

LABORATORIO DI FONDAMENTI DI INFORMATICA

Incontro 5 di 5 – Liste Monodirezionali, File

Elemento di una lista monodirezionale

Predisporre un tipo di variabile denominato `s_elem` adatto ad essere usato come tipo di elemento di una lista monodirezionale. Il record deve contenere:

- un carattere denominato `codice`
- un numero intero denominato `quantita`
- un puntatore all'eventuale elemento successivo nella lista denominato `next`

Funzione “stampalista”

Implementare una funzione denominata `stampalista` che, dato un puntatore alla testa di una lista con elementi del tipo specificato sopra e una etichetta testuale (ad esempio "Listal"), mostri a video l'etichetta e il contenuto di tutti gli elementi della lista.

Listal: [F, 9] -> [U, 5] -> [Q, 2] -> [F, 8] -> [L, 9] ->NULL

Funzione “creaelem”

Implementare una funzione denominata `creaelem` che, dati come parametri un carattere `codice` e un numero intero `quantita`, allochi un nuovo elemento e ne ritorni il puntatore (o `NULL` se l'allocazione fallisce).

Funzione “inseriscitest”

Implementare una funzione denominata `inseriscitest` che, dati come parametri la testa di una lista e un nuovo elemento, vada ad inserire l'elemento in testa alla lista.

Nella funzione `main` del programma occorre inizializzare a `NULL` la testa di una lista denominata `listal` e richiamare `DIM 10` volte le funzioni `creaelem` e `inseriscitest` passando come parametri un `codice` e una `quantita` generati casualmente nel seguente modo:

```
codice = 'A' + rand() % ('Z' - 'A' + 1); //codice A-Z  
quantita = rand() % 10; //quantita' 0-9
```

Richiamare poi la funzione `stampalista` per verificare l'avvenuto inserimento.

[L, 9]; [F, 8]; [Q, 2]; [U, 5]; [F, 9];

Listal: [F, 9] -> [U, 5] -> [Q, 2] -> [F, 8] -> [L, 9] ->NULL

Funzione “inseriscicoda”

Implementare una funzione denominata `inseriscicoda` analoga ad `inseriscitest` ma che vada ad inserire un nuovo elemento in coda alla lista. Nella funzione `main` del programma testare il funzionamento andando a creare e riempire una lista denominata `lista2`.

[K, 5]; [Y, 2]; [V, 3]; [R, 9]; [Y, 2];

Lista2: [K, 5] -> [Y, 2] -> [V, 3] -> [R, 9] -> [Y, 2] ->NULL

Funzioni “cercaelem” e “raggruppa”

Implementare una funzione denominata `cercaelem` che, presi come parametri la testa di una lista e un carattere, ritorni il puntatore al primo elemento della lista che contiene il `codice` passato come parametro (o `NULL` se tale elemento non esiste).

Implementare poi una funzione denominata `raggruppa` che, presa come parametro la testa di una lista, crei una nuova lista come indicato di seguito e ne ritorni la testa. La nuova lista da creare deve essere una copia della lista di origine in cui però gli elementi con lo stesso `codice` sono “raggruppati” sommando le `quantita`. Ad esempio, se nella lista di origine ci sono gli elementi [A,3]...[A,2]...[A,5]..., nella nuova lista si avrà un unico elemento [A, (3+2+5)] = [A,10]. Per creare la nuova lista, la funzione `raggruppa` deve “scorrere” la lista originale, richiamare la funzione `cercaelem` per verificare se nella nuova lista esiste già un elemento con il `codice` corrispondente all'elemento corrente e in caso affermativo aggiornarne la `quantita`, in caso

negativo aggiungere un nuovo elemento chiamando `creaelem e inseriscicoda`.

Nella funzione main del programma creare la lista `list3` passando come parametro alla funzione `raggruppa` la lista `list1`.

`List1: [F, 9] -> [U, 5] -> [Q, 2] -> [F, 8] -> [L, 9] -> NULL`

Genero `Lista3` come "raggruppamento" di `List1`.

`Lista3: [F, 17] -> [U, 5] -> [Q, 2] -> [L, 9] -> NULL`

Funzione "filtralista"

Implementare una funzione denominata `filtralista` che prenda come parametri la testa di una lista e un codice. La funzione deve eliminare dalla lista tutti gli elementi che contengono il codice passato come parametro. La memoria occupata dagli elementi eliminati va liberata tramite la funzione di libreria `free`.

Nella funzione main del programma occorre chiedere all'utente quale codice desideri eliminare dalla lista `list1` e poi richiamare le funzioni `filtralista` e `stampalista`. Si consiglia di eseguire un test di eliminazione del primo elemento della lista, dell'ultimo e di uno in mezzo.

`List1: [F, 9] -> [U, 5] -> [Q, 2] -> [F, 8] -> [L, 9] -> NULL`

Quale codice vuoi eliminare da `List1` (A-Z) ? L

Elimino da `List1` 'L'.

`List1: [F, 9] -> [U, 5] -> [Q, 2] -> [F, 8] -> NULL`

Funzione "dividilista"

Implementare una funzione denominata `dividilista` che prenda come primo parametro la testa di una lista contenente degli elementi e come secondo e terzo parametro le teste di altre due liste vuote da popolare all'interno della funzione (attenzione alle modalità di passaggio dei parametri perché tutte le liste verranno modificate). La funzione deve analizzare la lista passata come primo parametro e "dividerla" nelle altre due liste, senza allocare memoria ma solo "spostando" i puntatori, eliminando gli elementi con quantità multipla di 5. La prima lista `pari` deve contenere tutti gli elementi con quantità pari, la seconda lista `dispari` deve contenere tutti gli elementi con quantità dispari (eccetto i multipli di 5). Non è necessariamente richiesto mantenere l'ordine originale degli elementi (cioè si possono anche spostare gli elementi dalla lista di origine a quella dei pari o dei dispari "agganciandoli" in testa perché più semplice; se lo si desidera si può comunque provare l' "aggancio" in coda).

Nella funzione main occorre richiamare questa funzione passando come parametro la lista `list3` e due nuove liste vuote `pari` e `dispari`.

Le tre liste vanno stampate a video in main per verifica (dopo lo spostamento la lista di origine `list3` dovrebbe risultare vuota).

`Lista3: [F, 17] -> [U, 5] -> [Q, 2] -> [L, 9] -> NULL`

Divido `Lista3` in `LPari` e `LDispari` eliminando i multipli di 5.

`ListaPari: [Q, 2] -> NULL`

`ListaDispari: [L, 9] -> [F, 17] -> NULL`

`Lista3: NULL`

Funzioni "salvalista" e "caricalista"

Implementare due funzioni denominate `salvalista` e `caricalista` per salvare e caricare da file una lista.

Le funzioni ricevono come parametri la testa di una lista (che in caso di salvataggio deve contenere elementi, mentre in caso di caricamento deve essere vuota e modificabile), il path del file da salvare o caricare (massimo `PATH_LEN` 30 caratteri) e la modalità di apertura del file (testuale o binaria) definita tramite il seguente tipo enumerato:

```
typedef enum {txt, bin} t_file;
```

Le due funzioni ritornano 0 in caso l'operazione sia andata a buon fine, -1 se è avvenuto qualche errore.

Nella modalità testuale gli elementi vanno salvati nel file uno per riga, separando il codice e la quantità con uno spazio, utilizzando le funzioni `fprintf` e `fscanf` della libreria `<stdio.h>` con stringa di formato "%c %d\n".

Esempio di file testuale con due elementi:

L 9

F 17

Nella modalità binaria gli elementi della lista si possono salvare e caricare direttamente utilizzando le funzioni `fread` e `fwrite` (anche se sarebbe inutile salvare e caricare il loro campo "next" perché, essendo un indirizzo che cambia a seconda della posizione degli elementi nella memoria, ogni volta va "ricalcolato").

Nella funzione main provare a salvare la lista pari dei numeri pari in un file testuale e la lista dispari dei numeri dispari in un file binario.

Passaggio di parametri

La funzione main può ricevere dei parametri che vengono passati "dall'esterno" al momento del lancio del file eseguibile del programma.

```
int main(int argc, char *argv[]) { ... }
```

Il valore di `argc` è il numero di parametri ricevuti e `argv[]` è un vettore di stringhe che contiene i valori degli `argc` parametri.

Il primo parametro `argv[0]` esiste sempre (e quindi `argc` è sempre ≥ 1) e corrisponde al nome (eventualmente comprensivo di tutto il percorso nel filesystem) del file eseguibile.

Gli altri parametri `argv[1], ..., argv[argc - 1]` invece esistono solo se sono stati specificati da chi ha lanciato il programma.

I parametri vengono tutti ricevuti come stringhe nel vettore `argv`. Se necessario, per convertire un parametro stringa in numero intero, si può utilizzare la funzione `atoi` presente in `<stdlib.h>` che ritorna un numero intero derivante dalla conversione o 0 in caso la stringa non rappresenti un numero valido.

Provare a modificare il programma in modo che possa ricevere dall'esterno tre parametri:

- il numero di elementi generati casualmente da inserire nelle liste (prima stabilito in modo costante tramite `#define DIM 10`)
- il nome del file testuale per la lista dei pari
- il nome del file binario per la lista dei dispari

Occorre quindi controllare che `argc` sia = 4, convertire `argv[1]` in numero intero e considerare come path per i file `argv[2]` e `argv[3]`.

```
Controllo parametri...
argv[0] = "C:\lab\liste.exe"
argv[1] = "10"
argv[2] = "pari.txt"
argv[3] = "dispari.bin"
Numero elementi = 10
Path File testuale = "pari.txt"
Path File binario = "dispari.bin"
```

NB: E' possibile consultare il documento "Passaggio di parametri" contenente le istruzioni su come passare parametri al programma tramite IDE o tramite terminale.