PROGETTAZIONE

Sommario

Progettazione Main

Il main si compone di 5 funzioni:

- Main (principale)
- menuCliente
- menuCorso
- menuPrenotazione
- report

e include diversi moduli, oltre a quelli standard, come:

- iscritto
- corso
- prenotazione
- lista (corsi)
- listaPrenotazioni
- hash
- utils
- test
- data

main

1. Definisco le strutture dati che mi conteranno le mie tre strutture
    - hClienti: hashtable di 30 slot che conteranno i clienti.
    - listaCorsi: lista che conterrà i corsi
    - lPrenotati: lista che conterrà le prenotazioni
2. Carico le strutture appena definite, da file.
3. Eseguo il main test (per il risultato del test cercare il file testResult.txt)
4. Stampo il menu principale con le relative opzioni, ognuno di essi riporterà al menu indicato.
5. Riscrittura dei file al termine dell'esecuzione, per il salvataggio del lavoro svolto.

menuCliente

Per ogni punto del menu, viene controllata l'esistenza del cliente e la validità dell'abbonamento.

1. Crea Cliente:
    a. Input: nome, cognome, data di iscrizione e durata abbonamento.
    b. Controllo sulla data e sulla durata, la richiede se sbagliata.
    c. Generazione automatica dell'ID cliente
    d. Creazione del cliente, tramite creaIscritto
    e. Aggiunta nella hashtable e nel file iscritti.txt.
    f. Stampa del nuovo cliente.

2. Rinnovo abbonamento
    a. Input: ID cliente
    b. Ricerca per ID tramite la funzione hashSearch.
    c. Se trovato, richiede l'estensione dell'abbonamento esistente.
    d. Ricalcolo automatico della data di scadenza.
    e. Aggiornamento della modifica riportata anche su file iscritti.txt
3. Ricerca abbonamento
    a. Ricerca per ID
        i. Funzione hashSearch (accesso diretto via hash)
        ii. Se l'abbonamento non è scaduto, viene stampato.
    b. Ricerca per Nome
        i. Funzione ricercaGenerica, criterio di ricerca 0
        ii. Se l'abbonamento non è scaduto, viene stampato.
    c. Ricerca per Cognome
        i. Funzione ricercaGenerica, criterio di ricerca 1
        ii. Se l'abbonamento non è scaduto, viene stampato.
    d. Ricerca per durata abbonamento
        i. Funzione ricercaGenerica, criterio di ricerca 2
4. Elenco Clienti
    a. Versione compatta: stampa di ID, cognome e nome dei clienti
    b. Versione estesa: tutti i dati dei clienti
5. Cancella Cliente
    a. Input: ID cliente
    b. Visualizzazione compatta dei clienti per facilitare l'operazione
    c. Cancellazione del cliente e di tutti le prenotazioni da lui effettuate
    d. Aggiornamento automatico dei partecipanti ai corsi, a seguito della cancellazione

menuCorso

Per ogni operazione viene controllata l'esistenza del corso prima di continuare l'operazione.

1. Aggiungi Corso
    a. Input: nome, data, orario
    b. Controllo su data e orario, la data deve essere valida e non antecedente alla data odierna.
    c. Generazione automatica del ID Corso.
    d. Creazione del corso (il numero di partecipanti viene inizializzato a 0)
    e. Aggiunta del nuovo corso nella lista e nel file corsi.txt.
    f. Stampa del nuovo corso.

2. Ricerca Corso
    a. Ricerca per ID
        i. Funzione ricercaGenericaLista, criterio di ricerca 0.
    b. Ricerca per Nome
        i. Funzione ricercaGenericaLista, criterio di ricerca 1
        ii. Se esistono più corsi con lo stesso nome, vengono stampati tutti
    c. Ricerca per Data
        i. Controllo validità della data inserita.
        ii. Funzione ricercaData.
        iii. Se esistono più corsi nella stessa data, vengono stampati tutti.
    d. Ricerca per Orario
        i. Funzione ricercaOrario.
        ii. Se esistono più corsi nello stesso orario, vengono stampati tutti.
3. Elenco Corsi
    a. Versione Compatta (stampa orizzontale)
    b. Vesione Estesa (stampa verticale)
4. Elimina Corso
    a. Input: ID del Corso.
    b. Visualizzazione compatta del corso per facilitare l'utilizzo.
    c. Cancellazione del corso con le relative prenotazioni.

menuPrenotazioni

Ogni funzione controlla l'esistenza del cliente/corso, prima di continuare l'esecuzione.

1. Prenota corso
    a. Input: ID Cliente, ID Corso
    b. Visualizzazione compatta dei clienti per facilitare l'utilizzo.
    c. Stampa dei dati del cliente se trovato e con abbonamento valido.
    d. Visualizzazione compatta del corso per facilitare l'utilizzo.
    e. Controllo che la data della lezione non sia antecedente ad oggi, in tal caso è impossibile prenotare la lezione.
    f. Genero l'ID della prenotazione
    g. Creo la prenotazione con i dati ottenuti
    h. Inserisco la prenotazione nella lista e nel file prenotazioni.txt
    i. Incremento il numero di partecipanti al corso selezionato.
    j. Riscrivo il file corsi.txt, per aggiornare il numero di partecipanti.
2. Ricerca Prenotazione
    a. Ricerca prenotazioni di un cliente
        i. Funzione ricercaListaPrenotati, criterio di ricerca 0.
        ii. Se il cliente ha più prenotazioni, ne verranno stampate tutte.
    b. Ricerca prenotazioni di un corso
        i. Funzione ricercaListaPrenotati, criterio di ricerca 1.
        ii. Se il corso ha più prenotazioni, ne verranno stampate tutte.

3. Elenco Prenotazioni
    a. Stampa tutte le prenotazioni della palestra
4. Elimina Prenotazioni
    a. Input: ID Cliente, ID Prenotazione.
    b. Controllo la validità dell'abbonamento.
    c. Per facilitare l'operazione, stampo tutte le prenotazioni del cliente.
    d. A seguito della cancellazione della prenotazione, viene anche decrementato il numero di partecipanti al corso della prenotazione cancellata.
    e. Aggiorno i file corso.txt e prenotazioni.txt con le nuove modifiche.

report

1. Outpur: report_[meseCorrente].txt
2. Il file contiene:
    - Il numero di prenotazioni del mese corrente, ottenibile dalla size della funzione ricercaMesePrenotazione.
    - Le tre lezioni del mese con più partecipanti, ottenibile dalla funzione lezioniInEvidenza.

ADT Iscritto

Struttura Iscritto

- ID (stringa)
- Nome (stringa)
- Cognome (stringa)
- dataIscrizione (Data)
- dataScadenza (Data)
- durata (intero, espresso in mesi)
- next (puntatore all'iscritto successivo)

Funzioni Iscritto

CreaIscritto:

1. Controllo che i valori: nome, cognome, dataIscrione, durata e ID siano validi (caso negativo ritorno NULL).
2. Creo e alloco dinamicamente lo spazio per la variabile iscritto.
3. Controllo che la memoria sia stata allocata correttamente.
4. Alloco e controllo lo spazio per i campi della variabile iscritto (caso negativo ritorno NULL).
5. Copio i valori passati come parametri nei campi dell'iscritto.
6. Richiamo la funzione copiaData e calcolaDataScadenza per i rispettivi campi, e controllo che l'operazione sia andata a buon fine (caso negativo ritorno NULL).
7. Ritorno la variabile iscritto appena creata.

getID:

1. Controllo che l'iscritto esista.
2. Ritorno l'ID dell'iscritto.

getNome:

1. Controllo che l'iscritto esista.
2. Ritorno il nome dell'iscritto.

getCognome:

1. Controllo che l'iscritto esista.
2. Ritorno il cognome dell'iscritto.

getDurata:

1. Controllo che l'iscritto esista.
2. Ritorno la durata dell'abbonamento dell'iscritto.

getDataScadenza:

1. Controllo che l'iscritto esista.
2. Ritorno la data di scadenza dell'iscritto.

getNext:

1. Controllo che l'iscritto esista.
2. Ritorno il puntatore all'iscritto successivo dell'iscritto.

setNext:

1. Controllo che l'iscritto esista.
2. Copio l'indirizzo dell'iscritto next, nel campo next dell'iscritto.

eliminaIscritto:

1. Controllo che l'iscritto esista.
2. Dealloco tramite la funzione free tutti i campi allocati dinamicamente di iscritto.

rinnovaAbbonamento:

1. Controllo che l'iscritto esiste e che la durata sia valida.
2. Somma alla durata dell'iscritto la nuova durata passata come parametro.
3. Ricalcolo la data di scadenza dell'iscritto e controllo che l'operazione sia andata a buon fine.

controlloAbbonamento:

1. Controllo che l'iscritto esista.
2. Richiamo la funzione confrontaData
   - Se uguale a -1 ritorno 1 Abbonamento Scaduto
   - Se diverso da 0 ritorno 0 Abbonamento Valido

stampaCliente

1. Controllo che l'iscritto esista.
2. Stampo i valori dei iscritto (per le date richiamo stampaData).

stampaMinimaCliente:

1. Controllo che l'iscritto esista.
2. Stampo in linea i valori: ID, cognome e nome dell'iscritto.

scriviCliente:

1. Controllo che l'iscritto esista.
2. Scrive sul file passato come parametro i campi dell'iscritto in linea (tranne dataScadenza, che viene calcolata).

ADT Corso

Struttura Corso

- ID (stringa)
- Nome (stringa)
- dataLezione (Data)
- dataScadenza (Data)
- orario (Orario)
- numPartecipanti (int)

Struttura Orario

- Ora (int)
- Minuti (int)

Funzioni Corso

creaCorso:

1. Controllo che: ID, nome, dataLezione, ora, minuti e numPartecipanti; esistano e che siano validi.
2. Se non lo sono, ritorno NULL.
3. Creo una variabile corso e l'alloco dinamicamente.
4. Controllo che la memoria sia stata allocata correttamente.
5. Alloco le variabili stringhe del corso dinamicamente e controllo l'allocazione (caso negativo ritorno NULL).
6. Assegno i valori passati come parametri ai campi della variabile corso.
7. Richiamo la funzione copiaData e creaOrario per i rispettivi campi, e controllo che l'operazione sia andata a buon fine (caso negativo ritorno NULL).
8. Ritorno la variabile corso creata.

creaOrario:

1. Creo una variabile di tipo orario e alloco la memoria dinamicamente (non controllo i parametri ora e minuti in quanto già controllati esternamente da creaCorso).
2. Controllo che la memoria sia stata allocata correttamente.
3. Assegno i valori passati come parametri ai campi della variabile orario.
4. Ritorno orario.

stampaCorso:

1. Controllo che il corso esista.
2. Richiamo la funzione Disponibilità, per stampare solo i corsi disponibili.
3. Stampo i valori del corso.

stampaCorsoCompatta:

1. Versione compatta di stampaCorso.
2. Vengono stampati in linea tutti i dati del corso.

Disponibilità:

1. Controllo che il corso esista.
2. Se il numero di partecipanti del corso è minore della macro MaxPartecipanti(=20), ritorno 1 altrimenti 0.

getIDCorso:

1. Controllo che il corso esista (caso negativo ritorno NULL).
2. Ritorno ID del corso.

getNomeCorso:

1. Controllo che il corso esista (caso negativo ritorno NULL).
2. Ritorno il nome del corso.

getDataCorso:

1. Controllo che il corso esista (caso negativo ritorno NULL).
2. Ritorno la data del corso.

getNumPartecipantiCorso:

1. Controllo che il corso esista (caso negativo ritorno NULL).
2. Ritorno il numero di partecipanti del corso.

getOrario:

1. Controllo che il corso esista (caso negativo ritorno NULL).
2. Ritorno l'orario del corso.

confrontaOrario:

1. Controllo che i due orari esistano:
   - -1 se o1<o2
   - 0 se o1=o2
   - 1 se o1>o2
2. Confronto prima l'ora:
   - Se o1→ora > o2→ ora ritorno 1
   - Se o1→ora < o2→ ora ritorno -1
3. Stessa ora, confronto i minuti:
   - Se o1→minuti > o2→ minuti ritorno 1
   - Se o1→minuti < o2→ minuti ritorno -1
4. orari uguali, ritorno 0.

incrementaPartecipanti:

1. Controllo che il corso esista.
2. Incremento il numero di partecipanti del corso.

decrementaPartecipanti:

1. Controllo che il corso esista.
2. Decremento il numero di partecipanti del corso.

scriviCorso:

1. Controllo che il corso esista.
2. Scrive sul file passato come parametro i campi del corso in linea.

ADT Prenotazione

Struttura Prenotazione

- IDPrenotazione (stringa)
- IDCorso (stringa)
- IDCliente (stringa)
- DataPrenotazione (Data)

Funzioni Prenotazione

creaPrenotazione:

1. Controllo che i valori IDPrenotazione, IDCorso, IDCliente e dataPrenotazione siano validi (caso negativo ritorno NULL).
2. Creo e alloco dinamicamente lo spazio per la variabile prenotazione.
3. Controllo che la memoria sia stata allocata correttamente.
4. Alloco e controllo lo spazio per i campi della variabile prenotazione (caso negativo ritorno NULL).
5. Copio i valori passati come parametri nei campi della variabile prenotazione.
6. Richiamo la funzione copiaData per il rispettivo campo, e controllo che l'operazione sia andata a buon fine (caso negativo ritorno NULL).
7. Ritorno la variabile prenotazione appena creata.

eliminaPrenotazione:

1. Controllo che la prenotazione esista.
2. Dealloco lo spazio tramite la funzione free di tutti i campi allocati dinamicamente di prenotazione.

getDataPrenotazione:

1. Controllo che la prenotazione esista.
2. Ritorno la data della prenotazione.

getIDCorsoPrenotazione:

1. Controllo che la prenotazione esista.
2. Ritorno l'ID Corso della prenotazione.

getIDClientePrenotazione:

1. Controllo che la prenotazione esista.
2. Ritorno l'ID Cliente della prenotazione.

getIDPrenotazione:

1. Controllo che la prenotazione esista.
2. Ritorno l'ID della prenotazione.

stampaPrenotazione:

1. Controlla che la prenotazione esista.
2. Stampa tutti i campi della prenotazione.

scriviPrenotazione:

1. Controlla che la prenotazione esista.
2. Scrive sul file passato come parametro i campi della prenotazione in linea.

ADT Data

Struttura Data

- Giorno (int)
- Mese (int)
- Anno (int)

Funzioni Data

creaData:

1. Controllo che i valori di giorno e mesi passati dai parametri sia validi
2. Se non sono validi, richiedo una nuova data, finché non inserisce una data valida
3. Dichiaro la variabile data e la alloca dinamicamente
4. Controllo che la memoria sia stata allocata correttamente
5. Assegno ad ogni campo della data, il corrispettivo passato dai parametri
6. Ritorno data

calcoloDataScadenza:

1. Controllo che la data e la durata siano validi.
2. Se non esiste, o la durata è minore di zero, ritorno NULL
3. Dichiaro una variabile scadenza di tipo data e l'alloco dinamicamente
4. Controllo che la memoria sia stata allocata correttamente
5. Copio i valori di giorno e anno della data passata come parametro e sommo ai mesi la durata passato come parametro.
6. Controllo che, se il numero di mesi è maggiore di 12.
7. Se è maggiore incremento di uno l'anno e sottraggo 12 ai mesi.
8. Ritorno la data di scadenza

copiaData:

1. Controllo che la data esista
2. Se non esiste, ritorno NULL
3. Creo una nuova variabile di tipo data che sarà la copia della data originale
4. Controllo che la memoria sia stata allocata correttamente
5. Copio i valori della data originale nella data duplicata
6. Ritorno la data duplicata

stampaData:

1. Controllo che la data esista
2. Stampo i valori della data

getGiorno:

1. Controllo che la data esista
2. Ritorno il valore del giorno della data

getMese:

1. Controllo che la data esista
2. Ritorno il valore del mese della data

getAnno:

1. Controllo che la data esista
2. Ritorno il valore dell'anno della data

dataOggi:

1. Uso la libreria time.h per ottenere la data corrente
2. Poi converto i valori nel tipo data da me creato
3. Ritorno la data creata

confrontaData:

1. Controllo che le due date esistano
    - -1 se d1<d2
    - 0 se d1=d2
    - 1 se d1>d2
2. Confronto prima gli anni
    - Se d1→anno > d2→ anno ritorno 1
    - Se d1→anno < d2→ anno ritorno -1
3. Stesso anno, confronto i mesi
    - Se d1→mese > d2→ mese ritorno 1
    - Se d1→mese < d2→ mese ritorno -1
4. Stesso mese, confronto i giorni
    - Se d1→giorno > d2→giorno ritorno 1
    - Se d1→giorno < d2→ giorno ritorno -1
5. Date uguali, ritorno 0

ADT Utils

Funzioni Utils

caricaFileClienti:

1. Apro il file iscritti.txt in modalità lettura, se l'apertura fallisce il programma termina.
2. Inizializzo una variabile statica intera maxIDCliente=0, servirà per trovare l'ultimo ID Cliente presente nel file.
3. Dichiaro e alloco lo spazio per le variabili di iscritto (le stringhe vengono allocate dinamicamente e vengono controllate)
4. Apro un while
5. Per leggere implemento una fscanf per assegnare i valori scritti alle variabile allocate precedentemente.
6. Uso la funzione creaData per compormi la data.
7. Assegno alla mia variabile iscritto il ritorno di CreaIscritto (puntatore a iscritto).
8. Controllo che la creazione non abbia avuto problemi.
9. Tramite la funzione insertHash, inserisco il nuovo iscritto nella tabella hash, passata come parametro, e controllo che l'output della funzione sia diverso da zero (0=Errore).
10. Assegno ad una variabile intera temp, il valore dell'ID convertito a int.
11. Se maggiore del maxIDCliente, maxIDCliente diventa uguale a temp.
12. Chiudo il while.
13. Assegno alla variabile statica globale IDCounterCliente il valore di maxIDCliente, che mi servirà per la funzione generaIDCliente.
14. Chiudo il file e dealloco le variabili di appoggio.

generaIDCliente

1. Incremento di uno la variabile statica globale IDCounterCliente (la precedente corrisponde ad un ID già utilizzato).
2. Alloco spazio per l'IDCliente, 7 caratteri, 3 per i caratteri, 3 per i numeri e 1 per il \0.
3. Controllo che la memoria sia stata allocata correttamente.
4. Costruisco l'IDCliente tramite il comando snprintf.
5. Ritorno l'IDCliente (Sintassi IDCliente: CLI001).

caricaFileCorso:

1. Apro il file corsi.txt in modalità lettura, se l'apertura fallisce il programma termina.
2. Inizializzo una variabile statica intera maxIDCorso=0, servirà per trovare l'ultimo ID Corso presente nel file.
3. Dichiaro e alloco lo spazio per le variabili di corso (le stringhe vengono allocate dinamicamente e vengono controllate)
4. Apro un while
5. Per leggere implemento una fscanf per assegnare i valori scritti alle variabile allocate precedentemente.
6. Uso la funzione creaData per compormi la data della corso.
7. Assegno alla mia variabile corso il ritorno di creaCorso(puntatore a corso).
8. Controllo che la creazione non abbia avuto problemi.
9. Tramite la funzione insertList, inserisco il nuovo corso nella lista, passata come parametro, e controllo che l'output della funzione sia diverso da zero (0=Errore).
10. Assegno ad una variabile intera temp, il valore dell'ID convertito a int.
11. Se maggiore del maxIDCorso, maxIDCorso diventa uguale a temp.
12. Chiudo il while.
13. Assegno alla variabile statica globale IDCounterCorso il valore di maxIDCorso, che mi servirà per la funzione generaIDCorso.
14. Chiudo il file e dealloco le variabili di appoggio.

generaIDCorso

1. Incremento di uno la variabile statica globale IDCounterCorso (la precedente corrisponde ad un ID già utilizzato).
2. Alloco spazio per l'IDCorso, 7 caratteri, 3 per i caratteri, 3 per i numeri e 1 per il \0.
3. Controllo che la memoria sia stata allocata correttamente.
4. Costruisco l'IDCorso tramite il comando snprintf.
5. Ritorno l'IDCorso (Sintassi IDCorso: CRS001).

caricaFilePrenotazioni:

1. Apro il file prenotazioni.txt in modalità lettura, se l'apertura fallisce il programma termina.
2. Inizializzo una variabile statica intera maxIDPrenotazione=0, servirà per trovare l'ultimo ID Prenotazione presente nel file.
3. Dichiaro e alloco lo spazio per le variabili di prenotazione (le stringhe vengono allocate dinamicamente e vengono controllate)
4. Apro un while
5. Per leggere implemento una fscanf per assegnare i valori scritti alle variabile allocate precedentemente.
6. Uso la funzione creaData per compormi la data della prenotazione.
7. Assegno alla mia variabile prenotazione il ritorno di creaPrenotazione(puntatore a prenotazione).
8. Controllo che la creazione non abbia avuto problemi.
9. Tramite la funzione insertListPrenotati, inserisco la nuova prenotazione nella lista, passata come parametro, e controllo che l'output della funzione sia diverso da zero (0=Errore).
10. Assegno ad una variabile intera temp, il valore dell'ID convertito a int.
11. Se maggiore del maxIDPrenotazione, maxIDPrenotazione diventa uguale a temp.
12. Chiudo il while.
13. Assegno alla variabile statica globale IDCounterPrenotazione il valore di maxIDPrenotazione, che mi servirà per la funzione generaIDPrenotazione.
14. Chiudo il file e dealloco le variabili di appoggio.

generaIDPrenotazione

1. Incremento di uno la variabile statica globale IDCounterPrenotazione (la precedente corrisponde ad un ID già utilizzato).
2. Alloco spazio per l'IDPrenotazione, 7 caratteri, 3 per i caratteri, 3 per i numeri e 1 per il \0.
3. Controllo che la memoria sia stata allocata correttamente.
4. Costruisco l'IDPrenotazione tramite il comando snprintf.
5. Ritorno l'IDPrenotazione (Sintassi IDPrenotazione: PRT001).

ADT Hash

Struttura Hash

- Size (intero)
- Table (puntatore ad un array di puntatori a iscritto)

Funzioni Hash

newHashtable:

1. Alloco lo spazio per una variabili di tipo hashtable.
2. Controllo che la memoria sia stata allocata correttamente.
3. Assegno alla size della tabella creata, la size passata tra i parametri.
4. Inizializzo dinamicamente tutti gli elementi della tabella come puntatori a iscritto.
5. Ritorno la tabella hash.

insertHash:

1. Dichiaro una variabile intera idx, che mi rappresenta l'indice di posizione in cui andro ad inserire il mio iscritto.
2. Assegno a idx il valore di ritorno generato dalla funzione di hashing, hashfun, passandogli tra i parametri l'ID dell'iscritto e la size della tabella.
3. Imposta 'curr' e 'head' (puntatori a iscritto) al primo elemento della lista in questo indice.
4. Apro un ciclo per scorrere la lista, per capire se è gia presente un iscritto con lo stesso ID, se presente ritorna 0 e il duplicato viene scartato.
5. Se l'elemento non è presente, viene inserito in cima alla lista e ritorna 1.

hashDelete:

1. Dichiaro una variabile intera idx e ne calcolo il valore tramite la funzione hashfun, che prende come stringa l'IDCliente.
2. Imposto i tre puntatori al primo elemento della lista in questo indice.
3. Scorro la lista con un while.
4. Se si trova il nodo con IDCliente uguale a quello passato come parametro, viene rimosso, se in testa viene aggiornato il primo elemento della lista, se nella lista viene fatto scalare.
5. Se trovato il cliente viene restituito, toccherà a noi richiamare la funzione eliminaCliente per deallocare lo spazio occupato.
6. Se non viene trovato nulla viene restituito NULL.

destroyHastable:

1. Viene chiamato un for che intera da 0 alla dimensione della tabella.
2. Nel corpo del for viene richiamata la funzione deleteList, che prende la lista di iscritti della posizione i della tabella.
3. Termina il for.
4. Viene deallocata la tabella hash.

deleteList:

1. Controlla che l'iscritto passato come parametro non sia NULL (caso di terminazione)
2. Richiama se stessa, ma come parametro l'elemento successivo della lista.
3. Dealloca l'iscritto.

hashSearch:

1. Controllo che la tabella e l'ID sia validi (caso negativo ritorno NULL).
2. Calcolo la posizione dall'ID tramite l'hashFun.
3. Scorro la lista affinché non trovi un iscritto con lo stesso ID.
4. Quando la trovo ritorno l'iscritto stesso, altrimenti restituisco NULL.

ricercaGenerica:

1. Inizializzo una variabile intera trovato a 0.
2. Scorro tutta la tabella (non utilizzo hashfun in quanto non sto ricercando per chiave primaria).
3. Tramite un selettore, comando lo switch in base a che tipo di ricerca devo svolgere:
   - Ricerca per nome → 0
   - Ricerca per cognome → 1
   - Ricerca per durata Abbonamento → 2
4. Quando trovo l'elemento, stampo i dettagli del cliente tramite la funzione stampaCliente e imposto trovato a 1.
5. Ritorno trovato, per capire se l'elemento fosse presente o meno nella tabella.

stampaHash:

1. Controllo che la tabella non sia vuota.
2. Scorro l'intera tabella e stampo, quando li trova, stampa i dettagli del cliente tramite la funzione stampaCliente.

stampaMinimaHash:

1. Controllo che la tabella non sia vuota.
2. Scorro l'intera tabella e stampo, quando li trova, stampa i dettagli del cliente tramite la funzione stampaMinimaCliente.

scriviFileClienti:

1. Apro il file iscritti.txt in modalità scrittura, e controllo la corretta apertura del file.
2. Scorro la tabella tramite un for.
3. Tramite la funzione scriviCliente, scrivo su iscritti.txt i campi di iscritto.
4. Chiudo il file iscritto.txt.

hashFun:

1. Algoritmo di Hashing FNV-1a

ADT List

Struttura List

- First (puntatore a nodo)
- Size (intero)

Struttura Node

- C (puntatore a corso)
- Next (puntatore a nodo)

Funzioni List

newList:

1. Creo e alloco una variabile lista.
2. Se la creazione è andata a buon fine, imposto il primo elemento della lista a NULL e la size della lista a 0
3. Ritorno la lista appena creata.

isEmpty:

1. Controllo se la lista esiste (caso negativo ritorno 1).
2. Se il primo elemento della lista non esiste, ritorno 1, altrimenti 0.
   - Ritorno 1 Lista vuota o inesistente
   - Ritorno 0 Lista piena

insertList:

1. Assegno ad una variabile nodo il ritorno della funzione insertNode, che prende come parametri, il puntatore alla testa della lista, la posizione e la variabile corso da inserire nella lista.
2. Controllo che l'operazione sia andata a buon fine (caso negativo ritorno 0).
3. Aggiorno il puntatore al primo nodo della lista con il nuovo nodo.
4. Incremento di uno la dimensione della lista.
5. Restituisco 1 per indicare il corretto inserimento nella lista.

insertNode:

1. Dichiaro due variabili nodo, e faccio puntare uno dei due nodi all nodo in testa alla lista, passato dai paramentri.
2. Alloco lo spazio per il nodo che conterrà il corso e controllo che la memoria sia stata allocata correttamente (caso negativo ritorno NULL).
3. Copio la variabile corso all'interno del nodo.
4. Controllo se l'inserimento deve essere in testa, o all'interno della lista.
   - Se in testa (pos=0), assegno al puntatore next del nodo creato, l'indirizzo del nodo in testa alla lista. Ritorno il nodo creato (diventato la testa della lista).
   - Se la posizione è diversa da 0, imposto un ciclo che scorre la lista fino alla posizione inserita o fino alla fine della lista.
     Se il nodo è uguale a NULL, vuol dire che la lista è più corta della posizione indicata, dealloco lo spazio per il nodo che contiene il corso e ritorno NULL.
     Se invece non è NULL, il nuovo nodo punterrà al nodo successivo di della posizione indicata, e aggiorno il nodo precedente al nuovo facendolo puntare al nodo stesso.
     Ritorno la lista.

removeList:

1. Controlla se la lista è vuota o non inizializzata e ritorno 0.
2. Altrimenti richiamo la funzione removeNode, che prende la lista e la posizione e restituisce la lista aggiornata.
3. Decremento di uno la dimensione della lista.
4. Ritorno 1 per indicare la corretta rimozione dell'elemento.

removeNode:

1. Creo una variabile nodo, che punterà al nodo da eliminare.
2. Se l'eliminazione è in testa (pos=0):
   a. Assegno l'indirizzo della lista al nuovo nodo creato.
   b. Aggiorno la lista, assegnando al nodo in testa alla lista l'indirizzo del nodo successivo.
   c. Libero il nodo creato che conteneva il corso da cancellare.
3. Se l'eliminazione non è in testa:
   a. Scorro la lista fino alla posizione indicata.
   b. Controllo che il nodo alla posizione esista.
   c. Se esiste assegno l'indirizzo del nodo da cancellare al nodo creato, collego il nodo precedente al nodo successivo al nodo da cancellare e libero il nodo da cancellare.
4. Ritorno la lista aggiornata.

reverseList:

1. Creo la lista che conterrà gli elementi invertiti.
2. Scorro la lista passata come parametro.
3. Man mano che scorro inserisco nella nuova lista gli elementi, tramite la funzione insertList, con posizione uguale a 0, per l'inserimento in testa.
4. Se l'inserimento fallisce il programma si chiude.
5. Ritorno la lista con gli elementi invertiti.

ricercaGenericaLista:

1. Controllo che la lista esista (caso negativo ritorno 0).
2. Creo una lista che conterrà i riscontri della ricerca (questo per gestire corsi che hanno lo stesso nome).
3. Scorro ogni elemento della lista e tramite un selettore imposto il parametro di ricerca:
   - Selettore=0 Ricerca per ID
   - Selettore=1 Ricerca per Nome
4. Quando trovo l'elemento lo inserisco nella nuova lista tramite il comando insertList, con posizione uguale a 0.
5. Al termine del ciclo ritorno la lista.

ricercaData:

1. Controllo che la lista esista (caso negativo ritorno 0).
2. Creo una lista che conterrà i riscontri della ricerca (questo per gestire corsi che hanno la stessa data).
3. Scorro ogni elemento della lista.
4. Per confrontare le date uso la funzione confrontaData, che ritorna 0 quando sono uguali.
5. Quando trovo l'elemento lo inserisco nella nuova lista tramite il comando insertList, con posizione uguale a 0.
6. Al termine del ciclo ritorno la lista.

ricercaMese:

1. Controllo che la lista esista (caso negativo ritorno 0).
2. Creo una lista che conterrà i riscontri della ricerca (questo per gestire corsi che hanno lo stesso mese).
3. Scorro ogni elemento della lista.
4. Quando trovo l'elemento lo inserisco nella nuova lista tramite il comando insertList, con posizione uguale a 0.
5. Al termine del ciclo ritorno la lista.

ricercaOrario:

1. Controllo che la lista esista (caso negativo ritorno 0).
2. Creo una lista che conterrà i riscontri della ricerca (questo per gestire corsi che hanno lo stesso orario).
3. Scorro ogni elemento della lista.
4. Per confrontare gli orari uso la funzione confrontaOrario, che ritorna 0 quando sono uguali.
5. Quando trovo l'elemento lo inserisco nella nuova lista tramite il comando insertList, con posizione uguale a 0.
6. Al termine del ciclo ritorno la lista.

lezioniInEvidenza:

1. Controllo che la lista esista.
2. Inizializzo tre variabili intere e tre variabili corso, che serviranno per tenere traccia delle lezioni più frequentate.
3. Scorro la lista per analizzare uno ad uno il numero di partecipanti di ogni corso.
   - Se il numero di partecipanti è maggiore del valore della variabile primo, i valori vengono scalati, sia le variabili che contengono il numero di partecipanti che i corsi stessi.
   - Stesso discorso viene applicato per il secondo e il terzo posto.
4. Se le variabili corso esistono, stampo in ordine i tre corsi più frequentati, tramite la funzione stampaCorso.

stampaLista:

1. Controllo che la lista esista.
2. Scorro la lista elemento per elemento, per poi stampare le informazioni del corso tramite la funzione stampaCorso.

stampaListaEssenziale:

1. Controllo che la lista esista.
2. Scorro la lista elemento per elemento, per poi stampare le informazioni del corso tramite la funzione stampaCorsoCompatta.

scriviFileCorso:

1. Apro il file corsi.txt in modalità scrittura, e controllo la corretta apertura del file.
2. Scorro la lista elemento per elemento.
3. Tramite la funzione scriviCorso scrivo su corsi.txt i campi di corso.
4. Chiudo il file corso.txt.

getFirstCorso:

1. Controllo che la lista esista (caso negativo ritorno NULL).
2. Ritorno il corso del primo nodo della lista.

cancellaCorso:

1. Controllo che la lista esista (caso negativo ritorno 0).
2. Scorro la lista elemento per elemento, affinche non trovo il corso con lo stesso ID passato come parametro.
3. Per cancellarlo richiamo la fuzione removeList e controllo l'uscita del valore:
   - remoreList==0 ritorno 0 errore nella cancellazione.
   - removeList==1 ritorno 1 cancellazione avvenuta.
4. Se al termine del ciclo non viene trovato l'elemento da cancellare ritorno 0.

scriviLezioniInEvidenza

1. Controllo che la lista esista (caso negativo, scrivo su file "Non ci sono corsi in evidenza").
2. Scorro tutta la lista e ne stampo l'ID, il nome, la data e il numero di partecipanti del corso.

ADT ListaPrenotazione

Struttura ListaPrenotazione

La struttura è identica alla sua controparte per i corsi (liste.c) l'unica differenza sono la presenza di variabili Prenotazioni invece delle variabili Corso.

Funzioni ListaPrenotazioni

- newListPrenotazioni
- insertListPrenotazioni
- insertNode
- removeListPrenotati
- removeNode
- reverseListPrenotati
- cancellaPrenotazione
- getFirstPrenotazione
- isEmptyPrenotazione
- ricercaMesePrenotazione
- stampaListaPrenotazioni
- scriviFilePrenotazioni

Presentato la stessa progettazione della loro controparte per i corsi (liste.c), l'unica differenza sono la presenza di variabili Prenotazione invece delle variabili Corso.

ricercaListaPrenotati:

1. Controllo che la lista esista (caso negativo ritorno 0).
2. Creo una lista che conterrà i riscontri della ricerca (questo per gestire le prenotazioni di un cliente o di un corso).
3. Scorro ogni elemento della lista e tramite un selettore imposto il parametro di ricerca:
   - Selettore=0 Ricerca per ID Cliente
   - Selettore=1 Ricerca per ID Corso
   - Selettore=2 Ricerca per ID Prenotazione
4. Quando trovo l'elemento lo inserisco nella nuova lista tramite il comando insertList, con posizione uguale a 0.
5. Al termine del ciclo ritorno la lista.

cancellaPrenotazioniDi:

1. Controllo che la lista esista (caso negativo ritorno 0).
2. Inizializzo una variabile contatore elementiCancellati=0, che mi servirà per tener conto di quante prenotazioni sono state cancellate a seguito della cancellazione di un cliente o della cancellazione del corso.
3. Scorro ogni elemento della lista è tramite il selettore passato come parametro imposto il parametro per scegliere cosa eliminare:
   - Selettore=0 Cancello tutte le prenotazioni di un cliente
   - Selettore=1 Cancello tutte le prenotazioni di un corso
4. Se trovo l'elemento, richiamo la funzione removeListPrenotati e incremento il contatore elementiCancellati.
5. Ritorno il contatore elementiCancellati.

ADT Test

Funzioni Test

mainTest:

1. Apro in lettura testSuite.txt e in scrittura testResult.txt
2. Controllo che l'apertura dei file sia avvenuta correttamente
3. Dichiaro una stringa che mi conterrà il nome del test che andrò ad effettuare e ne controllo l'allocazione.
4. Avvio un ciclo che termina quando il file viene letto completamente.
   a) Ad ogni lettura assegno ad una variabile intera result il ritorno di runTest, che prende: il nome del test, il selettore, che mi indica quale test sto facendo, le tre strutture dati che contengono gli iscritti, i corsi e le prenotazioni.
   b) Se result è uguale ad 1, scrivo nel file testResult il nome del test e Pass (TC1_Prenotazioni: Pass), che me ne indica il successo; altrimenti il nome del file e Fail (TC1_Prenotazioni: Fail), che ne rappresenta il fallimento.
5. Chiudo il file aperti all'inizio e libero la memoria del nome del test.

runTest:

1. Costruisco le tre stringhe che mi rappresentano il nome del file di input, di output e l'oracolo del test che eseguiremo:
   - TC1_Prenotazione_Input.txt
   - TC1_Prenotazione_Output.txt
   - TC1_Prenotazione_Oracle.txt
2. Tramite il valore del selettore passato tra i parametri, uno switch sceglie che tipo di test devo eseguire:
   a. Verifica della corretta registrazione delle prenotazioni e dell'aggiornamento delle disponibilità.
   b. Test della gestione degli abbonamenti e della verifica della validità.
   c. Verifica che il report generato contenga informazioni corrette e complete sulle prenotazioni.
3. Ognuno di questi casi ritorna il valore della funzione chiamata, le funzioni sono:
   a. testCorrettaRegistrazionePrenotazione
   b. testValiditàAbbonamento
   c. testReport

testCorrettaRegistrazionePrenotazione:

1. Apro il file fileInput in lettura, e fileOutput in scrittura.
2. Controllo che vengano aperti correttamente.
3. Alloco spazio per due stringhe, che conterranno l'IDCorso e l'IDCliente, e controllo che l'allocazione sia andata a buon fine.
4. Creo una nuova lista di corsi, che mi servirà per controllare che il corso esista, successivamente mi servirà anche per incrementare e decrementare il numero di partecipanti al corso.
5. Avvio un ciclo che termina quando viene letto l'intero fileInput.
6. Ad ogni iterazione controllo che gli ID letti da file esistano:
    a. Caso Negativo: scrivo sul fileOutput "Cliente o Corso Inesistente"
    b. Caso Positivo:
        1. Creo la nuova prenotazione con i dati letti dal file.
        2. Controllo la corretta creazione della prenotazione.
        3. Incremento il numero di partecipanti del corso a cui fa riferimento la prenotazione.
        4. Scrivo sul fileOutput i due ID e il numero di partecipanti al corso.
        5. Decremento il numero di partecipanti.
7. Chiudo i due file e dealloco lo spazio dedicato ai due ID.
8. Richiamo la funzione confrontaFile, per confrontare l'oracolo e l'output, se uguali ritorno 1, altrimenti 0.

testValiditàAbbonamento:

1. Apro il file fileInput in lettura, e fileOutput in scrittura.
2. Controllo che vengano aperti correttamente.
3. Alloco spazio per una stringa, che conterranno l'IDCliente, e controllo che l'allocazione sia andata a buon fine.
4. Creo una variabile iscritto i, che mi conterrà l'iscritto dell'ID letto da file.
5. Avvio un ciclo che termina quando viene letto l'intero fileInput.
6. Ad ogni iterazione assegno ad i, l'iscritto trovato dalla funzione hashSearch, che data la tabella hash e l'ID dell'iscritto, restituisce un puntatore ad esso.
    a. Se i è uguale a NULL, scrivo sul fileOutput "Cliente inesistente".
    b. Se i è diverso da NULL, controllo che l'abbonamento sia valido, tramite la funzione controlloAbbonamento, se il ritorno della funzione è 0, scrivo "Abbonamento valido", altrimenti "Abbonamento Scaduto".
7. Chiudo i due file e dealloco lo spazio dedicato all'ID.
8. Richiamo la funzione confrontaFile, per confrontare l'oracolo e l'output, se uguali ritorno 1, altrimenti 0.

testReport:

1. Apro il file fileInput in lettura, e fileOutput in scrittura.
2. Controllo che vengano aperti correttamente.
3. Creo un array di stringhe contenenti tutti i mesi, mi servirà poi per determinare l'output del programma.
4. Assegno ad una variabile intera meseR, il valore del mese corrente.
5. Creo due liste, una per Corsi e una per Prenotazioni, e le carico tramite le funzioni caricaFileCorso e caricaFilePrenotazioni. (non uso le strutture già riempite dal programma, in quanto le modifiche che devo apportare per effettuare il test dovrammo poi essere cancellate, in questo modo non dovrò riaggiornare le strutture che vengono riempite all'inizio del main).
6. Alloco spazio per due stringhe, che conterranno l'IDCorso e l'IDCliente, e controllo che l'allocazione sia andata a buon fine.
7. Creo tre liste di Corsi:
    - resultCorso: per controllare che il corso esista.
    - corsiDelMese: lista contenente solo i corsi del mese corrente.
    - Classifica: lista dei tre corsi più richiesti del mese.
9. Ad ogni iterazione controllo che gli ID letti da file esistano:
    a. Caso Negativo: scrivo sul fileOutput "Cliente o Corso Inesistente"
    b. Caso Positivo:
        1. Creo la nuova prenotazione con i dati letti dal file.
        2. Controllo la corretta creazione della prenotazione.
        3. Aggiungo la prenotazione alla lista Prenotazioni già precedentemente riempita.
        4. Incremento il numero di partecipanti del corso a cui fa riferimento la prenotazione.
10. Assegno a corsi del mese, il ritorno della ricercaMese.
11. Assegno a classifica, la lista proveniente dalla funzione lezioniInEvidenza, che restituisce una lista con i tre corsi più richiesti del mese corrente.
12. Stampo il mese corrente, il numero di prenotazioni del mese, ottenibile grazie alla size della lista prenotati, e i primi tre corsi del mese, tramite la funzione scriviLezioniInEvidenza, che prende la lista classifica e il puntatore a file del file sul cui scrivere.
13. Chiudo i file precedentemente aperti e dealloco le variabili allocate dinamicamente
14. Richiamo la funzione confrontaFile, per confrontare l'oracolo e l'output, se uguali ritorno 1, altrimenti 0.

confrontaFile:

1. Apro due file in lettura, con i nomi dei file presi dai parametri.
2. Controllo la corretta apertura dei file.
3. Alloco lo spazio per due stringhe, la prima stringa leggerà dal primo file, la seconda stringa dal secondo file. Ne controllo l'allocazione.
4. Pongo di base che i due file siano uguali, ponendo una variabile intera uguali=1.
5. Avvio un ciclo che termina quando entrambi i file sono terminati.
6. Ad ogni iterazione confronto le due stringhe tramite strcmp, se sono diversi cambio il valore di uguali, ponendolo a 0 ed esco dal ciclo.
7. Libero lo spazio occupato dalle due stringhe e chiudo i due file.
8. Ritorno il valore di uguali.