**SARC – Supplemental Instruction**

**COP3223 – Fall 2016**

**Exam 2 – Review**

**NOTE:** *This is a peer created document and may contain errors. Neither the author of this document nor Student Academic Resource Center can be held liable for its content. Use this as a general study tool and as preparation for the exam. If you ascertain any errors, you may inform the SI leader.*

*Section I.*

Answer the following questions with the most appropriate choice. **Unless stated otherwise, assume all variables used in the options are properly declared and initialized in the code.**

1. When using printf the conversion character (%f, %lf, %d, %s,…) do not always need to match the data type of the variable used.
2. True
3. False
4. Arrays cannot be passed by reference because they are already pointers.
5. True
6. False

**Recall:** the name of the array is the base address of the array, thus if you pass it into a function you are always passing by reference. You can pass individual cells of an array by value by specifying a cell.

1. An array can have at most how many dimensions?
2. 1
3. 2
4. 3
5. Unlimited
6. Once you declare an array in C like so : “int my\_array[100]” it is possible to change the size during program execution.
7. True
8. False
9. In the original C standards, variable length arrays have always been legal.
10. True
11. False

**Note:** while some (not all) compilers may allow this, this is **discouraged** programming practice. In the original C standards, under no circumstances was this allowed.

1. Use defined functions under ANY circumstance require function prototypes to be declared before the main function.
2. True
3. False.

**Note:** Functions can be defined before main and thus when you call them from main, prototypes are not required.

1. Pointer arithmetic is generally meaningless outside of arrays.
2. True
3. False

**Note:** Visit PowerPoint 6, slide 15 for more information on this. But in general iterating through arrays is the main reason we use pointer arithmetic.

1. The only way to initialize all cells of an array to a set of values is to use a loop.
2. True
3. False

**Note:** while you can in fact use a loop to accomplish this, you may also do this in the declaration of the array by using curly brackets and inputting values.

1. Which of the following is the correct way to pass an array into a function with the following definition: int find\_max( int array [], int size)
2. int find\_max ( array [], 10);
3. find\_max (array, 10);
4. find\_max (array [], 10);
5. int find\_max (array [], 10);
6. Functions defined by the programmer can be defined inside of other functions.
7. True
8. False

Note: They can be called but not defined inside of other functions.

1. Pointer variables contain:
2. The address of another variable
3. Indirection operators
4. Integers
5. The value of another variable
6. The operator \* indicates what, outside of a variable declaration?
7. A pointer variable
8. Dereferencing
9. Address of
10. Pass by reference
11. When a integer variable is used as an argument in a function call, what are you passing into the function?
12. The address of the variable
13. The value contained inside the variable

**Note:** passing in a normal variable as an arguments indicates you are passing by value and therefore a copy of the value inside that variable and not its address

1. What is the correct way to set a pointer variable equal to an address of a variable?
2. ptr = x;
3. &ptr = x;
4. \*ptr = x;
5. ptr = &x;
6. In C, you can use variables before you have to declare them in memory
7. True
8. False
9. The function strlen does which of the following:
10. returns an integer indicating the length of the string including the null terminator
11. returns an integer indicating the length of the string excluding the null terminator
12. returns an integer indicating the length of your character array
13. returns an integer indicating the lexicographical value of the string
14. Functions relating to strings are found in which library in C?
15. #include <stdlib.h>
16. #include <stdio.h>
17. #include <string.h>
18. #include <math.h>

**Note:** I originally had a typo in option c, I added an extra s, when it’s actually string.h. string.h contains all the important functions we use relating to strings, as the name might suggest.

1. Which of the following would return a value that is less than zero if str1 = “cat” and str2 = “dog”?
2. strmp (str1, str2);
3. strcmp (str1, str2);
4. strcpy (str1, str2);
5. strcmp (“str1”,”str2”)

**Note:** strcmp is a string function that compares both strings lexicographically, thus because “c” has a lower value than “d” in the ASCII table (because it comes first alphabetically) cat is deemed to be less than dog and this will result in a value less than zero.

1. When printing a floating point variable, what would be the correct way to restrict the precision to only two places past the decimal?
2. printf (“%.2lf”, my\_var);
3. printf(“%2f”, my\_var);
4. printf(“%.2f”, my\_var);
5. printf(“%4.2lf”, my\_var);

**Note:** the correct formatting for floating point variables would be %f[field width].[precision]

1. When using a pseudo random number generator in our program, why do we use srand()?
2. In order to restrict the range of our generator
3. In order to put values into rand
4. In order to prevent it from generating the same sequence of numbers
5. In order to enable us to set a variable equal to the output of rand()

**Note:** in programs that you use a random number generator, if you wish to use the function rand() more than one iteration through a loop, for example, seeding it will make sure the same sequence of numbers don’t get repeatedly generated

1. Which of the following segments restrict our range in our random number generator from 0 – 120?
2. my\_var = rand()/120;
3. my\_var = rand()%121;
4. my\_var = rand()%120;
5. my\_var = rand()+120;

**Note:** to restrict the lower bound of the array, mod by the difference of the lower and upper bound and then add the lower bound. The modulus operator will always at max return one less than the number you are modding by thus we use 121 so that it it is possible to get a value of 120.

1. Structure definitions (templates) can only be accessed from the function from which they were created
2. True
3. False

**Note:** *For the purposes of this course*, you are to make all your structure templates global variables such that you can instantiate a new instance of a structure from anywhere. Restricting your template to only one function is possible, though it is not good programming practice, as it defeats the intended purpose of using a structure to begin with.

1. Which of the following is a correct way to declare a structure definition and a data type called car?
2. struct car {

char make[100];

int year;

};

1. typedef struct {

char make[100];

int year;

} car;

1. struct {

char make [100];

int year;

}car;

1. car struct {

char make [100];

int year;

};

**Note:** You may *optionally* add a structure tag after struct, though in this case I chose not to. If it makes it more clear to you that car is the datatype and not the tag, then please do so when you write your programs.

1. What function uses your systems clock to return the time?
2. srand();
3. time();
4. rand();
5. scanf();
6. What is the correct way to access the member variable (called speed) of an instantiated structure called car1 (the template is below)?

struct Car

{

int speed;

char make[100];

};

int main (void)

{

typedef struct Car car;

car car1;

}

1. car1.speed;
2. Car.speed;
3. struct.speed;
4. struct Car.speed;

**Note:** Because the instantiated variable “car1” is not a pointer, you will use the dot operator and not the arrow operator to access individual members of that instance (individual variable).

1. Determine the output of the following segment of code: Assume you are using CodeBlocks and the GCC Compiler.

#include <stdio.h>

#include <stdlib.h>

int main (void)

{

// Assume the user will not type a name that will not fit in here

char name[100];

printf(“What is your name?\n”);

scanf (“%c”, name);

printf (“%c”, name);

return 0;

}

1. What is your name?

<User’s Response>

1. What is your name?

<Garbage value>

1. Nothing would print
2. The program would not run

**Note:** Since code is executed sequentially, the print statement would definitely print, and because I used the wrong conversion character “%c” instead of “%s” , this will result in garbage output and not the intended <user’s name>

1. Determine the output of the following segment of code: Assume you are using CodeBlocks and the GCC Compiler.

#include <stdio.h>

#include <stdlib.h>

typedef struct

{

int speed;

char make [200];

} car;

int main (void)

{

int count = 0, i;

car array[3];

do

{

printf(“What is your top speed?\n”);

scanf(“%d”, &array[count].speed);

printf(“What is your make?\n”);

scanf(“%s”, array[count].make);

count++;

} while (count != 3);

for ( i = 0 ; i < 3; i ++)

{

printf(“%d\n”, array[i].speed);

printf(“%s\n”, array[i].make);

}

return 0;

}

Write output below:

**Note:** *I originally had a typo in my struct template I meant to put “make” not “model”*. As discussed in session, since I only want to ask 3 times on account of how I only have three spaces in the array, using a do while loop could potentially make it ask one too many times, thus using a for loop would be a better idea as we would be absolutely certain of how many time it will run.

If the user types in (in this exact order): “120” “Mazda” “135” “Ford” “200” “Lambo” it would print sequentially to the screen.

28 - Determine the output of the following segment of code: Assume you are using CodeBlocks and the GCC Compiler.

#include<stdio.h>

int main()

{

int k, num=30;

k = (num>5 ? (num <=10 ? 100 : 200): 500);

printf("%d\n", num);

return 0;

}

a. 200

b. 30

c. 100

d. 500

**Note:** The printf statement uses the variable num and not k. num never changes.

The **?** would indicate the course of action to take if the statement preceding it is true and the **:** would indicate the course of action to take it is is false, in this case I have a nested statement.

*Section II. Write a program*

**Topic(s) :** FILE I/O, Character Representations

Write a program that reads text from a corpus named ***plain\_text.txt*** character by character, stores it in character array, and encrypts it by modding each individual character in the text by 26 and adding 1 to it, then outputting the new encrypted data to another file that we will name ***cipher\_text.txt***. You may assume all letters are lower case and the corpus does not contain any punctuation and it will contain anywhere from 0 to 1000 characters . You must first convert each character to its decimal representation before you perform arithmetic then convert the resulting number back to a character to print them to the output file.

Note: Using the ASCII representations in C, you may convert a character to an integer with the following code: *char[i] – ‘a’*

Write your code below:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main (void)

{

int counter = 0, length, i=0;

char plain[1000];

FILE \*input, \*output;

input = fopen ("plain\_text.txt", "r");

// If the file fails to open for whatever reason, let the user know and exit the program

if (input == NULL)

{

printf("The file failed to open!\n");

exit(0);

}

// Take in the text one character at a time. Use the counter i to go through each cell of the //array

while((fscanf(input,"%c",& plain[i]))!=EOF) //scanf and check EOF

{

i++;

}

length = strlen(plain);

for (i= 0 ; i < length ; i++)

{

printf("%c", plain[i]);

}

// We are done reading the characters, close the file for safety reasons

fclose(input);

for (i=0 ; i < length ; i++)

{

//Convert from char to int representation

plain[i] = plain[i] - 'a';

// Apply our simple, albeit kind of lame, encryption algorithm

plain[i] = (plain[i]%26)+1;

// Convert back to char

plain[i] = plain[i] + 'a';

}

// We are creating a new file called cipher\_text to print our output to

output = fopen("cipher\_text.txt", "w");

// Once again check to see if the file failed to initialize

if (output == NULL)

{

printf("Output file failed to be created\n");

exit(0);

}

for (i = 0; i < length; i++)

{

fprintf(output, "%c", plain[i]);

}

fclose(output);

return 0;

}