Esercitazione di Laboratorio del 28/05/2021: Heap

Esercizio 1

Nel file `heapify.c` implementare la definizione della seguente funzione:

```
extern Heap *HeapMinHeapify(const ElemType *v, size_t v_size);
```

La funzione `HeapMinHeapify()` prende in input un vettore di `ElemType`, `v`, e la sua dimensione, `v_size`. La funzione crea dinamicamente una min-heap contenente tutti gli elementi del vettore e ne ritorna il puntatore. Se il vettore di input è vuoto, la funzione ritorna una heap vuota.

Si testi la funzione con un opportuno `main()` di prova.

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int ElemType;

struct Heap {
    ElemType *data;
    size_t size;
};
typedef struct Heap Heap;
```

e le seguenti funzioni primitive e non:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
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);

int HeapLeft(int i);
int HeapRight(int i);
int HeapParent(int i);
Heap *HeapCreateEmpty(void);
bool HeapIsEmpty(const Heap *h);
void HeapDelete(Heap *h);
void HeapWrite(const Heap *h, FILE *f);
void HeapWriteStdout(const Heap *i);
ElemType *HeapGetNodeValue(const Heap *h, int i);
void HeapMinInsertNode(Heap *h, const ElemType *e);
void HeapMinMoveUp(Heap *h, int i);
void HeapMinMoveDown(Heap *h, int i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `minheap.h` e `minheap.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `heapify.c`

Esercizio 2

Nel file `move_up.c` implementare la definizione della funzione:

```
extern void HeapMinMoveUpRec(Heap *h, int i);
```

La funzione prende in input un min-heap e l'indice dell'**unico** nodo che viola le proprietà heap. La funzione deve spostare il nodo verso l'alto, ovvero scambiarlo con il padre, fino a quando le proprietà min-heap non sono rispettate. A differenza della primitiva `HeapMinMoveUp()`, la funzione deve essere ricorsiva.

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int ElemType;

struct Heap {
    ElemType *data;
    size_t size;
};
typedef struct Heap Heap;
```

e le seguenti funzioni primitive e non:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
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);

int HeapLeft(int i);
int HeapRight(int i);
int HeapParent(int i);
Heap *HeapCreateEmpty(void);
bool HeapIsEmpty(const Heap *h);
void HeapDelete(Heap *h);
void HeapWrite(const Heap *h, FILE *f);
void HeapWriteStdout(const Heap *i);
ElemType *HeapGetNodeValue(const Heap *h, int i);
void HeapMinInsertNode(Heap *h, const ElemType *e);
void HeapMinMoveUp(Heap *h, int i);
void HeapMinMoveDown(Heap *h, int i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `minheap.h` e `minheap.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `move_up.c`

Esercizio 3

Nel file `move_down.c` implementare la definizione della seguente funzione:

```
extern void HeapMinMoveDownRec(Heap *h, int i);
```

La funzione prende in input un min-heap e l'indice dell'**unico** nodo che viola le proprietà heap. La funzione deve spostare il nodo verso il basso, ovvero scambiarlo con il figlio minore, fino a quando le proprietà min-heap non sono rispettate. A differenza della primitiva `HeapMinMoveDown()`, la funzione deve essere ricorsiva.

Si testi la funzione con un opportuno main() di prova.

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int ElemType;

struct Heap {
    ElemType *data;
    size_t size;
};
typedef struct Heap Heap;
```

e le seguenti funzioni primitive e non:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
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);

int HeapLeft(int i);
int HeapRight(int i);
int HeapParent(int i);
Heap *HeapCreateEmpty(void);
bool HeapIsEmpty(const Heap *h);
void HeapDelete(Heap *h);
void HeapWrite(const Heap *h, FILE *f);
void HeapWriteStdout(const Heap *i);
ElemType *HeapGetNodeValue(const Heap *h, int i);
void HeapMinInsertNode(Heap *h, const ElemType *e);
void HeapMinMoveUp(Heap *h, int i);
void HeapMinMoveDown(Heap *h, int i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `minheap.h` e `minheap.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `move_down.c`

Esercizio 4

Nel file `pop.c` implementare la definizione della seguente funzione:

```
extern bool HeapMinPop(Heap *h, ElemType *e);
```

La funzione prende in input un min-heap ed estrae l'elemento minimo dallo heap, deallocando opportunamente la memoria e assicurandosi che le proprietà min-heap siano rispettate al termine dell'operazione.

La funzione ritorna `false` se l'heap è vuoto, `true` altrimenti. La funzione salva in `e` l'elemento estratto.

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int ElemType;

struct Heap {
    ElemType *data;
    size_t size;
};
typedef struct Heap Heap;
```

e le seguenti funzioni primitive e non:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
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);

int HeapLeft(int i);
int HeapRight(int i);
int HeapParent(int i);
Heap *HeapCreateEmpty(void);
bool HeapIsEmpty(const Heap *h);
void HeapDelete(Heap *h);
void HeapWrite(const Heap *h, FILE *f);
void HeapWriteStdout(const Heap *i);
ElemType *HeapGetNodeValue(const Heap *h, int i);
void HeapMinInsertNode(Heap *h, const ElemType *e);
void HeapMinMoveUp(Heap *h, int i);
void HeapMinMoveDown(Heap *h, int i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `minheap.h` e `minheap.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `pop.c`

Esercizio 5

Nel file `heapsort.c` implementare la definizione della seguente funzione:

```
extern void HeapMinHeapsort(Heap *h);
```

La funzione prende in input un min-heap e lo trasforma in modo tale che al termine dell'esecuzione l'array dei dati sia ordinato in senso decrescente. La funzione deve sfruttare le proprietà dell'heap e non deve fare uso di altri algoritmi di ordinamento.

Si testi la funzione con un opportuno `main()` di prova. Al termine dell'esecuzione l'heap rispetta ancora le proprietà?

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int ElemType;

struct Heap {
    ElemType *data;
    size_t size;
};
typedef struct Heap Heap;
```

e le seguenti funzioni primitive e non:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
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);

int HeapLeft(int i);
int HeapRight(int i);
int HeapParent(int i);
Heap *HeapCreateEmpty(void);
bool HeapIsEmpty(const Heap *h);
void HeapDelete(Heap *h);
void HeapWrite(const Heap *h, FILE *f);
void HeapWriteStdout(const Heap *i);
ElemType *HeapGetNodeValue(const Heap *h, int i);
void HeapMinInsertNode(Heap *h, const ElemType *e);
void HeapMinMoveUp(Heap *h, int i);
void HeapMinMoveDown(Heap *h, int i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `minheap.h` e `minheap.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `heapsort.c`

Esercizio 6

Si modifichino le primitive min-heap in modo tale che diventino primitive per la struttura dati max-heap. Si risolvano quindi gli esercizi precedenti facendo riferimento alle nuove primitive:

- `Heapify()` deve costruire un max-heap,
- `Pop()` deve estrarre l'elemento massimo dall'heap
- `Heapsort()` deve ordinare il vettore di dati in senso crescente.

N.B. Per questo esercizio non sono disponibili test su OLJ.