Structuri de Date
Laboratorul 1: Mulțimi ordonate

Tudor Berariu

22 februarie 2016

1. Introducere

Scopul acestui laborator îl reprezintă implementarea unei structuri de date simple, **mulțimea ordonată**, precum și reîmprospătarea unor noțiuni de programare în limbajul C: structuri, pointeri și alocarea memoriei.

2. Mulțimi ordonate

O **mulțime ordonată** este o structură de date ce reține elemente de un anumit tip, **fără duplicate**. Structura de date gestionează memoria în așa fel încât la finalul fiecărei operații, elementele vor fi **ordonate în memorie**.

Deoarece vom folosi C, pentru a obține o implementare generică vom presupune un tip de date `T` al elementelor definit anterior.

Cerința 1 Citiți definiția tipului `OrderedSet` și funcția `createOrderedSet` din fișierul `OrderedSet.h`. Lămuriți cu asistentul orice aspect neclar.

```c
typedef struct OrderedSet {
  T* elements;           // where the elemets are in memory
  long size;             // the number of elements in the set
  long capacity;         // current capacity (allocated memory)
} OrderedSet;

OrderedSet* createOrderedSet(long initialCapacity) {
  OrderedSet* newSet = (OrderedSet*) malloc(sizeof(OrderedSet));
  newSet->size = 0;
  newSet->capacity = initialCapacity;
  newSet->elements = (T*) malloc(initialCapacity * sizeof(T));
  return newSet;
}
```

3. Operații de bază

Pentru a lucra cu structura de date deja definită se vor implementa două operații de bază: (1) verificarea faptului că un element aparține unei mulțimi ordonate; (2) adăugarea unui element într-o mulțime ordonată.

Desigur, ar fi utile și alte operații: verificarea dacă o mulțime ordonată este vidă, ștergerea unui element, etc., dar din motive de timp se vor implementa doar cele două operații cerute mai sus.

Cerința 2 Implementați funcțiile descrise mai jos și testați-le folosind programul `charTest`:

1. Implementați funcția `contains` care primește o mulțime ordonată `set` și un element `element` și întoarce 1 dacă acel element aparține mulțimii și 0 altfel. Cum elementele sunt ordonate, puteți folosi căutare binară. Antetul funcției este dat:

   ```
   int contains(OrderedSet* set, const T element);
   ```

2. Implementați funcția `add` care primește o mulțime ordonată `set` și un element `newElement` și adaugă acel element mulțimii dacă acesta nu există (într-o mulțime ordonată nu pot exista duplicate). Înainte de a adăuga elementul se verifică dacă *mai este loc*. Dacă mulțimea este plină, atunci se dublează memoria alocată înainte de adăugarea noului element (folosind `realloc`). Antetul funcției este dat:

   ```
   void add(OrderedSet* set, const T newElement)
   ```

3. Compilați programul `charTest.c` și verificați că se construiesc corect mulțimea ordonată a literelor din cuvântul *mississippi* și mulțimea ordonată a literelor din cuvântul *small*. Rezultatul așteptat la ieșire este:

   ```
   student $ gcc -std=c99 charTest.c -o charTest
   student $ ./charTest
   mississippi: There are 4 letters: i m p s
   small: There are 4 letters: a l m s
   ```

4. Operații cu mulțimi ordonate

În cele ce urmează se vor implementa operații între mulțimi ordonate cu elemente de același tip: reuniunea și intersecția. Ambele operații vor construi o mulțime ordonată nouă (nu vor modifica structurile de date primite ca argumente).

Cerința 3

1. Implementați o funcție care primește două mulțimi ordonate `s1`, `s2` și construiește o mulțime ordonată nouă reprezentând reuniunea celor două. Antetul funcției este dat:

   ```
   OrderedSet* unionOrderedSets(OrderedSet* s1, OrderedSet* s2)
   ```

2. Implementați o funcție care primește două mulțimi ordonate `s1`, `s2` și construiește o mulțime ordonată nouă reprezentând intersecția celor două. Antetul funcției este dat:

   ```
   OrderedSet* intersectOrderedSets(OrderedSet* s1, OrderedSet* s2)
   ```

3. Recompilați programul `charTest.c` și verificați că reuniunrea și intersecția celor două mulțmi de litere sunt construite corect.

   ```
   student $ gcc -std=c99 charTest.c -o charTest
   student $ ./charTest
   mississippi: There are 4 letters: i m p s
   small: There are 4 letters: a l m s
   reunion: There are 6 letters: a i l m p s
   intersection: There are 2 letters: m s
   ```

5. Reutilizarea codului

Observați că structura de date pe care ați implementat-o gestionează elemente care aparțin unui tip `T` generic. `T` poate fi orice tip atât timp cât putem compara elementele lui (pentru ordonare).

Cerința 4 Scrieți un program `longTest.c` în care să refolosiți structura de date `OrderedSet` definită în `OrderedSet.h`, de data aceasta pentru a gestiona elemente de tipul `long`.

1. implementați o funcție care construiește mulțimea ordonată a multiplilor unui număr dat dintr-un interval;
2. implementați o funcție care afișează elementele unei mulțimi ordonate cu elemente de tip `long`;
3. construiți și afișați mulțimea multiplilor lui 3 din intervalul $[4, 25]$;
4. construiți și afișați mulțimea multiplilor lui 4 din intervalul $[5, 30]$;
5. construiți și afișați intersecția și reuniunea celor două mulțimi.

Un posibil exemplu al ieșirii programului `longTest`:

```
student $ gcc -std=c9x longTest.c -o longTest
student $ ./longTest
There are 7 elements: 6 9 12 15 18 21 24
There are 6 elements: 8 12 16 20 24 28
There are 11 elements: 6 8 9 12 15 16 18 20 21 24 28
There are 2 elements: 12 24
```

6. Fișierul `Makefile`

Cerința 5 Scrieți un fișier `Makefile` care să permită următoarele comenzi:

- `make build` pentru recompilarea (dacă este nevoie) a programelor `charTest` și `longTest`;
- `make clean` pentru ștergerea celor două executabile (dacă acestea există)

Atenție: modificarea fișierului `OrderedSet.h` va duce la recompilarea ambelor programe. Un fișier `Makefile` scris corect ar trebui să producă următorul comportament:

```
student $ make clean
rm -f charTest longTest
student $ make build
gcc -std=c9x charTest.c -o charTest
gcc -std=c9x longTest.c -o longTest
student $ make build
make: Nothing to be done for ‘build‘.
student $ touch charTest.c
student $ make build
gcc -std=c9x charTest.c -o charTest
student $ touch OrderedSet.h
student $ make build
gcc -std=c9x charTest.c -o charTest
gcc -std=c9x longTest.c -o longTest
```