National University of Computer and Emerging Sciences
Karachi Campus

Programming Fundamentals (CS1002) — Final Exam Solution

Date: 12/23/2024

Total Time: 3hr
Total Marks: **50**
Total Questions: 05

Course Instructor(s)

Dr. Farrukh Shahid, Basit Ali, Farooq Zaidi, Fahad Hussain, Nauraiz Subhan, Kariz Kamal, Kashif, Sumaiya, Bakhtawar, Rafia, Zain Noreen, Iqra Fahad.

_____ _____ _____

Roll No Section Student Signature

Do not write below this line.

Attempt all questions.

CLO 1: Describe fundamental concepts of structured and procedural programming, use pseudo codes and simple programs to understand control structures, iterative structures and functions using C language.

[Marks 10, 1 each, 25 min]

Q1: Choose one best answer for the following MCQs. Mention the question number and your choice clearly on answer script. Give reasons in 2 lines only for your choice.

(i) What is the primary factor that limits the depth of recursion in C?

a) Number of lines of code **b) Size of the stack**

c) Number of variables in the function d) Number of recursive calls in the main function

(ii) You can create a structure in C that contains itself as one of its members.

a) True **b) False** c) Sometimes true d) Can not be determined

(iii) A function in C cannot return a pointer to a local variable, because the local variable will be destroyed once the function returns.

a) True b) False c) Sometimes true d) Can not be determined

(iv) What is the effect of using the & (address-of) operator in a function call?

a) It passes the argument by value

b) It passes the address of the argument to the function.

c) It modifies the local variable in the calling function.

d) It dereferences the argument inside the function.

(v) Which of the following is not valid in a while loop?

a) A condition can be an expression.

b) The loop executes until the condition evaluates to false.

c) A while loop executes at least once, regardless of the condition.

d) The condition is checked before each iteration.

(vi) In C, the sizeof operator can be used to determine the size of a dynamically allocated memory block.

a) True **b) False** c) Sometimes true d) Can not be determined

(vii) Which of the following will result in undefined behavior in C?

a) Dereferencing a NULL pointer. b) Using an uninitialized pointer in a function.

c) Assigning a pointer to NULL. d) Using malloc and not checking if it returns NULL.

(viii) A for loop in C can only be used when the number of iterations is known beforehand.

a) True **b) False** c) Sometimes true d) Can not be determined

(ix) Which of the following is true regarding dynamic memory allocation in C?

a) calloc initializes the allocated memory to zero.

b) malloc initializes the memory to zero.

c) free allocates new memory for the program.

d) malloc and calloc cannot be used in the same program.

(x) C-Language only has the option to use functions as pass by value. Passing by reference is achieved (tricked) by using pointers and explicitly passing address (reference) of variables.

a) True b) False c) True on some compilers and versions of C-Lang

NOTE: In Question 1, 0.25 marks **for getting the correct answer and** 0.75 marks **for correct reasoning.**

CLO 2: *Examine code writing, compiling, debugging and program execution.*

Q2: Perform and show dry runs using tables or stacks or any way you see fit. Then give the output of the following programs. (**NOTE: There are no compile time errors in the code.**)

[Marks 10, 2.5 each, 35 min]

(i)

```
#include <stdio.h>
int mysteryRec(int a, int b) {
    if (b == 1)
        return a;
    else
        return a + mysteryRec(a, b - 1);
}

int main(void) {
    printf("The result is %u\n",
             mysteryRec(4, 5));
    return 0;
}
```

Output

```
The result is 20
```

(ii)

```
#include <stdio.h>
void processNums(int nums[], int n) {
    int i = -1, j;
    for (j=0; j<n; ++j) {
        if (nums[j] == 1) {
            i++;
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }//end if
    }//end for
}
int main() {
    int arr[] = {0,1,0,0,1,1,0,1,0,0};
    int n = 10;
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    processNums(arr, n);
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Output

```
0 1 0 0 1 1 0 1 0 0
1 1 1 1 0 0 0 0 0 0
```

(iii)

```
#include <stdio.h>
void my_print(char *str) {
    char *ptr; int i, j;
    for (i = 0; *(str + i)!='\0'; i++) {
        ptr = str;
        for (j = 0; j < (5 - i); j++) {
            printf("%c", *(ptr + j));
        }
        printf("\n");
    }
}

int main() {
    char text[] = "ABCDE";
    my_print(text);
    return 0;
}
```

Output

```
ABCDE
ABCD
ABC
AB
A
```

(iv)

```
#include <stdio.h>

int mystery(int bits ) {
    int i, total = 0;
    int mask = 1 << 31;
    for (i = 1; i <= 32; ++i,bits<<=1){
        if ((bits & mask) == mask )
            ++total;
    }
    return !(total % 2) ? 1 : 0;
}

int main( void ){
  printf("Result is %d\n", mystery(53));
  printf("Result is %d\n", mystery(8));
  printf("Result is %d\n", mystery(18));
  return 0;
}
```

Output

```
Result is 1
Result is 0
Result is 1
```

CLO3: *Justify problem solving techniques and analytical thinking by identifying the concepts and properties of algorithm.*

Q3: Consider the code snippet below and answer the questions that follow accordingly.
[Marks 10, 2.5 each, 30 min]

```
int **create_matrix(int m, int n){
  int **p, i;
  p = (int**)malloc(sizeof(int*)*m);

for (int i = 0; i < m; ++i){
  p[i] = (int*)malloc(sizeof(int)*n);}

  return p;
}

void destroy_matrix(int **p){
  free(p);
}
```

(i) If m is 5 and n is 4 how many function calls to malloc will be made in the function above?
Answer: *6, one before for loop and 5 inside the for loop.*

(ii) What and where if any error checking should have been performed in the create_matrix function?
Answer: ***What:*** *Check for NULL pointer returned from the malloc functions in case of unsuccessful at getting DMA.* ***When:*** *just before the for loop. Second within the for loop after each malloc call.*

(iii) The code compiles and runs, but gives memory leakage. What is the fault and how can it be corrected?
Answer: ***Fault:*** *The function destroy_matrix does not free up the individual arrays allocated dynamically inside the for loop.* ***Correction:*** *Use a for loop to first free all the individual array and then free up p.*

(iv) Write an updated version of create_matrix that creates matrixes for any values of m and n with just 2 calls to malloc function and despite that **p can be used as 2D array.
Answer:

```
int **create_matrix2(int m, int n){
      int **p, *t, i;
      p = (int**)malloc(sizeof(int*)*m);
      t = (int*)malloc(sizeof(int)*m*n);
      for (int i = 0; i < m; ++i)   {
            p[i] = &t[i*n];
      }
      return p;
}//end function create_matrix
```

CLO 4: *Design basic problems of the real world through small/medium size programs given as course projects.*

[Marks 10, 40 min]

Q4: You are given ***n jobs*** and ***m workers***. Each job has a difficulty and a profit, and each worker has an ability which represents the maximum difficulty level they can handle. The task is to assign jobs to workers in such a way that maximizes the total profit. Write a C program that performs the above-mentioned task and returns only the maximum total profit possible. Function prototype is given below with explanation of all the argument and return value: -.

Sample Output:

Input: difficulty = [2,4,6,8,10], profit = [10,20,30,40,50], worker = [4,5,6,7]
Output: 100

Input: difficulty = [85,47,57], profit = [24,66,99], worker = [40,25,25]
Output: 0

```
#include <stdio.h>

// Function to calculate maximum profit
int maxProfit(int difficulty[], int profit[], int n, int worker[], int m) {
    int totalProfit = 0;

    // Step 1: For each worker, assign the most profitable job they can handle
    for (int i = 0; i < m; i++) {
        int maxProfitForWorker = 0;  // Keep track of the highest profit for this worker
        for (int j = 0; j < n; j++) {
            // If the worker can do the job (worker's ability >= job difficulty)
            if (worker[i] >= difficulty[j]) {
                if (profit[j] > maxProfitForWorker) {
                    maxProfitForWorker = profit[j];  // Assign the best job (highest profit)
                }
            }
        }
        totalProfit += maxProfitForWorker;  // Add the maximum profit this worker can earn
    }

    return totalProfit;
}

int main() {
    // Example Input
    int difficulty[] = {2, 4, 6, 8, 10};
    int profit[] = {10, 20, 30, 40, 50};
    int worker[] = {6, 2, 6, 9};

    int n = sizeof(difficulty) / sizeof(difficulty[0]);  // Number of jobs
    int m = sizeof(worker) / sizeof(worker[0]);          // Number of workers

    int result = maxProfit(difficulty, profit, n, worker, m);
    printf("Maximum Profit: %d\n", result);

    return 0;
}
```

CLO 4: *Design basic problems of the real world through small/medium size programs given as course projects.*

[Marks 10, 2 each, 30 min]

Q5: Write a C program to work with a 3D coordinate system. The program should include the following:

(i) Define a struct named **Point3D** to represent a point in 3D space with three floating-point members: x, y, and z.

```
typedef struct {
    float x, y, z;
} Point3D;
```

(ii) Define a struct named **Cuboid** to represent a cuboid in 3D space. This structure should contain:

a. A Point3D for the back-bottom-left corner of the cuboid.
b. Three float values representing the length, width, and height of the cuboid.

```
typedef struct {
    Point3D corner; // Back-bottom-left corner of the cuboid
    float length, width, height; // Dimensions of the cuboid
} Cuboid;
```

Implement the following functions:

(iii) **float distanceBetweenPoints(Point3D p1, Point3D p2)**: Calculates the Euclidean distance between two 3D points.

```
#include <math.h> // For sqrt

float distanceBetweenPoints(Point3D p1, Point3D p2) {
    return sqrt((p2.x - p1.x) * (p2.x - p1.x) +
                (p2.y - p1.y) * (p2.y - p1.y) +
                (p2.z - p1.z) * (p2.z - p1.z));
}
```

(iv) **float cuboidVolume(Cuboid c):** Calculates the volume of the given cuboid.

```
float cuboidVolume(Cuboid c) {
    return c.length * c.width * c.height;
}
```

(v) **int isPointInsideCuboid(Point3D p, Cuboid c)**: Checks if a given point lies inside the cuboid (returns 1 if true, 0 otherwise).

```
int isPointInsideCuboid(Point3D p, Cuboid c) {
    float x_min = c.corner.x;
    float y_min = c.corner.y;
    float z_min = c.corner.z;

    float x_max = x_min + c.length;
    float y_max = y_min + c.width;
    float z_max = z_min + c.height;

    return (p.x >= x_min && p.x <= x_max &&
            p.y >= y_min && p.y <= y_max &&
            p.z >= z_min && p.z <= z_max);
}
```