Return type – data type function returns
- Used to validate functions and function calls
- In Structured Programming - function prototypes must always be given

```
int maximum( int x, int y, int z );
```

  - Receives 3 **int**'s
  - Returns an **int**

5.6 Function Prototypes II

Promotion rules and conversions

- Casting to different types:
  - Using an integer as a real number: (float) int1
  - Using a real number as an integer: (int) float1
- Converting to "lower" types can lead to errors
- Convert type

```
int mval;
float a,b,c;
a = 10.0;
b = 12.0;
c = 11.0;
mval = maximum( ( int ) a, ( int ) b, ( int ) c );
```


5.6 Function Prototypes III

Promotion hierarchy for data types

Data types	printf conversion specifications	scanf conversion specifications
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
short	%hd	%hd
char	%c	%c

Perspective

Today

Functions I

- Introduction to functions
- `math` library functions
- Benefits of functions
- Function definitions
- Function prototypes

Next lecture

Functions II

- Header files
- Function calls: by value and by reference
- Example

Homework

1. Study Sections 5.1-5.6 in Deitel & Deitel
2. Do Self Review Exercises 5.1(a)-(h), 5.3-5.5 in Deitel & Deitel
3. Do Exercises 5.8, 5.15, 5.18, 5.25 in Deitel & Deitel