1. Show the memory layout of the following C struct definition taking into consideration the SPARC data type memory alignment restrictions discussed in class. Fill bytes in memory with the appropriate struct member/field name. For example, if member/field name `p` takes 4 bytes, you will have 4 `p`'s in the appropriate memory locations. If the member/field is an array, use the name followed by the index number. For example, some number of `p[0]`s, `p[1]`s, `p[2]`s, etc. Place an `X` in any bytes of padding. Structs and unions are padded so the total size is evenly divisible by the most strict alignment requirement of its members. [14pts]

```c
struct snoop {
    short  a[3];
    double * b;
    double  c[2];
    char   d;
};

struct snoop dogg;
```

What is the `offsetof( struct snoop, d )`? ______

What is the `sizeof( struct snoop )`? ______

---

2. Given the following C union declaration and taking into consideration the SPARC data type memory alignment restrictions discussed in class, identify the size of the union when `TYPE` is replaced with each of the data types listed below: [6pts]

```c
union okgo {
    TYPE a;
    char b[10];
};
```

`TYPE` => char: `sizeof( union okgo )`? ______

`TYPE` => int: `sizeof( union okgo )`? ______

`TYPE` => double: `sizeof( union okgo )`? ______

(over)
3. Given the following C++ program and taking into consideration how struct constructors and destructors are called as discussed in class, determine the output generated by this program: [16pts]

```cpp
#include <iostream>

using namespace std;

int id = 0;

struct FOO {
    int id;
    FOO() {
        this->id = ::id++;
        cout << "FOO CTOR: " << this->id << endl;
    }
    ~FOO() {
        cout << "FOO DTOR: " << this->id << endl;
    }
};

struct BAR {
    int id;
    struct FOO f1;
    struct FOO f2;
    BAR() {
        this->id = ::id++;
        cout << "BAR CTOR: " << this->id << endl;
    }
    ~BAR() {
        cout << "BAR DTOR: " << this->id << endl;
    }
};

int main() {
    cout << "MAIN: 1" << endl;
    struct BAR myBar1;
    cout << "MAIN: 2" << endl;
    if ( true ) {
        cout << "ID: " << id << endl;
        struct BAR myBar2;
    }
    cout << "MAIN: 3" << endl;
    return 0;
}
```

4. Handling array comparison and assignment in C:

In C, one should call the library function __________________________ in order to compare if two arrays of arbitrary data type are equal or not. [2pts]

In C, one should call the library function __________________________ in order to assign the contents of one array of arbitrary data type into another array of that same data type and size, where the two arrays may or may not overlap memory addresses. [2pts]