Esercitazione di Laboratorio del 30/04/2021: Ordinamento ed ElemType

Esercizio 1

Sono dati i file `elemtype.h` ed `elemtype.c`, contenenti la definizione del tipo `ElemType` e delle seguenti funzioni:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
void ElemSwap(ElemType *e1, ElemType *e2);
void ElemDelete(ElemType *e);
int ElemRead(FILE *f, ElemType *e);
int ElemReadStdin(ElemType *e);
void ElemWrite(const ElemType *e, FILE *f);
void ElemWriteStdout(const ElemType *e);
```

La documentazione è disponibile al link https://federicobolelli.it/fondamenti-ii/elemtype/int/html/elemtype_8h.html

Si creino i file `vettore.h` e `vettore.c` che consentano di definire la `struct`:

```
typedef struct {
    size_t capacity;
    size_t size;
    ElemType *data;
} Vector;
```

e le funzioni:

```
Vector *VectorRead(const char *filename);
Vector *VectorReadSorted(const char *filename);
```

La struct `Vector` rappresenta un vettore di `ElemType` contenente `size` elementi ed una capacità totale pari a `capacity`. Il primo elemento si trova alla posizione `data[0]`, il secondo alla posizione `data[1]`, e così via.

La funzione `VectorRead()` accetta come parametro una stringa C contenente il nome di un file che deve essere aperto in lettura in modalità tradotta. Il file contiene una sequenza di `ElemType`, che possono essere letti tramite la funzione `ElemRead()`. La funzione deve creare un nuovo `Vector`, allocato dinamicamente su heap, contenente la sequenza di `ElemType` presente sul file. Infine, la funzione deve ritornare il puntatore al `Vector` creato. Se il file di input non esiste, o non è possibile aprirlo, la funzione ritorna `NULL`.

La funzione `VectorReadSorted()` ha un comportamento analogo a `VectorRead()`, ma deve produrre un vettore **ordinato** in senso crescente: un vettore è ordinato in senso crescente se ogni elemento è $\leq$

del successsivo e $\geq$ del precedente. Ogni nuovo elemento letto dal file seve essere direttamente inserito nel vettore alla posizione corretta, affinché l'ordinamento parziale sia sempre rispettato; questo può richiedere di spostare verso destra parte degli elementi già presenti nel vettore. Per confrontare due elementi, si usi la funzione `ElemCompare()`.

Su OLJ dovete sottomettere solamente i file `vector.h` e `vector.c`.

Esercizio 2

Sono dati i file `elemtype.h` ed `elemtype.c`, contenenti la definizione del tipo `ElemType` e delle seguenti funzioni:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
void ElemSwap(ElemType *e1, ElemType *e2);
void ElemDelete(ElemType *e);
int ElemRead(FILE *f, ElemType *e);
int ElemReadStdin(ElemType *e);
void ElemWrite(const ElemType *e, FILE *f);
void ElemWriteStdout(const ElemType *e);
```

La documentazione è disponibile al link https://federicobolelli.it/fondamenti-ii/elemtype/int/html/elemtype_8h.html

Si creino i file `vettore.h` e `vettore.c` che consentano di definire la `struct`:

```
typedef struct {
    size_t capacity;
    size_t size;
    ElemType *data;
} Vector;
```

e le funzione:

```
void VectorSort(Vector* v);
```

La struct `Vector` rappresenta un vettore di `ElemType` contenente `size` elementi ed una capacità totale pari a `capacity`. Il primo elemento si trova alla posizione `data[0]`, il secondo alla posizione `data[1]`, e così via.

La funzione `VectorSort()` accetta come parametro un puntatore a `Vector`, e deve ordinare il vettore in senso crescente, tramite un algoritmo di ordinamento a scelta (ma non funzioni di libreria). Per confrontare ed eventualmente scambiare di posto due elementi, usare rispettivamente le funzioni `ElemCompare()` e `ElemSwap()`.

Se `v == NULL` la funzione termina senza far nulla.

Su OLJ dovete sottomettere solamente i file `vector.h` e `vector.c`.

Esercizio 3

Sono dati i file `elemtype.h` ed `elemtype.c`, contenenti la definizione del tipo `ElemType` e delle seguenti funzioni:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
void ElemSwap(ElemType *e1, ElemType *e2);
void ElemDelete(ElemType *e);
int ElemRead(FILE *f, ElemType *e);
int ElemReadStdin(ElemType *e);
void ElemWrite(const ElemType *e, FILE *f);
void ElemWriteStdout(const ElemType *e);
```

La documentazione è disponibile al link https://federicobolelli.it/fondamenti-ii/elemtype/int/html/elemtype_8h.html

Si creino i file `vettore.h` e `vettore.c` che consentano di definire `struct`:

```
typedef struct {
    size_t capacity;
    size_t size;
    ElemType *data;
} Vector;
```

e la funzione:

```
int VectorFind(const Vector *v, const ElemType *e);
```

La struct `Vector` rappresenta un vettore di `ElemType` contenente `size` elementi ed una capacità totale pari a `capacity`. Il primo elemento si trova alla posizione `data[0]`, il secondo alla posizione `data[1]`, e così via.

La funzione `VectorFind()` accetta come parametri un puntatore ad un `Vector` **ordinato** in senso crescente, `v`, ed un puntatore a `ElemType`, `e`. La funzione deve cercare l'elemento `e` all'interno di `v`, e ritornarne la posizione. Se `e` compare più volte nel vettore, ritornare la posizione della prima occorrenza. Se `e` non è presente nel vettore, ritornare `-1`.

È utile sapere che il vettore è ordinato?

Su OLJ dovete sottomettere solamente i file `vector.h` e `vector.c`.

Esercizio 4

Si realizzi un programma da riga di comando con la seguente sintassi:

```
ordina_interi <input_file> <output_file>
```

Il programma `ordina_interi` accetta come parametri due stringhe. La prima stringa, `input_file`, è il nome di un file di testo che contiene una sequenza di numeri interi espressi in base 10 con caratteri ASCII, separati da caratteri `<whitespace>`.

Il programma deve scrivere nel file `output_file` (creandolo se non esiste, altrimenti sovrascrivendolo) gli stessi numeri presenti in `input_file`, ma ordinati in senso crescente, separati dal carattere `<a capo>`.

Se non è possibile aprire `input_file` o non è possibile creare `output_file`, il programma deve terminare con codice di errore 1.

Se il numero di parametri è diverso da 2, il programma deve stampare un messaggio di errore su `stderr` e terminare con codice di errore 2.

Ad esempio, dato `input_file`:

```
1 5 9 -3 0 11 -453
```

La funzione dovrà scrivere in `output_file`:

```
-453
-3
0
1
5
9
11
```

Per realizzare questo programma si utilizzino le funzioni create per l'esercizio 1 (e, volendo, il 2).