Arizona State University
Lecturer Acuña (5), UGTA Karaliova (4)

SER334: Operating Systems & Networks
Revised 8/26/2019

# Unit 1 Sample Problems - C Programming (SOLN)

In this sample problem set, we will practice concepts of the C programming language.

- Length: 50 minutes with discussion.
- Questions: Q1-Q2, Q4, Q6, and Q8. (optional: Q3, Q5, Q7, and Q9)

## A Simple C Program

1. [Acuña] Consider the following implementations of the linear search algorithm in both Java and C:

```java
import java.util.Scanner;

public class LinearSearchExample {
    public static boolean find(int target, int[] pool) {
      for(int i = 0; i < pool.length; i++)
        if(pool[i] == target)
          return true;
      return false;
    }

    public static void main(String args[]) {
        Scanner scanner = new Scanner(System.in);
        int[] data = {4, 45, 8, 1, 3, 3, 22, 9};
        int target;

        System.out.println("What is the target number?");
        target = scanner.nextInt();
        if(find(target, data))
          System.out.println("Found.");
        else
          System.out.println("Missing.");
    }
}
```

```c
#include <stdio.h>

int find(int target, int[] pool, int n) {
    for(int i = 0; i < n; i++)
    if(pool[i] == target)
        return 1;
    return 0;
}

void main(int argc, char* argv[]) {
    int[] data = {4, 45, 8, 1, 3, 3, 22, 9};
    int target;

    printf("What is the target number?");
    scanf("%d", &target);
    if(find(target, data, 8))
        printf("Found.");
18         else
19             printf("Missing.");
20 }
```

(a) Identify 2 lines that are different because of the library (or run-time) being used: [2 points]

**Ans: [Acuña]**

(There are several answers to this question.)

Line 1 (Java): The package containing the Scanner object is imported instead of the stdio.h library.

Line 19 (Java): Instead of using System.out.println, the C program uses printf.

(b) Identify 2 lines that are different because of the language being used: [2 points]

**Ans: [Acuña]**

(There are several answers to this question.)

Line 4 (Java): the find function has a return type of int instead of boolean.

Line 5 (Java): instead of being able to do .length to get the number of elements in the array, we have to pass a separate variable called n with the size.

2. [Acuña] Consider the following program. It compiles without any compile-time errors in GCC, yet it contains a total of 4 issues (a combination of syntax, and logical problems). Study the program to identify all the issues. For each issue, list its type (syntactic, logical), what the problem is, and how to fix it.

```
1  #include <stdio.h>
2  int main() {
3          int input;
4          int result;
5          printf("Enter an integer number:\n");
6          scanf("d", input);
7
8          result = input % 2;
9
10         if (result = 0)
11                 printf("\nNumber %d is even.", input);
12         else
13                 printf("\nNumber %d is odd.", input);
14 }
```

(a) First issue: [1 point]

Ans: [Acuña] Syntactic. The percent sign is missing before the d control symbol for the scanf. Add the percent sign to fix.

(b) Second issue: [1 point]
Ans: [Acuña] Syntactic. The ampersand is missing for the scanf's parameter. Add the ampersand.

(c) Third issue: [1 point]
Ans: [Acuña] Semantic/logical. In the conditional of the if-statement, an assignment is used instead of a comparison. We need to change the = to ==.

(d) Forth issue: [1 point]
Ans: [Acuña] Semantic/Logical. It is missing a return statement. A return 0; should be added at the end of the program.

## Memory in C

3. [Acuña] Consider the following program:

```
#include <stdio.h>

int badfunction(int* num) {
    printf("num: %d\n", *num);
    *(num+1) = 7;
}

void main(void) {
    int dont = 1;
    int do = 2;
    int this = 3;

    printf("#1 dont: %d, do: %d, this: %d\n", dont, do, this);
    badfunction(&do);
    printf("#2 dont: %d, do: %d, this: %d\n", dont, do, this);
}
```

Trace this program, and give it's output. Explain why it generates that specific output. [2 points]

**Ans: [Acuña]**

```
#1 dont: 1, do: 2, this: 3
num: 2
#2 dont: 7, do: 2, this: 3
```

The memory for *do* and *dont* are stored contiguously in the stackframe for main(). When badfunction() is called, it gets an address to *do*, which it can manipulate to access anything else in the stack frame for main(). (Note: you would not be expected to know which of *dont* or *this* changed, since that depends on the structure of the stack frame as generated by the compiler.)

## C-Style Strings

4. [Karaliova] Consider the following declarations in C and Java. Answer the following for each declaration: 1) What data type is declared? 2) What value would we get if we attempt to access myExample[6]? 3) What value would we get if we attempt to access myExample[7]? [3 points]

**Ans: [Karaliova]**

(a) C:
char myExample[] = {'s', 'e', 'r', '3','3','4'};
1) Declared data type: array of characters.
2) myExample[6] is whatever is stored in memory at that location.
3) myExample[7] is whatever is stored in memory at that location.

(b) C:
char myExample[] = "ser334";
1) Declared data type: C-style string or null-terminated array of characters.
2) myExample[6] is null ( or \0)
3) myExample[7] is whatever is stored in memory at that location.

(c) Java:
char[] myExample = {'s', 'e', 'r', '3','3','4'};
1) Declared data type: array of characters.
2) Java throws IndexOutOfBound Exception that prevents myExample[6] from being accessed.
3) Java throws IndexOutOfBound Exception that prevents myExample[7] from being accessed.

5. [Karaliova] What is the difference between '\0' and '0' in context of C-style string? What could happen if we replace '\0' with '0' in a C-style string? [1 point]

**Ans: [Karaliova]**

\0 represents null character, and in context of C-style string, it is used as a flag (referred to as null character terminator) indicating the end of the string. If we replace '\0' with '0' in a C-style string, we could encounter a segmentation fault (memory leak) or a security issue as our program will gain access to bytes in memory that it should not be accessing in the first place.

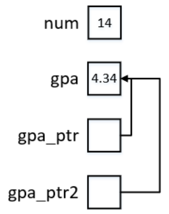
## Pointers

6. [Acuña] Consider the following fragment of code. Using box and arrow notation (boxes represent variables in memory, numbers in boxes are values, arrows show when a value is the address of another variable), draw out the variables and values that will exist during the program's execution.

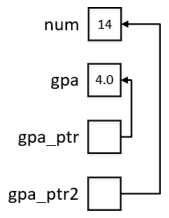
```
int num = 14;
float gpa = 4.34;
float* gpa_ptr = &gpa;
float* gpa_ptr2 = &gpa;
//point a

*gpa_ptr = 4.0;
gpa_ptr2 = (float*)(&num)
//point b
```

(a) What will exist at point a? [1 points]



(b) What will exist at point b? [1 points]



7. [Karaliova] In C, ** notation represents a double pointer (a pointer to a pointer). What is the difference between && and & in context of pointers? [1 point]

   **Ans: [Karaliova]**

   In context of pointers, & is a unary operator which returns address of variable following it. && is not used in context of pointers (as it wouldn't make much sense to have a double address or address of an address operator). Like in many other programming languages, && in C is a binary logical AND operator.

8. [Acuña] For the following C code, fill in the value of each variable at the given point in the code. If the value of the variable cannot be determined at a given point in the code, write unknown. If the value is an address, write "address of ____". (Hint: There is only one unknown value in this code.) [4 points]

```
int a, *b, **c, d;
a = 5;
d = a + 1;
b = &a;
c = &b
// Point 1
*b = 8;
b = &d;
**c = 3;
// Point 2
b++;
// Point 3
```

| Type | int | int (dereferenced) | int pointer | int pointer (dereferenced) | int double pointer | int |
|---|---|---|---|---|---|---|
| Variable Name | a | *b | b | *c | c | d |
| Point 1 | 5 | 5 | address of a | address of a | address of b | 6 |
| Point 2 | 8 | 3 | address of d | address of d | address of b | 3 |
| Point 3 | 8 | unknown | address of d+1 | address of d+1 | address of b | 3 |

## Memory Allocation

9. [Karaliova] Provide a use case example for situations when it makes sense to use a static variable inside a function versus a global variable and a use case example of the opposite (when it makes sense to use a global variable for the program instead of a static variable inside a function). For both use cases, explain your logic. [2 points]

   **Ans: [Karaliova]**

   There could be a multitude of use cases for these two alternative scenarios. The underlying logic is that static variable prevents other functions from accessing the variable if declared locally as opposed to a global variable, where all functions can read and modify the variable. My example is a calculator program. It would make sense to use a global variable for a counter that keeps track of all modifications, so that, for example, methods that perform addition, subtraction, multiplication, division, etc, all can increment this global modification counter whenever an operation is performed. On the other hand, static variables can be declared inside of each method to represent counters for a specific operation. That is, addition method can have a local static variable for addition counter only. Making addition counter static variable inside addition function ensures that other functions like subtraction, etc, cannot access it.