Question 1 [5 Marks] : True or False
Circle either True or False on this sheet for each of the following statements.

a) The signed integer `01101111000101000111000110001101O` is a negative number. `True` `False`

b) `#include <stdio.h>` is needed to allocate memory from the heap. `True` `False`

c) Every `char*` must end with a `'\0'` character. `True` `False`

d) `&x` gives the memory location of variable `x`. `True` `False`

e) Dangling pointers always point to freed memory on the heap. `True` `False`

Question 2 [16 Marks] : Short Answers
For each of the following questions provide a **short answer** in the booklet provided. Clearly indicate which question you are answering in the booklet (E.g., 2a) 2b), etc).

a) At least one of the True/False statements in Question 1 is False. Choose a True/False statement that is false and give a short explanation why it is false. Clearly indicate which True/False question you are referring to.

a) A negative signed integer has a 1 as the most significant bit. This integer does not.
b) stdlib.h provides the functions for memory allocation (like malloc and free)
c) Every string must end with '\0'. An array of chars need not.
e) A dangling pointer may point to the stack (or a portion of the stack that is no longer on the stack).

b) Briefly describe two things that are different between C and Java. (While it is true that their names are different, you will receive no marks for this difference.)

There are many to choose from.

o) Java is an object oriented language. Everything is an Object (except primitive data types). There are no Objects in C.

o) Java takes care of memory leaks for you. When allocating memory on the heap (with new in Java) it automatically (garbage collector) frees the memory once it is no longer being referenced by anything. In C, we have to do this manually with the free() function.

o) When you compile a C program, you get machine code specific to the machine you compiled on. In Java, when you compile, you get Java bytecode, which runs on any Java Virtual Machine (on any computer).

o) You work explicitly with pointers and addresses in C. You do not explicitly use these in Java.

o) Strings have their own class in Java; in C they are character arrays with a special terminating char.

o) etc...

c) Briefly describe one benefit and one potential danger of using a `union` in C.

Benefit: you can save memory.

Danger: you have to keep track of what type is being currently stored; C will let you interpret the bits as any of types in the union.

d) What does `10100011010011110011001100010101` mean? Why?

Without a bit model it doesn't mean anything at all.

(It could be anything at all; an int, an unsigned int, a float, garbage, a machine language instruction, a part of an mp3 or jpg, ...)

e) Briefly explain what memory leak is. Give two (different) examples in which a memory leak can occur.

A memory leak occurs whenever you have memory that was allocated, not freed, and not reachable from the program.

Ex1:
int* x = (int*) malloc(sizeof(int));
x = NULL;

Ex2:
creating memory in a function without freeing it (and without returning a pointer to it)

f) Briefly list one way in which you think this course course could be improved for the rest of the semester.

g) Describe the difference between `char s1[]="cat";` and
`char s2[]={'c','a','t'};`

s1 is a string. It contains 4 chars, the last being '\0' which is automatically added.
s2 is an array of chars. It contains 3 chars, as given in the initialization.

Question 3 [30 Marks] : C code and memory models
Use the following datatypes to answer the following questions. Assume that these are in a file called datatypes.h in the same directory as your code.

-----datatypes.h-----

```
typedef struct Person_t Person;
struct Person_t {
   char* name;
   int   age;
};
```

```
typedef struct Node_type Node;
struct Node_type {
     Person* p;
     Node*   next;
};
```

You may include <stdio.h>, <stdlib.h>, <string.h> and <stdbool.h> at your discretion.

a) Write a complete C program that does the following: reads in data for `num` people (Person type), where `num` is given as a command line argument, and stores the data.

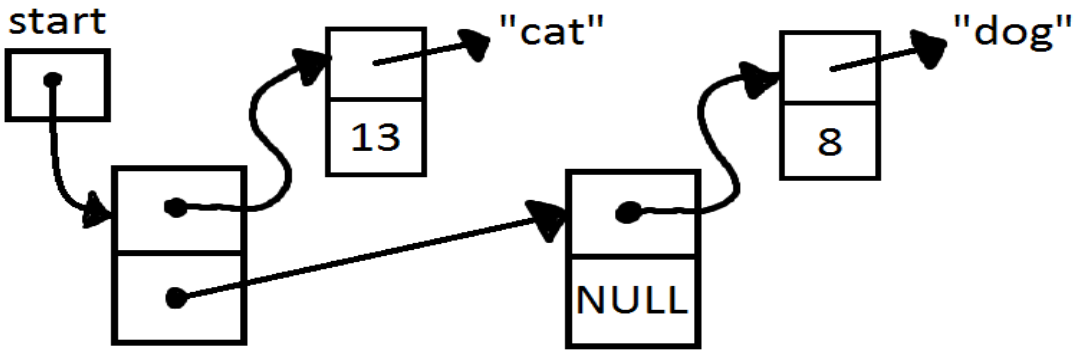
   Read the data from standard input using scanf(), where the data for a given person will consist of a string (for the name) followed by some whitespace and then an integer (for the age) followed by a newline. After reading all the people the program outputs, to standard output, the number of people having the same name as the last person read. There will be at least one person read.

   Note: There are several aspects to this problem. Break them down into a sequence of tasks (write out these tasks in your solution) and if you cannot solve one them just comment what you would do and continue working on the next task.

   Note: You do not need to use the Node type for this question (but you may). You can store your people any way you wish.

```
// one possible solution
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// don't bother retyping type definitions
#include "datatypes.h"

int main(int argc, char* argv[]){
    // get the command line argument that gives us num
    int num = atoi(argv[1]);

   // use an array to store the people
   Person people[num];

   // populate the array with the num people (read from std in)
   for( int i=0; i<num; i+=1){
      char name[1024];                // assumption here!
      int age;
      scanf("%s", name);
      people[i].name = (char*) malloc( strlen(name) );
      strcpy(people[i].name, name);
      scanf("%d", &age);
      people[i].age = age;
   }

   // count how many people have same name as last person
   int count = 1;

   for(int i=0; i<num-1; i+=1){
      if( strcmp(people[i].name, people[num-1].name) == 0 )
         count +=1;
   }

  // output number
   printf("%d\n", count);

  return 0;

}
```

b) Consider the following memory model:



You are given the variable `Node* start`. Give a sequence of statements that adds a new Node in between the two Nodes already there. The new Node's person will have name "eel" and age that is equal to the sum of the ages of the "cat" and "dog" nodes (don't just hardcode this age, you need to compute it).

```
Node* newNode = (Node*) malloc( sizeof(Node));
newNode->name = "eel";
newNode->age = start->p->age + start->next->p->age;
newNode->next = start->next;
start->next = newNode;
```

c) Free all the memory as it exists at the end of the previous question. (Assume malloc was used for all Persons, names and Nodes.)

```
Node* next = start->next;
Node* curr = start;
while( next != NULL){
   free(curr->p->name);
   free(curr->p);
   free(curr);
   curr = next;
   next = next->next;
}
```

Alternatively, we can explicitly free everything from start.

```
free(start->next->next->p->name);
free(start->next->next->p);
free(start->next->next);
free(start->next->p->name);
free(start->next->p);
free(start->next);
free(start->p->name);
free(start->p);
free(start);
```