Această lucrare reprezintă varianta tipărită a unei părţi din informaţiile puse la dispoziţia studenţilor pe site-ul disciplinei Structuri de Date, predată la anul I CB, facultatea de Automatică şi Calculatoare, Universitatea Politehnica din Bucureşti.

Recomandăm studenţilor să implementeze toţi algoritmii prezentaţi în lucrare.

Autorii

IRINA GEORGIANA MOCANU, EUGENIA KALISZ

STRUCTURI DE DATE

- variante de implementare în C -

EDITURA UNIVERSITARĂ
Bucureşti - 2012

Redactor: Gheorghe Iovan
Tehnoredactor: Eugenia Kalisz
Coperta: Angelica Mălăescu

Editură recunoscută de Consiliul Național al Cercetării Științifice (C.N.C.S.)

Descrierea CIP a Bibliotecii Naționale a României
MOCANU, IRINA GEORGIANA
Structuri de date : variante de implementare în C /
Irina Georgiana Mocanu, Eugenia Kalisz. - București : Editura Universitară, 2012
Bibliogr.
ISBN 978-606-591-397-4

I. Kalisz, Eugenia

004.43 C

DOI: (Digital Object Identifier): 10.5682/9786065913974

Editura Universitară
Director: Vasile Muscalu
B-dul. N. Bălcescu nr. 27-33, Sector 1, București
Tel.: 021 – 315.32.47 / 319.67.27
www.editurauniversitara.ro
e-mail: redactia@editurauniversitara.ro

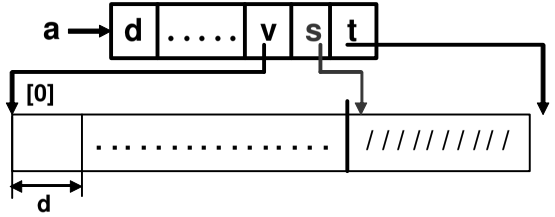
Distribuție: tel.: 021-315.32.47 /319.67.27 / 0744 EDITOR / 07217 CARTE
comenzi@editurauniversitara.ro
O.P. 15, C.P. 35, București
www.editurauniversitara.ro

CUPRINS

Mulţimi generice

Reprezentarea tipului mulțime:

- **`d`** – dimensiunea elementelor
- **`fid`**– funcția ce verifică identitatea
- **`ord`**– funcția de ordonare (NULL dacă vectorul nu este sortat)
- **`v`** – adresa vectorului de elemente,
- **`s`** – adresa sfârșitului zonei utile (sfârșitul ultimului element)
- **`t`** – adresa sfârșitului spațiului disponibil.



Cazuri particulare: (**`a->s == a->p)`** - mulțime vidă, (**`a->s == a->t)`** - mulțime completă

Operaţii de bază

Constructori - au ca rezultat o colecţie nouă sau modifică o colecţie existentă	**Operaţii de caracterizare** - furnizează informaţii despre o colecţie
➢ iniţializare () → colecţie ➢ adăugare (element, colecţie) → colecţie ➢ eliminare (element, colecţie) → colecţie ➢ reuniune (colecţie, colecţie) → colecţie ➢ intersecţie (colecţie, colecţie) → colecţie ➢ diferenţă (colecţie, colecţie) → colecţie	➢ cardinal (colecţie) → întreg ➢ apartenenţă (element, colecţie) → 1 / 0 (da / nu) ➢ identice (colecţie, colecţie) → 1 / 0 ➢ include (colecţie, colecţie) → 1 / 0 ➢ disjuncte(colecţie, colecţie) → 1 / 0

```
/*-- multimeV.h --*/
/*-- Multimi generice (elemente de orice tip) memorate ca vectori --*/

#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>

#ifndef  _TMULTIME_
#define  _TMULTIME_

typedef int(*TFComp)(const void *, const void *);

typedef struct
{ size_t d;         /* dimensiune elemente */
  TFComp fid;       /* verifică identitatea elementelor */
  TFComp ord;       /* functia de ordonare sau NULL */
  void *v, *s, *t; /* adrese vector, sfarsit zona utila, disponibila */
} TMultime;

#define Cardinal(m) (((char*)((m)->s)-(char*)((m)->v))/(m)->d)
```

```
/*--- functii de initializare ---*/
TMultime *InitD(size_t n, size_t d, TFComp fid, TFComp ord);
     /* creeaza multime, alocand dinamic spatiu pentru descriptor
        si n elemente de dimensiune d; intoarce adr.multime sau NULL */
void InitS(TMultime *m, void *v, size_t n, size_t d,
           TFComp fid, TFComp ord);
     /* initializeaza multimea m, cu maxim n elemente de dimensiune d,
        memorate in vectorul v, deja alocat static sau dinamic */

/*--- operatii asupra multimilor NESORTATE ---*/

int Adauga(void *nou, TMultime *m);
     /* obiectiv: adaugarea unui nou element la sfarsitul vectorului;
        intoarce 1/0/-1 - succes/exista deja/lipsa spatiu */
int Elimina(void *x, TMultime *m);
     /* intoarce 1/0 cu semnificatia eliminat / nu exista */
void* Loc(void *x, TMultime *m);
     /* intoarce adresa elementului cautat sau NULL */

/*-- functii cu rezultat 1/0 cu semnificatia adevarat/fals*/
#define Apartine(x,m) ((Loc(x,m)!=NULL))
int Identice(TMultime *m1, TMultime *m2);
int Include(TMultime *m1, TMultime *m2);
```

```
/*-- spatiul pentru multimea rezultat alocat in prealabil, la adresa r;
     rezultat intors = cardinalul multimii rezultat (>= 0) */
int Reuniune(TMultime *m1, TMultime *m2, TMultime *r);
int Diferenta(TMultime *m1, TMultime *m2, TMultime *r);
int Intersectie(TMultime *m1, TMultime *m2, TMultime *r);

/*--- operatii asupra multimilor SORTATE ---*/

int Inserare(void *nou, TMultime *m);
     /* obiectiv: inserarea noului element in vectorul ordonat;
        intoarce 1/0/-1 - succes/exista deja/lipsa spatiu */
int ElimO(void *x, TMultime *m);
     /* intoarce 1/0 cu semnificatia eliminat / nu exista */
void* LocO(void *x, TMultime *m);
     /* cautare secventiala, cu oprire la elem cautat sau la succesor
        sau la sfarsit; intoarce adresa elementului gasit sau NULL */
void *CautBin(void *x, TMultime *m, int *r);
     /* cautare binara in vector sortat; daca elementul cautat exista,
        atunci intoarce adresa si 1 (la adresa r),
        altfel intoarce adresa primului succesor si 0 */
/*-- functii cu rezultat 1/0 cu semnificatia adevarat/fals*/
int IdenticeO(TMultime *m1, TMultime *m2);
int IncludeO(TMultime *m1, TMultime *m2);
```

```
/*-- spatiul pentru multimea rezultat alocat in prealabil, la adresa r;
     rezultat intors = cardinalul multimii rezultat (>= 0) */
int ReuniuneO(TMultime *m1, TMultime *m2, TMultime *r);
int DiferentaO(TMultime *m1, TMultime *m2, TMultime *r);
int IntersectieO(TMultime *m1, TMultime *m2, TMultime *r);

/*--- functii auxiliare ---*/
void *DeplDr(void *a, size_t dim, size_t d);
      /* deplaseaza cu d octeti la dreapta un pachet de dim octeti */
void DeplSt(void *a, size_t dim, size_t d);
      /* deplaseaza cu d octeti la stanga un pachet de dim octeti */
void Copie(void *dest, void *sursa, size_t n);
      /* copiaza la destinatie n octeti de la sursa */
void Invers(void *a, void *b, size_t n);
      /* inversearza n octeti intre a si b */
#endif

/*-- declaratii necesare pentru generarea de valori aleatoare --*/
#ifndef randomize
#include <time.h>
#define random(num)    (rand() % (num))
#define randomize()    srand((unsigned)time(NULL))
#endif
```

Exemplu de utilizare în cazul mulțimilor de întregi nesortați

```
/*-- test-mInt.c --*/
#include "multimeV.h"

void afiI(TMultime *m)  /*-- functie de afisare multime --*/
{ int *x = (int*)(m->v), n = Cardinal(m), i = 0;
  printf("[");
  for( ; i < n; i++) printf(" %i,", x[i]);
  printf("\b ]\n\n");
}

int compI(const void *a, const void *b) /*-- functie de comparare --*/
{ return *(int*)a - *(int*)b; }

int main()
{ int v[10] = {-1, 23, 4, 6, 3, 10}, z[15] = {0};
  TMultime m1 = {sizeof(int),compI,NULL,v,v+6,v+10}, *a = &m1,
           m2, *b = &m2,
           m3, *c = &m3;
  int v1 = 0, v2 = 6, v3 = 11, rez, i;
  int *pr;
```

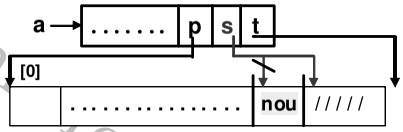
```
/*- afisare multime, test apartenenta si localizare -*/
  afiI(a);
  rez = Apartine(&v1, a);
  printf("elementul %i %sapartine multimii\n", v1, rez? "" : "nu ");
  pr = (int*)Loc(&v2, a);
  if(!pr)
    printf("elementul %i nu apartine multimii\n\n", v2);
  else
    printf("elementul %i are adresa %p si indice %i\n\n",v2,pr,pr-v);

/*- adaugare 2 elemente: unul care nu exista si altul care exista -*/
  rez = Adauga(&v1, a);
  printf("%i %s\n", v1, rez? "adaugat" : "exista deja");
  rez = Adauga(&v2, a);
  printf("%i %s\n\n", v2, rez? "adaugat" : "exista deja");
  afiI(a);

/* - eliminare 2 elemente - unul exista, celalat nu -*/
  rez = Elimina(&v3, a);
  printf("%i %s\n", v3, rez? "eliminat" : "nu exista");
  rez = Elimina(&v2, a);
  printf("%i %s\n\n", v2, rez? "eliminat" : "nu exista");
  printf("a: "); afiI(a);
```

```
/*- initializare multime alocata static cu valori aleatoare -*/
  InitS(b, z, 15, sizeof(int),compI,NULL);
  randomize();
  for(i = 10; i > 0; i--) { rez = random(15); Adauga(&rez, b); }
                           printf("b: "); afiI(b);
  printf("Cardinal(b) = %i\n", Cardinal(b));

/*- initializare multime c vida, alocata dinamic; c = Reuniune(a,b) -*/
  rez = InitD(&m3,30,sizeof(int),compI,NULL);
  if(!rez)
  { printf("Initializate dinamica esuata\n");
    getch(); return 1;
  }
  rez = Reuniune(a, b, c);
  printf("\nc = reuniune(a,b): "); afiI(c);
  printf("Cardinal(c) = %i\n", Cardinal(c));

  getch(); return 0;
}
```

Implementarea unor operaţii de bază

```
TMultime *InitD(size_t n, size_t d, TFComp fid, TFComp ord)
{ TMultime *m = (TMultime*)calloc(1,sizeof(TMultime));
  if(!m) return NULL;                          /* alocare esuata */
  m->v = calloc(n, d);
  if(!m->v) { free(m); return NULL;}           /* alocare esuata */
  m->d = d;
  m->s = m->v, m->t = m->v + d * n;
  m->fid = fid, m->ord = ord;
  return m;                                    /* initializare reusita */
}
```

Operaţii asupra mulţimilor nesortate

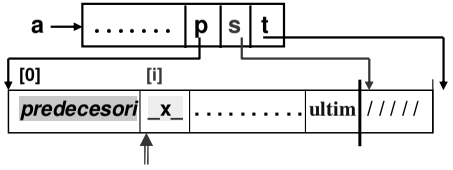



```
int Adauga(void *nou, TMultime *m)
{ char* dest = (char*)(m->s), *x = (char*)nou;
  if(Apartine(nou,m)) return 0; /* nou exista */
  m->s = (char*)(m->s) + m->d;  /* nu exista -> adauga la sfarsit m */
  for(; dest < (char*)m->s; dest++, x++) *dest = *x;
  return 1;
}
```

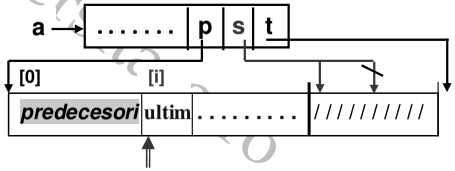
```
void* Loc(void *x, TMultime *m)
{ void *p = m->v;
  for(; p < m->s; p += m->d) if(m->fid(p, x) == 0) return p;
  return NULL;
}

int Include(TMultime *m1, TMultime *m2)
{ void *p1, *p2;
  for(p2 = m2->v; p2 < m2->s; p2 += m2->d)    /* pentru fiecare e2 din m2 */
    if(!Apartine(p2,m1)) return 0;            /* e2 nu a fost gasit in m1 */
  return 1;                                   /* toate elementele din m2 exista in m1 */
}
```


Implementați funcția de eliminare a unui element, care se realizează în două etape:

1. localizare (căutare)



2. eliminare efectivă, dacă a fost găsit



Ipoteză: nu este necesară păstrarea ordinii relative a elementelor. Numai în acest caz elementul eliminat poate fi înlocuit cu ultimul.

```
int Reuniune(TMultime *m1, TMultime *m2, TMultime *r)
   /* Preconditii: (m1->d == m2->d) && (m1->fid == m2->fid)*/
{ void *x, *p1 = m1->v, *p2 = m2->v;
  /* actualizeaza r si copiaza m1 in r */
  r->d = m1->d, r->fid = m1->fid, r->ord = m1->ord, r->s = r->v;
  for(x = p1; x < m1->s; x++, (r->s)++)
    *(char*)(r->s) = *(char*)x;
  for(; p2 < m2->s; p2 += m2->d) /* pentru fiecare element e2 */
  { if(Apartine(p2,m1))continue; /* daca exista in m1 continua parcurgere */
    Copie(r->s, p2, r->d);      /* nu exista -> copiaza in r */
    r->s += r->d;               /* si avans in r */
  }
  return (r->s - r->v)/r->d;    /* intoarce cardinal reuniune */
}
```

Exemple:

```
a: [ -1, 23, 4, 6, 3, 10]
b: [ 5, 0, 3, 1, 4, 7]
c = a U b: [ -1, 23, 4, 6, 3, 10, 5, 0, 1, 7]

d: [7, 0, 1]
e = d U b: [ 7, 0, 1, 5, 3, 4]
```

Operaţii asupra mulţimilor sortate

În cazul acestor operații trebuie să se țină seama de faptul că funcțiile de comparare și ordonare pot fi diferite - două elemente similare din punctul de vedere al criteriului de sortare pot să nu fie identice. De exemplu, în cazul unei mulțimi de produse sortate după preț, pot exista mai multe produse cu același preț, dar unul singur dintre ele este cel căutat.

Din acest motiv, în cazul în care funcțiile de ordonare și comparare sunt diferite, nu se pot aplica algoritmii specifici mulțimilor ordonate, ci trebuie apelate funcțiile care implementează operațiile pentru mulțimi nesortate.

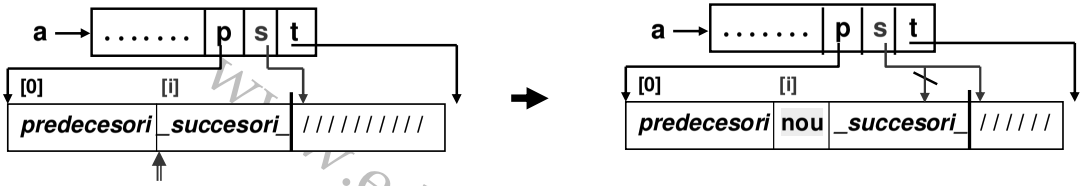
Următoarea implementare a funcției de **căutare secvențială** tratează corect aceste situații.


```
void* LocO(void *x, TMultime *m)
{ void *p = m->v;
  int r;
  if(m->ord != m->fid) return Loc(x,m);  /* similar != identic */
  for(; p < m->s; p += m->d)        /* pentru fiecare element din mutime */
  { r = m->ord(p, x);               /*   verifica relatia de ordine      */
    if(r == 0) return p;            /*   p identic cu elementul cautat   */
    if(r > 0) break;                /*   p succesor -> oprire cautare    */
  }
  return NULL;                      /* elementul cautat nu exista in m   */
}
```

```
Algoritm Căutare_binară(x, m)
{ inițializări inf,sup - limite zonă de căutare în m;
  cât timp zonă de căutare nevidă
  { determină y = elementul de la mijlocul zonei de cautare;
    dacă  elementele x si y sunt identice atunci ➔ succes
      altfel dacă x predecesor y
               atunci sup = y; /* cautare in jumatatea stanga  */
               altfel inf = y; /* cautare in jumatatea dreapta */
  }
  ➔ eșec
}

int IdenticeO(TMultime *m1, TMultime *m2)
{ void *p1 = m1->v, *p2 = m2->v;
  if(m1->ord != m1->fid) return Identice(m1,m2);/* test criteriu ordonare */
  if(Card(m1) != Card(m2)) return 0;            /* test acelasi cardinal  */
  for(; p1 < m1->s ; p1 += m1->d, p2 += m2->d)  /* parcurge multimile     */
    if(m1->fid(p1, p2) != 0) return 0;          /* elemente diferite -> 0 */
  return 1;              /* toate elementele identice -> multimi identice */
}
```

Observație. Deoarece mulțimile au același cardinal, testul de sfârșit parcurgere se face pentru una singură.

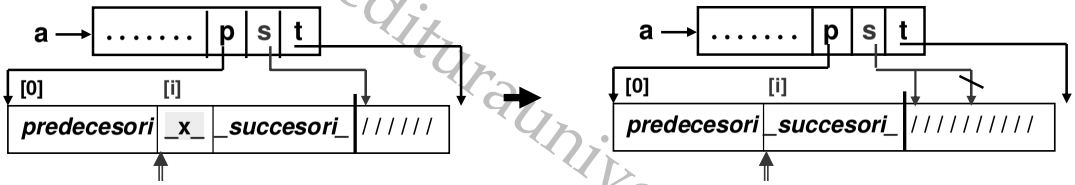
Inserarea se realizează în interiorul vectorului, între predecesori și succesori. Inserarea la sfârșit este doar un caz particular.




```
int Inserare(void *nou, TMultime *m)
{ void *p = m->v, *sp;
  int r;
  if(m->ord != m->fid) return Adauga(nou,m); /* test criteriu ordonare */
  for(; p < m->s; p += m->d)    /* pentru fiecare element din multime  */
  { r = m->ord(p, nou);            /* verifica relatia de ordine       */
    if(r < 0) continue;            /* p predecesor -> continua cautare */
    if(r == 0) return 0;           /* nou exista deja in m -> gata */
    /* p succesor -> nou va fi inserat  */
    if(PlinaM(m)) return -1;       /* multime plina -> inserare esuata */
    DeplDr(p, m->s - p, m->d);     /* deplasare dreapta succesori nou  */
    break;
  }
```

```
  /* copiaza nou la adresa p (la sfarsit sau inaintea succesorilor) */
  Copie(p, nou, m->d);
  m->s += m->d;                          /* actualizeaza sfarsit m */
  return 1;
}
```

Implementați funcția de **eliminare** din mulțime sortată.




```
/*--- functii auxiliare ---*/

void *DeplDr(void *a, size_t dim, size_t d)
      /* deplaseaza cu d octeti la dreapta un pachet de dim octeti */
{ char* sursa = (char*)a + dim - 1, *dest = sursa + d;
  while(sursa >= (char*)a) *(dest--) = *(sursa--);
  return ++dest;
}
```

```
void DeplSt(void *a, size_t dim, size_t d)
      /* deplaseaza cu d octeti la stanga un pachet de dim octeti */
{ char* sursa = (char*)a, *dest = sursa - d, *sf = sursa + dim;
  while(sursa < sf) *(dest++) = *(sursa++);
}

void Copie(void *dest, void *sursa , size_t n)
      /* copiaza la destinatie n octeti de la sursa */
{ void *sf = sursa + n;
  for(; sursa < sf; sursa++, dest++)
    *(char*)dest = *(char*)sursa;
}

void Invers(void *a, void *b, size_t n)
      /* inversearza n octeti intre a si b */
{ void *sf = a + n;
  char temp;                  /* variabila tampon */
  for(; a < sf; a++, b++)
  { temp = *(char*)a;
    *(char*)a = *(char*)b;
    *(char*)b = temp;
  }
}
```