Dolgozatok

Készítsen programot telefonszámok nyilvántartására. Az adatok: *név* és *telefonszám*, ezeket egy struktúra-tömbben tárolja. A tömböt tetszőleges számú elememmel töltse fel. A feltöltéshez és a kiíráshoz használjon **függvényt**.
A programot egészítse ki egy **függvénnyel**, amely telefonszám szerint keres a tömbben. A megtalált telefonszámhoz tartozó nevet írja ki a képernyőre.

Megoldás:

```
#include <stdio.h>
#include <iostream>
struct telefon{
	char nev[20];
	int telszam;
};
int beolvasas(telefon *);
void kiir(int, telefon *);
void keres(int, telefon *);
int main() {
	setlocale(LC_ALL, "hun");
	telefon tomb[100];
	int darab = beolvasas(tomb);
	kiir(darab, tomb);
	keres(darab, tomb);
}
int beolvasas(telefon *t) {
	int db = 0;
	printf("Kérek egy nevet: ");
	while (scanf("%[^\n]", t->nev) != EOF) {
		printf("Kérek egy telefonszámot: "); scanf("%d", &t->telszam);
		fflush(stdin);
		db++; t++;
		printf("Kérek egy nevet: ");
	}
	return db;
}
void kiir(int db, telefon *t) {
	for (int i = 0; i < db; t++, i++)
		printf("Név: %s\t Telefonszám: %d\n", t->nev, t->telszam);
}
void keres(int db, telefon *t){
	int szam;
	printf("Kérem a keresendő telefonszámot: "); scanf("%d", &szam);
	for (int i = 0; i < db; t++, i++) {
		if (t->telszam == szam) {
			printf("A %d számhoz tartozó név: %s", szam, t->nev);
			return;
		}
	}
	printf("Nincs ilyen telefonszám!\n");
}
```

Készítsen programot, amelyik tanulók nevét és érdemjegyét egy struktúra-tömbben tárolja. A tömböt legalább 5 adattal töltse fel. A feltöltéshez és a kiíráshoz használjon **függvényt**.

Feltöltés után **függvénnyel** számítsa ki az osztályátlagot.

Megoldás:

```
#include <stdio.h>
#include <iostream>
struct jegyek{
	char nev [80];
	int jegy;
};
int beolv(jegyek*);
float atlag(int, jegyek*);
void kiir(int, jegyek*);
int main()
{
	setlocale (LC_ALL, "");
	jegyek tomb[100];
	int db;
	db=beolv(tomb);
	kiir(db, tomb);
	printf("\nA jegyek átlaga: %.2f" ,atlag(db, tomb));
}
int beolv(jegyek *t){
	int n = 0;
	printf("Kérem az adatokat * végjelig!\n\n");
	printf("Kérek egy nevet: ");scanf("%[^\n]", t->nev);
	while (t->nev[0] != '*') {
		printf("Kérem a jegyet: "); scanf("%d", &t->jegy);
		fflush(stdin);
		n++; t++;
		printf("Kérek egy nevet: ");scanf("%[^\n]", t->nev);
	}
	return n;
}
void kiir(int db, jegyek *t){
	for (int i = 0; i < db; t++, i++)
		printf("Név: %s\t Telefonszám: %d\n", t->nev, t->jegy);
}
float atlag(int db, jegyek *t){
	float osszeg=0;
	for (int i = 0; i < db; t++, i++)
		osszeg+=t->jegy;
	return osszeg/db;
}
```

Készítsen programot fizetések nyilvántartására. Az adatok: *név* és *fizetés*, ezeket egy struktúra-tömbben tárolja. A tömböt tetszőleges számú elememmel töltse fel. A feltöltéshez és a kiíráshoz használjon **függvényt**. **Függvény** segítségével határozza meg és írja ki a legkevesebbet és a legtöbbet kereső nevét.

Megoldás:

```
#include <stdio.h>
#include <iostream>
#define IDIOT  "csaba"
struct fizetes{
	char nev[30];
	float fizet;
};
int beolv(fizetes*);
void kiir(int, fizetes*);
void szamol(int, fizetes*);
int main() {
	setlocale (LC_ALL, "");
	fizetes tomb[100];
	int db=beolv(tomb);
	kiir(db, tomb);
	szamol(db, tomb);
}
int beolv(fizetes *t){
	int db = 0;
	printf("Kérek egy nevet: ");
	while (scanf("%[^\n]", t->nev) != EOF) {
		printf("Kérem a fizetést: "); scanf("%f", &t->fizet);
		fflush(stdin);
		db++; t++;
		printf("Kérek egy nevet: ");
	}
	return db;
}
void kiir(int db, fizetes *t) {
	for (int i = 0; i < db; t++, i++)
		printf("Név: %s\t Fizetés: %f.2\n", t->nev, t->fizet);
}
void szamol(int db, fizetes *t) {
	int i=0, maxi=0, mini=0;
	float max = t[i].fizet;
	float min = t[i].fizet;
	for (int i = 1; i < db; i++){
		if(t[i].fizet > max) {
			max = t[i].fizet;
			maxi = i;}
		if(t[i].fizet < min) {
			min = t[i].fizet;
			mini = i;}
	}
	printf("\nA legtöbbet kereső személy neve: %s\n", t[maxi].nev);
	printf("\nA legkevesebbet kereso személy neve: %s\n", t[mini].nev);
}
```

Az objektumorientált programozás

A számítógépes programokkal a körülöttünk lévő világot és eseményeit próbáljuk modellezni. A számítógépes programozás fejlődése során – ahogy a számítógépek teljesítménye, „tudása” lehetővé tette – egyre inkább olyan programnyelvek készültek, amelyek jobban illeszkednek az emberi gondolkodáshoz. Manapság már nemcsak az adott eseményeket modellezük, hanem azok környezetét is. Hiszen egy esemény, tevékenység egy adott környezetben történi, függ a környezetétől. Ezért célszerű olyan programozási modellt alkalmazni, amelynél maga a programozási nyelv elemei és azok tulajdonságai, illetve viselkedésük is illeszkedik a környezethez.

Ezt a módszert valósítja meg az objektumorientált programozás (Object-Oriented Programming), röviden OOP.

Nézzünk példát az objektumokra:

- Játékprogramok esetén a szereplő (karakter) is egy objektum. Vannak tulajdonságai, pl. élet, erőnlét, ruházat, fegyverzet stb. Valamint különböző módon viselkedik, pl. adott irányban elmozdul, ugrik, lő stb.
- Az iskolában tanuló diákoknak vannak tulajdonságaik (adataik) pl. név, születési hely és idő, lakcím stb. A diák tevékenykedik (viselkedik) pl. a tanórán figyel, felel, beszélget, chattel stb. Ha a diák objektummodelljét szeretnénk elkészíteni, akkor ezeket az adatokat (tulajdonságokat) és tevékenységeket (viselkedést) kell felhasználnunk
- Egy grafikus felhasználói felület is objektumokat tartalmaz. Például egy parancsgombnak van helye, mérete, felirata, színe, valamint tartozik hozzá egy eseményleírás, amit akkor kell végrehajtani, mikor azt lenyomjuk (klikkelünk).

Az objektumok használatával egységbe foglaljuk (egységbezárás) az adatokat tartalmazó változókat és az azokon végzett műveletekhez szükséges függvényeket. A változókat tagváltozónak attribútumnak vagy tulajdonságnak nevezzük. Az objektum viselkedését a benne elhelyezkedő tagfüggvények (metódusok) határozzák meg.

Az objektumokat struktúraként vagy osztályként írjuk le. Az objektumok futási időben, a struktúra vagy az osztály példányosításával jönnek létre.

1. Egységbezárás

Példa:

Készítsünk egy struktúrát az SVGA (1024x768) grafikus képernyő tetszőleges pixelének (koordinátáinak) tárolására, annak beállításához és lekérdezéséhez szükséges metódusokkal.

Megbeszélés:

1. Standard input és output, *iostream* használata.
2. Metódusok leírása struktúrán belül és kívül, metódusok viselkedése példányosításkor.

Megoldás:

```
#include <iostream>
#define MAX_X 1024
#define MAX_Y 768
using namespace std;
struct point{
        int x;
        int y;
        int setx(int adat);
        int sety(int adat);
};
int point::setx(int adat){
        if(adat >= 0 && adat < MAX_X){      //Képernyő pont
                x = adat;
                return 0;
        }
        return -1;                  //A megadott érték a képernyőn kívül van
}
int point::sety(int adat){
        if(adat >= 0 && adat < MAX_Y){      //Képernyő pont
                y = adat;
                return 0;
        }
        return -1;                  //A megadott érték a képernyőn kívül van
}
int main(){
      point p1, p2;                 //Példányosítás
        p1.setx(100);
        p1.sety(200);
        p2.x = 5000;
        p2.y = 3000;
        cout << p1.x << " - " << p1.y << endl;
        cout << p2.x << " - " << p2.y << endl;
}
```

```
100 - 200
5000 - 3000

--------------------------------
Process exited after 0.08952 seconds with return value 0
Press any key to continue . . .
```

2. Adatrejtés

Azért, hogy a tagváltozóknak közvetlenül (metódus nélkül) ne lehessen értéket adni, alkalmazzuk az adatrejtést. Ez azt jelenti, hogy a tagváltozókhoz csak a metódusokon keresztül férhetünk hozzá.

Példa:

Módosítsuk az előző programot úgy, hogy a tagváltozókhoz ne férhessünk hozzá.

Megbeszélés:

1. Privát és publikus tulajdonságok, metódusok.
2. Struktúra és osztály közötti különbség

Megoldás:

```
#include <iostream>
#define MAX_X 1024
#define MAX_Y 768
using namespace std;
class point{
        int x;
        int y;
    public:
        int setx(int adat);
        int sety(int adat);
        int getx(){return x;}
        int gety(){return y;}
};
int point::setx(int adat){
        if(adat >= 0 && adat < MAX_X){
                x = adat;
                return 0;
        }
        return -1;
}
int point::sety(int adat){
        if(adat >= 0 && adat < MAX_Y){
                y = adat;
                return 0;
        }
        return -1;
}
int main(){
    point p1, p2;
        p1.setx(100);
        p1.sety(200);
        p2.setx(500);
        p2.sety(300);
        cout << p1.getx() << " - " << p1.gety() << endl;
        cout << p2.getx() << " - " << p2.gety() << endl;
}
```

```
100 - 200
500 - 300

--------------------------------
Process exited after 0.04859 seconds with return value 0
Press any key to continue . . .
```

3. Konstruktor, destruktor

A konstruktor olyan metódus, amely a példányosítással egyidőben, automatikusan elindul. Neve megegyezik az osztályéval, visszaadott értéke, így típusa sincs. A C-n alapuló nyelvekben lehetőség van azonos nevű függvények (metódusok) létrehozására (függvénytúlterhelés). Ilyenkor a paraméterlistának eltérőnek kell lennie.

A destruktor típus és paraméter nélküli metódus, így csak egy lehet belőle, neve: ~osztálynév. A program befejezése előtt automatikusan indul. Automatikusan általában az objektum élettartama során lefoglalt memóriát szabadítja fel, a megnyitott fájlokat lezárja.

Példa:

Módosítsuk az előző programot úgy, hogy az adatokat konstruktorral is megadjuk. Destruktorral írassuk ki a „Program vége” szöveget.

Megbeszélés:

1. Konstruktor tulajdonsága, paraméterátadás.
2. Destruktor tulajdonsága.

Megoldás:

Standard

```
#include <iostream>
using namespace std;
#define MAX_X 1023
#define MAX_Y 768
class point{
        unsigned int x;
        unsigned int y;
    public:
        point(){x=1; y=2;}
        point(unsigned int x, unsigned int y){this->x=x; this->y=y;}
        ~point(){cout<<"program vege"<<endl;}
        int setx(int adat);
        int sety(int adat);
        int getx(){return x;}
        int gety(){return y;}
};
int point::setx(int adat){
        if(adat >= 0 && adat < MAX_X){
                x = adat;
                return 0;
        }
        return -1;
}
int point::sety(int adat){
        if(adat >= 0 && adat < MAX_Y){
                y = adat;
                return 0;
        }
        return -1;
}
```

```
int main(){
        point p1, p2(40,50);
        cout<<p1.getx()<<" - "<<p1.gety()<<endl;
        cout<<p2.getx()<<" - "<<p2.gety()<<endl;
        p1.setx(100);
        p1.sety(200);
        p2.setx(500);
        p2.sety(300);
        cout<<p1.getx()<<" - "<<p1.gety()<<endl;
        cout<<p2.getx()<<" - "<<p2.gety()<<endl;
}
```

```
1 - 2
40 - 50
100 - 200
500 - 300
program vege
program vege

--------------------------------
Process exited after 0.04494 seconds with return value 0
Press any key to continue . . .
```

4. Öröklés

Amikor egy osztály forráskódja nem áll rendelkezésünkre, vagy nem akarjuk azt megváltoztatni, viszont szeretnénk új tulajdonságokkal kiegészíteni, akkor az öröklés a megoldás. Az öröklés segítségével egy osztályból több leszármazottat hozhatunk létre, amelyeket más és más tulajdonsággal ruházunk fel (polimorfizmus).

Példa:

Módosítsuk az előző programot úgy, hogy az adatok közé bevesszük a pont színkódját, és annak kezeléséhez metódusokat készítünk.

Megbeszélés:

1. Öröklés és annak tulajdonságai.
2. Protected típus.

Megoldás:

```
#include <iostream>
using namespace std;
#define MAX_X 1023
#define MAX_Y 768
class point{
    protected:
        unsigned int x;
        unsigned int y;
    public:
        int setx(int adat);
        int sety(int adat);
        void ki(){cout<<x<<" - "<<y<<endl;}
};
class color: public point{
        int c;
    public:
        void setc(unsigned short int adat){c=adat;}
        void ki(){cout<<x<<" - "<<y<<" - "<<c<<endl;}
};
int point::setx(int adat){
        if(adat >= 0 && adat < MAX_X){
                x = adat;
                return 0;
        }
        return -1;
}
int point::sety(int adat){
        if(adat >= 0 && adat < MAX_Y){
                y = adat;
                return 0;
        }
        return -1;
}
```

```
int main(){
    color p1;
    point p2;
        p1.setx(100);
        p1.sety(200);
        p2.setx(500);
        p2.sety(300);
        p1.setc(55);
        p1.ki();
        p2.ki();
}
```

```
100 - 200 - 55
500 - 300

--------------------------------
Process exited after 0.03454 seconds with return value 0
Press any key to continue . . .
```

Feladatok:

1. Készítsen objektum orientált programot, amely egy 10 elemű tömböt billentyűzetről feltölt, majd kiírja a tartalmát. A tömb az osztály tagváltozója, a beolvasás és kiírás metódusok legyenek! Hozzon létre a fenti osztályból egy leszármazott osztályt, amelyben a kiírást úgy módisítja, hogy az indexeket is megjeleníti.
2. Készítsen objektum orientált programot, amely egy karakterláncot (sztringet) beolvas a billentyűzetről, majd visszaírja a képernyőre. Hozzon létre a fenti osztályból egy leszármazott osztályt, amelyben megszámolja és kiírja a sztringben lévő kisbetűk számát. A tömb az osztály tagváltozója, a beolvasás és a kiírás metódusok legyenek!
3. Készítsen objektum orientált programot, amelyben egy karakterláncot konstruktor segítségével adunk meg. A karakterláncot destruktor írja ki a képernyőre.
4. Készítsen objektum orientált programot, amely egy karakterláncot (sztringet) beolvas a billentyűzetről, majd visszaírja a képernyőre. Hozzon létre a fenti osztályból egy leszármazott osztályt, amelyben megszámolja és kiírja a sztringben lévő számkarakterek számát. A tömb az osztály tagváltozója, a beolvasás és a kiírás metódusok legyenek!
5. Készítsen objektum orientált programot, amely egy téglatest adatait (három oldalának hosszát) billentyűzetről bekéri, majd kiszámítja és kiírja a felszínének értéket. Hozzon létre a fenti osztályból egy leszármazott osztályt, amely a térfogatot is kiírja. A téglatest paraméterei az osztály tagváltozója, a beolvasás és kiírás metódusok legyenek!
6. Készítsen objektum orientált programot, amelyben egy téglatest adatait (három oldalának hosszát) konstruktor segítségével, a példányosításkor adjuk meg. Egy metódussal kiszámítja a téglatest felszínét és térfogatát, majd az eredményt destruktor írja ki.
7. Készítsen objektum orientált programot, amely bekéri egy háromszög oldalainak hosszát, majd kiértékeli, hogy az adatokból szerkeszthető-e háromszög (bármely két oldal összege nagyobb a harmadiknál). Az oldalak hossza az osztály attribútuma, az értékét konstruktorral adja meg, a beolvasás és kiírás metódusok legyenek!
8. Készítsen objektum orientált programot, amelyben egy háromszög oldalainak hosszát konstruktor segítségével, a példányosításkor adjuk meg. Határozza meg, hogy az adatokból szerkeszthető-e háromszög (bármely két oldal összege nagyobb a harmadiknál). Az eredmény destruktorral irassa ki.
9. Készítsen objektum orientált programot, amely egy 10 elemű tömböt feltölt, majd kiszámítja és kiírja az elemek összegét. Hozzon létre a fenti osztályból egy leszármazott osztályt, amelyben az összeg helyett az átlagot írja ki. A metódus nevét ne változtassa meg! A tömb az osztály attribútuma, a beolvasás és kiírás metódusok legyenek!
10. Készítsen objektum orientált programot, amely egy henger adatait (sugár és magasság) bekéri, majd kiszámítja és kiírja a felszínének és térfogatának értéket. A henger paraméterei az osztály tagváltozója, a beolvasás és kiírás metódusok legyenek!
11. Készítsen objektum orientált programot, amelyben egy henger adatait (sugár és magasság) kostruktor segítségével adja meg, majd kiszámítja és destruktor segítségével kiírja a felszínének és térfogatának értéket.
12. Készítsen objektum orientált programot, amely adott hőmérsékleten meghatározza a víz halmazállapotát („jég”, „víz”, „gőz”) és kiírja azt. A hőmérséklet az osztály tagváltozója, a beolvasás és kiírás metódusok legyenek!
13. Készítsen objektum orientált programot, amely egy téglatest adatait, a három oldalának hosszát konstruktorral adja meg. Számítsa ki és írja ki a felszínének és térfogatának értéket. A téglatest paraméterei az osztály tagváltozója, a kiírás metódus legyen!