Complementi di Programmazione

Esercitazione 7

- Tutti gli esercizi devono essere risolti con funzioni **ricorsive**.
- Scrivere dei test nel main per verificare che le funzioni scritte siano corrette.
- E' consentito l'utilizzo di funzioni ausiliarie, purché siano ricorsive.
- Nel caso di utilizzo di funzioni ausiliarie ricorsive, allora la funzione principale può essere non ricorsiva in quanto richiama la funzione ausiliaria ricorsiva. In ogni caso, è vietato usare istruzioni di ciclo in entrambe le funzioni.
- Non è consentito l'utilizzo di variabili globali o static o qualsiasi altro meccanismo analogo di condivisione di memoria tra tutte le attivazioni delle funzioni ricorsive.

SCL float

```
typedef float TipoInfoSCL;
struct ElemSCL {
     TipoInfoSCL info;
     struct ElemSCL *next;
};

typedef struct ElemSCL TipoNodoSCL;
typedef TipoNodoSCL * TipoSCL;
```

Esercizio 7.1

Implementare la funzione:

```
int scl_len(TipoSCL scl);
```

che calcola la lunghezza struttura collegata `scl`.

Esercizio 7.2

Implementare la funzione:

```
float scl_sum(TipoSCL scl);
```

che calcola la somma degli elementi contenuti nella lista `scl`.

Esercizio 7.3

Implementare la funzione:

```
float scl_media(TipoSCL scl);
```

che restituisce il valore medio degli elementi della lista. L'esercizio deve essere svolto **senza** l'ausilio delle precedenti due funzioni.

Esercizio 7.4

Implementare la funzione:

```
float scl_dot(TipoSCL scl1, TipoSCL scl2);
```

che ritorna il prodotto scalare, risultato della moltiplicazione degli elementi delle due liste.

Esercizio 7.5

Scrivere una funzione:

```
void scl_duplicate_pos(TipoSCL scl, int pos);
```

che modifichi la SCL in input duplicando l'elemento in posizione `pos` (si inserisca un nuovo nodo un nodo nella posizione successiva).

Ex:
Scl: [1,2,3,4,5]
Pos : 2
Risultato : [1,2,3,3,4,5]

Esercizio 7.6

Scrivere una funzione:

(versione con side-effect)
```
void scl_positives(TipoSCL scl, TipoSCL *ris);
```

(versione funzionale)
```
TipoSCL scl_positives(TipoSCL scl);
```

che modifica la SCL `ris` (versione con side-effect) oppure che ritorna una nuova SCL (versione funzionale) contenente solo gli elementi con valori maggiori o uguali di zero di `scl`.

SCL char

Sia data una struttura collegata lineare definita come nella Parte 1, in cui ora il tipo di `info` è definito:

```
typedef char TipoInfoSCL;
```

Esercizio 7.7

Scrivere la funzione:

```
void sclstring_print(TipoSCL s);
```

che data in input la struttura `s`, ne stampi a schermo tutti i caratteri. Una SCL vuota corrisponderà alla stringa vuota “”.

Esercizio 7.8

Scrivere la funzione:

(versione con side-effect)

```
void sclstring_create(const char *s, TipoSCL *ris);
```

(versione funzionale)
```
TipoSCL sclstring_create(const char *s);
```

che data in input una stringa, generi una struttura SCL che la rappresenti.

```
Es:
Input: "SCL"
Output: 'S'->'C'->'L'
```

Esercizio 7.9

Scrivere una funzione:

```
bool sclstring_equals(TipoSCL scl, const char *s);
```

che restituisca true se e solo se la stringa rappresentata da `scl` è uguale a `s`.
Nota: *non* utilizzare `strcmp`.
Nel main, utilizzare questa funzione come ulteriore controllo per il funzionamento di `sclstring_create`.

Esercizio 7.10

Scrivere una funzione:

```
void sclstring_remove(TipoSCL *scl_p, char val);
```

che modifichi la struttura puntata da `scl_p`, eliminando tutti gli elementi con valore uguale a `val`. Se `val` non è presente, non va fatta alcuna modifica.
Si usi `sclstring_equals` per verificare il corretto funzionamento.
Nota: attenzione a liberare correttamente la memoria.