**Final Exam Fall 2020 Name: Ian Kaiserman**

**CSCD 240 (C and Unix Programming) Date: 12/10/20**

1. Explain the following pointer declaration.

double (\*ptr)[10][20][30][40];

**This is a pointer to a 4D array of doubles, the dimensions containing 10, 20, 30, and 40 double values, respectively.**

-------------------------------------------------------------------------------------------------------------------------------------

1. The following array needs to be sorted. The **sort** function needs the number of elements (here, n) in the array. How will you find ‘**n**’?

int num[] = {9, 7, 8, 6, 4, 5, 3, 1, 2, 0};

sort(num, n);

**sizeof(num)/sizeof(int)**

-------------------------------------------------------------------------------------------------------------------------------

1. Take a look at the following code. What will be printed by the printf statement?

#include <stdio.h>

int main(){

int \*ptr[10];

int a[10] = {0,1,2,3,4,5,6,7,8,9};

ptr[0] = a;

printf("%d\n", \*ptr[2]);

return 0;

}

**Memory address of index 2 of ptr**

1. Consider the following expressions. What will be the value of ‘**r**’?

int i = 3, j = 5, k;

int \*p = &i, \*q = &j, \*r;

\*(r = &k) = \*p \* \*q;

**R is the memory address of k whose value is now 15**

-----------------------------------------------------------------------------------------------------------------------

1. What will be printed by the following program?

#include <stdio.h>

void function(int array[]);

int main(){

int x[] = {1,2,3,4,5,6,7,8,9};

printf("size=%d\n", sizeof(x));

printf("x[5]=%d\n", x[5]);

function(x);

return 0;

}

void function(int array[]){

printf("size=%d\n", sizeof(array));

printf("x[5] = %d\n", array[5]);

array = array + 1;

printf("x[5] = %d\n", array[5]);

}

**size=36**

**x[5]=6**

**size=8**

**x[5] = 6**

**x[5] = 7**

------------------------------------------------------------------------------------------------------------------------

1. What is the difference between **++\*ptr** and **\*ptr++** ? Explain in the context of the following code snippet.

int i;

int array[SIZE];

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

array[i] = i;

int \*ptr;

ptr = &array[0];

**++\*ptr makes the computation before the following statement, \*ptr++ makes the computation after the following statement. In the code snippet, the i++ is applied but doesn’t affect the read value of i until after the code block has completed, causing the last runthrough of the code to be 1 less than SIZE.**

1. What will be printed by the following program?

#include <stdio.h>

int main(){

int total = 23;

int count = 10;

double average = total/count;

printf(“Average is: %f\n”, average);

return 0;

}

**Average is: 2.0**

-----------------------------------------------------------------------------------------------------------------

1. What will be printed by the following program?

#include <stdio.h>

#define SIZE 10

int main(){

int i;

int array[SIZE];

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

array[i] = i;

int \*ptr;

ptr = &array[5];

ptr++;

printf("%d ", ptr[0]);

putchar('\n');

return 0;

}

**6**

1. In the following code, what is the difference between ptr++ and p++?

#include <stdio.h>

int main(){

int \*ptr;

int (\*p)[5];

int arr[5];

ptr = arr;

p = &arr;

ptr++;

p++;

return 0;

}

**since ptr is a pointer to a singular int, ptr++ is changing the value that it is pointing to in terms of arr. However, p is a pointer to an array of integers, so p++ is traversing itself, which happens to also be pointing at arr, but array indexing notation also works on p, where it doesn’t work on ptr.**

Use the following program to answer Question 10 and Question 11.

#include <stdio.h>

int main(){

int a = 2, b=3, c =4, d = 5;

int (\*p)[5];

int \*pOne[5];

int arr[5] = {1,2,3,4,5};

p = &arr;

pOne[0] = arr;

pOne[1] = &a;

pOne[2] = &b;

pOne[3] = &c;

pOne[4] = &d;

printf("\*\*p is %d\n", \*\*p);

printf("(\*p)[4] is %d\n", (\*p)[4]);

printf("\*\*pOne is %d\n", \*\*pOne);

printf("\*pOne[4] is %d\n", \*pOne[4]);

return 0;

}

1. Explain ‘**int (\*p)[5]**’ and ‘**int \*pOne[5]**’.

**Int (\*p)[5] is a pointer to an array of 5 integers, while int \*pOne[5] is an array of 5 integer pointers**

1. What will be printed by the program?

**\*\*p is 1**

**(\*p)[4] is 5**

**\*\*pOne is 1**

**\*pOne[4] is 5**

1. Write down two differences between **gets** and **fgets**.

**Fgets is a safer and more flexible version of gets. Gets only allows for an output destination to be specified, while fgets allows a max size, a custom stream to read from, and also ends the string with null terminated \0 rather than the default newline \n which can cause problems.**

1. Consider the following two dimensional array. What is the value of array[2][3]?

int array[3][4] ={ 1, 2, 3, 4, 5, 6, 7, 8, 9};

**0**

----------------------------------------------------------------------------------------------------------------

1. Consider the following code. What output will be generated by the program?

#include <stdio.h>

#include <stdlib.h>

struct LinkedList{

int data;

struct LinkedList \*next;

};

void insert(struct LinkedList \*root, int item){

struct LinkedList \*temp = malloc(sizeof(struct LinkedList));

temp->data=item;

temp->next=NULL;

struct LinkedList \*p = root;

while(p->next!=NULL){

p=p->next;

}

p->next=temp;

}

/\* Function to print linked list \*/

void printList(struct LinkedList \*head){

struct LinkedList \*temp = head;

while (temp != NULL){

printf("%d ", temp->data);

temp = temp->next;

}

}

int main(){

struct LinkedList \*A = malloc(sizeof(struct LinkedList));

A->data = 5;

A->next = NULL;

insert(A, 6);

insert(A, 8);

printList(A);

return 0;

}

**5 6 8**

1. Explain the following **gdb** commands: ‘**list**’, ‘**next**’ and ‘**step**’.

**List lists lines in the program, or lists the specified contents i.e. function names, line numbers, etc**

**Next steps into the next function call silently, without showing inner program lines**

**Step steps into the next function call and shows inner program lines individually**

1. Assume there are three jobs running in the background at the same time. 1) **infinite**, 2) **prog** and 3) **factorial**. All 3 jobs were started from the current shell. Presuming the job id is the same as the order listed, give the unix command that would bring **factorial** into the foreground.

**fg 3**

1. Write a command that shows the lines with line numbers the string “**bird**” exists in a file named “**The Rhyme of Ancient Mariner**” in your current directory.

**Grep -ni “bird” “The Rhyme of Ancient Mariner”**

1. What are the differences among the following commands?

cat

cat < filename

cat > filename

cat >> filename

**standalone “cat” repeats everything typed into the shell until the command is terminated**

**cat < filename takes all the contents of the file and pastes it in the shell**

**cat > filename takes all the contents entered into the shell and pastes it into the file, overriding original contents**

**cat >> filename does the same as above, but appends the contents rather than replaces it**

---------------------------------------------------------------------------------------------------------------

1. Somewhere in some subdirectory of your home directory, there is a **.c** file named **tester.c**. Give the command that will locate occurrences of all files having this name.

**Find ~ tester.c**

---------------------------------------------------------------------------------------------------------------

1. What will the following ‘**cp**’ command do?

**cp test1.txt test2.txt**

**the command will copy the contents of test1.txt into test2.txt, overriding original contents. if the file does not exist, it will create one with the contents.**

**Identify whether the following statements are True or False**

1. In ‘**int a[10][10]**’, **&a[0]+1** refers to the address of row with index no. ‘1’.

**true**

1. You can start a foreground job by putting ‘&’ at the end of a command.

**False**

1. A pointer to a pointer stores the address of another pointer.

**true**

-------------------------------------------------------------------------------------------------------------------------------

1. Any function written to receive an array argument is given the address of the first element of the array passed to it.

**true**

-------------------------------------------------------------------------------------------------------------------------------

1. Consider an array declared as follows: int x[10]. The following declaration is correct:

printf(“Value of element at index ‘1’ is: %d\n”, 1[x]);

**false**

-------------------------------------------------------------------------------------------------------------------------------

1. In the following code snippet, the line marked with an arrow is a valid expression.

register int i;

scanf(“%d”, &i); **🡨**

**false**

------------------------------------------------------------------------------------------------------------------------------

1. Data held by a pointer is the address of a location in memory.

**true**

-----------------------------------------------------------------------------------------------------------------------------

1. A function declaration is about writing the function body and completing the function.

**false**

-----------------------------------------------------------------------------------------------------------------------------

1. When a program is compiled with **–c** option, relocatable object files are generated.

**true**

30) The following command will print “Hello Joseph”.

avar = Joseph

echo Hello $avar

**true**

1. Inserting an element at any point in the middle of an array is easier than inserting an element at any point in the middle of a linked list.

**false**

32) The following expression is correct.

struct temp{

float x;

float y;

struct temp z;

};

**false**

-----------------------------------------------------------------------------------------------------------------

33) The following expression is true.

**sizeof(char) > sizeof(float)**

**false**

----------------------------------------------------------------------------------------------------------------------------

34) Arrays are stored in the memory in row-major order.

**true**

------------------------------------------------------------------------------------------------------------------------------

35) A null character (‘\0’) is appended to the end of the string when a sentence is read from stdin

using fgets().

**true**

-------------------------------------------------------------------------------------------------------------------------------

36) The following command will redirect the standard output for the listing of all **.c** files to **output.txt**.

ls –l \*.c 2> output.txt

**false**

-------------------------------------------------------------------------------------------------------------------------------

37) The following unix command adds read and execute permissions for user and group.

**chmod ug+rx myﬁle**

**true**

-------------------------------------------------------------------------------------------------------------------------------

38) The metacharacter “^” is used in regular expressions to match a particular string at the beginning of a line.

**true**

-------------------------------------------------------------------------------------------------------------------------------

39) ‘**grep**’ command is recursive by default.

**false**

-------------------------------------------------------------------------------------------------------------------------------

40) A relative path is a path that starts from the home directory.

**false**

-------------------------------------------------------------------------------------------------------------------------------