Esame di Fondamenti di Programmazione
27-1-09 – Sterbini – AD
Compito A
Prima Parte

(da svolgere se non avete esonero o se volete migliorarne il voto)

Esercizio 1

Si supponga di dover realizzare il gioco del Sudoku. Si scriva la funzione **isValid** che:

- riceve come argomenti una matrice 9x9 di interi **M** e tutti gli altri argomenti che ritenete necessari
  - gli elementi della matrice possono contenere il valore **0** (che sta per "vuoto") oppure i valori da 1 a 9
- calcola se la configurazione contenuta nella matrice è valida (ciascuno dei 9 sottoquadrati da 9 caselle, ed ogni riga ed ogni colonna, contengono lo stesso numero solo una volta)
- nel caso in cui la configurazione NON sia valida individua una delle caselle che contengono il valore ripetuto (nella riga o nella colonna o nel quadretto)
- torna 3 valori (per riferimento): se la conf. è valida (vero/falso), la riga (0-8) e la colonna (0-8) della casella individuata

Seconda Parte

(obbligatoria)

Esercizio 2

Sia dato un tipo struct definito come segue:

```
typedef struct nodo {
        long val;
        struct nodo * figlio_sx;
        struct nodo * figlio_dx;
} Nodo;
```

Scrivere una funzione `controlla_albero` che riceve in input un puntatore a `Nodo` (e tutti gli altri argomenti che ritenete necessari) e verifica se l'albero radicato in quel nodo è un albero binario di ricerca. La funzione deve restituire 0 se l'albero è un albero binario di ricerca, 1 se ci sono nodi i cui campi `val` contrastano con le regole che un albero binario di ricerca deve rispettare. (In particolare, un albero vuoto è un albero di ricerca valido.)

Suggerimento: usate due flag e due variabili (max e min); i flag indicano se i valori di max e min sono inizializzati, mentre max e min indicano il range valido per il nodo corrente – aggiornate correttamente i valori di min e max quando visitate i sottoalberi.

Esercizio 3

Sia dato un tipo struct definito come segue:

```
typedef struct nodo {
        double val;
        struct nodo * succ;
} Nodo;
```

Scrivere una funzione `elimina_intervallo` che riceve in input una lista (non ordinata) di elementi di tipo `Nodo`, un `double min` e un `double max` e rimuove dalla lista tutti gli elementi i cui campi `val` siano `>min` e `<max`. I nodi rimossi devono essere inseriti in una nuova lista di nodi che verrà restituita dalla funzione `elimina_intervallo`.

Esame di Fondamenti di Programmazione
27-1-09 – Sterbini – AD
Compito B
Prima Parte

(da svolgere se non avete esonero o se volete migliorarne il voto)

Esercizio 1

Si supponga di dover realizzare uno sparatutto 3D in prima persona. Nel gioco è necessario visualizzare istante per istante tutti gli oggetti presenti nell'ambiente in ordine di distanza decrescente dal giocatore, per fare in modo che gli oggetti più vicini "coprano" i più lontani e si ottenga una visualizzazione realistica. Si scriva la funzione **ordina3D** che riceve in ingresso:

- un vettore **oggetti** di struct definite come segue, che contengono la posizione e la grandezza di un oggetto:

```
typedef struct oggetto3D {
        int x; int y; int z;        // coordinate della posizione dell'oggetto
        int raggio;                 // raggio della sfera che contiene l'oggetto
} Oggetto3D;
```

- le coordinate del personaggio nel sistema (3 interi)
- tutti gli altri argomenti che ritenete necessari

La funzione deve ordinare il vettore di struct in ordine decrescente di distanza dalla posizione del giocatore, e tornare, in tre variabili passate per riferimento, le coordinate dell'oggetto più vicino.
NOTA: ciascun oggetto occupa una sfera, per cui la distanza dell'oggetto dal giocatore ne deve tener conto.

Seconda Parte

(obbligatoria)

Esercizio 2

Sia dato un tipo struct definito come segue:

```
typedef struct nodo {
        char * val;
        struct nodo * figlio_sx;
        struct nodo * figlio_dx;
} Nodo;
```

Scrivere una funzione `inserisci` che riceve in input un albero binario di ricerca e una stringa e inserisce un nuovo elemento di tipo `Nodo` nell'albero (qualora non sia già presente un nodo con lo stesso valore). L'ordine fra i nodi è dato dall'ordine lessicografico fra i campi `val`.

Suggerimenti: le stringhe nel campo val dei nuovi nodi non devono essere nuovamente allocate; i confronti fra stringhe possono essere effettuati utilizzando la funzione strcmp della libreria string.h.

Esercizio 3

Sia dato un tipo struct definito come segue:

```
typedef struct nodo {
        long val;
        struct nodo * succ;
} Nodo;
```

Scrivere una funzione `pad` che riceve in input una lista di elementi di tipo `Nodo`, ordinati per valori non decrescenti del campo val e un `long max_diff` e inserisce nella lista un nuovo elemento con campo `val` pari alla parte intera inferiore della metà della somma fra i valori dei campi `val` del nodo precedente e successivo, qualora questi valori differiscano di più di `max_diff`. Ipotizziamo che nella lista ci siano degli elementi a e b, dove a->succ = b; se b->val – a->val è maggiore di max_diff, deve essere inserito fra a e b un nuovo elemento c con campo val pari ad (a->val+b->val)/2.

La funzione deve restituire il numero di elementi aggiunti.

Esame di Fondamenti di Programmazione
27-1-09 – Sterbini – AD
Compito C
Prima Parte

(da svolgere se non avete esonero o se volete migliorarne il voto)

Esercizio 1

Si supponga di dover trovare tutti gli anagrammi di una parola (una parola è un anagramma di un'altra se contiene tutte le lettere dalla parola, ma in ordine diverso).
Si scriva una funzione **stampaAnagrammi** che riceve come argomenti:

- una stringa che contiene la parola da anagrammare
- un vettore di stringhe, ordinato, che contiene il vocabolario di parole conosciute
- tutti gli altri argomenti che pensate possano servire

La funzione deve stampare i primi 10 anagrammi della parola (o quelli che trova, se sono di meno).

Seconda Parte

(obbligatoria)

Esercizio 2

Sia dato un tipo struct definito come segue:

```
typedef struct nodo {
        long val;
        struct nodo * figlio_sx;
        struct nodo * figlio_dx;
} Nodo;
```

Scrivere una funzione elimina che riceve in input un albero binario di ricerca con elementi di tipo Nodo e un long valore ed elimina dall'albero di ricerca (se presente) il nodo con campo val pari a valore, insieme a tutto il suo sottoalbero. Ricordate di deallocare la memoria per ogni elemento eliminato dall'albero.

Suggerimento: eliminate gli elementi solo dopo averli resi foglie.

Esercizio 3

Sia dato un tipo struct definito come segue:

```
typedef struct nodo {
        char * val;
        struct nodo * succ;
} Nodo;
```

Scrivere una funzione inserisci che riceve in input una lista di elementi di tipo Nodo e una stringa e inserisce un nuovo nodo nella lista. La lista in origine è ordinata rispettando l'ordine lessicografico per il campo val, e gli inserimenti devono mantenere questa proprietà.

Le stringhe nel campo val dei nuovi nodi non devono essere nuovamente allocate; i confronti fra stringhe possono essere effettuati utilizzando la funzione strcmp della libreria string.h.

Esame di Fondamenti di Programmazione
27-1-09 – Sterbini – AD
Compito D
Prima Parte

(da svolgere se non avete esonero o se volete migliorarne il voto)

Esercizio 1

Si supponga di dover realizzare il gioco delle parole crociate. Si scriva la funzione **trovaParola** che:

- riceve come argomenti:
  - una matrice RxC di caratteri **M** che rappresenta la tabella delle parole crociate, che potrebbe essere già parzialmente riempita. Gli elementi della matrice possono contenere i valori: **' '** (spazio, che sta per "vuoto") oppure 'X' (che sta per casella nera) oppure una qualsiasi lettera minuscola
  - una stringa **parola** e tutti gli altri argomenti che ritenete necessari
- cerca nella matrice la **parola**, che può essere disposta orizzontalmente, verticalmente o in diagonale
- torna il valore 1 (orizzontale) 2 (verticale) 3 (diagonale) o 0 (parola non trovata)
- inoltre torna, in due variabili passate per riferimento, le coordinate (riga e colonna) della prima lettera della parola trovata

Seconda Parte

(obbligatoria)

Esercizio 2

Sia dato un tipo struct definito come segue:

```
typedef struct nodo {
        long val;
        struct nodo * figlio_sx;
        struct nodo * figlio_dx;
} Nodo;
```

Un albero binario di ricerca è bilanciato se la lunghezza del cammino più lungo da radice a foglia e la lunghezza del cammino più corto differiscono al più di 1. Questo significa che per ogni nodo dell'albero, il suo sottoalbero sinistro ha circa lo stesso numero di nodi del suo sottoalbero destro.

Scrivere una funzione `costruisci_bilanciato` che riceve in input un vettore ordinato di long, il numero di elementi nel vettore, un puntatore a Nodo (inizialmente tale puntatore varrà NULL per identificare un albero vuoto) e tutti gli altri argomenti che ritenete necessari, e inserisce gli elementi del vettore nell'albero in maniera tale che l'albero risulti bilanciato.

Suggerimento: il primo valore da inserire, che sarà la radice dell'albero, deve essere l'elemento mediano del vettore, e il ragionamento va iterato nelle due metà rimanenti.

Esercizio 3

Sia dato un tipo struct definito come segue:

```
typedef struct nodo {
        int val;
        struct nodo * succ;
} Nodo;
```

Scrivere una funzione `clona_shift` che riceve in input una lista di elementi di tipo `Nodo` e un intero `i` e restituisce una nuova lista i cui elementi hanno come valore la somma fra il campo `val` del corrispondente elemento nella lista ricevuta in input e il valore `i`. La lista originale NON deve essere modificata.

Gli elementi devono essere aggiunti alla nuova lista senza scorrere l'intera lista ad ogni inserimento (quindi utilizzando una pila e inserendoli in ordine inverso, oppure utilizzando una coda).