Un `Nastro` rappresenta un nastro trasportatore circolare di un ristorante, sul quale la cucina colloca dei piatti che i clienti a loro volta possono scegliere, rimuovere e consumare. Ad ogni tavolo del ristorante è associato uno ed un solo nastro, non condiviso con nessun altro tavolo. Questo permette di registrare il conto totale di ciascun tavolo direttamente sul `Nastro`. I piatti presenti sul `Nastro` sono caratterizzati dalla coppia (nome, prezzo), dove il nome è al massimo di 25 caratteri e il prezzo è un intero non negativo. Sullo stesso `Nastro` è possibile che ci siano piatti identici, ma non piatti con lo stesso nome e prezzo diverso.

Implementare le seguenti operazioni che possono essere effettuate su un `Nastro`.

--- PRIMA PARTE ---

✓ `Nastro n;`
Costruttore di default che inizializza un `Nastro n` vuoto e con conto pari a zero.

✓ `n.aggiungi(s,c);`
Operazione che permette alla cucina di aggiungere il piatto di nome `s` e prezzo `c` al `Nastro n`. Ogni piatto deve sempre rispettare la massima lunghezza prevista per i nomi e un prezzo non negativo. Qualora ciò non si verifichi, provvedere a sanitizzare con una stringa vuota e/o un prezzo pari a zero. **L'operazione lascia il conto invariato**.

✓ `n.consuma(s);`
Operazione che permette al cliente di consumare la pietanza di nome `s`. L'operazione fallisce se un piatto con tale nome non è presente sul `Nastro`, altrimenti il piatto (solo uno nel caso ce ne siano più di uno) viene tolto dal nastro e il suo prezzo aggiunto al conto. Deve essere permessa la concatenazione delle operazioni: `n.consuma(s1).consuma(s2);`

✓ `cout << n;`
Operatore di uscita per il tipo `Nastro`. I piatti vengono riportati **in ordine crescente di prezzo** e, in caso di parità, **in ordine lessicografico** ("AA" precede "BB" ma è successivo ad "A"). L'uscita ha la forma seguente:

```
Piatti sul nastro:
    Sashimi 5
    Imperia roll 7
    Philadelphia roll 7
     9
    Tori no karaaghe 12
Conto: 0
```

In questo esempio, il `Nastro` contiene 5 piatti ed il conto è ancora pari a zero (nessun piatto è stato ancora consumato). I piatti sono indentati con **4 spazi**. Si noti come il quarto piatto ha nome vuoto. Tra il nome ed il prezzo c'è un unico spazio bianco. NB: il piatto con nome vuoto (quello di prezzo 9, nell'esempio) deve poter essere rimosso mediante l'operazione `n.consuma("")`.

--- SECONDA PARTE ---

✓ `n += n1;`
Operatore di somma e assegnamento che aggiunge tutti i piatti presenti sul `Nastro n1` al `Nastro n`. Come detto in precedenza, su un `Nastro` non possono esserci piatti con lo stesso nome ma diverso prezzo. Quindi i piatti di `n1` che, se aggiunti ad `n`, violerebbero questo requisito non vengono aggiunti.

✓ `n | n1;`
Operatore di or bit a bit che ritorna un `Nastro` contenente i piatti presenti o sul `Nastro n` o sul `Nastro n1`. Come detto in precedenza, su un `Nastro` non possono esserci piatti con lo stesso nome ma diverso prezzo. Quindi se `n` e `n1` contengono piatti con lo stesso nome, solo quelli del `Nastro` in cui costano meno vanno a far parte del risultato. Il conto del `Nastro` risultato va impostato a zero.

✓ `~Nastro();`
Distruttore.

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

```cpp
// file main.cpp
#include <iostream>
#include "compito.h"
using namespace std;

int main(){

    cout<<"--- PRIMA PARTE ---"<<endl;
    cout<<"Test del costruttore"<<endl;
    Nastro n;
    cout<<n<<endl;

    cout<<"Test della aggiungi"<<endl;
    n.aggiungi("B", 5);
    cout<<n<<endl;

    n.aggiungi("C", 8);
    cout<<n<<endl;

    n.aggiungi("A", 6);
    cout<<n<<endl;

    cout<<"Test della consuma"<<endl;
    n.consuma("A");
    cout<<n<<endl;

    Nastro n1;
    n1.aggiungi("AA", 6);
    n1.aggiungi("D", 7);
    n1.aggiungi("C", 7); // piatto con lo stesso prezzo ma nome precedente a "D"
    n1.aggiungi("E", 7); // piatto con lo stesso prezzo ma nome successivo a "D"
    n1.aggiungi("B", 5);
    n1.aggiungi("B", 8); // piatto con stesso nome di "B" ma prezzo diverso
    n1.aggiungi("B", 5); // piatto identico a "B"
    cout<<n1<<endl;

    cout<<endl<<"--- SECONDA PARTE ---"<<endl;
    cout<<"Test dell'operatore +="<<endl;
    n += n1;
    cout<<n<<endl;

    cout<<"Test dell'operatore |"<<endl;
    cout<<( n | n1 )<<endl;

    {
        cout<<"Test del distruttore"<<endl;
        Nastro n2;
        n2.aggiungi("AA", 6);
        n2.aggiungi("D", 7);
        n2.aggiungi("B", 5);
        n2.aggiungi("C", 7);
        cout << n2 << endl;
    }
    cout<<"(l'oggetto n2 e' appena stato distrutto)"<<endl;

}
```

USCITA ATTESA

```
--- PRIMA PARTE ---
Test del costruttore
Piatti sul nastro:
Conto: 0

Test della aggiungi
Piatti sul nastro:
    B 5
Conto: 0

Piatti sul nastro:
    B 5
    C 8
Conto: 0

Piatti sul nastro:
    B 5
    A 6
    C 8
Conto: 0

Test della consuma
Piatti sul nastro:
    B 5
    C 8
Conto: 6

Piatti sul nastro:
    B 5
    B 5
    AA 6
    C 7
    D 7
    E 7
Conto: 0
```

```
--- SECONDA PARTE ---
Test dell'operatore +=
Piatti sul nastro:
    B 5
    B 5
    B 5
    AA 6
    D 7
    E 7
    C 8
Conto: 6

Test dell'operatore |
Piatti sul nastro:
    B 5
    B 5
    B 5
    B 5
    B 5
    AA 6
    AA 6
    C 7
    D 7
    D 7
    E 7
    E 7
Conto: 0

Test del distruttore
Piatti sul nastro:
    B 5
    AA 6
    C 7
    D 7
Conto: 0

(l'oggetto n2 e' appena stato distrutto)
```