

Dezvoltarea unei aplicații care utilizează liste

Șl. Dr. Ing. Șerban Radu

Departamentul de Calculatoare

Facultatea de Automatică și Calculatoare

Cerințele aplicației

- Să se implementeze un program pentru managementul candidaților care se înscriu la un concurs de admitere cu 3 probe
- Fiecare candidat se identifică prin:
 - ☐ Numărul curent de înscriere
 - ☐ Numele și prenumele
 - ☐ Numărul legitimației
 - ☐ Notele la cele 3 probe și media lor

Tabel cu lista candidaților

Nr.	Nume și prenume	Nr. Legitimație	Nota 1	Nota 2	Nota 3	Medie
1	Popescu Adrian	120	9	7	9	8.33
2	Ionescu Victor	354	10	6	5	7.00
3	Georgescu Maria	872	8	5	3	5.33

Cerințele aplicației

- Conținutul listei se poate modifica prin:
 - Adăugarea de elemente noi la sfârșitul listei - înscrierea de candidați poate fi reprezentată cronologic, caz în care fiecare candidat nou înscris este adăugat la sfârșitul listei
 - Inserarea de elemente noi în orice loc din listă – dacă lista este păstrată în ordine alfabetică, înscrierea unui nou candidat duce la reorganizarea completă a listei

Cerințele aplicației

- ☐ Ștergerea de elemente din orice poziție a listei – pot exista candidați care renunță
- ☐ Modificarea unui element dintr-o poziție dată – este posibilă corectarea informațiilor introduse greșit
- ☒ Printre operațiile care modifică structura listei este și inițializarea unei liste ca o listă vidă – la începutul înscrierilor, lista candidaților este o listă vidă



Operații de caracterizare a listei

- Acestea nu modifică structura listelor doar furnizează informații despre acestea:
 - ☐ Determinarea lungimii listei
 - ☐ Localizarea elementului din listă care îndeplinește o anumită condiție

Operații primitive

- Mulțimea de operații primitive utilizează poziția curentă în listă, definită ca poziția elementului accesibil la un moment dat
- Operațiile de prelucrare se referă la elementul din poziția curentă
- De asemenea, sunt necesare operații care asigură modificarea poziției curente

Operații de modificare a poziției curente

- PoziționareÎnceput – poziționare pe primul element din listă
- PoziționareSfârșit – poziționare pe ultimul element din listă
- Următor – poziționare pe succesorul elementului curent
- Precedent – poziționare pe predecesorul elementului curent

Operații de modificare a conținutului listei

- AdaugăLaSfârșit – adăugarea unui element la sfârșitul listei
- InsereazăElement – inserare înaintea elementului curent
- ModificăInformație – modificarea elementului curent
- ȘtergeElement - ștergerea elementului curent din listă




Alte operații

- InițializareListă – inițializarea unei liste ca o listă vidă
- AdresăInformație – copierea adresei informațiilor din elementul curent

Alte operații

■ Operații de caracterizare

- TestListăVidă – testarea unei liste dacă este vidă
- TestÎnceput – testul poziționării pe primul element
- TestSfârșit – testul poziționării pe ultimul element
- LungimeListă – determinarea lungimii listei



```
#define bool int
```

```
#define True 1
```

```
#define False 0
```

```
#define NULL 0
```

```
struct celula {  
    void *pTipElement;  
    struct celula *prec;  
    struct celula *urm;  
};
```



```
struct LISTA {
```

```
    int lungime;
```

```
    struct celula *inceput;
```

```
    struct celula *curent;
```

```
    struct celula *sfarsit;
```


```
};
```

```
bool InitializareLista(struct LISTA *pL);
```

```
bool TestListaVida(struct LISTA *pL);
```

```
bool PozitionareInceput(struct LISTA *pL);
```

```
bool PozitionareSfarsit(struct LISTA *pL);
```



```
bool TestInceput(struct LISTA *pL);  
bool TestSfarsit(struct LISTA *pL);  
bool Urmator(struct LISTA *pL);  
bool Precedent(struct LISTA *pL);  
bool AdaugaLaSfarsit(struct LISTA *pL, void *x);  
bool AdresaInformatie(struct LISTA *pL, void **x);  
bool ModificaInformatie(struct LISTA *pL, void *x);  
bool StergeElement(struct LISTA *pL);  
bool InsereazaElement(struct LISTA *pL, void *x);  
int LungimeLista(struct Lista *pL);
```

Reprezentarea listelor în memorie

- Structura LISTA este reprezentată prin lungimea ei și prin trei pointeri:
 - La începutul listei
 - La sfârșitul listei
 - La elementul curent din listă

Reprezentarea listelor din memorie

- Pentru ca operațiile de inserarea și ștergere să se facă la fel pentru orice element din listă, se folosesc două elemente santinelă, plasate la începutul și la sfârșitul listei
- În acest fel toate elementele utile din listă au atât predecesor, cât și succesor

Implementarea operațiilor primitive

- Acestea sunt definite ca funcții booleene, care întorc rezultatul adevărat dacă operația se desfășoară normal și fals, în caz că se va produce o eroare
- Celelalte rezultate sunt transmise prin lista de parametrii




Implementarea operațiilor primitive

- Operațiile primitive sunt grupate într-un fișier header
- Operațiile definite se referă la liste cu orice fel de elemente
- Acest lucru este posibil datorită utilizării pointerilor generici către celule



Implementarea operațiilor primitive

- În aplicațiile cu liste, tipul informației din celule trebuie să fie precizat
- Aceasta cere programatorului să transforme referințele la valori de tip nespecificat în referințe la date de tip precizat, folosind conversia de tip



```
#include <stdlib.h>
```

```
#include "listed.h"
```

```
/*inializzare lista vida*/
```


```
bool InializzareLista(struct LISTA *pL) {
```

```
    pL->inceput = (struct celula *)malloc(sizeof(struct celula));
```

```
    if (!(pL->inceput)) return False;
```

```
    pL->sfarsit = (struct celula *) malloc (sizeof(struct celula));
```

```
    if (!(pL->sfarsit)) return False;
```



```
pL->incept->urm = pL->sfarsit;
```

```
pL->sfarsit->prec = pL->incept;
```

```
pL->incept->prec = NULL;
```

```
pL->sfarsit->urm = NULL;
```

```
pL->lunime = 0;
```

```
pL->curent = pL->sfarsit;
```


```
return True;
```

```
}
```

```
bool TestListaVida(struct LISTA *pL) {  
    return pL->lungime == 0;  
}
```

```
bool PozitionareInceput(struct LISTA *pL) {  
    if (TestListaVida(pL)) return False;  
    else {  
        pL->curent = pL->inceput->urm;  
        return True;  
    }  
}
```

```
bool PozitionareSfarsit(struct LISTA *pL) {  
    if (TestListaVida(pL)) return False;  
    else {  
        pL->curent = pL->sfarsit->prec;  
        return True;  
    }  
}
```



```
bool TestInceput(struct LISTA *pL) {  
    return pL->inceput->urm == pL->curent;  
}
```


```
bool TestSfarsit(struct LISTA *pL) {  
    return pL->curent == pL->sfarsit;  
}
```




```
bool Urmator(struct LISTA *pL) {  
    if (TestListaVida(pL)) return False;  
    else {  
        if (pL->curent != pL->sfarsit) {  
            pL->curent = pL->curent->urm;  
            return True;  
        }  
        else return False;  
    }  
}
```

```
bool Precedent(struct LISTA *pL) {  
    if (TestListaVida(pL)) return False;  
    else {  
        if (pL->curent != pL->inceput) {  
            pL->curent = pL->curent->prec;  
            return True;  
        }  
        else return False;  
    }  
}
```

```
bool AdaugaLaSfarsit(struct LISTA *pL, void *x) {  
    struct celula *p;  
    p = (struct celula *) malloc (sizeof(struct celula));  
    if (!p) return False;  
    else { p->prec = pL->sfarsit->prec;  
        p->urm = pL->sfarsit; p->pTipElement = x;  
        pL->sfarsit->prec->urm = p;  
        pL->sfarsit->prec = p;  
        pL->lungime += 1;  
        return True; } }
```



```
bool AdresaInformatie(struct LISTA *pL, void **x) {  
    if (TestListaVida(pL)) return False;  
    else {  
        *x = pL->curent->pTipElement;  
        return True;  
    }  
}
```



```
bool ModificaInformatie(struct LISTA *pL, void *x) {  
    if (TestListaVida(pL)) return False;  
    else {  
        pL->curent->pTipElement = x;  
        return True;  
    }  
}
```

```
bool StergeElement(struct LISTA *pL) {  
    struct celula *p;  
    if (TestListaVida(pL)) return False;  
    else {    p = pL->curent;  
        p->prec->urm = p->urm;    p->urm->prec = p->prec;  
        if (p->urm == pL->sfarsit) pL->curent = p->prec;  
        else pL->curent = p->urm;  
        free (p->pTipElement);    free(p);  
        pL->lungime -= 1;  
        return True;    }    }
```

```
bool InsereazaElement(struct LISTA *pL, void *x) {  
    struct celula *p;  
    p = (struct celula *) malloc (sizeof(struct celula));  
    if (!p) return False;  
    p->pTipElement = x;    p->urm = pL->curent;  
    p->prec = pL->curent->prec;    pL->curent->prec->urm = p;  
    pL->curent->prec = p;    pL->curent = p;  
    pL->lungime += 1;  
    return True;  
}
```

```
int LungimeLista(struct LISTA *pL) {  
    return pL->lungime;  
}
```




Operațiunile specifice aplicației

- Înscrierea candidaților
- Modificarea informațiilor referitoare la un candidat
- Retragera unui candidat înscris
- Înregistrarea notelor la probele de concurs și calculul mediei fiecărui candidat
- Crearea unei liste de candidați ordonată după medie
- Afișarea rezultatelor finale



Lista de candidați

- Fiecare element al listei, reprezentând un candidat, conține următoarele informații:
 - Numele candidatului
 - Numărul legitimației de concurs
 - Notele obținute la probele de concurs și media lor
- Utilizatorul își selectează operația dorită dintr-un meniu de comenzi

Descrierea aplicației

- Repetă
 - Afișare meniu
 - Selecție și validare comandă
 - Alege comandă dintre
 - Înscriere – execută Funcție_Înscriere
 - Modificare – execută Funcție_Modificare
 - Retrageră – execută Funcție_Retrageră
 - Note – execută Funcție_Note
 - Ordonare – execută Funcție_Ordonare
 - Afișare – execută Funcție_Afișare
- Până când comandă = Opre

Descrierea aplicației

- Dispunem de o descriere generală a rezolvării problemei și cunoaștem că structura de date utilizată este o listă dublu înlănțuită cu elemente santinelă
- Pentru listă cunoaștem deja operațiile primitive
- Utilizarea lor în aplicație nu este directă – efectul acestor primitive trebuie completat cu acțiuni referitoare la informația din celulele listei

Descrierea aplicației

- Între comenzi există anumite condiționări
 - Toate operațiile (exceptând înscrierea și oprirea) presupun existența unei liste nevide de candidați
 - Ordonarea după medii și afișarea rezultatelor trebuie să fie precedate de operația de trecere a notelor
 - După trecerea notelor nu mai pot fi făcute înscrieri, modificări și nu mai pot fi trecute alte note



Descrierea aplicației

- Selecția și validarea unei comenzi citește un număr de comandă și stabilește dacă acesta reprezintă o comandă validă în contextul dat, conform condițiilor exprimate în tabel

Tabelul condiționărilor

Operația	Trecut Note	Lista Vidă
Înscriere	F	F/T
Modificare	F	F
Retragere	F	F
Note	F	F
Ordonare	T	F
Afișare	T	F
Terminare	F/T	F/T

Descrierea aplicației

- Înscrierea, retragerea sau modificarea datelor unui candidat folosesc aceeași operație și anume căutarea unui nume în lista de candidați
- Aceasta se face prin parcurgerea listei
- Operația se încheie cu succes, dacă se găsește numele căutat, caz în care elementul care conține numele devine elementul curent din listă, sau cu eșec, în caz că s-a ajuns la sfârșitul listei, fără a găsi numele

Căutarea unui nume în listă

Prezent (L, nume)

- Execută PoziționareÎnceput(L)
- găsit \leftarrow false
- Cât timp !TestSfârșit(L) și !găsit
- Extrage cheia din elementul curent
 - Dacă nume = cheia extrasă atunci găsit \leftarrow true
 - Dacă !găsit atunci avans la următorul element
- Întoarce găsit



Funcția de înscriere

- Comanda de înscriere acceptă informații (nume candidat și număr legitimație) de la tastatură
- Terminarea înscrierii este marcată printr-o linie vidă

Funcția de înscriere

- După introducerea fiecărui nume, se verifică mai întâi dacă acesta nu este deja prezent în listă, situație semnalată printr-un mesaj de atenționare și ignorarea numelui introdus
- Dacă numele nu figurează în listă, se citesc și celelalte informații relative la candidat

Funcția de înscriere

- Se creează o celulă, la care se leagă printr-un pointer informațiile introduse și se adaugă această celulă la sfârșitul listei de candidați

Funcția de înscriere

- Execută InițializareListă(Listă_candidați)
- Repetă
 - Citește nume
 - Dacă nu este terminator atunci
 - Dacă Prezent(Listă_candidați, nume) atunci scrie mesaj “Candidatul a fost înscris înainte”
 - Altfel
 - Citește număr
 - Creează o celulă și leagă la ea informația citită
 - Adaugă celula la sfârșitul listei de candidați
- Până când s-a citit terminator



Funcția de modificare

- Comanda de modificare a datelor unui candidat verifică dacă numele introdus este prezent în listă, caz în care acceptă modificările datelor introduse de la tastatură, care vor constitui informația atașată celulei curente

Funcția de modificare

- Citește nume
- Dacă `Prezent(Listă_candidați, nume)` atunci
 - Citește date modificate
 - Atașează la celula curentă informația modificată



Funcția de trecere a notelor

- Comanda de trecere a notelor parcurge lista de candidați și extrage din fiecare celulă numele și numărul candidatului
- Notele introduse și media lor actualizează informația atașată celulei curente

Funcția de trecere a notelor

- Execută PoziționareÎnceput(Listă_candidați)
- Cât timp !TestSfârșit(Listă_candidați) execută
 - Extrage nume și număr din celula curentă
 - Citește note
 - Calculează medie
 - Atașează la celula curentă informația modificată
 - Execută Următor(Listă_candidați)



Funcția de ordonare a listei

- Comanda de creare a unei liste ordonate după cheia medie presupune inserarea de celule din lista neordonată în lista ordonată, astfel încât să fie respectată relația de ordine mai mare între mediile a două celule vecine

Funcția de ordonare(L1,L2)

- Execută InițializareListă(L2)
- Execută PoziționareÎnceput(L1)
- Cât timp !TestSfârșit(L1) execută
 - Copiază elementul curent din L1
 - Caută primul element din L2 având cheia mai mică decât elementul curent
 - Dacă nu s-a găsit atunci adaugă element la sfârșitul listei L2
 - Altfel inserează element înaintea celui cu cheie mai mică din L2
 - Execută Următor(L1)



Funcția de afișare

- Vizualizarea conținutului unei liste presupune parcurgerea elementelor listei și afișarea informațiilor conținute în acestea

Funcția de afișare

- Execută PoziționareÎnceput(L)
- Cât timp !TestSfârșit(L) execută
 - Extrage informația din elementul curent
 - Afișează informația extrasă
 - Execută Următor(L)