Un sistema per la gestione degli ordini di una sagra può essere rappresentato come una coda attraverso le seguenti costanti e strutture:

```
const int MAXN = 10;
const int MAXL = 100;

enum Tipo {PRIMO, SECONDO, CONTORNO, DOLCE};

struct Ordine {
  char nome[MAXL];
  int tav;
  Tipo tipo;
  int quanti;
};

struct CodaOrdini {
  Ordine ord[MAXN];
  int front, back;
};
```

La struttura **CodaOrdini** rappresenta una coda di ordini ed è realizzata mediante un array di dimensione **MAXN** (quindi il numero massimo di ordini che possono essere memorizzate è pari a **MAXN−1**). **front** e **back** sono gli indici utilizzati rispettivamente per le operazioni di estrazione (dalla testa) e inserimento (in coda). Quando viene creata una istanza di **CodaOrdini** a **front** e a **back** viene assegnato il valore *0*. Ogni ordine è caratterizzato da: una stringa **nome** che indica il nome del piatto, un campo **tav** di tipo intero corrispondente al numero del tavolo, un intero **quanti** che indica il numero di porzioni da preparare, un campo **tipo** che indica se il piatto è un primo, un secondo e così via.

Scrivere il corpo delle seguenti funzioni C++.

1. **`void quantePorzTipo(const CodaOrdini* pc, int nn[])`** che riempie l'array **nn** con il numero complessivo di porzioni di tipo **PRIMO** (in **nn[0]**), di tipo **SECONDO** (in **nn[1]**), e così via.
2. **`bool salva(const CodaOrdini* pc, const char nf[])`** che salva i dati relativi agli ordini presenti in coda nel file con nome specificato da **nf**. Ogni ordine viene salvato su una riga separata, secondo il formato indicato dal seguente esempio:
   cinghiale in umido x 3, tav. 5
   crostata di mele x 2, tav. 2
   ...
   La funzione restituisce il valore *true* se l'operazione va a buon fine, *false* in caso di errore.
3. **`bool piuPorzioni(const CodaOrdini* pc, Ordine* po)`** che cerca all'intrerno della coda l'ordine in cui è necessario preparare più porzioni. Tale ordine viene restituito al chiamante riempiendo l'oggetto puntato dal puntatore **po**. La funzione restituisce il valore *false* se la coda è vuota, *true* altrimenti.
4. **`int ordinaPerTavolo(const CodaOrdini* pc, Ordine vv[])`** che riempie l'array **vv** con gli ordini presenti in coda ordinandoli per numero di tavolo crescente. La funzione restituisce il numero di ordini inseriti in **vv**.

5. **`bool pronto(CodaOrdini* pc, char nome[])`** che cerca, partendo dalla testa, un ordine avente nome **`nome`**. Nel caso in cui tale ordine venga trovato, la funzione deve decrementare di uno il suo campo **`quanti`**. Se quest'ultimo diviene pari a zero, la funzione deve anche rimuovere l'ordine dalla coda. Il valore booleano restituito dalla funzione indica se un ordine con nome **`nome`** è stato trovato (*true*) o meno (*false*).

NOTE SULLO SVOLGIMENTO DELLA PROVA:

- PER SVOLGERE L'ELABORATO, APRIRE il Dev-C++ (dal Menù **`Avvio`** (o **`Start`**) nella barra degli strumenti in fondo allo schermo, selezionare Programmi e quindi Dev-C++);
- PRIMA DI INIZIARE LO SVOLGIMENTO DELL'ELABORATO, selezionare la voce **`Identifica studente`** nel menù **`Strumenti`** all'interno dell'ambiente Dev-C++ e inserire i dati richiesti;
- per svolgere l'elaborato, aprire il progetto *esaInf.dev* presente nel directory *c:\esame\esaInf* e scrivere le funzioni richieste nel file *compito.cpp*, già presente nel progetto;
- per una corretta stampa dell'elaborato bisogna mantenere il codice entro i margini imposti dall'ambiente Dev-C++ (linea verticale presente alla destra della pagina);
- SE L'ELABORATO È STATO COMPILATO SENZA ERRORI, PRIMA DELLA CONSEGNA, selezionare la voce **`Consegna Compito`** nel menù **`Strumenti`** all'interno dell'ambiente Dev-C++ e premere il tasto INVIO fino a quando non sparisce la finestra che è stata attivata.