Question 1
(15 marks, each part is 5 marks)

a) Using an example, explain the difference between initialization and declaration in c. You can use pseudocode for your explanation.

When we declare a variable: int a;

A space of the size of int i.e. 4 bytes is allocated in the memory. Since we have not assigned any value to it a default value i.e. 0 will be stored there. Hence declaration assigns memory space and default value.

When we initialize the variable i.e. int a = 12;

Here, memory is allocated, and initial non-zero value is assigned to the variable.

b) Consider the code snippet given below. The variable "sum" is of type int. Will this compile? Explain why or why not (3-4 sentences).

```c
int sum = 2 + "2";
```

Yes, this will compile. "2" is a string, which means it's a char array. Since arrays are addresses (from end of Lecture 4), the variable sum ends up being 2 + the address of the char array. This may result in overflow, since addresses may be longer than the number of bytes an int can hold.

c) Is it possible for a pointer to refer an address of another pointer? If yes, explain (3-4 sentences) with an example.

Yes, it's possible for a pointer to contain the address of the second pointer. This is a form of multiple indirection.

```c
#include <stdio.h>

void main () {
    int a = 54;
    int *b;
    int **c;
    b = &a; // pointer b is pointing to the address of a
    c = &b; // pointer c is a double pointer pointing the address of pointer b
    printf("value stored at b: %d\n", *b); // 54
    printf("value stored at c: %d\n", **c); // 54
}
```
Question 2
(3 marks for parts a and b, 4 marks for part c)

Based on your knowledge of pointers and memory addresses, answer the questions following the code snippet below:

```c
#include <stdio.h>

int hcf(int *x, int *y) {
    *x = *y;
    int a = *x;
    int b = *y;
    while (b != 0) {
        int tmp = b;
        b = a % b;
        a = tmp;
    }
    return a;
}

int main() {
    int a = 54;
    int b = 24;
    int result = hcf(&a, &b);
    return 0;
}
```

a) What will be the value of the result variable after calling hcf function?

24

b) What will be the value of the variables a and b just before the program terminates?

Value of a: 24
Value of b: 24

c) Briefly explain (in 1 or 2 sentences) your answers to part a and part b of this question.

We are passing by reference and re-assigning values in the function, hence a gets assigned the value of b in the first line of hcf. Note that b doesn’t change since the rest of the hcf function does not alter any variables outside of the function, except for the return value which is assigned to the variable result.
Question 3
(20 marks, each part is 5 marks)

For the following snippets, how many times is the `printf` statement executed? Briefly explain (up to 3 sentences).

a)
```c
int i, j;
int n = 100;
for (i = 1; i <= n; i++) {
    for (j = 3*i; j <= n; j++) {
        printf("programming is fun\n");
    }
}
```

The inner loop involving j is executed for j from 3i to 100, which is 100 – 3i + 1 times. This repeats for i from 1 to 33. So in total, printf is called 98 + 95 + 92 + ⋯ + 2 = 1650 times.

b)
```c
int i, j;
int n = 1000000;
for (i = 1; i <= n; i++) {
    for (j = 1; j <= 10000; j++) {
        printf("%d %d\n", i, j);
    }
}
```

The inner loop involving j is executed 10000 times, for j from 1 to 10000. This is repeated n = 1000000 times in the outer loop, so printf is called 1000000 × 10000 = 10^{10} times.

c)
```c
int i = 0;
int n = 10;
int j;

while (i < n) {
    i++;
    j = i;
    while (i < n) {
        printf("hello %d\n", i);
    }
}
```
\begin{verbatim}
int i = 0;
int n = 10;
int j;

while (i < n) {
    i++;
    j = i;
    while (i < n) {
        printf("hello %d\n", i);
        i++;
        break;
    }
    break;
}

Due to the break statement, the inner while loop, when entered, only calls printf once. From the outer loop, as before we have j going from 1 to 9, so printf gets called 9 times.
\end{verbatim}

Since j is equal to i before the inner loop, whenever the inner loop is executed, i goes from j to 9 for a total of \(9 - j + 1\) calls of printf. After every iteration of the outer loop, the starting point j is 1 greater than before, with the last iteration being when j is 9, so j goes from 1 to 9. This is a total of \(9 + 8 + 7 + \cdots + 1 = 45\) calls to printf.
Question 4
(10 marks)

Write a function that draws an isosceles triangle of given height using for/while loop.

```
#include <stdio.h>

int pyramid(int height)
{
    int spaces = height;
    int star;
    for (int i = 1; i <= height; i++)
    {
        star = 2*i-1;
        int j = 0;
        spaces = spaces - 1;
        while(j < spaces + star)
        {
            if(j < spaces)
            {
                printf(" ");
            }
            else{
                printf("*");
            }
            j++;
        }
        printf("\n");
    }
    int main()
    {
        pyramid(3);
    }
```