Question #2 - Store

Assignment

In this question we will write a program that can manage a convenience store. The program will support managing the store's products by,

1. Create product.
2. Adding products to the store.
3. Removing existing products.
4. Printing available products.
5. Printing expired products.
6. Printing a category of products.

A product will be described as follows,

```
typedef struct {
    char *name;
    char *category;
    char *barcode;
    int amount;
    double price;
    date_t *expire_date;
} product_t;
```

where date is described as follows,

```
typedef struct {
    int year;
    int month;
    int day;
} date_t;
```

We will represent our convenience store as,

```
typedef struct {
    product_t *products[MAX_PRODUCTS];
    int number_of_products;
} store_t;
```

> Define MAX_PRODUCTS to 25.

Functions

1. Create product

```
product_t *create_product(const char *name, const char *category, const char
*barcode, int amount, double price, const date_t *expire_date);
```

This function allocates and creates a product, on success returns a pointer to the new allocated product o.w returns NULL.

> Note you should deep copy the arguments given.

Validations

1. name, category and barcode are not empty.
2. amount is positive.
3. expire_date is not NULL.

> If memory allocations fails free all previously allocated memory and return NULL.

2. Add product

```
int add_product(store_t *store, product_t *product);
```

This function adds a product to the store if the barcode exists only add the amount to the existing product, on success returns 0 o.w -1.

Theoretical questions

Write in the functions documentation the answer to questions.

1. What changes in the function are to be made to prevent statically allocating MAX_PRODUCTS and make it dynamically.

Assumptions

1. store has enough space

Validations

1. store is not `NULL`.
2. product is not `NULL`.

Requirements

1. if product exists free the product passed to function.

3. Remove product

```
int remove_product(store_t *store, const char *barcode);
```

This function removes a product from the store, on success returns 0 o.w -1. The function should move the last valid product to the index of the removed product to maintain a sequential array.

Validations

1. store is not `NULL`.
2. barcode is not empty.
3. barcode exists in the store.

Requirements

1. free product.

4. Print products

```
void print_products(store_t *store);
```

This function prints all products in the store, if store is `NULL` do nothing.

5. Print expired products

```
void print_expired_products(store_t *store, date_t *now);
```

This function prints all expired products in the store (with reference to `now`) if the date is the same the product will not count as expired.

1. if store is `NULL` do nothing.
2. if date is `NULL` do nothing.

> The year in date is represented as YYYY.

6. Print a category of products

```
void print_category(store_t *store, const char *category);
```

This function prints all products belonging to the category.

1. if store is `NULL` do nothing.
2. if category is `NULL` print **all** products.

> Notice that `print_products` is a special case of `print_category` where category is NULL

Printing a product

You are **required** to use the following snippet to print a product,

```
void print_product(const product_t *product)
{
    printf("Product %s [%s]\n", product->name, product->barcode);
    printf("Category: %s\n", product->category);
    printf("Price:    %lf\n", product->price);
    printf("Amount:   %d\n", product->amount);
    printf("Expires:  %d/%d/%d\n\n", product->expire_date->day, product-
>expire_date->month, product->expire_date->year);
}
```

> Please just copy it exactly as it is to avoid mistakes.