Dolgozatok

Egy ***függvény*** segítségével tetszőleges számú fizetést olvasson be egy tömbbe (pl. 0 végjelig). A függvény paramétere a tömb, értéke a beolvasott adatok száma. Egy másik ***függvény*** segítségével határozza meg az átlagfizetést. A függvény paramétere a tömb és a darabszám, a függvény értéke az átlag legyen. Az eredményt, a függvényértéknek megfelelően a ***main*** írja ki.

Megoldás:

```
#include<stdio.h>
#include<iostream>
int beker(float*);
float atlag(int,  float*);
int main(){
	setlocale(LC_ALL, "hun");
	float tomb[100];
	printf("Átlag = %8.2f\n", atlag(beker(tomb), tomb));
}
int beker(float *t){
	int db=0;
	printf("Kérek fizetéseket, 0 végjelig:\n");
	scanf("%f",t);
	while(*t){
		t++;
		db++;
		scanf("%f",t);
	}
	return db;
}
float atlag(int n, float *t){
	float ossz = 0.0 ;
	for(int i = 0 ; i < n ; i++ ) ossz += *t++;
	return (n)? ossz/n : 0;
}
```

Egy ***függvény*** segítségével olvasson be egy tetszőleges szöveget. Egy másik ***függvény*** segítségével számolja meg, hogy mennyi kisbetű karakter található a szövegben. A függvények paramétere a szöveg, a második függvény értéke a kisbetűk száma. Az eredményt, a függvényértéknek megfelelően a ***main*** írja ki.

Megoldás:

```
#include <stdio.h>
#include <iostream>
void beolv(char*);
int kisbetu(char*);
int main(){
	char szoveg[80];
	setlocale(LC_ALL, "");
	beolv(szoveg);
	printf("A kisbetűk száma = %d\n",kisbetu(szoveg));
}
void beolv(char *s){
	printf("Kérek egy sor szöveget:");
	scanf("%[^\n]", s);
}
int kisbetu(char *s){
	int db = 0;
	while(*s){
		if(*s >= 'a' && *s <= 'z')db++;
		s++;
	}
	return db;
}
```

Egy ***függvény*** segítségével olvasson be egy szöveget és egy karaktert, a függvény paramétere a szövegtömb, a függvényérték a karakter. Egy másik ***függvény*** segítségével határozza meg, hogy a beolvasott karakter hányszor fordul elő a szövegben. A függvény paramétere a szöveg és a karakter, a függvényérték a darabszám. Az eredményt, a függvényértéknek megfelelően a ***main*** írja ki.

Megoldás:

```
#include <stdio.h>
#include <stdio.h>
#include <iostream>
char beolv(char*);
int vizsgal(char*, char);
int main(){
        setlocale (LC_ALL, "");
        char szoveg[80];
        printf("A megadott karakter előfordulásainak száma = %d" ,vizsgal(szoveg, beolv(szoveg)));
}
char beolv(char *s){
        char c;
        printf("Kérek egy sor szöveget szöveget:\n"); scanf("%[^\n]", s);
        fflush(stdin);
        printf("Kérek egy karaktert: "); scanf("%c", &c);
        return c;
}
int vizsgal(char *s ,char c){
        int db = 0;
        while(*s){
                if(*s == c) db++;
                s++;
        }
        return db;
}
```

Összetett adattípus

A korábban ismertetett változók egyszerű vagy elemi típusúak voltak. Vagyis egy változóban egyetlen adatot lehetett tárolni. A tömböket is ide sorolhatjuk, mivel minden eleme csak azonos (egyszerű) típus lehet.

Összetett adattípust akkor használunk, amikor több összetartozó adatot akarunk együtt tárolni.

Az összetett típusok olyan változók, amelyek különböző elemi adattípusokból vagy más összetett típusokból épülnek fel. A programozó határozza meg, az összetett típus felépítését (struktúráját).

Az első lépés az adattípus összeállítása (megkomponálása). Ekkor meghatározzuk, hogy az adattípust milyen (elemi vagy összetett) tagok alkotják.

Példa:
Tároljuk el egy összetett típusú változóban a mai dátumot.

Megbeszélés:
1. Az összetett adattípus, struktúra elkészítése.
2. Összetett típusú változó deklarálása
3. Összetett típus tagváltozói.

Megoldás:

```
#include <stdio.h>
#include <iostream>
struct date{
        int day;
        char month[10];
        int year;
        };
int main(){
        date d;
        setlocale (LC_ALL, "");
        printf("év:\t");scanf("%d",&d.year);
        printf("hónap:\t");scanf("%s",d.month);
        printf("nap:\t");scanf("%d",&d.day);
        printf("\n%d - %s - %d\n", d.year, d.month, d.day);
}
```

Példa:
Alakítsuk át az előző programot úgy, hogy a dátum beolvasásához függvényt használunk.

Megbeszélés:
1. Összetett típus tagváltozói pointer használatakor.

Megoldás:

```
#include <stdio.h>
#include <iostream>
struct date{
        int day;
        char month[10];
        int year;
        };
void feltolt(struct date*);
int main(){
        date d;
        setlocale (LC_ALL, "");
        printf("\n%d - %s - %d\n", d.year, d.month, d.day);
}
void feltolt(struct date *f){
        printf("év:\t");scanf("%d",&f->year);
        printf("hónap:\t");scanf("%s",f->month);
        printf("nap:\t");scanf("%d",&f->day);
}
```

Példa:

Alakítsuk át az előző programot úgy, hogy több dátumot olvasunk be (tömböt használunk).

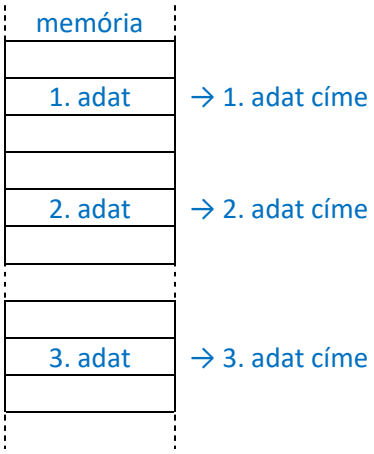
Megoldás:

```
#include <stdio.h>
#include <iostream>
struct date{
        int day;
        char month[10];
        int year;
        };
int feltolt(struct date*);
int main(){
        struct date tomb[10];
        int n,i;
        setlocale (LC_ALL, "");
        n=feltolt(tomb);
        for(i=0;i<n;i++)
                printf("\n%d - %s - %d\n", tomb[i].year, tomb[i].month, tomb[i].day);
}
int feltolt(struct date *f){
        int db=0;
        printf("kérem az évet: ");scanf("%d",&f->year);
        while (f->year){
                printf("kérem a hónapot: ");scanf("%s",f->month);
                printf("kérem a napot: ");scanf("%d",&f->day);
                db++; f++;
                printf("kérem az évet: ");scanf("%d",&f->year);
        }
        return db;
}
```

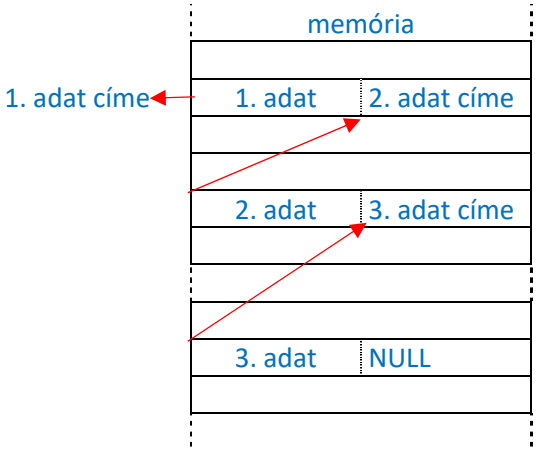
A dinamikus memória, láncolt lista

Az eddig megismert változók statikusak, vagyis a fordítás során (fordítási időben) jönnek létre. Ezért a programozónak előre ismernie kell, hogy mennyi adatot fognak tárolni a program használata során.

A dinamikus változók viszont futási időben jönnek létre, és csak akkora helyet foglalnak el a memóriában, amekkorára az adatok tárolására szükség van. Az adatok helyét a memóriában – a futás alatt – az operációs rendszer dönti el, az adat kezdőcímeit átadja a programnak.



Azért, hogy ne kelljen minden egyes memóriacímet a programban tárolni, egy láncba feltűzzük az elemeket úgy, hogy az első elem címét megjegyezzük egy mutatóban, a következő címét pedig mindig az előző elemben tároljuk egy pointerben. A lánc végét jelzi: az utolsó elemnél, a következő elemre mutató pointert NULL értékkel töltjük fel.



Példa:

Tároljunk láncolt listába tetszőleges mennyiségű egész számot, majd írjuk ki a tárolt adatokat a képernyőre.

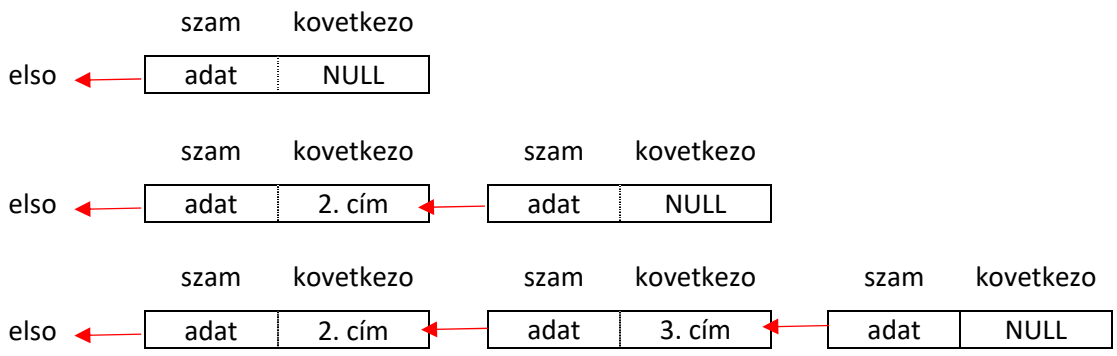
Megbeszélés:

1. Lista, láncolt lista kezelése, működése.
2. NULL memóriacím.
3. Memória allokálás, *malloc()* és *sizeof()* függvények.
4. Típuskényszerítés.
5. A *system()* függvény.

Megoldás:

Olvassunk be tetszőleges mennyiségű számot, zárásként adjunk nullát. A dinamikus memóriában helyet foglalunk, és ellenőrizzük, hogy az sikerült-e.

```
#include <stdio.h>
#include <stdlib.h>
struct rec{
	int szam;
	rec *kovetkezo;
};
int main(){
	int adat;
	rec *uj, *elso=NULL, *aktualis;
	printf("Kerek egy számot: "); scanf("%d",&adat);
	while(adat){
		uj=(rec*)malloc(sizeof(rec));		//Típuskényszerítés, helyfoglalás
		if(!uj){		//Ha nincs cím (NULL), akkor nem sikerült a helyfoglalás
			printf("Nincs elég memória\n");
			system("pause");
			return -1;
		}
		uj->szam=adat;
		uj->kovetkezo=NULL;
		if(!elso) elso=uj;		//Ha legelső elem
		else aktualis->kovetkezo=uj;
		aktualis=uj;
		printf("Kerek egy számot: "); scanf("%d",&adat);
	}
//Adatkiírás
	aktualis=elso;	//Ez első elem címe
	while(aktualis){	//A lista végén NULL
		printf("%d\n",aktualis->szam);
		aktualis=aktualis->kovetkezo;
	}
}
```



Példa:

Módosítsuk az első feladatot úgy, hogy a listában rendezetten tároljuk a számokat.

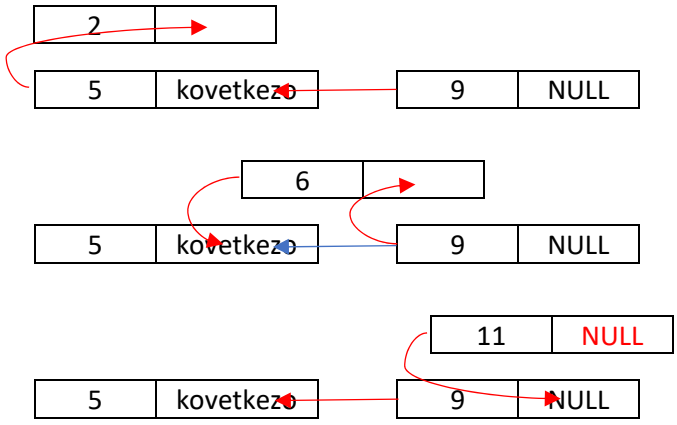
Megbeszélés:

1. A beszúrásos rendezés.
2. Adatkeresés láncolt listában.
3. Láncolt lista átcímzése úgy, hogy azon végighaladva rendezetten kapjuk meg az adatokat.

Megoldás:

Új mutató bevezetése: az új elem mindig az *elozo* és az *aktualis* mutatóval címzett adatok közé kerül.

```
#include <stdio.h>
#include <stdlib.h>
struct rec{
        int szam;
        rec *kovetkezo;
};
int main(){
        int adat;
        rec *uj, *elso=NULL, *elozo, *aktualis;
        printf("Kerek egy számot: "); scanf("%d",&adat);
        while(adat){
                uj=(rec*)malloc(sizeof(rec));
                if(!uj){
                        printf("Nincs elég memória\n");
                        system("pause");
                        return -1;
                }
                uj->szam=adat;
                elozo=NULL;
                aktualis=elso;
                while(aktualis && adat>aktualis->szam){ //Az új érték helyének megkeresése
                        elozo=aktualis;
                        aktualis=aktualis->kovetkezo;
                }
                if(!elozo) elso=uj;             //Ha a lista első eleme
                else elozo->kovetkezo=uj;
                uj->kovetkezo=aktualis;
                printf("Kerek egy számot: "); scanf("%d",&adat);
        }
//Adatkiírás
        aktualis=elso;
        while(aktualis){
                printf("%d\n",aktualis->szam);
                aktualis=aktualis->kovetkezo;
        }
}
```



Példa:

Egészítsük ki az előző programot úgy, hogy a listában megkeresünk egy tetszőleges értéket, és töröljük azt.

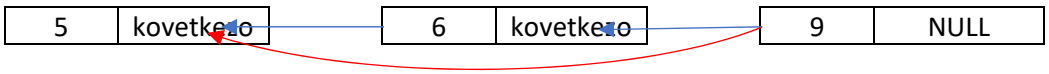
Megbeszélés:

1. Törlés a listából, lista átcímzése.
2. A nem használt memória felszabadítása.

Megoldás:

```
#include <stdio.h>
#include <stdlib.h>
struct rec{
	int szam;
	rec *kovetkezo;
};
int main(){
	int adat;
	rec *uj, *elso=NULL, *elozo, *aktualis;
//Rendezett feltöltés
	printf("Kerek egy számot: "); scanf("%d",&adat);
	while(adat){
		uj=(rec*)malloc(sizeof(rec));
		if(!uj){		//Ellenőrzés
			printf("Nincs elég memória\n");
			system("pause");
			return -1;
		}
		uj->szam=adat;
		elozo=NULL;
		aktualis=elso;
		while(aktualis && adat>aktualis->szam){
			elozo=aktualis;
			aktualis=aktualis->kovetkezo;
		}
		if(!elozo) elso=uj;
		else elozo->kovetkezo=uj;
		uj->kovetkezo=aktualis;
		printf("Kerek egy szamot: "); scanf("%d",&adat);
	}
// Lista kiírás
	aktualis=elso;
	while(aktualis){
		printf("%d\n",aktualis->szam);
		aktualis=aktualis->kovetkezo;
	}
```

```
// Egy elem törlése
        printf("Kérem a törlendő azonosítót: "); scanf("%d",&adat);
        aktualis=elso; elozo=NULL;
        while(aktualis){            //Keresés a listában
                if(adat==aktualis->szam)break;
                elozo=aktualis;
                aktualis=aktualis->kovetkezo;
        }
        if(!aktualis)printf("A keresett elem nem található\n");
        else{               //Megvan a keresett elem
                if(!elozo) elso=aktualis->kovetkezo;        //Ha az első elemet töröltük
                else elozo->kovetkezo=aktualis->kovetkezo;
                free(aktualis);     //A törölt elem helyének felszabadítása
        }
// Az új lista kiírás
        aktualis=elso;
        while(aktualis){
                printf("%d\n",aktualis->szam);
                aktualis=aktualis->kovetkezo;
        }
}
```



Feladatok:

1. Készítse el a következő struktúrát:

   név karaktertömb,
   életkor egész,
   testmagasság lebegőpontos.

   Töltsön fel egy struktúratömböt legalább három különböző adattal.

   Függvény segítségével határozza meg a legmagasabb ember nevét. A függvény paramétere a tömb, függvényérték a név legyen. Az eredményt a **main** írja ki.

2. Készítse el a következő struktúrát:

   név karaktertömb,
   életkor egész,
   testmagasság lebegőpontos.

   Töltsön fel egy struktúratömböt legalább három különböző adattal.

   Függvény segítségével határozza meg a legidősebb ember magasságát. A függvény paramétere a tömb, függvényérték a magasság legyen. Az eredményt a **main** írja ki.

3. Készítse el a következő struktúrát:

   azonosító egész,
   név karaktertömb,
   fizetés lebegőpontos.

   Töltsön fel egy struktúratömböt legalább három különböző adattal.

   Függvény segítségével határozza meg a legtöbbet kereső ember nevét. A függvény paramétere a tömb, függvényérték a név legyen. Az eredményt a **main** írja ki.

4. Készítse el a következő struktúrát:

   név karaktertömb,
   életkor egész,
   testmagasság lebegőpontos.

   Töltsön fel egy struktúratömböt legalább három különböző adattal.

   Függvények segítségével határozza meg az átlagmagasságot és az átlagéletkort. A függvények paramétere a tömb, függvényérték az átlag legyen. Az eredményeket a **main** írja ki.

5. Készítse el a következő struktúrát:

   azonosító egész,
   életkor egész,
   fizetés lebegőpontos.

   Töltsön fel egy struktúratömböt legalább három különböző adattal.

   Függvény segítségével határozza meg a legidősebb ember fizetését. A függvény paramétere a tömb, függvényérték a fizetés legyen. Az eredményt a **main** írja ki.

6. Készítsen egy programot e-mail címek nyilvántartására, az adatok: *név* és *e-mail,* ezeket egy struktúra-tömbben tárolja. A tömböt tetszőleges számú elememmel töltse fel. A feltöltéshez és a kiíráshoz használjon **függvényt**.

   Az előző programot egészítse ki olyan függvénnyel, amelyik név szerint keres a tömbben.

   Az előző feladatokat egészítse ki olyan függvénnyel, amely a megtalált nevet és a hozzá tartozó címet kitörli a tömbből.

7. Írjon programot, amelyben a következő feladatokat végzi el:

   - Hozza létre a következő struktúrát:

     ```
     int     azonosito;     /*sorszám*/
     char    szoveg[80];    /*egy sornyi szöveg*/
     int     hossz;         /*az aktuális sor hossza*/
     ```

   - Hozzon létre egy 100 elemű tömböt a fenti struktúra-típussal.
   - Töltse fel a tömböt tetszőleges számú elemmel (az elemek számát előre **ne** kérje be!). Az azonosítót és a szöveget billentyűzetről kérje be, a beírt szöveg hosszát **függvénnyel** határozza meg és írja be a „hossz" tagba. (A függvény paramétere a szöveg, érteke a hossz.)
   - Készítsen **függvényt**, amely megkeresi a tárolt elemek közül a leghosszabb szöveget és kiírja azt. A függvény paramétere a tömb, és az adatok száma.
   - Készítsen **függvényt**, amely megkeresi a tárolt elemek közül a legrövidebb szöveget és kiírja azt. A függvény paramétere a tömb, és az adatok száma.
   - Készítsen **függvényt**, amely kiszámítja a tárolt szövegek átlaghosszát, a függvény paramétere a tömb, és az adatok száma, a függvényérték az átlag, amit a main ír ki.
   - Készítsen **függvényt**, amely kiszámítja a szöveg hosszak szórását, a függvény paramétere a tömb, és az adatok száma, a függvényérték a szórás. A szórás képlete: $\sqrt{\frac{\sum_{i}^{n}(x-\bar{x})^2}{n-1}}$, a gyökvonást az **sqrt(x)** függvénnyel végezheti el, amelyhez a **math.h** headert kell használni!

8. Készítsen egy programot, amely létrehoz egy tömböt szövegek nyilvántartására. A tömb típusa a következő struktúrájú legyen:

   ```
   char    szoveg[80];    /*egy sornyi szöveg*/
   int     hossz;         /*az aktuális szöveg hossza*/
   int     kisbetu;       /*a sorban lévő kisbetűk száma*/
   ```

   - Hozzon létre egy 100 elemű tömböt a fenti struktúra típussal.
   - Töltse fel a tömböt tetszőleges számú elemmel. (Az elemek számát előre **ne** kérje be!) A szöveget billentyűzetről kérje be, a beírt szöveg hosszát és a benne lévő kisbetűk számát **saját függvényekkel** határozza meg és írja be a „hossz" illetve a „kisbetu" tagba. A függvények paramétere a szöveg, az értékük a hossz illetve a darabszám.
   - Készítsen **függvényt**, amely megkeresi a tárolt elemek közül a leghosszabb szöveget és kiírja azt. A függvény paramétere a tömb, és az adatok száma.
   - Készítsen **függvényt**, amely megkeresi a tárolt elemek közül a legrövidebb szöveget és kiírja azt. A függvény paramétere a tömb, és az adatok száma.
   - Készítsen **függvényt**, amely kiszámítja a tárolt szövegek átlaghosszát. A függvény paramétere a tömb, és az adatok száma, a függvényérték az átlag, amit a **main** ír ki.
   - Készítsen **függvényt**, amely kiszámítja a szöveg hosszának és a benne lévő kisbetűk számának arányát. Majd megkeresi azt a szöveget, amelyben ez az arány a legalacsonyabb. A függvény paramétere a tömb, és az adatok száma, a függvényérték a megtalált szöveg sorszáma (indexe), az indexnek megfelelő szöveget a **main** írja ki.