UNIVERSITÀ DEGLI STUDI DI BERGAMO

Informatica
Modulo di Programmazione

Informatica
Modulo di Programmazione
Lists

mauro.pelucchi@gmail.com
Mauro Pelucchi

2023/2024

Agenda

- Lists
- Files

Liste semplicemente concatenate

- La definizione del tipo di dato astratto lista è intrinsecamente ricorsiva, infatti una lista può essere:
  - la lista vuota
  - oppure contiene un valore (di un certo tipo di dato) seguito da una lista
- Per poter definire in C un tipo di dato astratto "lista", si utilizza un puntatore a una struttura che contiene un campo di informazioni e un campo che è invece il puntatore alla struttura successiva.

Liste semplicemente concatenate

```
typedef struct Node {
        int data;
        struct Node *next;
} node;

typedef node* list;
```

Liste semplicemente concatenate

- Mentre gli "array" sono "tipi di dato" ad accesso casuale, cioè è possibile accedere in maniera diretta a ogni singolo elemento(con [ ]); una lista è un "tipo di dato" ad accesso sequenziale, cioè bisogna scorrere necessariamente tutti gli elementi che precedono l'elemento d'interesse prima di potervi accedere.

- Inoltre una lista semplice è sempre identificata con il puntatore al primo nodo e termina sempre con un puntatore a NULL.

Esercizio 1

Scrivere i sottoprogrammi di manipolazione delle liste concatenate semplici (di interi), indicate di seguito, utilizzando un procedimento ricorsivo e uno iterativo per ognuna di esse:

- Creazione di un nuovo nodo:
```
nodo *createNode(int d);
```
- Inserimento in testa di un nuovo nodo con prototipo seguente:
```
list insertHead(list h, node *el);
```
- Inserimento in coda di un nuovo nodo con prototipo seguente:
```
list insertTail(list h, node *el);
```
- Stampa di una lista :
```
void printList(list h);
```
- Calcolo della lunghezza di una lista:
```
int lenLista(list h);
```
- Ricerca elemento in una lista:
```
list searchList(list h, int x);
```
- Copia di una lista:
```
list copyList(list h);
```
- Concatenazione di due liste:
```
void appendLista(list& h1, list& h2);
```
- Inserimento in una lista ordinata:
```
list insertSorted(list h, int x);
```
- Cancellazione di una lista:
```
void  wipeList(list& h);
```
- Cancellazione 1ma occorrenza di un elemento da una lista:
```
void deleteList(list& h, int x);
```
- Cancellazione di tutte le occorrenze di un elemento in lista:
```
void  deleteListAll(list& h, int x);
```

Esercizio 2

Scrivere una funzione in linguaggio C che elimini tutti i
nodi di posto pari in una lista semplicemente concatenata di interi.
Es: 1 5 7 9 3 diventa 1 7 3

Esercizio 3

Scrivere una funzione in linguaggio C che elimini tutti i nodi di posto dispari in una lista semplicemente concatenata di interi.
Es: 1 5 7 9 3 diventa 5 9

Esercizio 4
Appello del 10 Luglio 2009

Si consideri un programma che gestisce gli orari dei treni di un servizio ferroviario.
In particolare, si definisca una lista dinamica che rappresenta le singole fermate di ciascun treno: ogni nodo della lista descrive un numero di treno, il numero progressivo della fermata (la stazione di partenza corrisponde alla fermata numero 1), il nome della stazione, l'ora e i minuti (si usino due campi separati di tipo intero).
Nella stessa lista possiamo perciò trovare le fermate di treni diversi.
Dopo aver definito la struttura dati, si scriva in C/C++ la funzione TreniAlternativi, che riceve come parametro la lista delle fermate, e un numero di treno; la funzione restituisce il numero dei treni che collegano i due capolinea del treno fornito come parametro (se il treno indicato come parametro parte dalla stazione A e arriva alla stazione B, vogliamo sapere quanti sono i treni che collegano A con B, anche se A e B non sono capolinea).
La funzione restituisce -1 se il treno indicato come parametro non esiste.

Esercizio 5

La funzione …toro(…) riceve come parametro un vettore di interi e la specifica della sua dimensione, e alloca e restituisce una lista dinamica "circolare" di interi che contiene solo i valori del vettore positivi e divisibili per 11. Ovviamente la lista può essere circolare solo se non è vuota, quindi si suggerisce di renderla circolare solo alla fine dell'analisi del vettore

[N.B. una lista è circolare se l'ultimo elemento invece di avere un puntatore a NULL ha un puntatore alla testa].

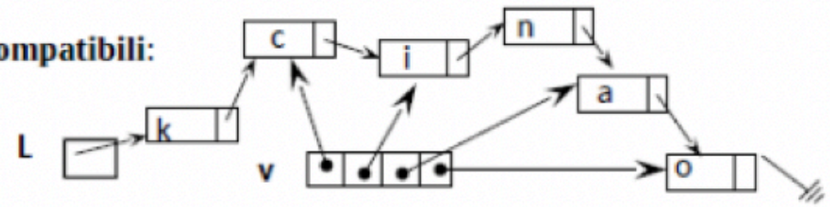
- Si definiscano la struttura della lista e il prototipo della funzione toro
- Si codifichi la funzione (unitamente alle eventuali funzioni di supporto)
- Si disegni, la lista ottenuta con la funzione passando come parametro il vettore

Esercizio 6

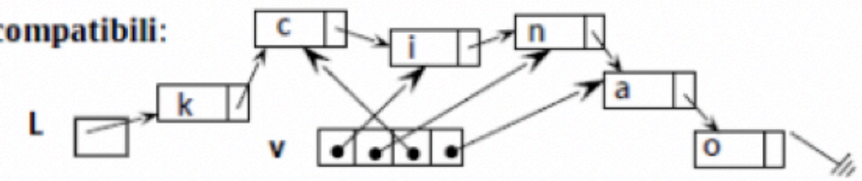
```
struct Nodo { char c;
              Nodo * next; };
typedef Nodo * Lista;
```

Si considerino una lista **L** di almeno N nodi e un vettore **v** di esattamente N puntatori a Nodo, tutti diversi da NULL e diversi tra loro, che puntano a un sottoinsieme dei nodi di **L**. Diciamo che **L** è *compatibile* con **v** se i nodi puntati da **v** compaiono in **L** in un ordine **compatibile** con l'ordine in cui i puntatori sono contenuti in **v** (cioè non succede mai che un nodo *n1* puntato dal puntatore **v**[i] preceda in **L** il nodo *n2* puntato dal puntatore **v**[j] che però si trovi "più avanti" in **v**).

Esempi (con N=4): **compatibili**:



incompatibili:



a) Si codifichi la funzione **…compatibili(…)** che controlla che una lista e un vettore siano compatibili secondo la precedente definizione.

b) Si codifichi la funzione **…vprintf(…)** che stampa su stdout la parola ottenuta seguendo nell'ordine i puntatori di **v** e accedendo ai caratteri contenuti nella lista, se **L** e **v** sono compatibili, e nulla se sono incompatibili (stamperebbe "ciao" nel caso dell'esempio soprastante).

Esercizio 7

Si consideri la seguente definizione:

```
struct Nodo { char * parola;
              Nodo * next;  };
typedef Nodo * Lista;
```

(a) Si disegni la struttura allocata dalle istruzioni del blocco di codice a lato, si indichino i valori di ogni variabile (sia statica sia dinamica: per i puntatori si usino le frecce, per i valori ignoti i punti interrogativi), e se ne calcoli la dimensione totale in byte.

(b) Due parole si dicono *simili* se hanno al più due caratteri diversi. Una catena di parole si dice *compatibile col telefono senza fili* (*cctsf*) se ogni parola è *simile* alle adiacenti.

(c) Si codifichi la funzione **int simili (char *s1, char *s2);**

(d) Si consideri poi la definizione di una lista di catene di parole (da ogni **NodoTesta** inizia una **Lista**).

```
struct NodoTesta { Lista catena;
                   NodoTesta * next;  };
typedef NodoTesta *ListaDiListe;
```

Si codifichi una funzione che riceve una lista di liste così definita e dealloca interamente dalla lista di liste le sequenze di parole che non sono cctsf.
Attenzione: si ipotizzi che nelle catene non ci siano parole allocate staticamente, e si badi a deallocare *tutta* la memoria dinamica.

Esercizio 8

Si consideri la definizione

```
struct Nodo { int numero;
              Nodo* next; };
typedef Nodo * Lista;
```

Si codifichi la funzione **…spargidivisori(…)**, che riceve come parametri una lista di interi, una matrice NxN di interi, *tutti strettamente positivi* (come matrice dinamica), e la lunghezza N.

La funzione deve cercare di copiare ogni valore **v** della lista nella matrice, inserendolo al posto di un valore che sia multiplo di **v**. *Se ci riesce, restituisce 1, e la matrice deve contenere tutti i valori modificati, se non ci riesce, però, oltre a restituire 0, deve lasciare* ***inalterata*** *la matrice*.

Attenzione:

i valori **v** devono sempre essere confrontati *con la versione iniziale* della matrice, non con le "versioni intermedie" derivanti dalla sostituzione di alcuni valori, se ci sono più multipli di **v**, se ne può sostituire uno a piacere (il primo che si incontra), si badi a definire chiaramente e/o dichiarare eventuali opportune strutture dati di appoggio o funzioni ausiliarie.