Programozás alapjai 1. (BMEVIEEA100)

Gyakorlat anyaga az 8. oktatási héten

Alább látható a félév teljes menete. Zölddel kiemelve a 9. hét, amikor a nagy ZH-t írják, ez a jövő hét!!! A 10. héten van a pót zh, ezért kapott kiemelést. A nagy ZH-ról részletesebben az eheti anyag ismertetése után.

Hét	Hétfő			Szerda			Péntek		
	Anyag	HF	ZH	Anyag	HF	ZH	Anyag	HF	ZH
2	Bevezetés	A		Bevezetés	A		Bevezetés	A	
3	Adattípusok			Adattípusok			Adattípusok	B	
4	Operátorok	B		Operátorok	B				
5	Állapotgép, többdimenziós tömbök	C, D		Állapotgép, többdimenziós tömbök	C, D		Operátorok	C	
6	Pointerek, dinamikus tömbök	E	A, B	Pointerek, dinamikus tömbök	E	A, B	Állapotgép, többdimenziós tömbök	D	A, B
7	Stringek	F		Stringek	F		Pointerek, dinamikus tömbök, stringek	E, F	
8	Rendezés, keresés, hashing	G	C, D, E	Rendezés, keresés, hashing	G	C, D, E	Rendezés, keresés, hashing	G	C, D, E
9	Gyakorlás			Függvénypointerek, rekurzió (qsort)	H		Függvénypointerek, rekurzió (qsort)	H	
10	Függvénypointerek, rekurzió (qsort)	H	F, G	Dinamikus adatszerkezetek - listák		F, G	Dinamikus adatszerkezetek - listák		F, G
11	Dinamikus adatszerkezetek - listák						Dinamikus adatszerkezetek – fák	I	
12	Dinamikus adatszerkezetek – fák	I		Dinamikus adatszerkezetek – fák	I				
13	Fájlkezelés	J	H, I	Fájlkezelés	J	H, I	Fájlkezelés	J	H, I
14	Gyakorlás			Gyakorlás			Gyakorlás		

A Bevezetés
B Azonosítók, operátorok
C Adatbevitel billentyűzetről, kiírás képernyőre
D A C nyelv utasításai
E Függvények használata
F Enum, tömb, struktúra, sztringek
G Keresés, rendezés, hashing
H Függvényérték paramétersoron, függvénypointer, rekurzió
I Pointerek, dinamikus adatszerkezetek
J Fájlkezelés

2. kis ZH

Ezen a héten írassuk a 2. kis ZH-t! A sablon a következő oldalon, feladatsor generálása az ismert módon. Az első feladat a régebbi anyagból keletkezik, a többi új.

2. ZH (használható hozzá kétoldalas C összefoglaló és kézzel írott puska).

Név:		FV sikeres:		× 3 pont=	
Neptun kód:		FV sikertelen:		×–3 pont=	
Hallgató aláírása:		MF sikeres:		× 2 pont=	
		MF sikertelen:		×–2 pont=	
		FV=FeleletVálasztós, MF=Megoldandó Feladat		Össz:	

Pontozás: -2: 1, 3-8: 2, 9-14: 3, 15-18: 4, 19-22: 5

Röviden

Ismét mindhárom csoport azonos anyagrésznél tart, de ez csak egy hétig van így, mert a szerdai és pénteki csoportokban el fog maradni egy-egy óra a TDK ill. a nyílt nap miatt, így a hétfőiek kapnak egy plusz órát gyakorlás céljára a nagy ZH előtt.

Előadáson a 6. ment a hashing és a keresés, a múlt héten a rendezés, ezen a megy/ment a láncolt lista és a bináris fa. Gyakorlaton a téma a keresés, rendezés, hashing, de csak a rendezésekkel kell részletesen foglalkozni. A bináris keresés előkerül a beszúró rendezésnél, hashing pedig eddig még sosem fordult elő nagy zh-ban és vizsgán, tehát elég az, amit előadáson hallottak. Poppe András az összes Pongor György-féle rendezőalgoritmust fel szokta adni: http://www.eet.bme.hu/~nagyg/pongor_sort.pdf, azonban ezek közül a buborék, közvetlen kiválasztás és közvetlen beszúrás az, amit a gyakorlaton nézzünk át, illetve a gyorsrendezés is lesz később, a rekurzióval együtt, tehát most még nem adjuk le.

Új anyag

A korábbi órák tapasztalataiból okulva a hallgatókkal úgy egyeztünk meg, hogy nem megyünk végig közösen a tesztkérdéseken, csak amiket problémásnak éreznek, azokat beszéljük meg.

Hogyan működik a közvetlen kiválasztásos rendezés?

1. lépés: Végigmegyünk a tömb elemein, és megnézzük, melyik a legkisebb. Ha megtaláltuk, akkor kicseréljük a tömb első elemére.
2. lépés: A tömb első eleme tehát a legkisebb, ebben a körben ezt már nem vizsgáljuk. A maradék elemek között megkeressük a legkisebbet, és ezt kicseréljük a tömb második elemével.
3. lépés: A maradékból kikeressük a legkisebbet, és ezt tesszük a maradék elejére.
4. lépés: A 3. lépést ismételjük, míg a tömb végére nem érünk.

Lássuk ezt C nyelven:

```
//******************************************************
void kivalasztas(double * t,int meret){
//******************************************************
        int i,j,minindex;

        for(i=0;i<meret-1;i++){
                minindex=i;
                for(j=i+1;j<meret;j++){
                        if(t[j]<t[minindex])minindex=j;
                }
                if(minindex!=i){
                        double temp=t[minindex]; t[minindex]=t[i]; t[i]=temp; // csere
                }
        }
   }
```

A buborékrendezés a következőképpen működik:

1. lépés: Megvizsgáljuk a tömb első és második elemét. Ha az első nagyobb, akkor felcseréljük őket. Ezután megvizsgáljuk a második és harmadik elemét, ha a második nagyobb volt, ismét cserélünk (figyeljük meg, hogy ha az első

lépésben és a másodikban is volt csere, akkor az eredetileg az első helyen álló elem már a harmadikon van: ezúttal tehát a legnagyobb elem mozog a tömb vége felé, nem pedig a legkisebb az eleje felé, mint a közvetlen kiválasztásos rendezésnél!). Ezt a cserélgetést a tömb végéig ismételgetjük.

2. lépés: A tömb utolsó eleme tehát a legnagyobb. A következő lépésben szintén a tömb legelejéről kezdjük (ellentétben a közvetlen kiválasztásossal), de ezúttal csak az utolsó előttiig megyünk, hiszen az utolsó a legnagyobb.
3. lépés: A maradékban végigcserélgetjük a szomszédos elemeket, így a maradék legnagyobb eleme a maradék végére kerül.
4. lépés: A 3. lépést ismételgetjük, míg a maradék egy elem nem lesz.

Az algoritmust gyorsabbá tehetjük, ha figyeljük, hogy az adott lépésben volt-e csere. Ha nem volt, akkor a tömb rendezett, tehát nem kell tovább folytatnunk a rendezést.

C nyelven az algoritmus a következőképp néz ki:

```
//******************************************************
void buborek(double * t,int meret){
//******************************************************
        int i,j,nemvoltcsere;
        double temp;

        for(i=0;i<meret-1;i++){
                nemvoltcsere=1;
                for(j=0;j<meret-1-i;j++){
                        if(t[j]>t[j+1]){
                                temp=t[j];t[j]=t[j+1];t[j+1]=temp; // csere
                                nemvoltcsere=0;
                        }
                }
                if(nemvoltcsere)break;
        }
}
```

A közvetlen beszúró rendezés a következőképpen működik:

1. A tömb elején vannak a rendezett adatok, a végén a rendezetlenek.
2. Vesszük a következő rendezetlen adatot, és egy ideiglenes változóban eltároljuk.
3. Megkeressük, hová kell beszúrni a rendezett részben, bináris keresést használunk.
4. Ettől a helytől a rendezett rész végéig eggyel hátrébb csúsztatjuk az adatokat, az ideiglenes változóba elmentett nem rendezett értéket a rendezett rész utolsó eleme felülírja.
5. A felszabadított helyre beírjuk az elmentett értéket.
6. 2-5 lépések ismétlődnek, míg el nem tűnik a rendezetlen rész.

C nyelven az algoritmus a következőképp néz ki:

```
void beszuras(double * t, int n){
    int i,also,felso,x;
    double temp;
    for (i=1; i<n; i++) {
        for(also=0,felso=i-1,temp=t[i]; also<=felso; ){
            x=(also+felso)/2;
            if(temp<t[x])felso=x-1;
            else also=x+1;
        }
        for(x=i; x>also; x--)t[x]=t[x-1];
        t[also]=temp;
    }
}
```

Könyvtári qsort

A C rendelkezik egy beépített rendező függvénnyel, a qsorttal. A q a quickre utal. Előbb megnézzük a beépített qsort függvény használatát, majd mi magunk írunk qsort függvényt.

Az alábbi program bemutatja a használatát:

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

//*******************************************************
void kiir(double * t,int meret){
//*******************************************************
        int i;
        for(i=0;i<meret;i++)printf("%d. elem: %.2f\n",i+1,t[i]);
}

//*******************************************************
int hasonlit(const void *a,const void *b){
//*******************************************************
        double *ia=(double *)a;
        double *ib=(double *)b;

        if(*ia<*ib)return -1;  //<0
        if(*ia==*ib)return 0;  //==0
        return 1;              //>0
}

//*******************************************************
int main(){
//*******************************************************
        double t[7]={863,-37,54,9520,-3.14,25,9999.99};

        printf("Rendezes elott:\n");
        kiir(t,7);

        qsort(t,7,sizeof(double),hasonlit);

        printf("Rendezes utan:\n");
        kiir(t,7);
        return 0;
}
```

Nagy ZH

Felépítése:

- Teszt: 10 db feleletválasztós + 5 db válaszadós feladat. A **maximális pontszám legalább felét** kell elérni a sikerhez. Az első 7 témakör tesztkérdéseiből lehetnek feladatok.
- 3 db kiskérdés. A **megszerezhető pontszám fele** szükséges a sikeres nagy ZH-hoz.
  1. Adattípusok, operátorok, ciklusszervezés. Jellegzetes feladatok a számelméleti példák: osztó, prím stb., továbbá a bitszintű műveletek ismeretét igénylő feladatok: maszkolás, biteltolás, pl.: bitszámlálás, paritásvizsgálat, egyes bitek kimaszkolása.
  2. Állapotgép, egy- és többdimenziós tömbök, dinamikus tömbök, pointerek, függvényírás. Bármilyen egyszerűbb szövegszűréses állapotgépes feladat előfordulhat, állapotgráfot vagy állapottáblát rajzolni tudni kell. Dinamikus tömb felszabadítását tudni kell. Tudni kell függvényt írni, függvényértéket visszaadni paramétersoron (pointerrel) vagy függvényértékként (returnnel). Előfordulhat olyan feladat, hogy függvény tömböt kap, és a tömb elemei közül ki kell válogatni bizonyos tulajdonságúakat (páros számokat, átlag alattiakat, egy paraméterként adott értéknél nagyobbakat stb.) Pointert visszaadó függvények.
  3. Stringek, rendezés, keresés: könyvtári sztringfüggvények (strlen, strcat, strcmp, strcpy stb.) saját megírása; sztringek összefésülése betűnként, szavanként, mondatonként stb.; bizonyos tulajdonságú sztringrészek törlése, pl. szóközök, kisbetűk, nagybetűk, számok. Keresés sztringen belül, részsztring előállítása stb. Tudni kell rendezni, ismerni kell legalább egy rendező algoritmust. (Fenn lehet a kézzel írott puskán, de tudni kell használni).
- 2 db bonyolultabb feladat. Ezek elkészítése nem szükséges, ha a teszt és a kisfeladatok kritériuma teljesül, akkor megvan az aláírás. Aki szeretne elővizsgázni, annak kell megírnia a két bonyolultabb feladatot. A 3 kisfeladat + a 2 nagyfeladat pontszámából kialakult rangsor alapján dől el, hogy ki írhat elővizsgát (a teszt pontszáma ebbe nem számít bele). A bonyolultabb feladatok mindazokat az anyagrészeket felölelik, amelyek a kisfeladatoknál fel lettek sorolva, de többet kell gondolkodni és/vagy több munkával jár az elkészítésük.

Három teljes nagy ZH feladatsor készült: egy gyakorlás céljára, egy a nagy ZH-ra, egy a pót ZH-ra. Előre nem határoztuk el, hogy hová melyik feladatsor kerül, így tehát a **mellékelt minta ZH** akár az éles ZH is lehetet volna (nehézségre, de a fenti listában felsorolt témakörök mindegyikét nézzék át a hallgatók, mert míg az egyik sorban pl. állapotgép a második feladat, egy másik feladatsorban tömbkezelő függvény, stb.). A sztringkezelő függvények önálló megírásának gyakorlása különösen ajánlott!

A teszt megoldása:

---

1. Ha egy double visszatérési típusú, egyetlen double paramétert váró függvényt meg akarunk hívni a main függvényből, akkor a forrásállományban a main függvény definíciója előtt:
[ ] nem szükséges semmi, amennyiben a fordító valahol máshol megtalálja a meghívandó függvényt.

[ ] a meghívandó függvény deklarációjának és definíciójának is szerepelnie kell (ill. egy header állományt kell #include-olni, amiben szerepel).
[X] a meghívandó függvénynek legalább a prototípus deklarációjának szerepelnie kell (ill. egy header állományt kell #include-olni, amiben szerepel).
---
2. Vizsgálhatjuk-e a túlcsordulást a következő kódrészlettel?
```
 int i,j; ...
 if ((i+j)>INT_MAX)) ...
```
[X] Nem, mert semmilyen egész művelet ideiglenesen sem lesz nagyobb, mint az ábrázolható legnagyobb szám, ezért ez a feltétel akkor sem teljesül a programban, ha matematikailag az összeg nagyobb, mint INT_MAX
[ ] Igen, ez így jó
[ ] Nem, mert nem INT_MAX a legnagyobb ábrázolható egész, hanem MAX_INT
---
3. Mit történik a continue utasítás hatására ciklusban?
[ ] Hatására while, for és do-while ciklusok megszakadnak és a vezérlés a ciklus utáni első utasításra adódik át.
[X] A while, for és do-while ciklusok soron következő iterációját indítja el, így az aktuális iteráció hátralévő utasításait átugorja.
[ ] Hatására a program végrehajtása a megadott címke utáni első utasítással folytatódik.
---
4. Végezhető-e bináris keresés olyan tömbön, amelyben az elemek fordított sorrendben szerepelnek?
[X] Igen.
[ ] Nem.
---
5. Helyes-e a következő program?
```
int valami(const int* i) { return *i = (*i)+1; }
int main(){ int i = 5; printf("%d", valami(&i)); return 0; }
```
[X] Nem, fordítási idejű hiba
[ ] Igen
[ ] Nem, futási idejű hiba
---
6. Az alábbi változódefiníciók közül melyik hibás?
[ ] int nice1x=0xface;
[X] int 0xface,nice1x;
[ ] int nice1x;
---
7. Mi lesz ennek az utasításnak a hatása?
```
a>5;
```
[X] Semmi; a kifejezés logikai értékét nem használjuk fel semmire.
[ ] Ha ezen a ponton, vagy valamikor később 'a' változó értéke 5, vagy annál kisebb, a program hibával kilép.
[ ] 'a' változóba egy 5-nél nagyobb érték kerül.
---
8. Teljesül a feltétel az alábbi kódban?
```
char s[] = "Hello!", t[] = "Hello!";
if(strcmp(s, t)) printf("A két sztring egyforma!");
```
[ ] Igen.
[X] Nem.
---
9. Helyes-e az alábbi kódrészlet?
```
int i;
double t[5];
for( i = 1; i <=5; i++ ){
   printf("\nt[%d]", i );
   scanf("%lf", &t[i]);
}
```
[X] Nem, mert kiindexelünk a tömbből
[ ] Igen.
---
10. Mit ír ki az alábbi program?
```
main(){
  char word[20];
  word[0] = 'B'; word[0]++;
  word[1] = 'Y';
```

```
  word[2] = 'E'; --word[2];
  word[3] = 0;
  word[4] = 'B';
  printf("The contents of word[] is -->%s\n", word);
}
```

[ ] The contents of word[] is --> BYD
[ ] The contents of word[] is --> BYE
[X] The contents of word[] is --> CYD

---

11. Mi az fv(8) értéke?

```
int fv(unsigned x) {
  int c=0;
  for(; x; x>>=1) c+=x&1;
  return c;
}
```

Megoldás: 1

---

12. Hány karakterre fogja kiírni a következő kódrészlet a számot?

```
double d = 95.931;
printf("%13.2f", d);
```

Megoldás: 13

---

13. Adott az alábbi értékadás:

```
int t[] = {15, 2,  9,  2, 1, 2};
```

Mi az értéke a (t+2)[t[3]+1] kifejezésnek?

Megoldás: 2

---

14. Mit ír ki az alábbi program?

```
#include <stdio.h>
typedef enum {hetfo, kedd, szerda, csutortok, pentek, szombat, vasarnap} napok;
napok nap = 15 % 7;
void main(void) {nap = (nap +  kedd) % 7; printf("%d", nap);}
```

Megoldás: 2

---

15. Mennyi lesz x értéke?

```
int t[] = {7, 62, 8, 10, 32, 157}
int *p = t + 2;
int x = p[1];
```

Megoldás: 10

-

Házi feladat

```
X ********************************************************************************
C Keresés, rendezés, hashing
X ********************************************************************************
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
2. Mi a bináris keresés alkalmazhatóságának feltétele?
[ ] A bináris keresés csak számokon alkalmazható.
[ ] A tömb elemei rendezve kell legyenek.
[ ] A tömb mérete kettő valamely hatványa kell legyen.
--------------------------------------------------------------------------------
3. Végezhető-e bináris keresés olyan tömbön, amelyben az elemek fordított
sorrendben szerepelnek?
[ ] Igen.
[ ] Nem.
--------------------------------------------------------------------------------
4. Melyik rendezést valósítja meg az alábbi programkód?
int i,j,minindex;
for(i=0;i<meret-1;i++){
  minindex=i;
  for(j=i+1; j<meret; j++) if(t[j] < t[minindex]) minindex =j ;
  if(minindex!=i){
     double temp = t[minindex];
     t[minindex]=t[i];
     t[i]=temp;
  }
}
[ ] Közvetlen kiválasztásos.
[ ] Buborék.
[ ] Közvetlen beszúrásos.
--------------------------------------------------------------------------------
5. Milyen irányban rendezi a tömböt az alábbi algoritmus?
void h(int *t, int s) {
  int i, j;
  for(i=s-1; i>=0 ;i--)
    for(j=i-1; j>=0; j--)
      if(t[i]<t[j]){
        int k = t[i];
        t[i] = t[j];
        t[j]=k;
      }
}
[ ] Növekvő.
[ ] Csökkenő.
--------------------------------------------------------------------------------
6. Az alábbiak közül melyik a legkevésbé hatékony?
[ ] Buborékrendezés
[ ] Gyorsrendezés
[ ] Bináris fa rendezés
--------------------------------------------------------------------------------
7. n adat esetén milyen nagyságrendű a gyorsrendezés átlagos lépésszáma?
[ ] logn
[ ] n*n
[ ] n*logn
--------------------------------------------------------------------------------
8. n adat esetén milyen nagyságrendű a buborékrendezés átlagos lépésszáma?
[ ] n*n
[ ] logn
[ ] n*logn
--------------------------------------------------------------------------------
9. Hash tárolási technikát alkalmazva listázhatóak-e sorrendhelyesen a táblában
tárolt adatok?
[ ] Igen.
[ ] További rendezés nélkül nem.
--------------------------------------------------------------------------------
10. Igaz vagy hamis: a hash-eléshez használt adatstruktúra csak tömb lehet?
[ ] Hamis.
[ ] Igaz.
--------------------------------------------------------------------------------
11. A hash technikán alapuló tárolási elv átlagos lépésszáma beszúrás esetén
(megfelelően nagy táblaméret mellett) ...
[ ] arányos az adathalmaz méretének négyzetével.
[ ] konstans.
[ ] arányos az adathalmaz méretével.
--------------------------------------------------------------------------------
12. A hash technikán alapuló tárolási elv átlagos lépésszáma keresés esetén
(megfelelően nagy táblaméret mellett) ...
[ ] konstans.
[ ] arányos az adathalmaz méretének négyzetével.
[ ] arányos az adathalmaz méretével.
--------------------------------------------------------------------------------
13. A hash technikán alapuló tárolási elv átlagos lépésszáma közel telített
táblában való keresés esetén ...
[ ] arányos az adathalmaz méretének négyzetével.
[ ] arányos az adathalmaz méretével.
[ ] konstans.
--------------------------------------------------------------------------------
14. Milyen műveletek lesznek hatékonyak hashing alkalmazásával?
[ ] Keresés, beszúrás, törlés és módosítás
[ ] Keresés, beszúrás, min-max kiválasztás és módosítás
[ ] Keresés, beszúrás, rendezés és törlés
--------------------------------------------------------------------------------
15. Melyik rendezést valósítja meg az alábbi programkód?
int i,j,buf;
for(i=0;i<n;i++){
  buf=a[i]; j=i-1;
  while(j>0 && buf<a[j]){ a[j+1]=a[j]; j=j-1; }
  a[j+1]=buf;
}
[ ] Buborék.
[ ] Közvetlen beszúrásos.
[ ] Közvetlen kiválasztásos.
--------------------------------------------------------------------------------
16. Melyik rendezést valósítja meg az alábbi programkód?
int i,j,t;
for(i=0;i<n;i++)
  for(j=i-1;j>=0 && a[j]>a[j+1];j--){ t=a[j]; a[j]=a[j+1]; a[j+1]=t; }
}
[ ] Közvetlen kiválasztásos.
[ ] Közvetlen beszúrásos.
[ ] Buborék.
--------------------------------------------------------------------------------
17. Lehet-e egy egyirányú láncolt listában binárisan keresni?
[ ] Igen, bármikor
[ ] Igen, de csak akkor ha az rendezett
[ ] Nem lehet, mert mindig csak a következő elemre lehet ugrani.
--------------------------------------------------------------------------------
18. Milyen adatszerkezeten végezhető el a gyorsrendezés?
[ ] mindkettőn
[ ] tömbökön
[ ] egy irányban láncolt listán
--------------------------------------------------------------------------------
```

```
19. Milyen adatszerkezeten végezhető el a gyorsrendezés?
[ ] mindkettőn
[ ] tömbökön
[ ] két irányban láncolt listán
-------------------------------------------------------------------------------
20. Milyen adatszerkezeten végezhető el a buborékrendezés?
[ ] egy irányban láncolt listán
[ ] tömbökön
[ ] mindkettőn
-------------------------------------------------------------------------------
21. Ha egy 100 elemű int tömböt használunk hash táblának, akkor megfelelő-e az
alábbi hash függvény?

int hash(int x){ return x*2%100; }
[ ] Nem, mert nem használja ki megfelelően a hash tábla értékeit.
[ ] Nem, mert nem megfelelő tartományban adja vissza az értékeit.
[ ] Igen
-------------------------------------------------------------------------------
22. Mitől lassú a buborékrendezés?
[ ] Ha egy elem nagyon messze van a végleges helyétől, sok lépésben kell
cserélgetni
[ ] Mivel a rendezett szakasz hossza lépésenként csak egyesével nő
-------------------------------------------------------------------------------
23. A gyorsrendezés azért gyors, mert...
[ ] egyszerre több helyen, párhuzamosan rendezgeti a tömböt.
[ ] egyszerű lépésekkel hamar a végleges helyük közelébe mozgatja az elmeket.
[ ] ötvözi a buborék algoritmus és a közvetlen kiválasztás előnyeit.
-------------------------------------------------------------------------------
24. Egy int t[100] rendezetlen tömbben a minimális elemet keressük,
milyen kezdőértéket kell adni a minimumot tároló változónak?
[ ] -65535
[ ] Nem kell kezdőérték
[ ] A tömb egyik elemére állítsuk be.
-------------------------------------------------------------------------------
25. Radix rendezést végzünk 5 jegyű számokon, azaz helyiértékenként stabil
ládarendezéseket (bin sort), összesen 5-öt. Milyen sorrendben kell végezni a
ládarendezéseket?
[ ] A nagyobb helyiérték felől a kisebb felé.
[ ] Tetszőleges sorrendben.
[ ] A kisebb helyiérték felől a nagyobb felé.
```

Példák

1. Írjunk C függvényt, amely paraméterül kap egy kétdimenziós karaktertömböt és a tömb első indexének méretét, a tömb második indexe 80. A tömb hallgatók neveit tartalmazza sztringként, ABC sorba rendezve. A függvény bináris kereséssel keresse meg az ugyancsak paraméterként érkező sztringben megadott nevű hallgatót, és adja vissza a megtalált hallgató nevének memóriacímét a kétdimenziós tömbben! Deklaráció:
   `char * keres1(char t[][80],int n, char nev[]);`
2. Írjunk C függvényt, amely paraméterül kap egy egydimenziós pointertömböt és a tömb méretét. A tömb hallgatók neveit tartalmazó sztringekre mutat, melyek ABC sorba vannak rendezve. A függvény bináris kereséssel keresse meg az ugyancsak paraméterként érkező sztringben megadott nevű hallgatót, és adja vissza a megtalált hallgató nevének memóriacímét! Deklaráció: `char * keres2(char * t[],int n, char nev[]);`
3. Írjunk C függvényt, amely paraméterül kap egy kétdimenziós karaktertömböt és a tömb első indexének méretét, a tömb második indexe 80. A tömb hallgatók neveit tartalmazza sztringként. A függvény rendezze sorba a neveket
   a. a könyvtári qsort függvénnyel
   b. saját maga által írt rendező algoritmussal (közvetlen kiválasztás, beszúrás, buborék)
4. Írjunk C függvényt, amely paraméterül kap egy egydimenziós pointertömböt és a tömb méretét. A tömb hallgatók neveit tartalmazó sztringekre mutat. A függvény rendezze sorba a neveket
   a. a könyvtári qsort függvénnyel
   b. saját maga által írt rendező algoritmussal (közvetlen kiválasztás, beszúrás, buborék)
5. Írjon függvényt, amely paraméterként kap egy data struktúrákat tartalmazó tömböt, valamint a tömb elemszámát, és sorszám szerint csökkenő/növekvő sorrendbe rendezi az elemeket. A data struktúra felépítése:
   `typedef struct {unsigned sorszam; char nev[60];} data;`
   Megjegyzés: a struktúrák másolhatók = operátorral, nem kell az adattagokat egyenként másolni, akkor sem, ha a struktúrában tömb található. Pl.: `data a,b; a=b;` művelet teljesen jó!