Part VII: Structures

1.- Provide the definition for each of the following structures and unions:

a) Structure `inventory` containing character array `partName[ 30 ]`, integer `partNumber`, floating point `price`, integer `stock` and integer `reorder`.

b) Union `data` containing `char c`, `short s`, `long b`, `float f` and `double d`.

c) A structure called `address` that contains character arrays `streetAddress[ 25 ]`, `city[ 20 ]`, `state[ 3 ]` and `zipCode[ 6 ]`.

d) Structure `student` that contains arrays `firstName[ 15 ]` and `lastName[ 15 ]` and variable `homeAddress` of type `struct address` from part (c).

e) Structure `test` containing 16 bit fields with widths of 1 bit. The names of the bit fields are the letters `a` to `p`.

2.- Given the following structure and variable definitions,

```
struct customer
{
    char lastName[ 15 ];
    char firstName[ 15 ];
    int customerNumber;
    struct
    {
        char phoneNumber[ 11 ];
        char address[ 50 ];
        char city[ 15 ];
        char state[ 3 ];
        char zipCode[ 6 ];
    } personal;
} customerRecord, *customerPtr;

customerPtr = &customerRecord;
```

write an expression that can be used to access the structure members in each of the following parts:

- a) Member `lastName` of structure `customerRecord`.
- b) Member `lastName` of the structure pointed to by `customerPtr`.
- c) Member `firstName` of structure `customerRecord`.
- d) Member `firstName` of the structure pointed to by `customerPtr`.
- e) Member `customerNumber` of structure `customerRecord`.
- f) Member `customerNumber` of the structure pointed to by `customerPtr`.
- g) Member `phoneNumber` of member `personal` of structure `customerRecord`.
- h) Member `phoneNumber` of member `personal` of the structure pointed to by `customerPtr`.
- i) Member `address` of member `personal` of structure `customerRecord`.
- j) Member `address` of member `personal` of the structure pointed to by `customerPtr`.
- k) Member `city` of member `personal` of structure `customerRecord`.
- l) Member `city` of member `personal` of the structure pointed to by `customerPtr`.
- m) Member `state` of member `personal` of structure `customerRecord.`
- n) Member `state` of member `personal` of the structure pointed to by `customerPtr`.
- o) Member `zipCode` of member `personal` of structure `customerRecord`.
- p) Member `zipCode` of member `personal` of the structure pointed to by `customerPtr`.

Mandatory: (part of CasioCAN project)

3.- Un-Packing bytes to serialize messages

Made a function that takes an structure with the elements below and place them byte by byte into and array made of 7 bytes, information shall be place in the following order: (*Where word0 is the less significant byte of word element and word3 is the most significant byte, same with hword0 and hword1*).

byte0 | byte1 | byte2 | byte3 | byte4 | byte5 | byte6

word0 | word1 | word2 | word3 | byte | hword0 | hword1

```
struct StructPack
{
    unsigned long word;
    unsigned char byte;
    unsigned short hword;
};

void UnpackingBytes( struct StructPack *message, unsigned char *array );
```

Mandatory: (part of CasioCAN project)

4.- Packing bytes from a serial message

Made a function that takes an array made of 11 bytes and place that information into a defined structured, information shall be place in the following order: (*Where word0 is the less significant byte of word element and word3 is the most significant byte, same with hword0 and hword1*).

- <StructPack>.word1 = byte0, byte1, byte2, byte3
- <StructPack>.byte = byte4
- <StructPack>.hword = byte5, byte6
- <StructPack>.word2 = byte7, byte8, byte9, byte10

```
struct StructPack
{
    unsigned long word1;
    unsigned long word2;
    unsigned char byte;
    unsigned short hword1;
};

void PackingBytes( unsigned char *array, struct StructPack *message );
```

Mandatory: (part of CasioCAN project)

5.- Time to design a very nice algorithm, if you dare Circular Buffers