(2-2) Functions I
H&K Chapter 3

Instructor – Beiyu Lin

CptS 121 (May 9th, 2019)

Washington State University

Review -- Functions

```
#include<stdio.h>    /* starting with including libraries*/
#include<stdlib.h>
double get_grade_point(void);

int main(void)
{    /*Instructions for the machine to execute*/

       double grade1 = 0.0;
       grade1 = get_grade_point();

       return 0;
}

double get_grade_point(void)
{
      double grade_point = 0.0;
      printf("Please enter your grade point for your course:");
      scanf("%lf", &grade_point);
      return grade_point;
}
```


Problem Solving Example (1)

- Problem Statement: Write a program that computes your grade point average after completion of 3 courses.
- Inputs:
  - Grade point and number of credits for course 1
  - Grade point and number of credits for course 2
  - Grade point and number of credits for course 3
- Outputs
  - Grade point average (GPA)
- Relevant formula: GPA = ((grade_point1 * num_credits1) + (grade_point2 * num_credits2) + (grade_point3 * num_credits3)) / total_num_credits

Problem Solving Example (2)

- Initial algorithm
  - Get the grade points earned from each class
  - Get the credit hours for each class
  - Compute the average of the grade points
  - Display the results
- Refined algorithm
  - Get the grade points earned from each class
  - Get the credit hours for each class
  - Compute the grade point weighted by credits hours

    ```
    weighted_gp = (grade_point1 * num_credits1) + (grade_point2 * num_credits2) +
        (grade_point3 * num_credits3);
    ```
  - Total the credits across the classes
  - Compute the average of the grade points

    ```
    gpa = weighted_gp / total_num_credits;
    ```
  - Display the results

Problem Solving Example (3)

- Implementation

```
#include <stdio.h>

int main (void)
{
        int num_credits1 = 0, num_credits2 = 0, num_credits3 = 0;
        double grade_point1 = 0.0, grade_point2 = 0.0, grade_point3 = 0.0,
                weighted_gp = 0.0, total_num_credits = 0.0, gpa = 0.0;

        /* Get the grade points and credits */
        printf ("Please enter your grade point for computer science course 1: ");
        scanf ("%lf", &grade_point1);
        printf ("Please enter the number of credits possible for computer science course 1: ");
        scanf ("%d", &num_credits1);

        printf ("Please enter your grade point for calculus course 2: ");
        scanf ("%lf", &grade_point2);
        printf ("Please enter the number of credits possible for calculus course 2: ");
        scanf ("%d", &num_credits2);

        printf ("Please enter your grade point for physics course 3: ");
        scanf ("%lf", &grade_point3);
        printf ("Please enter the number of credits possible for physics course 3: ");
        scanf ("%d", &num_credits3);
```


Problem Solving Example (4)

```
    /* Compute grade point weighted by credit hours */
    weighted_gp = (grade_point1 * num_credits1) + (grade_point2 * num_credits2)
    +  (grade_point3 * num_credits3);

    /* Total the credits across the classes */
    total_num_credits = num_credits1 + num_credits2 + num_credits3;

    /* Compute gpa */
    gpa = weighted_gp / total_num_credits;

    /* Display results */
    printf ("GPA: %.2lf\n", gpa);

    return 0;
}
```


Notes on Example

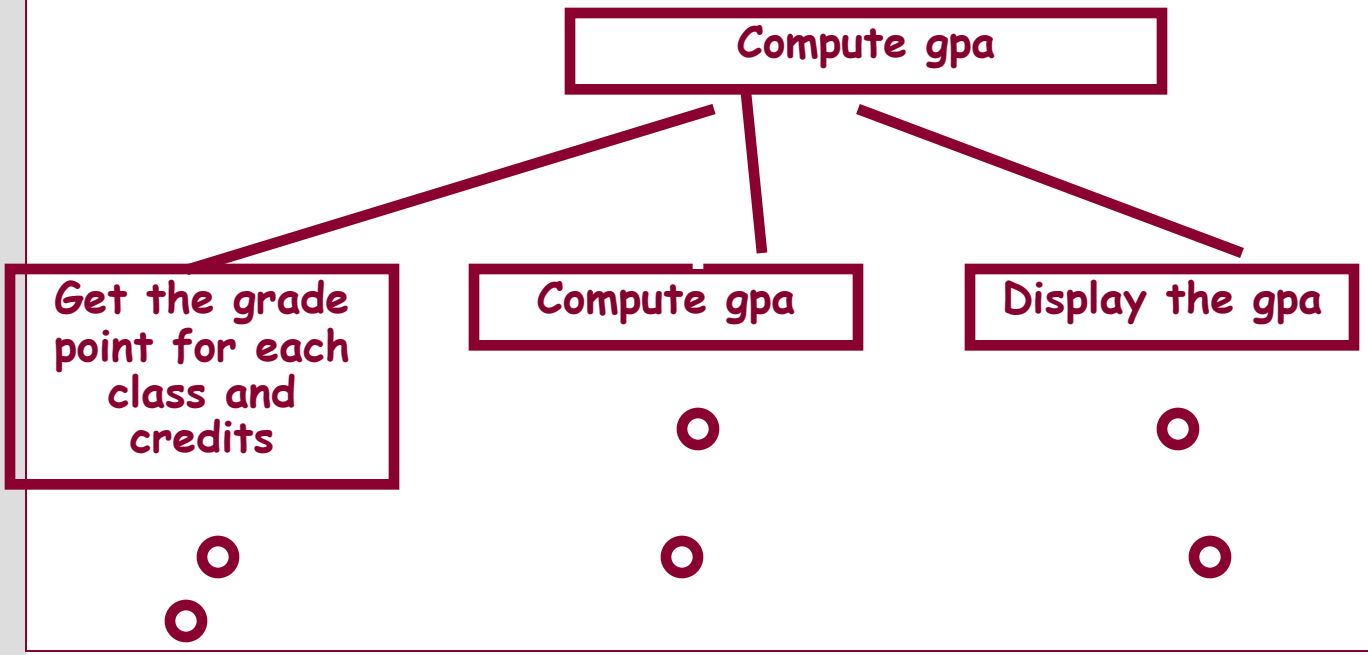
- It's redundant!
  - We're using the exact same sequence of commands (printf, scanf) to obtain the three grade points and credits
- Is there a better (less redundant, easier to read, more concise) way to approach this problem?

Top-Down Design (1)

- Allows us to manage the complexity of a problem by decomposing it into smaller, less complex subproblems
- A divide and conquer approach
- By solving each subproblem and combining solutions, we solve the overall problem
- We only need to solve each subproblem once, rather than having to "reinvent the wheel" each time

Top Down Design (2)

- Example: Compute and display the gpa




```
get_grade_point()  compute_weighted_gp ()      display_gpa ()
get_credits ()     compute_gpa ()
```


Functions (1)

- Enable us to implement top-down design
- Self-contained “mini-programs” that solve a problem
- General rule-of-thumb
  - 1 function = 1 task = 1 algorithm
- You already have some practical understanding of functions from your mathematical background
  - $f(x) = x^2 - 4x + 4$
  - In C, we pass the value of x into a function called “f” and get a result back

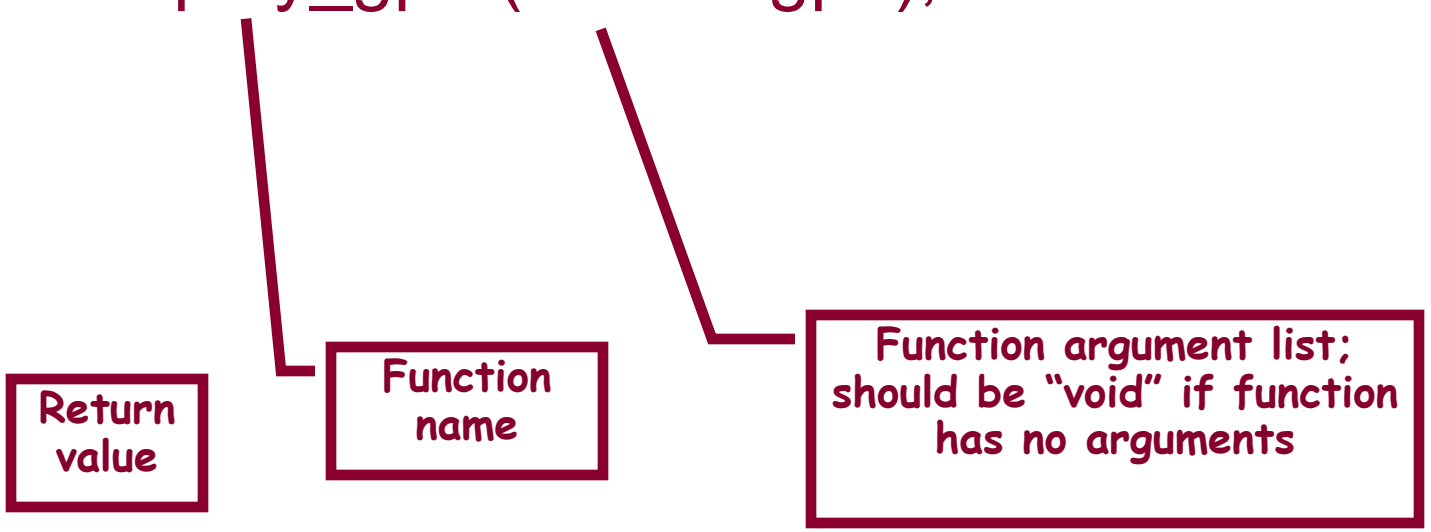
Functions (2)

- May have “input arguments” (also called “input parameters”)
  - The inputs to the function
- May return results in two ways:
  - Function result: the return statement specifies this
  - “output arguments” (also called “output parameters”): arguments into which the function places values to be passed back to the caller (more advanced; we’ll look at these later)

Functions (3)

- Anatomy of a function prototype:

```
void display_gpa (double gpa);
```



Functions (4)

- The GPA example revisited

```
double get_grade_point (void);
int get_credits (void);
double compute_weighted_gp (double grade_point1, double grade_point2, double grade_point3,
                            int num_credits1, int num_credits2, int num_credits3);
double compute_gpa (double weighted_credits, int total_num_credits);
void display_gpa (double gpa);

int main (void)
{
        int num_credits1 = 0, num_credits2 = 0, num_credits3 = 0;
        double grade_point1 = 0.0, grade_point2 = 0.0, grade_point3 = 0.0,
                        weighted_gp = 0.0, total_num_credits = 0.0, gpa = 0.0;

        /* Get the grade points and credits */
        grade_point1 = get_grade_point ();
        num_credits1 = get_credits ();

        grade_point2 = get_grade_point ();
        num_credits2 = get_credits ();

        grade_point3 = get_grade_point ();
        num_credits3 = get_credits ();

        /* Compute credit hours earned */
        weighted_gp = compute_weighted_credits (grade_point1, grade_point2, grade_point3,
                                                num_credits1, num_credits2, num_credits3);
- /* Should we have a new function for the sum? If so what would it look like?
        total_num_credits = num_credits1 + num_credits2 + num_credits3;
        /* Compute gpa */
        gpa = compute_gpa (weighted_gp, total_num_credits);

        /* Display results */
        display_gpa (gpa);

        return 0;
}
```


Functions (5)

- Definition of `get_grade_point ()`

```
/* Prompts the user for a grade point earned for a course */

double get_grade_point (void)
{
      double grade_point = 0.0;
      printf ("Please enter your grade point for your course: ");
      scanf ("%lf", &grade_point);
      return grade_point;
}
```


Functions (6)

- Definition of `get_credits ()`

```
/* Prompts the user for the number of credits for a course */

int get_credits (void)
{
        int num_credits = 0;

        printf ("Please enter the number of credits possible for your course: ");
        scanf ("%d", &num_credits);

        return num_credits;
}
```


Functions (7)

- Definition of `compute_weighted_gp ()`

```
double compute_weighted_gp (double grade_point1, double grade_point2, double
      grade_point3, int num_credits1, int num_credits2, int num_credits3)
{
      double weighted_gp = 0.0;

      weighted_gp = (grade_point1 * num_credits1) + (grade_point2 *
                              num_credits2) + (grade_point3 * num_credits3);

      return weighted_gp;
}
```


Functions (8)

- Definition of `compute_gpa ()`

```
double compute_gpa (double weighted_gp, int total_num_credits)
{
      double gpa = 0.0;

      gpa = weighted_gp / total_num_credits;

      return gpa;
}
```


Functions (9)

- Definition of `display_gpa ()`

```
/* Outputs the calculated gpa to the screen */

void display_gpa (double gpa)
{
    printf ("GPA: %.2lf\n", gpa);
}
```

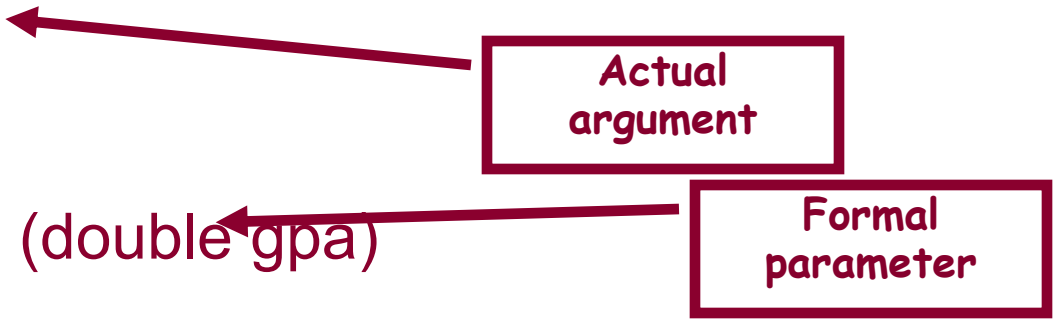

Functions (10)

- How Functions are compiled
  - Function prototypes tell compiler what functions are defined
  - When a function call is encountered within main, the compiler is already aware of it
  - After compilation of main function, each function is compiled
    - Machine language statement inserted at end of each function that transfers control back to caller (in main)
- How functions are executed
  - When a function is called, memory for local variables is allocated
  - Memory is released upon completion of function execution (→ local function variables do not “outlive” function)

Functions (11)

- Example 1: What happens when a function is called

```
display_gpa (3.4);
```




```
 void display_gpa (double gpa)
{
   printf ("GPA: %.2lf\n", gpa);
}
```


Functions (12)

- Why Use Functions: A Summary of Advantages
  - Break a large, complex solution into logical units
    - Individual members of a programming team can work on each unit independently
  - Procedural abstraction
    - The main function need not be aware of the details of *how* a function works—just *what* it does
    - Thus, during high-level problem solving activities, we won't get bogged down in detail
    - We can defer the details until we're ready to write the individual functions

Functions (13)

- Why Use Functions: A Summary of Advantages (cont.)
  - Reuse
    - Recall our comment on the original version of the program to compute and display the gpa of classes
      - Redundant: Much code was duplicated
    - Why re-write sections of code that have already been written and tested?
    - Functions allow us to package up a solution into a bite-size chunk that can be reused over and over

Functions (14)

- Why Use Functions: A Summary of Advantages (cont.)
  - Testing
    - Allows for more efficient testing and “bug” resolution
    - Each function is tested as it is implemented

C Math Functions

- The C math library `<math.h>` defines numerous useful mathematical functions
- This library is an excellent example of the power of functions
  - Commonly-used mathematical operations are packaged up in functions that can be re-used over and over

C Math Functions

- Some C Math Library Functions
  - `int abs(int x)  (<stdlib.h>)`
  - `double ceil(double)`
  - `double floor(double)`
  - `double cos(double)`
  - `double sin(double)`
  - `double tan(double)`
  - `double exp(double)`
  - `double fabs(double)`
  - `double log(double)`
  - `double log10(double)`
  - `double pow(double,double)`
  - `double sqrt(double)`

Next Lecture…

- More examples of top-down design involving
  - Functions with and without input arguments
  - Functions with and without output values
- The use of test drivers to verify that functions work
- Common programming errors

References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8th Ed.)*, Addison-Wesley, 2016

Collaborators

- Chris Hundhausen