CMPSC 297 - Introduction to C Programming

PennState

Systems and Internet Infrastructure Security Labratory

Week #1

Professor Patrick McDaniel

Logistics

PennState

- Canvas (if enrolled)
- https://psucmpsc311.github.io/ (if auditing)

- Piazza: http://piazza.com/psu/fall2021/cmpsc297

Practice – RUN 1

0) Clone git repository with Makefile for week#1 (https://classroom.github.com/a/7Qzr-KMS)

1) Open week1.c in an editor (create a new file)

2) Add the main function to the code (you can use void as the parameters)

3) Add a print statement "Hello, world\n" using the printf function. Add the include file for the printf function (stdio.h).

4) Return the value 0 from main

RUN 1 - save, file then make and run the program.

```
-- EXPECTED OUTPUT --
$ make
gcc -I. -c -g -Wall -I. week1.c -o
   week1.o
gcc  week1.o -o week1
$ ./week1
Hello, world
$
---
```

Practice – RUN 2

PennState

5) Create a functional prototype for the function **ABOVE** the main function. The function should be called function1, receive two int parameters (a and b), and return an int

6) Create the body of the function function1 **BELOW** the main function. The function should return a if a>b, b if b>a, and 0 if a == b. This will require you to use if and else if statements.

7) Add a comment above the function (neatly) that describes the function (you can past the test for #6 above).

8) Add the following calls to function1 in the main program after the hello printf.

```
printf("function 1 : A=8, B=6 -> %d\n", function1(8, 6));
printf("function 1 : A=3, B=6 -> %d\n", function1(3, 6));
printf("function 1 : A=6, B=6 -> %d\n", function1(6, 6));
```

RUN 2 - save, file then make and run the program.

```
-- EXPECTED OUTPUT --
$ make
gcc -I. -c -g -Wall -I. week1.c -o
week1.o
gcc  week1.o -o week1
$ ./week1
Hello, world
function 1 : A=8, B=6 -> 8
function 1 : A=3, B=6 -> 6
function 1 : A=6, B=6 -> 0
$
---
```

Practice – RUN 3

PennState

9) Create a second functional prototype above the main function, and the function body below main for the function function2. Function2 should receive a pointer to an array of integers, and the length of the array as an int. The function should return an int.

10) In the body of function2 walk the list of integers and count the number of even numbers and return that count.

11) Add the following global variables above main.

```
int arr1[5] = { 2, 4, 6, 9, 7 };
int arr2[6] = { 0x3, 0x6, 4, 8, 9, 11 };
```

12) Add the following calls to your main function (below the other calls):

```
printf("function 2 : arr1 has %d even integers\n", function2(arr1, 5));
printf("function 2 : arr2 has %d even integers\n", function2(arr2, 6));
```

RUN 3 - save, file then make and run the program.

```
-- EXPECTED OUTPUT --
$ make
gcc -I. -c -g -Wall -I. week1.c -o week1.o
gcc  week1.o -o week1
$ ./week1
Hello, world
function 1 : A=8, B=6 -> 8
function 1 : A=3, B=6 -> 6
function 1 : A=6, B=6 -> 0
function 2 : arr1 has 3 even integers
function 2 : arr2 has 3 even integers
$
---
```

Practice – RUN 4

13) Create a third functional prototype function3 which accepts one integer x and returns an integer. Create the body for the function.

14) Implement the function3 which computes the factorial as follows (for input x).

If x < 0, emit the following error message:

"function3 called with illegal input (%d)\n" (use printf)

If x == 0, return 0

If x > 0, then return the factorial value, which is the sum of the values from x to 1. e.g.,

x=3 -> return 3*2*1 = 6

x=3 -> return 5*4*3*2*1 = 120

* Note: You are NOT allowed to use a for statement. You can use a **while** or **do loop**, or implement this **recursively**.

RUN 4 - save, file then make and run the program.

```
-- EXPECTED OUTPUT --
$ make
gcc -I. -c -g -Wall -I. week1.c -o week1.o
gcc  week1.o -o week1
$ ./week1
Hello, world
function 1 : A=8, B=6 -> 8
function 1 : A=3, B=6 -> 6
function 1 : A=6, B=6 -> 0
function 2 : arr1 has 3 even integers
function 2 : arr2 has 3 even integers
Function3 (factorial) x=1 = 1
Function3 (factorial) x=3 = 6
Function3 (factorial) x=5 = 120
Function3 (factorial) x=0 = 0
function3 called with illegal input (-1)
Function3 (factorial) x=-1 = -1
$
---
```