Programozás alapjai 1. (BMEVIEEA100)

Gyakorlat anyaga az 11-12. oktatási héten

Hétfői csoportokban a feladat: láncolt listák gyakorlása, a részletek a múlt héten küldött anyagban. A **11. héten a szerdai csoportoknak elmarad az órája** a TDK miatt. A pénteki csoportoknak a feladat: bináris fák. A **12. héten a pénteki óra marad el** (nyílt nap a középiskolásoknak), a hétfői és szerdai csoportok ekkor fognak a bináris fákkal foglalkozni.

A 13. héten írják a 4. kis zh-t.

A bináris fák befejezése után adjuk házi feladatnak a Pointerek, dinamikus adatszerkezetek fejezetet a tesztkérdések közül. A 4. kis ZH-ba eddig bezárólag kerül az anyag.

Új anyag

Bináris fa építése és bontása.

A következő példa Vitéz Andrástól származik:

Tanuljuk meg a Morse ABC néhány kódját, és fejtsünk vissza egy szöveget:
S... O___ T_ E. A._ M__
... ___ . _ . . _ . ... _ .

A példa megoldása bináris fával:

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

/*********************************************************/
typedef struct dat{
/*********************************************************/
    char betu;
    struct dat *pont,*vonal;
}Morse,*MP;


/*********************************************************/
void delfa(MP e){
/*********************************************************/
    if(!e)return;
    delfa(e->pont);
    delfa(e->vonal);
    free(e);
}


/*********************************************************/
MP feltolt(){
/*********************************************************/
    int c,aktbetu=0;
    MP gyoker=NULL,futo;

    if(!(gyoker=(MP)malloc(sizeof(Morse)))){
        puts("Elfogyott a memória");
        exit(-1);
    }
    gyoker->pont=gyoker->vonal=0;
    gyoker->betu=0;
```

```
    futo=gyoker;

    printf("Kerem a definiciot!\n");
    while((c=toupper(getchar()))!=EOF){
        switch(c){
            case '.':{
                if(!(futo->pont)){
                    if(!(futo->pont=(MP)malloc(sizeof(Morse)))){
                        puts("Elfogyott a memória");
                        delfa(gyoker);
                        exit(-1);
                    }
                    futo=futo->pont;
                    futo->betu=0;
                    futo->pont=futo->vonal=0;
                }
                else futo=futo->pont;
                break;
            }
            case '_':{
                if(!(futo->vonal)){
                    if(!(futo->vonal=(MP)malloc(sizeof(Morse)))){
                        puts("Elfogyott a memória");
                        delfa(gyoker);
                        exit(-1);
                    }
                    futo=futo->vonal;
                    futo->betu=0;
                    futo->pont=futo->vonal=0;
                }
                else futo=futo->vonal;
                break;
            }
            default:{
                futo->betu=aktbetu;
                if((c>='A'&&c<='Z')||(c>='0'&&c<='9'))aktbetu=c;
                else aktbetu=0;
                futo=gyoker;
            }
        }
    }
    return gyoker;
}


/*******************************************************/
void keres(MP gyoker){
/*******************************************************/
    int c;
    MP futo=gyoker;
    int jo=1;

    printf("Kerem a megfejtendo kodot!\n");
    while((c=toupper(getchar()))!=EOF){
        switch(c){
            case '.':{
                if(jo){
                    if(futo->pont)futo=futo->pont;
                    else jo=0;
                }
                break;
            }
            case '_':{
                if(jo){
                    if(futo->vonal)futo=futo->vonal;
                    else jo=0;
                }
```

```
                break;
            }
            default:{
                if(futo->betu&&jo)putchar(futo->betu);
                else if(c==' ')putchar(' ');else putchar('?');
                futo=gyoker;
                jo=1;
            }
        }
    }
}


/*********************************************************/
void main(){
/*********************************************************/
    MP gyoker=feltolt();
    keres(gyoker);
    delfa(gyoker);
}
```

Hívjuk fel a figyelmet, hogy a fa építése nem rekurzív módon történt, sőt, ezúttal a bejárása sem! A lebontása viszont igen.

Bináris fa bejárása rekurzív függvényekkel

Hívjuk fel a figyelmet, hogy a vizsgák beugró részében nagyon gyakran fordul elő bináris, vagy kettőnél több ágat tartalmazó fát bejáró feladat.

Példa: adott az alábbi típusdefiníció, határozzuk meg, hogy hány eleme van a bináris fának!

```
/*********************************************************/
typedef struct dat{
/*********************************************************/
    double adat;
    struct dat *bal,*jobb;
}bifa,*pbifa;
```

Megoldás:

```
/*********************************************************/
int elemszam(pbifa gy){
/*********************************************************/
    if(gy==NULL)return 0;
    return elemszam(gy->bal)+elemszam(gy->jobb)+1;
}
```

Ugyanez a feladat, de nem bináris, hanem sokágú fára:

```
/*********************************************************/
typedef struct dat{
/*********************************************************/
    double adat;
    unsigned n;
    struct dat **t;
}fa,*pfa;
```

A fa ágaira a t pointertömb elemei mutatnak, a t tömb n elemű.

Megoldás:

```
/*******************************************************/
int elemszam(pfa gy){
/*******************************************************/
    int sum=0,i;
    if(gy==NULL)return 0;
    for(i=0;i<gy->n;i++)sum+=elemszam(gy->t[i]);
    return sum+1;
}
```

Bináris fa preorder, inorder és postorder bejárása

Egy fát általában valamely adat szerint (ez a kulcs) rendezve építjük fel, azaz egy új adat beszúrása esetén a gyökérelemtől balra indulunk, ha az adat kisebb nála, és jobbra, ha nagyobb. Oda szúrjuk be, ahol az ilyen bejárást követően először találunk szabad helyet. A következő bejárások esetében nem követelmény ez a felépítés, de ha így épül fel a fa, ezt a jellemzőt kihasználhatjuk.

Preorder:

```
{
    Adat feldolgozása
    Bal ág bejárása
    Jobb ág bejárása
}
```

Ha bináris fát fájlba akarunk menteni úgy, hogy eredeti formájában tudjuk visszaépíteni, ezt a bejárást alkalmazzuk. (A fa bejárása annál hatékonyabb, minél kiegyensúlyozottabb a fa, azaz lehetőség szerint minden elemtől annyi legyen balra, mint jobbra, ellenkező esetben a bejárás tovább tart.)

Inorder:

```
{
    Bal ág bejárása
    Adat feldolgozása
    Jobb ág bejárása
}
```

Ebben az esetben növekvő/csökkenő sorrendben érhetjük el az elemeket, ha rendezett a fa.

Postorder:

```
{
    Bal ág bejárása
    Jobb ág bejárása
    Adat feldolgozása
}
```

Postorder bejárást kell alkalmaznunk, amikor a fát töröljük.

Gyakran szabadon választhatunk a három bejárásmód közül.

-

Házi feladat

```
X ********************************************************************************
C Pointerek, dinamikus adatszerkezetek
X ********************************************************************************
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
2. Szintaktikailag típushelyes-e az alábbi utasítás?
**p = 25;
[ ] Nem.
[ ] Igen, amennyiben p egy egész mutatókra irányított mutató.
[ ] Igen, amennyiben p egy egész mutató.
--------------------------------------------------------------------------------
3. Melyik utasítás helyes?
[ ] double * d=(double*)calloc(100*sizeof(double));
[ ] double * d=(double*)malloc(100*sizeof(double));
[ ] double * d=(double*)malloc(100,sizeof(double));
--------------------------------------------------------------------------------
4. Mi történik, ha túl sok memóriát próbálunk foglalni a malloc függvénnyel?
[ ] Az operációs rendszer megszakítja a program futását.
[ ] NULL értéket kapunk vissza.
[ ] Annyi memóriát foglal, amennyi rendelkezésre áll.
--------------------------------------------------------------------------------
5. Adott az alábbi struktúra. Milyen dinamikus adatszerkezetet épít belőle az
adott függvény?
struct a {int b;struct a *c;struct a *d;};
struct a *ad(struct a *e, int f) {
  struct a *g = (struct a *)malloc(sizeof(struct a)), *h;
  g->b = f; g->c = g->d = NULL; if (e == NULL) return g;
  for (h = e; h->c != NULL; h = h->c);
  h->c = g; g->d = h;
  return e;
}
[ ] Egy rendezetlen, két irányban láncolt listát.
[ ] Egy rendezett, két irányban láncolt listát.
[ ] Egy bináris fát.
--------------------------------------------------------------------------------
6. Ekvivalens-e egymással a p->m és a (*p).m jelölés?
[ ] Nem.
[ ] Igen.
--------------------------------------------------------------------------------
7. Helyes-e az alábbi típusdefiníció?
struct node {
        struct node* left;
        struct node* right;
};
[ ] Nem, mert a definíció önhivatkozó (rekurzív).
[ ] Igen.
[ ] Nem, mert ebben a bináris fát megvalósító adatszerkezetben nincs a csomópont
értékét tároló tag.
--------------------------------------------------------------------------------
8. Mit csinál az alábbi függvény?
typedef struct bf { struct bf * b; struct bf * j; int a; } bifa;
int f(bifa*x){
   if(!x)
     return 0;
   else
     return f(x->j)+f(x->b)+1;
}
[ ] Megméri egy bináris fa mélységét.
[ ] Megszámolja egy bináris fa csúcspontjait.
[ ] Megszámolja egy bináris fa leveleit.
--------------------------------------------------------------------------------
9. Mit csinál az alábbi függvény?
typedef struct bf { struct bf * b; struct bf * j; int a; } bifa;
int f(bifa*x){
   if(!x)
      return 0;
   return f(x->j)+f(x->b)+(!x->b && !x->j?1:0);
}
[ ] Megméri egy bináris fa mélységét.
[ ] Megszámolja egy bináris fa csúcspontjait.
[ ] Megszámolja egy bináris fa leveleit.
--------------------------------------------------------------------------------
10. Láncolt lista esetén mit nevezünk strázsának?
[ ] Azt az érvényes értéket nem tartalmazó elemet, amellyel a lista elejét, vagy
végét jelöltük meg.
[ ] Minden olyan listaelemet, amely érvényes értéket tartalmaz.
[ ] Minden esetben a lista első elemét.
--------------------------------------------------------------------------------
11. Melyik tárolódik a heap területen?
[ ] dinamikusan allokált memória
[ ] globális változók
[ ] lokális változók
--------------------------------------------------------------------------------
12. Mit jelent ha valaminek a típusa void* ?
[ ] Ez egy általános mutató, amelyet használat előtt a megfelelő típusúra kell
konvertálni.
[ ] Ez egy olyan objektumra hivatkozó mutató, amely nem foglal memóriát.
[ ] Ez egy void típusú objektumra hivatkozó mutató.
--------------------------------------------------------------------------------
13. Melyik függvény (komponens) feladata a free() meghívása?
[ ] Mindegy, csak valamelyik hívja meg.
[ ] Senkié.
[ ] Amelyik a malloc()-ot hívta.
--------------------------------------------------------------------------------
14. Mit ír ki az alábbi program?
#include <stdio.h>
typedef struct a {unsigned val; struct a *next;} list;
list *put(list *head; unsigned a) {
   list *n = (list *)(malloc(sizeof(list)));
   n->val = a; n->next = head;
   return n;
}
unsigned get(list *head) {return head->val;}
void main(void) {
   list *head = NULL;
   head = put(head, %u1); head = put(head, %u2); head = put(head, %u3);
   printf("%u", get(head));
}

--------------------------------------------------------------------------------
15. Melyik utasítás helyes?
[ ] char* c=(char*)calloc(50*sizeof(char));
[ ] char* c=(char*)realloc(50*sizeof(char));
[ ] char* c=(char*)calloc(50,sizeof(char));
--------------------------------------------------------------------------------
16. Az alábbi kód helyesen járja-e be a láncolt listát a megadott elemtől
kezdődően? Az elem struktúra már definiálva van és tartalmaz egy mutatót
a következő elemre.
void Bejar( struct elem ** start ){
struct elem** iterator = start;
while( *iterator) iterator = &(*iterator)->kovetkezo;
```

```
}
```
[ ] Nem, mert nem járja be a listát.
[ ] Igen helyesen járja be.
[ ] Nem jó, nem is fordul a kód.

---

17. Mit csinál az alábbi függvény?
```
typedef struct le { struct le * next; int a; } lista;
int f(lista *x){ if(!x) return 0; return a+f(x->next);}
```
[ ] Összeszámolja egy láncolt listában tárolt elemek összegét.
[ ] Meghatározza egy láncolt listában tárolt elemek maximumát.
[ ] Megszámolja egy láncolt lista elemeit.

---

18. Mit csinál az alábbi függvény?
```
typedef struct bf { struct bf * b; struct bf * j; int a; } bifa;
int f(bifa *x){
   if(!x)
     return 0;
   return x->a+f(x->j)+f(x->b);
}
```
[ ] Megméri egy bináris fa mélységét.
[ ] Megszámolja egy bináris fa csúcspontjait.
[ ] Összeadja egy bináris fa csúcsaiban tárolt értékeit.

---

19. Milyen adatszerkezet felépítésére alkalmas a következő struktúra?
```
struct elem {struct elem * egy; struct elem * ketto; int ertek; };
```
[ ] Mindkettő
[ ] Kétszeresen láncolt lista
[ ] Bináris fa

---

20. Mi lesz a *p értéke a következő programrészlet végén?
```
    char t[] = "S178M", *p = t + 1;
    int i = 2, *q = &i;
    p += *q;
```

---

21. Mit ír ki a következő programrészlet?
```
   int i = 10, j = 8, *p = &i, *q = &i;
   (*q)++; q = &j; *q += *p;
   printf("%d", *q);
```

---

22. Szintaktikailag típushelyes-e az alábbi utasítás?
```
**p = *q;
```
[ ] Igen, amennyiben **p és *q típusa megegyezik.
[ ] Igen, amennyiben p egy egész típusú mutató.
[ ] Nem, csak abban az esetben működne, ha *p=*q vagy p=q lenne az értékadásban.

---

23. Mivel tér vissza az alábbi függvény?

```
struct Node { struct Node *a; struct Node *b; };
int f(struct Node *n) {  return f(n->a) + f(n->b); }
```
[ ] A paraméterként kapont bináris fa csúcspontjainak számával.
[ ] Hibával elszáll a program.
[ ] A paraméterként kapott bináris fa leveleinek számával.

---

24. Melyik sor okozza a futás idejű hibát?
```
1. char *p;
2. *p = (char *) malloc(20);    /* lefoglalunk 20 byte memóriát */
3. *p = 'x';
```
[ ] 2. sor hibás, nem kell * a p elé
[ ] 3. sor helyesen: p[0]='x';
[ ] 2. sor hibás, & kell a p elé

---

25. Egy fésűs láncolt listának minden eleme azonos típusú?
[ ] Nem igaz, a "fejlista" elemei mások.
[ ] Igaz.
[ ] Nem igaz, minden fogban különböző típusú elmek lehetnek.

---

26. Hogyan állapítható meg a sort függvényből, hogy az array paraméter dinamikus?
```
void sort(int *array, int size) {
  ...
```
[ ] sizeof operátorral.
[ ] cast operátorral int [] típusra.
[ ] Sehogy.

---

27. Mit jelent az int **i deklaráció?
[ ] Ez egy int típusú változó, mert a két * operátor "kiejti" egymást
[ ] Ez egy hibás deklaráció, két * operátor nem szerepelhet egymás után
[ ] Ez egy int típusú mutatóra mutató mutató deklarációja

---

28. Mit ír ki az alábbi program?
```
#include<stdio.h>
#include<stdlib.h>
int main(){
   struct lelem{ int a; struct lelem* kov;} *kezdo=NULL, *uj;
   int tomb[4]={ 25, 25, 17, 13};
   int i;
   for(i=0;i<3;i++){
     uj = (struct lelem*)malloc(sizeof(struct lelem));
     uj->a = tomb[i];
     uj->kov = kezdo;
     kezdo = uj;
   }
   for(uj=kezdo;uj;uj=uj->kov)printf("%d, ", uj->a);
   for(uj=kezdo;uj;uj=kezdo){kezdo=uj->kov;free(uj);}
}
```

---

29. Hivatkozhatunk-e a p2 pointer által mutatott adatra a következő kódrészlet után?
```
int *p1,*p2;
p1=(int *)malloc(5*sizeof(int)); ...
p2=p1+2; ...
p1=(int *)realloc(p1,10*sizeof(int));
```
[ ] Nem, mert egy olyan dinamikus adatterületre mutat, amelyet a realloc felszabadíthatott, és az arra való hivatkozás elszállást okozhat
[ ] Igen, legfeljebb nullát kapunk eredményül
[ ] Igen, mert a dinamikus tömb mérete legalább 5, mi pedig a 2. elemre hivatkozunk

---

30. Adott az alábbi struktúrákból felépülő bináris fa:
```
struct fa{ int kulcs; double ertek; struct fa *bal, *jobb; };
```
A fa egy érvényes csomópontjára mutató 'struct fa *p' pointeren végrehajtott p++; utasítás hova állítja a pointert?
[ ] Oda, ami az adott struktúra után következik a memóriában; potenciálisan egy érvénytelen, vagy nem 'struct fa'-t tartalmazó memóriaterületre.
[ ] A bal oldali ágon levő csomópontra.
[ ] A jobb oldali ágon levő csomópontra.

---

31. Mit mondhatunk a következő kódrészletről?
```
  char s1[]="alma";  char s2[10];  while( *s2++ = *s1++) ;
```
[ ] Az s1 és az s2 nem jobbérték, ezért a kód szintaktikailag hibás.
[ ] Az s1 sztringet átmásolja az s2 sztringbe.
[ ] A sztring első betűjének az értékét növeli eggyel, és a ciklusnak akkor lesz vége, ha a 8 bites érték túlcsordul.

```
32. Mit mondhatunk a következő kódrészletről?
      double (*m)[10];
      int i, j;
      m = ( double(*)[10] ) malloc( 10 * sizeof( *m ) );
      for(i=0; i<10; i++ )for( j=0; j<10; j++) m[i][j] = 5;
[ ] Az m változó definíciója szintaktikailag hibás.
[ ] Az m változó egy 10 elemű vektorra mutató pointer, ezért az m[i][j]
kifejezés hibás.
[ ] Az m egy olyan pointer, amely elemei 10 elemű lebegőpontos értékeket tartal-
mazó vektorok, tehát a kódrészlet egy 10X10-es mátrixot foglal le a heap-en.
```

Példák

1. Adott az alábbi típusdefiníció., a fa adat szerint rendezett (balra a kisebbek, jobbra a anagyobbak).

```
/*******************************************************/
typedef struct dat{
/*******************************************************/
    double adat;
    struct dat *bal,*jobb;
}bifa,*pbifa;
```

   a. Írjon függvényt, amely visszaadja a fa leveleinek számát! (Levél az az elem, amelynek nincs gyermeke.)
   b. Írjon függvényt, amely visszaadja a fa belső csomópontjainak számát! (Belső csomópont az az elem, amelynek legalább egy gyermeke van.)
   c. Írjon függvényt, amely visszaadja a fa szintjeinek számát! (A gyökérelem van a legfelső szinten, ennek gyerekei a másodikon, ezek gyerekei a harmadikon, stb.)
   d. Írjon függvényt, amely visszaadja azon elemek számát, melyeknek pontosan egy gyermeke van!
   e. Írjon függvényt, amely nagyság szerint növekvő/csökkenő sorrendben kiírja a fában tárolt adatokat!
   f. Írjon függvényt, amely kiírja a fában tárolt adatok átlagát! (Segédfüggvény(ek) írása megengedett.)
   g. Írjon függvényt, amely törli a fából azokat az elemeket, melyeknek adata egy paraméterként kapott értéknél kisebb!

2. Adott a következő típusdefiníció:

```
typedef struct fa{
       unsigned kulcs;
       char szo[50];
       struct fa *bal,*jobb;
}bifa,*pbifa;
```

a) Készítsen egy olyan szabványos ANSI C függvényt, amely egy bifa típusú elemekből álló, kulcs szerint rendezett bináris fa első elemére mutató pointert (gy) és egy beszúrandó, bifa típusú elemre vagy részfára mutató pointert (uj) vesz át paraméterként, és az uj elemet beszúrja a rendezett fa megfelelő helyére! A függvény adja vissza a módosított fa gyökérelemének címét! (5p)
b) Készítsen egy olyan szabványos ANSI C függvényt, amely egy bifa típusú elemekből álló, kulcs szerint rendezett bináris fa első elemére mutató pointert (gy) és egy előjel nélküli egészet (torol) vesz át paraméterként; megkeresi a fában a torol paraméterrel megegyező kulcsú elemet, és ha benne van, törli ezt az elemet! Csak ezt az elemet törölje, a belőle induló két részfát a fa más elemeihez kell csatolnia, méghozzá oly módon, hogy a fa továbbra is kulcs szerint rendezett maradjon! (5p)

3. Adottak a következő típusdefiníciók:

```
struct alagut;
typedef struct {
       struct alagut * kov;
       unsigned max,akt;
       double hossz;
}adat,*padat;
typedef struct alagut{
       padat tomb;
       unsigned n;
}alag,*palag;
```

Egy városi kábelcsatorna-hálózatot leíró program két függvényét készítjük el. Az alagut típusú struktúrák egy irányított gráf csomópontjai (a csatorna elágazásai). A csomópontok tomb adattagja a szomszédos csomópontokba vezető élek (csövek) adatait tartalmazzák, az n pedig a tömb elemszáma (ha n==0, akkor tomb==NULL). Az élek adatai: kov: a szomszédos csomópontra mutató pointer. NULL, ha nincs szomszéd; max: egy csőben hány adatkábel futhat; akt: jelenleg hány adatkábel fut; hossz: a cső hossza. Egy adott csomópontba több úton is el lehet jutni, de visszafelé nem, sem közvetlenül, sem közvetve.
a) Írjon egy olyan szabványos ANSI C függvényt, amely paraméterként kapja két csomópont címét, visszatérési értékke pedig a csomópontok közötti legrövidebb olyan út hossza, amelyen még lefektethető kábel a két csomópont között (akt<max)! Ha nincs ilyen út, -1-et adjon vissza! (6p)
b) Írjon egy olyan szabványos ANSI C függvényt, amely paraméterként kapja két csomópont címét, és behúz egy kábelt a két csomópont közötti útra, azaz az útvonal minden csomópontjának akt értékét eggyel megnöveli! Ha sikerült behúzni a kábelt, a függvény 1-et adjon vissza, ha nem, akkor 0-t! (4p)