Instructor: Professor Neena Kaushik

Office hours (in AT 203): Mondays: 3:15 to 4:15 p.m.
Wednesdays: 10 to 11 a.m.
and by appointment

Email: kaushikneena@fhda.edu

Course website: http://www.deanza.edu/faculty/kaushikneena

Lecture: 12:30 - 2:10 p.m. in AT 312 (1:30 – 1:35 p.m. break) (Mondays & Wednesdays)

Lab: 2:15 - 2:40 p.m. in AT 312 (Mondays & Wednesdays)

Lectures: 7 and 8

Dates: October 13 and October 15

Textbook Chapter: 4

Topics Covered: User defined functions and programming examples

Sentences in quotes have been taken as is from the textbook. The material in these notes has been prepared by referring the textbook and the slides for the instructor.

1) “In top-down design, a program is divided into a main module and its related modules.
2) Each module is in turn divided into submodules until the resulting modules are intrinsic, that is, until they are implicitly understood without further division.
3) In C a program is made up of one or more functions, one and only one of which is called main.
4) The execution of the program always starts with main but it can call other functions to do some parts of the job.”

Programmers create functions so that
1) Program is divided into smaller easily understood parts
2) Each function is a separate unit and can be tested independently. Once it is verified to work correctly, it can be distributed as a library.

Let’s take a look at the hello message program
Now let’s write a function which prints a hello message and make the main program call it.

#include “stdafx.h”
#include <stdio.h>
#include <conio.h>
#include <conio.h>
int main(void)
{
    printf("\n Hello CIS15AG-03 students !!");
    getch();
    return 0;
}

void print_hello(void) // Function declaration
int main(void)
{
    print_hello();
    getch();
    return 0;
}

/******************************************************************************
** Function: print_hello()
** Inputs:  None
** Returns: void
** What the function does: It prints a message
*******************************************************************************/

void print_hello(void) // Function definition
{
    printf("\n Hello CIS15AG-03 students !!");
    return;
}

Let's take a look at a program which reads a single assignment score
/******************************************************************************
** Program written by: Neena Kaushik
** Inputs:  The first assignment score
** Returns: An Integer
** What the program does: It reads the first assignment score and then prints it
*******************************************************************************/
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>
int main(void)
Now, let's write a function which reads an assignment score and returns it.

/**************************
** Program written by: Neena Kaushik
** Inputs: None
** Returns: An Integer
** What the program does: It prints the assignment score by first calling a function read_assignment_score()
**************************/

#include "stdafx.h"
#include <stdio.h>
#include <conio.h>

float read_assignment_score(void) // Function declaration

int main(void)
{
    float assignment_score_1;

    assignment_score_1 = read_assignment_score();
    printf("\n Assignment 1 score is %f", assignment_score_1);

    return 0;
}
getch();
return 0;
}

QUEUE END

/** Function: read_assignment_score(void)
** Inputs: None
** Returns: Floating point number
** What the function does: Reads the assignment score and returns it
**QUEUE END

float read_assignment_score(void) // Function definition
{
    float score;

    printf("\n Please enter your assignment score: ");
    scanf("%f", &score);

    return score;
}

Now, let’s write a program with a function which accepts a number and returns its least significant digit

QUEUE END

/** Program written by: Neena Kaushik
** Inputs: Accepts an integer
** Returns: An Integer
** What the program does: It prints the least significant digit by first calling a function least_significant_digit()
**QUEUE END
```c
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>

int least_significant_digit(int number) // Function declaration

int main(void)
{
    int number;
    int digit;

    printf("\n Please enter your number");
    scanf("%d", &number);
    digit = least_significant_digit(number);
    printf("\n Least significant digit is %d", digit);

    getch();
    return 0;
}

/*****************************************************/
** Function: least_significant_digit(int number)  
** Inputs:  An Integer                           
** Returns: The least significant digit        
** What the function does: Takes a number, finds the least    
** significant digit and returns it.           
*****************************************************/

int least_significant_digit(int number) // Function definition
```
{ 
    int digit; // Local Declaration 
    digit = number %10; 
    return digit; 
}

Function Declaration

return_type function_name (formal parameter list)

Function Definition

return_type function_name (formal parameter list)

{
    // Local Declarations 
    // Statements 
}

Examples of function Declaration

<table>
<thead>
<tr>
<th>Example</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>1   int multiply (int num1, int num2)</td>
<td>Multiplies two numbers.</td>
</tr>
<tr>
<td>2   void print_one(int x)</td>
<td>It prints the number x.</td>
</tr>
<tr>
<td>3   int get_quantity(void)</td>
<td>It reads a quantity and returns it.</td>
</tr>
<tr>
<td>4   int sqr(int x)</td>
<td>It returns the square of number x.</td>
</tr>
<tr>
<td>5   int get_num(void)</td>
<td>It reads a number.</td>
</tr>
<tr>
<td>6   int first_digit(int num)</td>
<td>It returns the first digit of</td>
</tr>
</tbody>
</table>
Let's analyze the function in print hello message program
Let’s analyze the function in the read assignment score program

<table>
<thead>
<tr>
<th>Construct</th>
<th>What it is</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 void read_assignment_score()</td>
<td>Function Declaration</td>
</tr>
<tr>
<td>2 float</td>
<td>Return Type</td>
</tr>
<tr>
<td>3 None</td>
<td>Formal Parameters</td>
</tr>
<tr>
<td>4 read_assignment_score()</td>
<td>Function call from main</td>
</tr>
</tbody>
</table>

Let’s analyze the function in the least significant digit program

<table>
<thead>
<tr>
<th>Construct</th>
<th>What it is</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 int least_significant_digit(int number)</td>
<td>Function Declaration</td>
</tr>
<tr>
<td>2 int</td>
<td>Return Type</td>
</tr>
<tr>
<td>3 int number</td>
<td>Formal Parameters</td>
</tr>
<tr>
<td>4 least_assignment_score(number)</td>
<td>Function call from main</td>
</tr>
</tbody>
</table>

The calling function and the called function can communicate in three different ways.

<table>
<thead>
<tr>
<th>Communication Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Downward</td>
<td>void print_one(int x)</td>
</tr>
<tr>
<td>2 Upward</td>
<td>float read_assignment_score(void) void read_assignment_scores(float *score1, float *score2)</td>
</tr>
<tr>
<td>3 Bi-directional</td>
<td>void swap(int *val_1, int *val_2)</td>
</tr>
</tbody>
</table>

Downward Communication
Now, let’s write a program with a function which accepts a number from the main program and prints it once to the screen.

```c
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>

void print_one(int number) // Function declaration

int main(void)
{
    int number;

    printf("\n Please enter your number");
    scanf("%d", &number);
    print_one(number);

    getch();
    return 0;
}

/** Function: print_one(int number) */
```
** Inputs: An Integer  
** Returns: void  
** What the function does: Takes a number and prints it  
*******************************************************************************/

void print_one(int number) // Function definition  
{  
    printf("\n The number is %d", number);  
    return;  
}

Upward Communication  
Now, let’s write a program with a function which accepts the first assignment score and returns it to the calling program.

*******************************************************************************/

** Program written by: Neena Kaushik  
** Inputs: Accepts an assignment score  
** Returns: An Integer  
** What the program does: It calls a function which reads an assignment score and returns it to the main program.  
*******************************************************************************/

#include “stdafx.h”  
#include <stdio.h>  
#include <conio.h>

float read_assignment_score(void) // Function declaration

int main(void)
{  
    float score;

score = read_assignment_score();
printf("\n Your assignment score is %f", score);

getch();
return 0;
}

/** Function: read_assignment_score(void)
 ** Inputs:  None
 ** Returns: A floating point number
 ** What the function does: Reads the assignment score and
returns it
***************************************************************************/

float read_assignment_score(void) // Function definition
{
    float score;
    printf("\n Please enter your assignment score: ");
    scanf("%f", &score);
    return score;
}

Now, let’s write a program with a function which accepts the first two assignment scores
and returns them to the calling program.

***************************************************************************/

/** Program written by: Neena Kaushik
 ** Inputs:  Accepts two assignment scores
 ** Returns: An Integer
*/
** What the program does: It calls a function which reads two assignment scores and returns them to the main program.

#include "stdafx.h"
#include <stdio.h>
#include <conio.h>

void read_assignment_scores(float *score1, float *score2)
    // Function declaration

int main(void)
{
    float score1;
    float score2;
    read_assignment_scores(&score1, &score2);
    printf("\n Your assignment scores are %f %f", score1, score2);

    getch();
    return 0;
}

******************************************************************************
** Function: read_assignment_scores(float *score1, float *score2)
** Inputs:  The address of two floating point numbers
** Returns: void
** What the function does: Reads the assignment score and since their addresses are passed, the values are stored in them.
******************************************************************************
void read_assignment_scores(float *score1, float *score2)
  // Function definition
  {
      printf("\n Please enter your assignment scores:");
      scanf("%f %f", &score1, &score2);
      return;
  }

If the scores were hard coded, the function would be

void read_assignment_scores(float *score1, float *score2)
  // Function definition
  {
      *score1 = 100;
      *score2 = 100;
      return;
  }

Bi-directional communication

Now, let’s write a program with a function which accepts two numbers, swaps them and returns them to the calling program.

	//*****************************************************************************
  ** Program written by: Neena Kaushik
  ** Inputs: None
  ** Returns: An Integer
  ** What the program does: It calls a function which accepts two numbers, swaps them, and returns them to the calling program.
  *****************************************************************************/
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>

void swap(int *val_1, int *val_2) // Function declaration

int main(void)
{
    int v_1;
    int v_2;

    v_1 = 1;
    v_2 = 2;

    printf("\n Before swap: v_1 = %d v_2 = %d", v_1, v_2);
    swap(&v_1, &v_2);
    printf("\n After swap: v_1 = %d v_2 = %d", v_1, v_2);

    getch();
    return 0;
}

/******************************************************************************
** Function: swap(int *v_1, int *v_2)
** Inputs:  Address of two integers v_1 and v_2
** Returns: void
** What the function does: Swaps the integers v_1 and v_2
*******************************************************************************/

void read_assignment_scores(int *v_1, int *v_2) // Function definition
```c
{  
    int temp;
    temp = *v_1;
    *v_1 = *v_2;
    *v_2 = temp;
    return;
}
```

### Rules

<table>
<thead>
<tr>
<th></th>
<th>Downward communication</th>
<th></th>
</tr>
</thead>
</table>
| 1 |                        | a) “Use values in the function call to pass data.  
b) Use appropriate data types in the function parameter list to receive data values.  
c) Use the parameter identifiers in the called function to access the local copies of the data.” |
|   | Upward and Bi-directional communication | a) “Use &variableName in the function call to pass a reference to the variable.  
b) Use type* in the function parameter list to receive the variable’s address.  
c) Use *parameterName in the function to reference the original variable.” |