REPORT PROGETTO DI PROGRAMMAZIONE & STRUTTURE DATI

Classe: PEU-Z

Traccia: Gestione degli ordini in un ristorante
1° Anno – Corso di laurea in Informatica A.A. 2023/2024

A Cura di:

- **Antonio Tino** - Matricola: 0512121211
- **Matteo Trinchese** - Matricola: 0512119333
- **Domenico Ricciardelli** - Matricola: 0512119238

INDICE

1. Scelta degli ADT

I diversi tipi di ADT che sono stati utilizzati per rappresentare le fasi degli ordini.

ADT Ordine:

Ordine è un ADT presente nel file header ordine.h.
Questo ADT è fondamentale per immagazzinare i dettagli dell'ordine.

Ha diversi vantaggi:

- **Riutilizzo:** Consente di riutilizzare il codice relativo all'ordine in diverse parti del programma, evitando duplicazioni.
- **Modularità:** Permette di raggruppare le informazioni relative agli ordini in una singola struttura, facilitando la gestione del programma.

Un gestore di ordini di un ristorante ha bisogno di poter aggiungere ordini, mandarli in elaborazione e consegnarli, motivo per cui L'ADT ordine risulta la scelta ideale per questo programma.

ADT Queue:

L'ADT queue è nato dal bisogno di avere una struttura dati che utilizza lo scheduling FIFO (First in First out).

L'ADT coda viene utilizzato due volte in istanze diverse:

- **Coda di attesa:** Contiene gli ordini in attesa prima che vengano mandati in elaborazione.
- **Coda consegnati:** Contiene gli ordini che sono stati marcati come consegnati.

ADT PQueue:

L'ADT PQueue è stato scelto per soddisfare il bisogno di una coda ordinata in base a una determinata variabile, il tempo di preparazione di ciascun ordine.

2. Progettazione

Obbiettivo del programma: Creare un programma in C che simuli il flusso di operazioni in un ristorante. Il sistema dovrebbe permettere l'aggiunta di ordini, la loro elaborazione in cucina e la marcatura degli ordini consegnati.

- **Contesto:** Il programma è stato concepito per facilitare la gestione degli ordini.

Compilazione:

```
test: ordine.o queue.o PQueue.o utili.o test.o
	gcc ordine.o queue.o PQueue.o utili.o test.o -o test


main: ordine.o queue.o PQueue.o utili.o main.o
	gcc ordine.o queue.o PQueue.o utili.o main.o -o main


ordine.o: ordine.c
	gcc -c ordine.c


queue.o: ordine.h queue.c
	gcc -c queue.c


PQueue.o: ordine.h PQueue.c
	gcc -c PQueue.c


utili.o: utili.c
	gcc -c utili.c


test.o: queue.h PQueue.h utili.h test.c
	gcc -c test.c


main.o: queue.h PQueue.h utili.h main.c
	gcc -c main.c


clean:
	rm -f *.o
```

- Per compilare digitare il comando:

```
make main
```

- Per eseguire il main digitare il comando:

```
./main
```

- Per eseguire il file test per eseguire i test case digitare il comando

```
./test Testsuite.txt
```

Struttura del programma:

- **Main:** Il file principale main.c è il punto d'ingresso del programma, include i file header necessari e chiama le diverse funzioni necessarie definite nei file di implementazione.
- **Header file:** Nel programma vengono utilizzati diversi file header dove vengono dichiarate le funzioni utili al corretto funzionamento del programma:
    - **Ordine.h:** Questo file serve a definire la struttura ordine che è composta dai seguenti campi:
        - **ID** (di tipo int, indica il nominativo dell'ordine)
        - **piatti** (di tipo int*, contiene il valore associato ai piatti presenti nell'ordine)
        - **descrizione** (di tipo char*, contiene la descrizione dell'ordine)
        - **t_preparazione** (di tipo int, indica il tempo di preparazione dell'ordine)

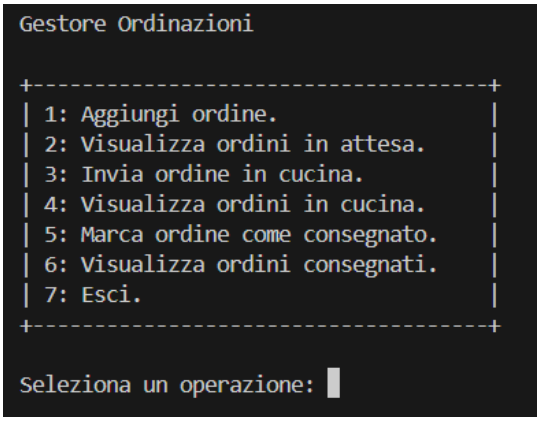
      Sono inoltre presenti funzioni in grado di creare, modificare e gestire l'ordine.
    - **Queue.h:** Questo file serve a definire la struttura dati queue che è composta da una struttura nodo con i seguenti campi:
        - **ord** (di tipo ordine, contiene un ordine)
        - **prossimo** (di tipo struct nodo*, contiene il puntatore al prossimo nodo della coda)

      E una struttura c_queue composta da:
        - **testa** (di tipo struct nodo*, contiene la testa della coda)
        - **coda** (di tipo struct nodo*, contiene l'ultimo elemento della coda)
        - **num_el** (di tipo int, contiene il numero di elementi della coda)

      Sono inoltre presenti funzioni in grado di creare, modificare e gestire la coda.
    - **PQueue.h:** Questo file serve a definire la struttura dati PQueue che è composta dalla struttura c_PQ contenenti i seguenti campi:
        - **ord** (di tipo ordine, è un array che contiene tutti gli ordini presenti nella coda di priorità)
        - **num_el** (di tipo int, contiene il numero di elementi della p_queue)

      Sono inoltre presenti funzioni in grado di creare, modificare e gestire la coda di priorità.
    - **utili.h:** Questo file contiene funzioni utili all'esecuzione del programma come la lettura delle righe di un file, la lettura delle descrizioni e una funzione di attesa che chiede all'utente di premere invio.
- **File di implementazione:** Le funzioni dichiarate nei file header vengono implementate nei corrispettivi file .c.

Comandi e menù:



menù dei comandi

- **1) Aggiungi Ordine:** Stampa il menù del ristorante tramite la funzione “stampa_menu” che legge dal file “MENU.txt”, in seguito chiede all’utente di selezionare i valori dei piatti corrispondenti alle pietanze che si vogliono ordinare e di aggiungere una descrizione facoltativa usando la funzione “crea_ordine”. Se creato correttamente, all’ordine verrà associato un tempo di preparazione attraverso la funzione “calcola_tempo_di_preparazione” che legge dal file “PREPARAZIONE.txt”. L’ordine sarà poi spostato nella coda degli ordini in attesa usando la funzione “aggiungi_in_queue”.

- **2) Visualizza ordini in attesa:** Controlla se sono presenti ordini in attesa tramite la funzione “queue_vuota” e in tal caso li stampa a schermo tramite la funzione “stampa_queue”.

- **3) Invia ordine in cucina:** Controlla se sono presenti ordini in attesa tramite la funzione “queue_vuota” e se la coda degli ordini in elaborazione non è piena attraverso la funzione “PQ_piena”, in tal caso ottiene l’ordine in testa alla coda di attesa tramite la funzione “ottieni_testa_queue”, lo inserisce nella coda di elaborazione attraverso la funzione “aggiungi_in_PQ” e in fine lo rimuove dalla coda di attesa tramite la funzione “rimuovi_testa_queue”

- **4) Visualizza ordini in cucina:** Controlla se sono presenti ordini in elaborazione tramite la funzione “PQ_vuota” e in tal caso li stampa a schermo tramite la funzione “stampa_PQ”.

- **5) Marca ordine come consegnato:** Controlla se sono presenti ordini in elaborazione tramite la funzione “PQ_vuota”, in tal caso ottiene l’ordine in testa alla coda di elaborazione tramite la funzione “ottieni_testa_PQ”, lo inserisce negli

ordini marcati come consegnati attraverso la funzione “aggiungi_in_queue” e in fine lo rimuove dalla coda di elaborazione tramite la funzione “rimuovi_testa_PQ”

- **6) Visualizza ordini consegnati:** Controlla se sono presenti ordini consegnati tramite la funzione “queue_vuota”, in tal caso li stampa a schermo tramite la funzione “stampa_queue”.

- **7) Esci:** Controlla la corretta deallocazione delle varie strutture dati usate attraverso le due funzioni “dealloca_queue” e “dealloca_PQ”, chiude i due file di testo aperti a inizio esecuzione del programma e infine ritorna il controllo al sistema operativo.

3. Specifica Sintattica e Semantica

- **Ordine.h:**
  - crea_ordine:
    - **Specifica sintattica:** crea_ordine (FILE *, FILE *, int) → ordine
      Tipi utilizzati: puntatore a file, ordine e intero.
    - **Specifica semantica:**
      - **Funzione:**
        crea_ordine (menu, tempo_di_preparazione, ID) → ordine.
      - **Descrizione:**
        Crea un nuovo ordine.
      - **Pre-condizioni:** I file devono essere stati correttamente aperti e il valore ID deve essere un intero.
      - **Post-condizioni:** La funzione restituisce il nuovo ordine creato, in caso in cui non vengono inseriti piatti la funzione restituisce NULL.
  - leggi_piatti:
    - **Specifica sintattica:** leggi_piatti(FILE *) → int *
      Tipi utilizzati: puntatore a file, puntatore ad intero e intero.
    - **Specifica semantica:**
      - **Funzione:** leggi_piatti(menu) → piatti.
      - **Descrizione:** Leggi il numero dei piatti inseriti dall'utente e li inserisce nel vettore piatti.
      - **Pre-condizioni:** Il file deve esser stato correttamente aperto.
      - **Post-condizioni:** La funzione restituisce il vettore piatti che contiene i numeri inseriti dall'utente, in caso in cui l'utente non inserisce un valore valido esso non viene inserito nel vettore piatti.
  - calcola_tempo_di_preparazione:
    - **Specifica sintattica:**
      calcola_tempo_di_preparazione(FILE *, int *) → int
      Tipi utilizzati: Puntatore a file, puntatore ad intero e char.
    - **Specifica semantica:**
      - **Funzione:**
        calcola_tempo_di_preparazione(tempo_di_preparazione, piatti) → t_preparazione.
      - **Descrizione:** Calcola il tempo stimato di preparazione in base ai patti inseriti nell'ordine
      - **Pre-condizioni:** Il file deve esser stato correttamente aperto
      - **Post-condizioni:** La funzione restituisce t_preparazione che contiene la somma dei tempi di preparazione di ciascun piatto.

- stampa_ordine:
  - **Specifica sintattica:** stampa_ordine(FILE *, ordine) → void
    Tipi utilizzati: Puntatore a file e ordine.
  - **Specifica semantica:**
    - **Funzione:** stampa_ordine(menu, ord) → void
    - **Descrizione:** Stampa a schermo le informazioni dell'ordine.
    - **Pre-condizioni:** Il file deve esser stato correttamente aperto e ord deve puntare a un ordine valido.
    - **Post-condizioni:** La funzione non restituisce valori ma stampa a schermo le informazioni dell'ordine.
- stampa_nome_piatti:
  - **Specifica sintattica:** stampa_nome_piatti(FILE *, int *) → void
    Tipi utilizzati: Puntatore a file, puntatore ad intero e intero
  - **Specifica semantica:**
    - **Funzione:** stampa_nome_piatti(menu, piatti) → void
    - **Descrizione:** Stampa a schermo il nome del piatto associato al numero inserito nel vettore piatti.
    - **Pre-condizioni:** Il file deve esser stato correttamente aperto.
    - **Post-condizioni:** La funzione non restituisce valori ma stampa a schermo il nome del piatto associato al valore inserito dall'utente nel vettore piatti.
- dealloca_ordine:
  - **Specifica sintattica:** dealloca_ordine(ordine) → void
    Tipi utilizzati: ordine
  - **Specifica semantica:**
    - **Funzione:** stampa_ordine(ord) → void
    - **Descrizione:** dealloca le informazioni contenute nell'ordine.
    - **Pre-condizioni:** ora deve puntare ad un ordine valido.
    - **Post-condizioni:** la funzione non restituisce valori ma dealloca le informazioni contenute nell'ordine.
- ottieni_tempo_di_preparazione:
  - **Specifica sintattica:**
    ottieni_tempo_di_preparazione(ordine) → int
    Tipi utilizzati: ordine
  - **Specifica semantica:**
    - **Funzione:** ottieni_tempo_di_preparazione(ord) → int
    - **Descrizione:** Restituisce il tempo di preparaizione.
    - **Pre-condizioni:** ord deve puntare ad un ordine valido.
    - **Post-condizioni:** La funzione restituisce il tempo di preparazione dell'ordine passato.

- ottieni_ID:
  - **Specifica sintattica:** ottieni_ID(ordine) → int
    Tipi utilizzati: ordine
  - **Specifica semantica:**
    - **Funzione:** ottieni_ID (ord) → int
    - **Descrizione:** Restituisce l'ID dell'ordine.
    - **Pre-condizioni:** ord deve puntare ad un ordine valido.
    - **Post-condizioni:** La funzione restituisce l'ID dell'ordine corrente
- leggi_ordine_da_file:
  - **Specifica sintattica:**
    leggi_ordine_da_file(FILE *, FILE *, FILE *, int) → ordine
    Tipi utilizzati: puntatore a file, interi ed ordine
  - **Specifica semantica:**
    - **Funzione:** leggi_ordine_da_file(menu, tempo_di_preparazione, input, ID) → ordine
    - **Descrizione:** Legge un ordine da un file di input e lo crea.
    - **Pre-condizioni:** I file devono essere stati correttamente aperti e il valore ID deve essere un intero.
    - **Post-condizioni:** La funzione restituisce il nuovo ordine creato leggendo le informazioni dal file di input, in caso in cui non vengono inseriti piatti la funzione restituisce NULL.

- leggi_piatti_da_file:
  - **Specifica sintattica:** leggi_piatti_da_file(FILE *, FILE *) → int *
    Tipi utilizzati: puntatore a file, puntatore a intero, intero e char
  - **Specifica semantica:**
    - **Funzione:** leggi_piatti_da_file(menu, input) → piatti
    - **Descrizione:** Legge i numeri dei piatti da un file di input e li inserisce nel vettore piatti.
    - **Pre-condizioni:** I file devono essere stati correttamente aperti.
    - **Post-condizioni:** La funzione restituisce il vettore piatti che contiene i numeri letti dal file di input, in caso in cui l'utente non inserisce un valore valido esso non viene inserito nel vettore piatti.

- stampa_ordine_file:
  - **Specifica sintattica:** stampa_ordine_file(FILE *, FILE *, ordine) → void
    Tipi utilizzati: puntatore a file e ordine
  - **Specifica semantica:**
    - **Funzione:** stampa_ordine_file(menu, output, ord) → void
    - **Descrizione:** Stampa su un file le informazioni dell'ordine.
    - **Pre-condizioni:** I file devono essere stati correttamente aperti e ord deve puntare a un ordine valido.
    - **Post-condizioni:** La funzione non restituisce valori ma stampa su un file le informazioni dell'ordine.
- stampa_nome_piatti_file:
  - **Specifica sintattica:** stampa_nome_piatti_file(FILE *, FILE *, int *) → void
    Tipi utilizzati: puntatore a file, puntatore ad intero, intero e char
  - **Specifica semantica:**
    - **Funzione:** stampa_nome_piatti_file(menu, output, piatti) → void
    - **Descrizione:** Stampa su un file il nome del piatto associato al numero inserito nel vettore piatti.
    - **Pre-condizioni:** I file devono essere stati correttamente aperti.
    - **Post-condizioni:** La funzione non restituisce valori ma stampa su un file il nome del piatto associato al valore inserito dall'utente nel vettore piatti.

- **queue.h:**
  - crea_queue:
    - **Specifica sintattica:** crea_queue() → queue
      Tipi utilizzati: queue
    - **Specifica semantica:**
      - **Funzione:** crea_queue() → queue
      - **Descrizione:** Alloca lo spazio necessario per la queue.
      - **Pre-condizioni:** Nessuna.
      - **Post-condizioni:** La funzione restituisce la nuova coda creata, in caso in cui non venga allocata allora restituirà NULL.

- queue_vuota:
  - **Specifica sintattica:** queue_vuota(queue) → int
    Tipi utilizzati: queue
  - **Specifica semantica:**
    - **Funzione:** queue_vuota(q) → int
    - **Descrizione:** Controlla se la queue è vuota.
    - **Pre-condizioni:** La queue deve essere correttamente creata ed allocata.
    - **Post-condizioni:** La funzione restituisce un intero, -1 se la queue è NULL,1 se la queue non ha elementi ed 0 se ha degli elementi.
- aggiungi_in_queue:
  - **Specifica sintattica:** aggiungi_in_queue(queue, ordine) → int
    Tipi utilizzati: queue e ordine
  - **Specifica semantica:**
    - **Funzione:** aggiungi_in_queue(q, ord) → int
    - **Descrizione:** Aggiunge un elemento in coda alla queue.
    - **Pre-condizioni:** La queue deve essere correttamente creata ed allocata, l'ordine deve essere allocato correttamente e deve contenere degli elementi.
    - **Post-condizioni:** La funzione restituisce un intero, 1 se ha inserimento correttamente l'ordine nella queue 0 invece se la queue è NULL.
- ottieni_testa_queue:
  - **Specifica sintattica:** ottieni_testa_queue(queue) → ordine
    Tipi utilizzati: queue ed ordine
  - **Specifica semantica:**
    - **Funzione:** ottieni_testa_queue(q) → ordine
    - **Descrizione:** Legge l'elemento in testa alla queue.
    - **Pre-condizioni:** La queue deve essere correttamente creata ed allocata.
    - **Post-condizioni**: La funzione restituisce l'ordine in testa alla queue se ha egli elementi, restituisce NULL se la queue non ha elementi e se la queueu è NULL.

- rimuovi_testa_queue:
  - **Specifica sintattica:** rimuovi_testa_queue(queue) → int
    Tipi utilizzati: queue e struct nodo
  - **Specifica semantica:**
    - **Funzione:** rimuovi_testa_queue(q) → int
    - **Descrizione:** Rimuove l'elemento in testa alla queue.
    - **Pre-condizioni:** La queue deve essere correttamente creata ed allocata.
    - **Post-condizioni:** La funzione restituisce un intero, 1 se la rimozione avviene con successo 0 se la queue è NULL o ha zero elementi.
- stampa_queue:
  - **Specifica sintattica:** stampa_queue(FILE *, queue) → void
    Tipi utilizzati: puntatore ad un file, queue e struct nodo
  - **Specifica semantica:**
    - **Funzione:** stampa_queue(menu, q) → void
    - **Descrizione:** Stampa a schermo le informazioni contenute nella queue.
    - **Pre-condizioni:** La queue deve essere correttamente creata ed allocata ed il puntatore al file deve essere aperto correttamente.
    - **Post-condizioni:** La funzione stampa a video tutta la queue contenente gli ordini in attesa.
- dealloca_queue:
  - **Specifica sintattica:** dealloca_queue(queue) → int
    Tipi utilizzati: queue ed ordine
  - **Specifica semantica:**
    - **Funzione**: dealloca_queue(q) → int
    - **Descrizione:** Dealloca lo spazio occupato dalla queue.
    - **Pre-condizioni:** La queue deve essere correttamente creata ed allocata.
    - **Post-condizioni:** La funzione restituisce un intero, 0 se la queue è NULL 1 se ha deallocato correttamente la memoria.

- stampa_queue_file:
  - **Specifica sintattica:** stampa_queue_file(FILE *, FILE *, queue) → void
    Tipi utilizzati: puntatore a FILE, struct nodo e queue
  - **Specifica semantica:**
    - **Funzione:** stampa_queue_file(menu, output, q) → void
    - **Descrizione:** Stampa le informazioni contenute nella queue su file.
    - **Pre-condizioni:** La queue deve essere correttamente creata ed allocata, il puntatore al file di input e output devono essere aperti correttamente.
    - **Post-condizioni:** Stampa su file tutte le informazioni degli ordini presenti nella queue.

- **PQueue.h:**
  - crea_PQ:
    - **Specifica sintattica:** crea_PQ() → Pqueue
      Tipi utilizzati: PQueue
    - **Specifica semantica:**
      - **Funzione:** crea_PQ() → Pqueue
      - **Descrizione:** Alloca lo spazio necessario per la queue a priorità.
      - **Pre-condizioni:** Nessuna.
      - **Post-condizioni:** La funzione restituisce la nuova queue creata, in caso in cui non venga allocata allora restituirà NULL.
  - PQ_vuota:
    - **Specifica sintattica:** PQueue_vuota(PQueue) → int
      Tipi utilizzati: PQueue
    - **Specifica semantica:**
      - **Funzione:** PQueue_vuota(q) → int
      - **Descrizione:** Controlla se la queue a priorità è vuota
      - **Pre-condizioni:** La PQueue deve essere correttamente creata ed allocata.
      - **Post-condizioni:** La funzione restituisce un intero, -1 se la queue a priorità è NULL, 1 se la queue a priorità non ha elementi e 0 se ha degli elementi.

- ottieni_testa_PQ:
  - **Specifica sintattica:** ottieni_testa_PQ(PQueue) → ordine
    Tipi utilizzati: PQueue
  - **Specifica semantica:**
    - **Funzione:** ottieni_testa_PQ(q) → ordine
    - **Descrizione:** Legge l'elemento in testa alla queue a priorità.
    - **Pre-condizioni:** La queue a priorità deve essere correttamente creata ed allocata.
    - **Post-condizioni:** La funzione restituisce l'ordine in testa alla queue a priorità se ha degli elementi, restituisce NULL se la queue a priorità non ha elementi o se è NULL.

- rimuovi_testa_PQ
  - **Specifica sintattica:** rimuovi_testa_PQ(PQueue) → int
    Tipi utilizzati: PQueue
  - **Specifica semantica:**
    - **Funzione:** rimuovi_testa_PQ(q) → int
    - **Descrizione:** Rimuove l'elemento in testa alla queue a priorità.
    - **Pre-condizioni:** La queue a priorità deve essere correttamente creata ed allocata.
    - **Post-condizioni:** La funzione restituisce un intero, 1 se la rimozione avviene con successo 0 se la queue a priorità è NULL o ha zero elementi.
- scorri_PQ_verso_il_basso:
  - **Specifica sintattica:** scorri_PQ_verso_il_basso(PQueue) → void
    Tipi utilizzati: Pqueue e int
  - **Specifica semantica:**
    - **Funzione:** scorri_PQ_verso_il_basso(q) → void
    - **Descrizione:** Scorre dall'alto verso il basso la queue a priorità.
    - **Pre-condizioni:** La queue a priorità deve essere correttamente creata ed allocata.
    - **Post-condizioni:** La funzione scorre dall'alto verso il basso la queue a priorità andando poi a scambiare gli indirizzi di memoria all'interno della queue stessa.

- aggiungi_in_PQ:
  - **Specifica sintattica:** aggiungi_in_PQ(PQueue, ordine) → int
    Tipi utilizzati: PQueue, ordine
  - **Specifica semantica:**
    - **Funzione:** aggiungi_in_PQ(q, ord) → int
    - **Descrizione:** Aggiunge un elemento in coda alla queue a priorità.
    - **Pre-condizioni:** La queue a priorità deve essere correttamente creata ed allocata, l'ordine deve essere allocato correttamente e deve contenere degli elementi.
    - **Post-condizioni:** La funzione restituisce un intero, 1 se ha inserimento correttamente l'ordine nella queue apriorità 0 invece se la queue a priorità è NULL oppure è piena.
- scorri_PQ_verso_alto:
  - **Specifica sintattica:** scorri_PQ_verso_alto(PQueue) → void
    Tipi utilizzati: Pqueue e int
  - **Specifica semantica:**
    - **Funzione:** scorri_PQ_verso_alto(q) → void
    - **Descrizione:** Scorre dal basso verso l'alto la queue a priorità.
    - **Pre-condizioni:** La queue a priorità deve essere correttamente creata ed allocata.
    - **Post-condizioni:** La funzione scorre dal basso verso l'alto la queue a priorità andando poi a scambiare gli indirizzi di memoria all'interno della queue stessa.
- stampa_PQ:
  - **Specifica sintattica:** stampa_PQ(FILE *, PQueue) → void
    Tipi utilizzati: puntatore ad un file, ordine, intero e PQueue
  - **Specifica semantica:**
    - **Funzione:** stampa_PQ(menu, q) → void
    - **Descrizione:** Stampa a schermo le informazioni contenute nella queue a priorità.
    - **Pre-condizioni:** La queue a priorità deve essere correttamente creata ed allocata ed il puntatore al file deve essere aperto correttamente.
    - **Post-condizioni:** La funzione stampa a video tutta la queue a priorità contenente gli ordini in attesa.

- dealloca_PQ:
  - **Specifica sintattica:** dealloca_PQ(queue) → int
    Tipi utilizzati: PQueue ed ordine
  - **Specifica semantica:**
    - **Funzione:** dealloca_PQ(q) → int
    - **Descrizione:** Dealloca lo spazio occupato dalla queue a priorità.
    - **Pre-condizioni:** La queue a priorità deve essere correttamente creata ed allocata.
    - **Post-condizioni:** La funzione restituisce un intero, 0 se la queue a priorità è NULL invece 1 se ha deallocato correttamente la memoria.
- PQ_piena:
  - **Specifica sintattica:** PQ_piena(PQueue) → int
    Tipi utilizzati: PQueue
  - **Specifica semantica:**
    - **Funzione:** PQ_piena(q) → int
    - **Descrizione:** Controlla se la queue a priorità è piena.
    - **Pre-condizioni:** La PQueue deve essere correttamente creata ed allocata.
    - **Post-condizioni:** La funzione restituisce un intero, -1 se la queue a priorità è NULL, 1 se la queue a priorità ha raggiunto la sua capacità massima e 0 se ha ancora spazio per altri elementi.
- stampa_PQ_file
  - **Specifica sintattica:** stampa_PQ_file(FILE *, FILE *, PQueue) → void
    Tipi utilizzati: puntatori a file, intero, ordine e PQueue
  - **Specifica semantica:**
    - **Funzione:** stampa_PQ_file(menu, output, q) -> void
    - **Descrizione:** Stampa a schermo e in un file le informazioni contenute nella queue a priorità.
    - **Pre-condizioni:** La queue a priorità deve essere correttamente creata ed allocata ed i puntatori ai file devono essere aperti correttamente.
    - **Post-condizioni:** La funzione stampa a video e scrive nel file output tutta la queue a priorità contenente gli ordini in attesa.

- **Utili.h:**
  - leggi_righe_file:
    - **Specifica sintattica:** leggi_righe_file(FILE *) → int
      Tipi utilizzati: puntatore a file, int e char
    - **Specifica semantica:**
      - **Funzione:** leggi_righe_file(file) → righe
      - **Descrizione:** Legge il numero di righe nel file in input alla funzione.
      - **Pre-condizioni:** Il file deve esser stato correttamente aperto.
      - **Post-condizioni:** La funzione restituisce il numero di righe presente nel file.
  - leggi_descrizione:
    - **Specifica sintattica:** leggi_descrizione(void) -> char *
      Tipi utilizzati: char *
    - **Specifica semantica:**
      - **Funzione:** leggi_descrizione() -> descrizione
      - **Descrizione:** Legge e alloca dinamicamente la descrizione inserita per l'ordine.
      - **Pre-condizioni:** nessuna.
      - **Post-condizioni:** La funzione resitutisce la stringa che contiene la descrizione, in caso in cui l'utente non inserisce una stringa (inserisce '\n'), la funzione restituisce NULL.
  - attesa:
    - **Specifica sintattica:** attesa(void) -> void
      Tipi utilizzati: Nessuno
    - **Specifica semantica:**
      - **Funzione:** attesa(void) -> void
      - **Descrizione:** Chiede all'utente di premere invio per proseguire.
      - **Pre-condizioni:** Nessuna.
      - **Post-condizioni:** La funzione non restituisce nessun valore però richiede all'utente di premere invio per proseguire nell'esecuzione del programma.

- leggi_descrizione_da_file:
  - **Specifica sintattica:** leggi_descrizione_da_file(FILE *) -> char *

    Tipi utilizzati: puntatore a file e puntatore a char
  - **Specifica semantica:**
    - **Funzione:** leggi_descrizione_da_file(input) -> descrizione
    - **Descrizione:** Legge dinamicamente la descrizione da un file di input
    - **Pre-condizioni:** il file input deve essere stato correttamente aperto in modalità lettura.
    - **Post-condizioni:** La funzione restituisce una stringa allocata dinamicamente che contiene la descrizione letta dal file. Se il file è vuoto o non contiene dati validi, la funzione restituisce NULL.

4. Relazione Casi di Test

Il testing è un'operazione fondamentale per garantire la qualità del programma, si tratta di un insieme di attività volte a verificare il corretto funzionamento del software.

Struttura dei casi di test:

I file usati per il testing sono:

- Un file di **Testsuite.txt** che contiene i nomi di tutti i file di input e operazione di ogni test case.
- Un file **risultato.txt** che conterrà "Successo" o "Fallimento" in base all'esito del test case.

Ogni caso di test del programma avrà:

- Un file input in cui vengono inseriti i dati relativi agli ordini.
- Un file operazione in cui vengono inseriti i dettagli dell'operazione da testare.
- Un file di output in cui verrà salvato il prodotto dell'operazione.
- Un file oracle in cui è salvato il risultato atteso dell'operazione.
- Un file risultato dove indicheremo se il test è andato a buon fine oppure no.

File di selezione operazione:

Caso generale:

- Operazione da eseguire (1, 2 o 3)
- Ulteriori informazioni che variano a seconda dell'operazione

Caso di aggiunta nella coda degli ordini in attesa:

- 1

Caso di aggiunta nella coda degli ordini in elaborazione:

- 2
- Numero degli ordini da spostare dalla coda di attesa a quella di elaborazione.

Caso di aggiunta nella coda degli ordini consegnati:

- 3
- Numero degli ordini da spostare dalla coda di attesa a quella di elaborazione.
- Numero degli ordini da spostare dalla coda di elaborazione a quella degli ordini consegnati.

Test effettuati:

- **Caso di test 1:**
  - **Input:** Inserimento di 2 ordini, con descrizione.
  - **Operazione:** Aggiunta di due ordini nella coda d'attesa.

- **Caso di test 2:**
  - **Input:** Inserimento di 2 Ordini, senza descrizione.
  - **Operazione:** Aggiunta di Ordini nella coda d’attesa.

- **Caso di test 3:**
  - **Input:** Aggiunge un ordine, con descrizione oltre il limite di caratteri.
  - **Operazione:** Aggiunta di un ordine nella coda d’attesa.

- **Caso di test 4:**
  - **Input:** Ordine vuoto.
  - **Operazione:** Aggiunta di un ordine nella coda d’attesa.

- **Caso di test 5:**
  - **Input:** Inserimento di Ordine con input char* inatteso.
  - **Operazione:** Aggiunta di un ordine nella coda d’attesa.

- **Caso di test 6:**
  - **Input:** Inserimento di ordine con input di valore 0.
  - **Operazione:** Aggiunta di un ordine alla coda d’attesa.

- **Caso di test 7:**
  - **Input:** Inserimento di un ordine con input di valore superiore al numero di elementi del menù.
  - **Operazione:** Aggiunta di un ordine alla coda d’attesa.

- **Caso di test 8:**
  - **Input:** Inserimento di un ordine con più di 20 cibi.
  - **Operazione:** Aggiunta di un ordine alla coda d’attesa.

- **Caso di test 9:**
  - **Input:** Inserimento di più di 20 ordini.
  - **Operazione:** Spostamento da coda di attesa a coda di elaborazione.

- **Caso di test 10:**
  - **Input:** Inserimento di 3 ordini.
  - **Operazione:** Spostamento da coda di attesa a coda di elaborazione, con rimozione dalla coda d’attesa.

- **Caso di test 11:**
  - **Input:** Inserimento di 3 ordini.
  - **Operazione:** Controlla il corretto ordinamento della coda di elaborazione.

- **Caso di test 12:**
  - **Input:** Inserimento di 3 ordini.
  - **Operazione:** Spostamento da coda d'elaborazione a coda di consegna, con rimozione dalla coda di elaborazione.

- **Caso di test 13:**
  - **Input:** File vuoto.
  - **Operazione:** Spostamento da coda in attesa a coda d'elaborazione.

- **Caso di test 14:**
  - **Input:** File vuoto.
  - **Operazione:** Spostamento da coda di elaborazione a coda di consegna.

- **Caso di test 15:**
  - **Input:** Inserimento di 6 ordini.
  - **Operazione:** Controllo del corretto ordinamento della coda di elaborazione in presenza di ordini con tempo di preparazione uguali.

Questi test hanno tutti esito positivo certificando la corretta esecuzione del programma.

5. Conclusione

In conclusione, in questo documento abbiamo analizzato il programma da noi sviluppato, la sua progettazione, le funzioni utilizzate e il perché dei determinati ADT.

Questo progetto ci ha concesso di applicare le nostre conoscenze attuali, mettendoci alla prova con il lavoro di gruppo e la progettazione del programma.

- *Matteo Trinchese*
- *Antonio Tino*
- *Domenico Ricciardelli*