Multi-Paradigm Programming

Shop Assignment

You are tasked to add some additional functionality to the base code in files :

References

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The shop CSV should hold the initial cash value for the shop.  See file called **stock.csv**   |  |  |  | | --- | --- | --- | | **1000.30** |  |  | | **Coke Can** | 1.10 | 100 | | **Bread** | 0.7 | 30 | | **Spaghetti** | 1.20 | 100 | | **Tomato Sauce** | 0.80 | 100 | | **Big Bags** | 2.50 | 4 |   Value is **1000.30** |
| Read in customer orders from a CSV file. |
| That file should include all the products they wish to buy and in what quantity.  It should also include their name and their budget.  See file called **customer.csv**   |  |  | | --- | --- | | **John** | **100.20** | | **Coke Can** | 10 | | **Bread** | 3 | | **Jam** | 1 |   Budget is **100.20** |
| The shop must be able to process the orders of the customer. Update the cash in the shop based on money received. |
| It is important that the state of the shop be consistent. |
| You should create customer test filles (CSVs) which cannot be completed by the shop e.g. customer wants 400 loaves of bread but the shop only has 20, or the customer wants 2 cans of coke but can only afford 1.  If these files don’t exist marks penalties will be applied. Know whether or not the shop can fill an order.  See files called  **customer\_too\_much\_bread.csv**   |  |  | | --- | --- | | **Mr Gobbler** | **500** | | **Bread** | 400 |   **customer\_not\_enough\_money.csv**   |  |  | | --- | --- | | **Mr Skint** | **1** | | **Coke Can** | 2 | |
| Thrown an appropriate error. |
| Operate in a live mode, where the user can enter a product by name, specify a quantity, and pay for it. The user should be able to buy many products in this way. |

BASE CODE C

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  struct Product {  char\* name;  double price;  };  struct ProductStock {  struct Product product;  int quantity;  };  struct Shop {  double cash;  struct ProductStock stock[20];  int index;  };  struct Customer {  char\* name;  double budget;  struct ProductStock shoppingList[10];  int index;  };  void printProduct(struct Product p)  {  printf("PRODUCT NAME: %s \nPRODUCT PRICE: %.2f\n", p.name, p.price);  printf("-------------\n");  }  void printCustomer(struct Customer c)  {  printf("CUSTOMER NAME: %s \nCUSTOMER BUDGET: %.2f\n", c.name, c.budget);  printf("-------------\n");  for(int i = 0; i < c.index; i++)  {  printProduct(c.shoppingList[i].product);  printf("%s ORDERS %d OF ABOVE PRODUCT\n", c.name, c.shoppingList[i].quantity);  double cost = c.shoppingList[i].quantity \* c.shoppingList[i].product.price;  printf("The cost to %s will be €%.2f\n", c.name, cost);  }  }  struct Shop createAndStockShop()  {  FILE \* fp;  char \* line = NULL;  size\_t len = 0;  ssize\_t read;  fp = fopen("../stock.csv", "r");  if (fp == NULL)  exit(EXIT\_FAILURE);  read = getline(&line, &len, fp);  float cash = atof(line);  // printf("cash in shop is %.2f\n", cash);    struct Shop shop = { cash };  while ((read = getline(&line, &len, fp)) != -1) {  // printf("Retrieved line of length %zu:\n", read);  // printf("%s IS A LINE", line);  char \*n = strtok(line, ",");  char \*p = strtok(NULL, ",");  char \*q = strtok(NULL, ",");  int quantity = atoi(q);  double price = atof(p);  char \*name = malloc(sizeof(char) \* 50);  strcpy(name, n);  struct Product product = { name, price };  struct ProductStock stockItem = { product, quantity };  shop.stock[shop.index++] = stockItem;  // printf("NAME OF PRODUCT %s PRICE %.2f QUANTITY %d\n", name, price, quantity);  }    return shop;  }  void printShop(struct Shop s)  {  printf("Shop has %.2f in cash\n", s.cash);  for (int i = 0; i < s.index; i++)  {  printProduct(s.stock[i].product);  printf("The shop has %d of the above\n", s.stock[i].quantity);  }  }  int main(void)  {  struct Shop shop = createAndStockShop();  printShop(shop);  return 0;  } |

Notes

The above described functionality should be completed in Python and C. This is to be done in a procedural programming style.

The live mode, and the input files, should have the exact same behaviour in ALL implementations.

For example I should be able to use the Python implementation in the same way as the C one i.e. same CSV files, and the same process when doing an order in live mode.

The user experience of each implementation should be identical.

Marking Scheme

Python Program (20%)

Good Procedural Programming (NOT OOP) (10%)

Level of functionality (5%)

Live Mode (5%)

C Program (30%)

Good Procedural Programming (10%)

Level of functionality (10%)

Live Mode (10%)