Note to Students: This file contains sample solutions to the term test. Please read the solutions carefully. Make sure that you understand why the solutions given here are correct, that you understand the mistakes that you made (if any), and that you understand *why* your mistakes were mistakes. For all remarking requests, please refer to the portal for instructions.

Question 1. [20 marks]

Please provide short answers to each one of the following sub-questions.

Part (a) [2 marks]

Assume that the number of bits allocated in a computer for `short int` is 16 bits. The range of short integers that can be stored in this computer is therefore [-32768, 32767]. Will a problem occur if this code snippet is executed in this computer? If so, what is the problem called and why is it caused?

```
short int i=32768;
printf("i is %d\n", i);
```

Sample Solution:

This is an integer overflow problem. short int is only allocated 16 bits in the computer. The value 32768 requires more than 16 bits of storage space.

Part (b) [3 marks]

What will the output of the following code snippet be? Why? What is a safer alternate implementation?

```
float a=4.5, c;
int b=3, d;
c=a+b;
d=a+b;
printf("c is %f\n", c);
printf("c is %d\n", d);
```

Sample Solution:

The output is 7.500000 for the first print statement and 7 for the second print statement. This is due to implicit typecasting. A proper way to write this code will be to explicitly typecast (i.e. `c=a+(float)b;` and `d=(int)a+b;`)

Part (c) [2 marks]

Write the logical expression required to evaluate whether a `double` variable `i` falls *within* the range `n=1.2` and `m=3.4`. This range should include the values `n=1.2` and `m=3.4`. Also, write the expression that evaluates whether `i` does not fall in this range.

Sample Solution:

Within the range: `i>=1.2 && i<=3.4`. Complement: `i<1.2 || i>3.4`

Part (d) [2 marks]

Re-write the equivalent code snippet of the following `for` loop in terms of a `while` loop.

```
for(i=0; i<n; i++)
{
   printf("%d\n", i);
}
```

Sample Solution:

```
i=0;
while(i<n)
{
   printf("%d\n", i);
   i++;
}
```

Part (e) [2 marks]

What problem can occur in the following function (assume that `i` is a positive number)? How can this be fixed?

```
int func(int i)
{
   if(i<1)
    return 1;
   else
    func(i+1);
}
```

Sample Solution:

Infinite regression. The recursive calls do not result in moving towards the base case. This can be fixed by changing `i` to approach closer and closer to `1` at every recursive call.

Part (f) [3 marks]

What problem can occur in the following code snippet? What are the two ways this problem be avoided?

```
int * i = (int *)malloc(sizeof(int));
int j=2;
i=&j;
```

Sample Solution:

Inaccessible memory location. Ensure that another pointer holds the memory address of the dynamically allocated space or the dynamically allocated space must be freed.

Part (g) [2 marks]

What problem can occur in the following code snippet? How can this problem be avoided?

```
int * i = (int *)malloc(sizeof(int));
free(i);
*i=3;
```

SAMPLE SOLUTION:

Dangling pointer. The freed pointer should be set to `NULL`.

Part (h) [4 MARKS]

What is the issue in this function implementation? Use dynamic allocation and write a correct implementation of this function.

```
int * func(void)
{
    int i, *j;
    j=&i;
    *j=0;
    return j;
}
```

SAMPLE SOLUTION:

Returning the memory address of a local variable declared within a function is incorrect as that variable will cease to exist when the function ends.

```
int * func(void)
{
    int i, *j;
    j=(int*)malloc(sizeof(int));
    *j=0;
    return j;
}
```

Question 2. [20 MARKS]

In the following, you will be asked to provide a series of function implementations and/or diagrams.

Part (a) [5 MARKS]

Implement the function `void initArray(int ** a, int n)` which dynamically allocates an array of size `n` for storing `n` integer elements. Each element in the array should be set to 0. Double pointer `a`, passed as an argument to the function, should be set to point to the dynamically allocated array.

SAMPLE SOLUTION:

```
void initArray(int **a, int n)
{
    int i;
    int *array=(int*)malloc(n*sizeof(int));
    for (i=0; i<n; i++) {
        array[i]=0;
    }
    *a=array;
}
```

Part (b) [5 MARKS]

Complete the `main` function below to call the function you have defined above and print the contents of the array. Pass the single pointer `a` declared in the `main` function to the `initArray` function in a suitable form. Ensure that there will be no memory leaks in your implementation.

SAMPLE SOLUTION:

```
int main()
{
    int *a;
    int i;
    initArray(&a, 5);
    for (i=0; i<5; i++) {
        printf("%d\n", a[i]);
    }
    free(a);
}
```

Part (c) [6 MARKS]

Implement the function `int factorial(int n)` using ***recursion***. A factorial, $n!$, is the multiplication of a decreasing sequence of consecutive integers ranging from `n` to `1` (i.e. $n! = n * (n-1) * ... * 1$). For instance, $4! = 4 * 3 * 2 * 1$.

SAMPLE SOLUTION:

```
int factorial(int n)
{
    int res;
    if (n==1) {
        res=1;
    }
    else{
        res=factorial(n-1)*n;
    }
    return res;
}
```

Part (d) [4 MARKS]

Draw the call tree resulting from the recursive implementation of the `factorial` function for the specific function call `factorial(4)`.

SAMPLE SOLUTION:

```
                     factorial(4)
                     24
                       /
              factorial(3)*4
              6*4=24
                 /
       factorial(2)*3
       2*3=6
            /
factorial(1)*2
   1
```

Question 3. [10 marks]

You will implement the function `int iIsPalidrome(int * a, int n)` in an ***iterative*** manner (i.e. without using recursion). This function will check whether an array of numbers represents a palindrome or not. A palindrome is a sequence of symbols that are read the same way in the forward and backward directions. For instance, the number sequence 1-2-4-2-1 is a palindrome as when you read this sequence in the forward direction you will read it as 1-2-4-2-1 and when you read it backwards you will also read this as 1-2-4-2-1. Similarly, the sequence 1-2-2-1 is also a palindrome. Variables `a` and `n` are passed as arguments to the function. `a` is an array containing `n` elements. If the sequence of numbers in the array is a palindrome, then the function should return `1`. Otherwise, it should return `0`.

Sample Solution:

```
int iIsPalidrome(int * a, int n)
{
    int res=1;
    for (int i=0; i<n/2; i++) {
        if (a[i]!=a[n-1-i]) {
            res=0;
        }
    }
    return res;
}
```

Question 4. [15 MARKS]

Finding the minimum element in an array problem.

Part (a) [10 MARKS]

In this question, you will implement the ***recursive*** function `int rFindMin(int * a, int i, int n, int *index)`. This function will recursively find the minimum element in an array, `a`, containing unique integer elements and return this value. The index at which this minimum element resides must also be indirectly passed to the calling function via the pointer variable `index`. Variables `a`, `n` and `index` are passed as arguments to the function. `i` and `n` are integers whose values will be $0 \leq i \leq n \leq size$ where $size$ is the size of the array `a`. For instance, in the array $\{5, 4, 7, 1, 3\}$, the minimum element is 1 and this element resides at index 3. An initial recursive call for this example can be `rFindMin(a, 0, 4, index)` where `a` is the array containing the elements and `index` is a single integer pointer variable.

SAMPLE SOLUTION:

```
int rFindMin(int * a, int i, int n, int *index)
{
    int res;
    if (i==n){
        res=a[n];
        *index=n;
    }
    else{
        res=rFindMin(a, i+1, n, index);
        if(a[i]<res){
            res=a[i];
            *index=i;
        }
    }
    return res;
}
```

Part (b) [5 MARKS]

Please provide the call tree diagram for the specific instance `rFindMin(a, 0, 4, index)` where `a` is an array containing integer elements $\{5, 4, 7, 1, 3\}$ and `index` is a single pointer variable.

SAMPLE SOLUTION:

```
                                rFindMin(a,0,4,index): res=1, *index=3

                                  /
                         rFindMin(a,1,4,index): res=1, *index=3

                           /
                 rFindMin(a,2,4,index): res=1, *index=3

                     /
         rFindMin(a,3,4,index): res=1, *index=3

               /
  rFindMin(a,4,4,index): res=3, *index=4
```