Un `Supermercato` rappresenta digitalmente un supermercato e ne implementa alcune funzionalità, sia lato proprietario che lato utente. In particolare, tiene traccia dei prodotti vendibili, della loro quantità esposta sugli scaffali e dei clienti presenti (massimo 5 contemporaneamente) con i loro carrelli. Ogni cliente è identificato da un valore intero positivo, ed ha un carrello che è identificato da un valore intero da 1 a 5. Implementare le seguenti operazioni che possono essere effettuate su un `Supermercato` (ogni situazione di errore deve lasciare la struttura dati inalterata):

**--- Metodi invocati nella PRIMA PARTE di main.cpp: ---**

- ✓ **`Supermercato s;`**

Costruttore di default che inizializza un `Supermercato s`. Inizialmente esso non contiene prodotti ed è privo di clienti. **Se necessario, implementare anche il relativo distruttore.**

- ✓ **`s.crea_prodotto(p, pu);`**

Metodo che crea un nuovo prodotto rappresentato dalla stringa `p` e avente prezzo per unità `pu` (positivo con virgola). Il prodotto non deve essere già stato creato in precedenza.

- ✓ **`s.esponi(p, q, pu);`**

Metodo che espone il prodotto `p` in quantità `q` (intero) sugli scaffali di `s`. Il prodotto deve essere stato creato in precedenza. Il valore `pu` rappresenta il prezzo per unità di prodotto e può essere assente. In caso non lo sia, la funzione: i) fallisce qualora `pu` sia negativo; ii) aggiorna il prezzo unitario qualora `pu` sia positivo; e iii) ignora l'aggiornamento qualora `pu` sia zero.

- ✓ **`cout << s;`**

Operatore di uscita per il tipo `Supermercato`. Esso deve mostrare un resoconto dei prodotti esistenti e dei clienti presenti con i loro carrelli. Un esempio di stampa a video è il seguente:

```
prodotti:
ananas 10 2.1
pere 8 2.3
sale 12 1.3

clienti:
[1-42951]: sale (1) |
[2-42930]: ananas (2) | pere (5) | sale (1) |
[5-42940]: ananas (2) |
```

La semantica della stampa è da intendersi come segue: i) sotto la scritta "`prodotti:`" compaiono i prodotti creati, con relativa quantità esposta (eventualmente zero) e prezzo per unità; ii) sotto la scritta "`clienti:`" vengono riportati gli identificativi dei carrelli attualmente in uso con il codice del relativo cliente (tra l'ultimo prodotto e la parola "clienti:" vi è una riga vuota); iii) a destra di ogni codice cliente è riportato il contenuto del relativo carrello elencando i prodotti (con la relativa quantità tra parentesi) e dividendoli mediante il simbolo "`|`"; iv) in caso non vi siano clienti presenti, stampare a video "`nessun cliente presente.`".

I prodotti esistenti e quelli nei carrelli devono essere mostrati in ordine lessicografico. I clienti devono essere mostrati in ordine di identificativo di carrello crescente.

**--- Metodi invocati nella SECONDA PARTE di main.cpp: ---**

- ✓ **`s += c;`**

Operatore di somma ed assegnamento che aggiunge il cliente identificato da `c` al `Supermercato s`. Perché l'inserimento vada a buon fine il cliente non deve essere già presente nel `Supermercato` e deve esserci un carrello libero da assegnargli. In caso ci sia più di un carrello a disposizione, gli verrà assegnato quello con identificativo minore.

- ✓ **`s.metti_nel_carrello(c, p, q);`**

Metodo che sposta il prodotto `p` in quantità `q` dagli scaffali al carrello del cliente `c`, se `p` esiste e `c` è presente all'interno del `Supermercato s`. Qualora la quantità superi quella esposta per tale prodotto, limitarsi all'aggiunta della quantità disponibile. L'aggiunta al carrello di prodotti in quantità nulla o negativa non è permessa.

- ✓ **`s.acquista(c);`**

Metodo che permette al cliente `c` di completare l'acquisto ed uscire dal supermercato, liberando il suo carrello. Il valore di ritorno è la spesa totale per i prodotti presenti nel carrello considerando i prezzi correnti. La spesa ammonta a -1.0 nel caso il cliente non sia presente.

Mediante il linguaggio C++, realizzare il tipo di dato astratto **`Supermercato`**, definito dalle precedenti specifiche. Non è permesso utilizzare funzionalità della libreria STL come il tipo `string`, il tipo `vector`, il tipo `list`, ecc. **Gestire le eventuali situazioni di errore**.

SUGGERIMENTO: L'implementazione di funzioni ausiliarie che eseguano la ricerca di un dato prodotto o di un dato cliente fornito in input può semplificare la programmazione.

**USCITA CHE DEVE PRODURRE IL PROGRAMMA**

```
--- PRIMA PARTE ---

Test del costruttore:
prodotti:

clienti:
nessun cliente presente.

Test della crea_prodotto:
prodotti:
ananas 0 2.1
pere 0 2.3
sale 0 1.3

clienti:
nessun cliente presente.

Test della esponi:
prodotti:
ananas 3 2.1
pere 10 2.3
sale 4 1.3

clienti:
nessun cliente presente.

Test dell'eventuale distruttore
```

```
--- SECONDA PARTE ---

Test dell'operatore +=:
prodotti:
ananas 3 2.1
pere 10 2.3
sale 4 1.3

clienti:
[1-42940]:
[2-42950]:
[3-42951]:

Test della metti_nel_carrello:
prodotti:
ananas 2 2.1
pere 8 2.3
sale 3 1.3

clienti:
[1-42940]:
[2-42950]: ananas (1) | pere (2) | sale (1) |
[3-42951]:

Altro test della esponi:
prodotti:
ananas 4 2.1
pere 9 2.3
sale 3 1.3

clienti:
[1-42940]:
[2-42950]: ananas (1) | pere (2) | sale (1) |
[3-42951]:

Altro test della metti_nel_carrello:
prodotti:
ananas 1 2.1
pere 9 2.3
sale 1 1.3

clienti:
[1-42940]: ananas (2) |
[2-42950]: ananas (2) | pere (2) | sale (2) |
[3-42951]: sale (1) |

Test della acquista:
Spesa sostenuta: 11.4

prodotti:
ananas 1 2.1
pere 9 2.3
sale 1 1.3

clienti:
[1-42940]: ananas (2) |
[3-42951]: sale (1) |

Altro test dell'eventuale distruttore
```

## Note per la consegna:

Affinché l'elaborato venga considerato valido, il programma **deve** produrre almeno la prima parte dell'output atteso. In questo caso, i docenti procederanno alla valutazione dell'elaborato **solo se** lo studente avrà completato l'autocorrezione del proprio elaborato.
In **tutti** gli altri casi (per esempio, il programma non compila, non collega, non esegue o la prima parte dell'output non coincide con quella attesa), l'elaborato è considerato **insufficiente** e, pertanto, **non verrà corretto**.