Esercizio 1 - Sintesi di codice: Algoritmica di base

Definiamo il grado minimo g e il grado massimo G di una parola P rispettivamente come il minimo e il massimo numero di occorrenze delle lettere di P in P. Ad esempio:

ISTANBUL g=1, G=1 (tutte le lettere della parola compaiono in essa una e una sola volta)
BOSFORO g=1, G=3 (B, S, F, R compaiono una sola volta, O compare tre volte)
GALATASARAY g=1, G=5 (G, L, T, S, R, Y compaiono una sola volta, A compare cinque volte)
MARMARA g=2, G=3 (M e R compaiono due volte, A compare tre volte)
G e g valgono convenzionalmente 0 per la parola vuota (cioè per una parola priva di caratteri).

(a) Si dichiari opportunamente il prototipo di una funzione C … **gradi(**…**)** che riceve come parametro una stringa di lunghezza generica che rappresenta P, calcola G e g, e li comunica al programma chiamante.
(b) Si completi poi il main riportato qui sotto con l'opportuna chiamata alla funzione dichiarata.

```
int main() { int
   max, min;
   char par[14]="PASSAMONTAGNA";
   ……
  cout << par << " g= " << min << "max = " <<
  max; return 0;
}
```

(b) Si descriva sinteticamente (ma in modo preciso, possibilmente per punti) un algoritmo per la funzione **gradi()** adatto ad essere codificato. Si trascuri la gestione dei caratteri maiuscoli e minuscoli.
(c) Si codifichi in C la funzione dichiarata al punto (a) secondo l'algoritmo descritto al punto (b).

Esercizio 2 - Sintesi di codice: Ricorsione

Scrivere per ciascuno dei problemi seguenti un sottoprogramma ricorsivo:

(a) Calcolare il numero di occorrenze di un numero intero, *x,* in un array di interi v[].
(b) Assumendo che v[] rappresenti una stringa, calcolare la sua lunghezza.
(c) Assumendo che v[] rappresenti una stringa, stabilire se la stringa è palindroma.
(d) Assumendo v[] un vettore di interi, stabilire se il vettore contiene una sequenza di numeri monotona crescente.
(e) Definire due sottoprogrammi ricorsivi che calcolino, rispettivamente, la somma e il prodotto di due numeri interi.
(f) Scrivere un sottoprogramma iterativo e uno ricorsivo che calcolino l' n-esimo termine della successione di Fibonacci: Fib(0) = 0, Fib(1) = 1, Fib(n) = Fib(n-1)+Fib(n-2).
(g) Scrivere una funzione che calcoli il massimo comun divisore di due naturali positivi (con algoritmo di Euclide).
(h) Scrivere una funzione ricorsiva per calcolare il resto della divisione intera tra due numeri interi.

Esercizio 3 - Sintesi di codice: Ricorsione

Si progetti la funzione ricorsiva che svolge il compito seguente. Siano dati due vettori V1 e V2, di dimensione N1 e N2, rispettivamente (con $1 \leq N2 \leq N1$). La funzione restituisce il valore 1 in uscita se tutti gli elementi del vettore V2 si trovano nel vettore V1 nell'ordine inverso rispetto a quello in cui essi figurano in V2, ma non necessariamente in posizioni immediatamente consecutive; altrimenti (ovvero se questo non si verifica), la funzione restituisce valore 0.
Si deve prima dare la stesura informale per punti della funzione ricorsiva e poi la codifica in C/C++, precisandone chiaramente la testata.

Esercizio 4 - Sintesi di codice: vettori e liste

Le telecamere di un parcheggio multi-livello riconoscono le targhe di ogni auto in ingresso. Se la targa è contenuta nel file *credito.txt* (formattato come nel riquadro) e il credito è di almeno 2,50 €, la funzione **int** `checkin(`**char*** `targa)` restituisce 1, altrimenti restituisce 0 (e una botola risucchia l'auto in un pozzo senza fondo).

```
EA456XB 14.75

CP313AF 1.15 Giuseppe Psaila
```

N.B. Non si trascuri, nel seguito, la definizione di eventuali funzioni ausiliarie, per rendere più facile il controllo del codice innanzitutto durante la sua creazione.

(a) Si codifichi in C la funzione `checkin`

(b) Le auto ammesse leggono su un display il piano a cui devono recarsi, che è calcolato automaticamente dal sistema come il primo che abbia almeno un posto libero.
La struttura dati del sistema è la seguente

```
typedef struct Car { char * targa;
                     struct Car * next;  } Auto;
typedef Auto * Lista;
typedef struct Level { int postiliberi;
                       Lista macchine;
                       struct Level * next; } Piano;
typedef Piano * ListaDiListe;
```

dove ogni piano è pieno se il suo attributo `postiliberi` vale 0 e vuoto se la sua lista `macchine` è vuota.
Si codifichi in C la funzione **int** assegnapiano**(**ListaDiListe L, **char** * targa**)**, che restituisce il numero del piano oppure -1 se non c'è posto. La funzione alloca anche il nodo corrispondente all'auto entrata.

(c) Si gestisca con la funzione `...checkout( ... )` l'uscita di un'auto dal parcheggio, in modo da lasciare la struttura dati in uno stato consistente. La funzione non si occupa dell'addebito del costo del parcheggio. Le soluzioni sostanzialmente ricorsive ricevono un punto di bonus.

Esercizio 5 - Memoria dinamica, liste

Sia data una struttura dati dinamica di tipo lista semplicemente concatenata. Ogni elemento della lista contiene un vettore di due numeri reali. Il puntatore al primo elemento della lista viene conservato in una variabile globale chiamata radice. Si svolgano i punti seguenti:

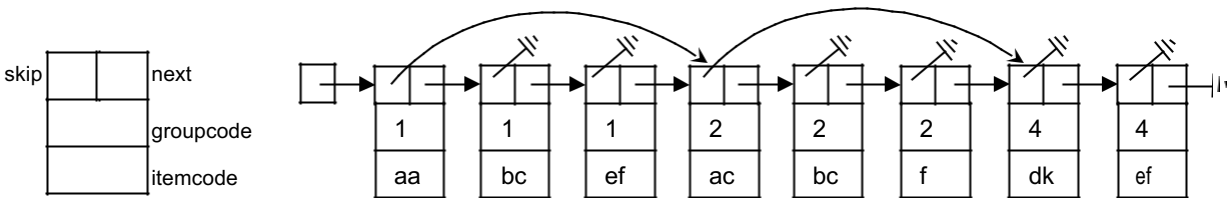
1. Si scrivano le strutture dati ricorsive che realizzano la lista in questione.
2. Si progetti una funzione che prende in ingresso come argomento la radice della lista e duplica la lista, ovvero crea una seconda lista, indipendente dalla prima ma identica ad essa (cioè avente la medesima lunghezza, contenente gli stessi elementi nel medesimo ordine); la funzione restituisce il puntatore al primo elemento della seconda lista. Si dia prima la stesura informale per punti della funzione, poi la testata e infine la codifica in C.
3. Si progetti una funzione che prende in ingresso come argomenti le radici di due liste indipendenti e restituisce in uscita il valore 1 se le due liste sono identiche, 0 altrimenti. Si dia prima la stesura informale per punti della funzione, poi la testata e infine la codifica in C/C++.

Per i punti (2) e (3) si possono usare funzioni ausiliarie, se lo si ritiene opportuno. Vanno prima descritte informalmente per punti, quindi codificate in C/C++ (testata e corpo).

Esercizio 6 - Memoria dinamica, liste

```
typedef struct Knt { int groupcode;
                     char * itemcode;
                     struct Knt *skip, *next; } Node;
typedef Node *SkipList;
```

La struttura soprastante definisce una *Skip List semplice* (Pugh, 1990) in cui a una normale lista concatenata (dal puntatore next) è sovrapposto un secondo ordine di puntatori a “blocchi in avanti” della stessa lista. Ad esempio i brani (*item*) di una playlist possono essere rappresentati in una lista di gruppi (ad esempio di album) e, a pari album (*gruppo*), in ordine di titolo, col puntatorie skip del primo brano di ogni album che punta al primo brano dell’album successivo. L’ultimo "gruppo" non ha gruppi successivi.



Queste strutture permettono una ricerca più rapida, "saltando" i gruppi non interessanti e poi cercando un elemento solo nel suo gruppo di appartenenza, ma permettono anche una normale scansione lineare.

(a) Si implementi in C/C++ un’opportuna funzione ... TrovaElem(...) che riceve come parametri una skiplist e i codici di un gruppo e di un elemento, e restituisce NULL o un puntatore all’elemento con gli estremi dati se la skiplist contiene un tale elemento.

(b) Se si è data al punto precedente una formulazione ricorsiva, se ne dia ora una iterativa, o viceversa

(c) Si spieghi come si può implementare e si codifichi in C/C++ una funzione ... Dealloca(...) che riceve come parametro un puntatore a un nodo da eliminare e lo dealloca completamente dalla lista. In particolare si presti attenzione a descrivere come gestire l’eliminazione del primo elemento di un gruppo, che può anche essere l’unico membro di tale gruppo.

Esercizio 7 - Memoria dinamica, liste

Per cercare un valore in una lista concatenata semplice, anche se ordinata, occorre scandirla dall'inizio. Per velocizzare l'accesso, si può affiancare alla lista un vettore ausiliario V che indica dove iniziano e finiscono vari segmenti della struttura. In particolare, la struttura qui sotto rappresenta una lista di parole ordinata alfabeticamente, in cui ogni elemento del vettore ausiliario punta al primo e all'ultimo nodo del segmento di lista relativo alle parole che iniziano con una lettera particolare: la cella V[0] è relativa all'iniziale 'A', v[1] alla 'B', ... v[25] alla 'Z'. Si considerino per semplicità parole di sole lettere maiuscole. Se non vi sono parole con una particolare iniziale, i due puntatori sono posti a NULL.

```
typedef struct Nd { char * word;
                    struct Nd *next; } Nodo;
typedef Nodo *Lista;
typedef struct Blc { Lista primo;
                     Lista ultimo } Blocco;
typedef Blocco Vettore[26]
```

Si codifichi in C/C++ la funzione

```
Lista inserisci(char * p, Lista L, Vettore V)
```

che inserisce una nuova parola p in ordine alfabetico nella struttura (L, V) aggiornando opportunamente il vettore ausiliario e restituendo la lista modificata.

Esercizio 8 - Memoria dinamica, liste

```
typedef struct ND { unsigned int matr;
                    char * nome;
                    struct ND *nxtMatr, *nxtNome; }
Nodo; typedef Nodo * ListPtr;
typedef struct DP { ListPtr headMatr, headNome; } Lista;
```

La struttura dati dinamica soprastante rappresenta un elenco di studenti contemporaneamente ordinato rispetto a due criteri indipendenti (ordine di nome e di matricola, entrambi crescenti e, per semplicità, entrambi univoci). Le matricole sono assegnate in modo progressivo ad ogni iscrizione.

(a) Si codifichi in C/C++ la funzione ... **aggiungistudente(**...**)** che riceve come parametro il nome di uno studente e la lista in cui inserirlo, e restituisce la lista modificata. (Soluzioni iterative o ricorsive sono considerate equivalenti ai fini dell'esercizio)

(b) L'elenco degli studenti è conservato su un file ordinato per matricola (crescente), in cui ogni studente è rappresentato da due righe: la prima contiene la matricola, la seconda il nome. Si codifichi in C/C++ la funzione ... **caricalista(**...**)** che apre il file archivio.txt per allocare e restituire la lista degli studenti ivi contenuti.

(c) Si codifichi ricorsivamente in C/C++ la funzione ... **miracolo(**...**)** che verifica (restituendo 1 in caso affermativo, 0 altrimenti) se l'ordinamento lessicografico della lista ricevuta come parametro coincide con quello per matricola.

(d) ***Facoltativo:*** Si consideri la seguente variante astratta della lista precedente, in cui i nodi hanno solo i due puntatori che inducono gli ordinamenti.

```
typedef struct ND { struct ND *nxt1, nxt2; } Nodo;
typedef Nodo * ListPtr;
typedef struct DP { ListPtr head1, head2; } Lista;
```

Si codifichi in modo puramente ricorsivo e senza allocare memoria nello heap (cioè senza invocare new…) la funzione ... **ordinamentiopposti(**...**)** che verifica se nella lista ricevuta come parametro gli ordinamenti rappresentati dai due puntatori sono esattamente opposti (cioè ogni nodo è, in base a uno degli ordinamenti, successivo del suo successivo in base all'altro ordinamento).

Esercizio 9

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.]

(a) Si definiscano la struttura della lista e il prototipo della funzione toro.

(b) Si codifichi la funzione (unitamente alle eventuali funzioni di supporto).

Esercizio 10

Un Testo è rappresentato tramite una lista dinamica di caratteri.
Una Parola è rappresentata da una lista dinamica di caratteri.

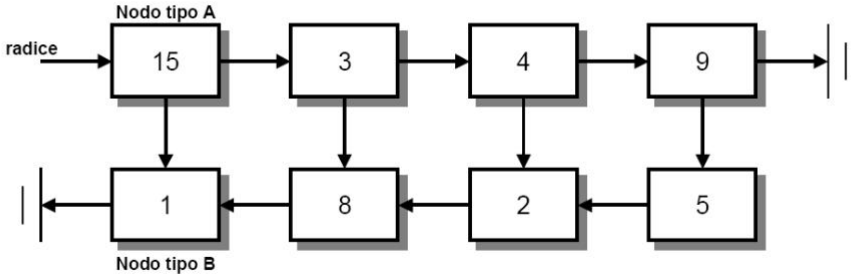
Si scriva la funzione ricorsiva **CercaParola** che riceve un Testo e una Parola come parametri d' ingresso ed esegue una ricerca dei caratteri della Parola all'interno del Testo. I caratteri trovati possono anche essere separati da altri caratteri nel Testo, ma devono apparire nell'ordine in cui appaiono nella Parola.
Scrivere poi una seconda versione della stessa funzione rinominandola come **ExtractParola**, che restituisca una lista di puntatori alle posizioni in cui i caratteri trovati appaiono nel Testo.

Ad esempio, la parola "camera" nel Testo: "**c**orsodifond**ame**ntidiinfo**r**m**a**tica" si trova nelle posizioni indicate in grassetto, mentre la parola "cane" non si trova.

Il programma principale dovrà quindi leggere dallo standard input un Testo (terminato dal carattere di "ritorno a capo") e una Parola (anch'essa terminata dal carattere di "ritorno a capo"); stampare a terminale le liste dinamiche che li memorizzano e stampare il risultato della funzione **CercaParola** e stampare i caratteri corrispondenti alla lista di puntatori restituita da **ExtractParola**.

Esercizio 11

Sia data una *struttura dati dinamica* formata da due liste concatenate, di uguale lunghezza, orientate in versi opposti (si veda la figura). La lunghezza minima di ogni lista è di 2 nodi. Ogni nodo contenga un numero intero e i puntatori necessari. Le due liste siano accoppiate nel modo seguente: da ogni nodo della lista superiore sia possibile immettersi nel corrispondente nodo della lista inferiore, tramite un puntatore apposito (si veda la figura).



Si chiede di svolgere i due punti seguenti:

1. Definire le strutture dati *ricorsive* opportune. che servono a gestire il sistema delle due liste accoppiate (vale a dire, definire il nodo di tipo A e quello di tipo B, si veda la figura).
2. Progettare e codificare in C una procedura *iterativa*, avente come argomento la radice della lista superiore, che si comporti nel modo seguente:
   - percorra la lista superiore e scriva a terminale il valore di ogni nodo attraversato;
   - quando incontri un nodo avente valore superiore a quello del nodo precedente, lo scriva a terminale, ma poi si immetta nella lista inferiore, la percorra e la scriva a terminale, a partire dal nodo di immissione fino al termine della lista.

Per esempio, in riferimento alla figura, la procedura così specificata scriverebbe a terminale la successione di valori: `15 3 4 2 8 1`; l'immissione nella lista inferiore avviene in corrispondenza dell'elemento `4`.

Qualora la lista superiore venga percorsa fino in fondo, al termine di essa la procedura si deve immettere comunque nella lista inferiore (e dunque stamparla tutta quanta a terminale).