

5. Gyakorlat

Rövid elméleti összefoglaló

Felhasználó által definiált adattípusok:

- A **typedef** egy speciális tárolási osztály, mellyel érvényes típusokhoz szinonim nevet rendelhetünk.

```
typedef létező_típus felhasználói_típus;

typedef double tomb[10];
tomb x;
```

A *tomb* olyan tömbtípus, amelynek 10 eleme (0-9 között indexelve) van, és minden eleme valós.

- Struktúra – összetartozó információ tárolása**

A struktúra több tetszőleges típusú (kivéve a **void** és függvénytípust) adatok együttese. Általános formája:

```
struct nev { // típusnév                // diak struktúra definiálása
    típus1 tag;
    ...
} változo; // változó definíció

struct diak {
    char nev[20];
    int kredit;
    double atlag;
};
```

Definiáljunk néhány változót a *diak* típus felhasználásával:

```
diak d2, *d3;
```

A statikus helyfoglalású *d2* változó adattagjainak a *(.)* operátorral adunk értéket:

```
strcpy(d2.nev, "Szilard Laszlo"); d2.kredit = 80;
```

Dinamikus memória-foglalás szükséges a *d3* mutató számára, majd a mutatóval kijelölt memória-területet fel kell szabadítanunk!

```
d3 = new diak;      delete d3;
```

Kétféle módon adhatunk értéket a *d3* által kijelölt struktúra adattagjainak:

- a nyíl operátor (*->*) biztosítja az adattag elérését:

```
strcpy(d3->nev, "Komaromi Anna"); d3->kredit = 100; d2.atlag = 4.2;
```
- Amikor a pont operátort a *d3* által mutatót objektumra alkalmazzuk, a precedencia-szabályok miatt zárójelek között kell megadnunk a **d3* kifejezést:

```
(*d3).atlag = 5;
```

Karaktertömbnek az **strcpy()** függvénnyel adhatunk értéket, használatához a *cstring* fejlécállomány szükséges.

```
//MINTA5_01
#include <iostream>
#include <cstring>
using namespace std;
struct diak{
    char nev[20];
    int kredit;
    double atlag;
};

int main()
{ struct diak d1;    // C/C++
  diak d2, *d3;      // C++
  strcpy(d1.nev, " Kiss Endre"); d1.kredit = 50; d1.atlag = 3.8;
  strcpy(d2.nev, " Szilard Laszlo"); d2.kredit = 80;
  d2.atlag = 4.2;
  d3 = new diak;
  strcpy(d3->nev, "Komaromi Anna"); d3->kredit = 100; (*d3).atlag = 5;
  cout << d1.nev << " " << d1.kredit << " " << d1.atlag << endl;
  cout << d2.nev << " " << d2.kredit << " " << d2.atlag << endl;
  cout << d3->nev << " " << d3->kredit << " "
      << (*d3).atlag << endl;
  delete d3;
  cin.get();
  return 0;
}
```

A program futásának eredménye:

```
Kiss Endre 50 3.8
Szilard Laszlo 80 4.2
Komaromi Anna 100 5
```

```
//MINTA5_02
#include <iostream>
#include <cstring>
using namespace std;
struct diak{ char nev[20];
             int kredit;
             double atlag;
             }tanulo;
int main()
{ strcpy(tanulo.nev,"Kiss Zoltan");
  tanulo.kredit = 75;
  tanulo.atlag = 3.8;
  cout << tanulo.nev << " " << tanulo.kredit << " "
        << tanulo.atlag << endl;
  cin.get();
  return 0;
}
```

Használjuk a *diak* struktúra típussal definiált *tanulo* változót!

A program futásának eredménye:

Kiss Zoltan 75 3.8

- Használjuk a szabványos *hibastream* **cerr** objektumát, melyet az *iostream* fejlécállomány tartalmaz!

cerr << "Hibas az adat!";

```
//MINTA5_03
#include <iostream>
using namespace std;
struct teglalap{
    double a, b;
    double ter, ker;
};
int main()
{
    teglalap t;
    cout << "Teglalap oldalai: \n";
    cout << "A oldal: "; cin >> t.a;
    cout << "B oldal: "; cin >> t.b;
    if (t.a >= 0 && t.b >=0)
    { t.ter = t.a * t.b;
      t.ker = 2*(t.a + t.b);
      cout << "Teglalap terulete: " << t.ter << endl;
      cout << "Teglalap kerulete: " << t.ker << endl;
    }
    else
        cerr << "Hibas adat ";
    cin.get(); cin.get();
    return 0;
}
```

A struktúra deklarációját záró kapcsos zárójel mögé pontosvesszőt kell tenni.

```
struct teglalap{
    double a, b;
    double ter, ker;
};
```

Olvassuk be a téglalap két oldalának hosszát!
Csak akkor számítjuk ki a területet és a kerületet, ha a téglalap oldalai nagyobbak nullánál, ellenkező esetben hibajelzést adunk a **cerr** segítségével.

A program futásának eredménye:

```
Teglalap oldalai:
A oldal: 1
B oldal: 2
Teglalap terulete: 2
Teglalap kerulete: 6
```

- Struktúra feltöltése konstansokkal: `teglalap t = {1,2};` // *MINTA5_04 feladat*
- A dinamikus helyfoglalású struktúra létrehozása és megszüntetése (**new** és a **delete** operátorok)

A C++ nyelvben a szükséges helyfoglalást a **new**, és a felszabadítást pedig a **delete** operátorokkal végezhetjük el.

```
//MINTA5_05
#include <iostream>
using namespace std;
struct teglalap{
    double a, b;
    double ter, ker;
};
int main()
{
    teglalap *t;
    t = new teglalap;
    if (t != 0 )
    {
        cout << "Teglalap oldalai: \n";
        cout << "A oldal: "; cin >> t->a;
        cout << "B oldal: "; cin >> t->b;
    }
```

A *teglalap *t* változónak a **new** operátorral foglaljunk helyet a *teglalap* típust megadva. Innen tudja a **new**, hogy a struktúra hány bájtot igényel.

<pre> if (t->a >= 0 && t->b >=0) { t->ter = t->a * t->b; (*t).ker = 2*((*t).a + (*t).b); cout << "Tegalalap terulete: " << t->ter << endl; cout << "Tegalalap kerulete: " << t->ker << endl; } else { cerr << "Hibas adat "; } } else { cerr << "Nincs eleg memoria"; cin.get(); exit(-1); } delete t; cin.get(); cin.get(); return 0; } </pre>	<p>A dinamikus struktúra adatait kétféleképpen érhetjük el:</p> <p style="margin-left: 40px;">t->a vagy (*t).a</p> <p>A delete operátorral szabadítjuk fel a lefoglalt memória-területet:</p> <p style="margin-left: 40px;">delete t;</p> <p>A program futásának eredménye:</p> <p>Tegalalap oldalai: A oldal: 3 B oldal: 4 Tegalalap terulete: 12 Tegalalap kerulete: 14</p>
--	--

- Struktúraelemeket tartalmazó tömb használata. Példaként használjuk fel a *teglalap* struktúrát! A struktúraelemek számára a **new** operátorral foglalhatunk helyet a dinamikusan kezelt memória-területen, a *db* változó tartalmazza az elemek számát:

```
t = new teglalap[db];
```

A tömbelemekben tárolt struktúrára a pont operátor segítségével hivatkozunk:

```
t[i].ter = t[i].a * t[i].b;
```

Feladatként keressük meg a legnagyobb területű és a legkisebb kerületű téglalapot, és eredményként írassuk ki a téglalap oldalainak hosszát is.

A **delete** [] operátorral szabadítsuk fel a lefoglalt memória-területet:

```
delete[] t;
```

<pre> //MINTA5_06 #include <iostream> using namespace std; struct teglalap{ double a, b; double ter, ker; }; int main() { teglalap *t; int db, maxIndex = 0, minIndex = 0; double maxTer, minKer; cout << "Tegalalapok szama: "; cin >> db; t = new teglalap[db]; if (t != 0) { cout << "Tegalalap oldalai: " << endl; for (int i = 0; i < db; i++) { cout << "\nA oldal: "; cin >> t[i].a; cout << "B oldal: "; cin >> t[i].b; t[i].ter = t[i].a * t[i].b; t[i].ker = 2*(t[i].a + t[i].b); cout << "Tegalalap terulete: " << t[i].ter << endl; cout << "Tegalalap kerulete: " << t[i].ker << endl; } maxTer = t[0].ter; minKer = t[0].ker; for (int i = 1; i < db; i++) { if (t[i].ter > maxTer) { maxIndex = i; maxTer = t[i].ter; } } } } </pre>	<p>A program futásának eredménye:</p> <p>Tegalalapok szama: 3 Tegalalap oldalai:</p> <p>A oldal: 1 B oldal: 1 Tegalalap terulete: 1 Tegalalap kerulete: 4</p> <p>A oldal: 2 B oldal: 2 Tegalalap terulete: 4 Tegalalap kerulete: 8</p> <p>A oldal: 3 B oldal: 3 Tegalalap terulete: 9 Tegalalap kerulete: 12</p> <p>Legnagyobb területu teglalap A oldal: 3 B oldal: 3 Terulet: 9 Legkisebb keruletu teglalap A oldal: 1 B oldal: 1 Kerulet: 4</p>
--	--

```

    if (t[i].ker < minKer)
    {
        minIndex = i; minKer = t[i].ker;
    }
}
cout << "\nLegnagyobb területu teglalap\n";
cout << "A oldal: " << t[maxIndex].a;
cout << "    B oldal: " << t[maxIndex].b<< endl;
cout << "Terület: " << t[maxIndex].ter<< endl;
cout << "Legkisebb kerületu teglalap\n";
cout << "A oldal: " << t[minIndex].a;
cout << "    B oldal: " << t[minIndex].b<< endl;
cout << "Kerület: " << t[minIndex].ker<< endl;
}
else
{
    cerr << "Nincs eleg memoria"; cin.get();
    exit(-1);
}
delete[] t;
cin.get(); cin.get();
return 0;
}

```

Feladatok

1. Használjuk fel az alábbi definíciót és deklarációt!

```

struct adat {
    double a, b, c;
    int jel;
    double ered;
};

int main()
{
    adat w;
    cout << "1. adat : "; cin >> w.a;
    cout << "2. adat : "; cin >> w.b;
    cout << "3. adat : "; cin >> w.c;
    . . .
}

```

Olvassuk be a *w* struktúra három valós adatát valamint a *jel* (0 vagy 1) értékét! Ha a *jel* értéke 0, akkor a három adat számtani közepét, ha a *jel* értéke 1, a három adat mértani közepét számítsuk ki! Ellenőrizzük az adatokat! Adjunk hibajelzést, ha nem megfelelő adatot olvastunk. (GYAK5_1)

A program futásának eredményei:

1. adat : 1	1. adat : 1
2. adat : 2	2. adat : 2
3. adat : 3	3. adat : 3
Valasztas	Valasztas
szamtani kozep(0), mertani kozep(1): 0	szamtani kozep(0), mertani kozep(1): 1
Szamtani kozep: 2	Mertani kozep: 1.81712

2. Módosítsuk az 1. feladatot, használjunk dinamikus struktúrát! (GYAK5_2)

```

struct adat {
    double a, b, c;
    int jel;
    double ered;
};

int main()
{
    adat *w;
    w = new adat;
    cout << "1. adat : "; cin >> w->a;
    cout << "2. adat : "; cin >> w->b;
    cout << "3. adat : "; cin >> w->c;
    . . .
}

```

3. Használjuk az alábbi definíciót és deklarációt!

```

struct meres{
    int db;
    double MertErtek[50];
    double atlag;
    double szazalek;
    int EltereseKszama;
};

meres madat;

```

Kérdezzünk rá a mérési adatok számára (max. 50), olvassuk be a mérési értékeket, és egy a *szazalek* értéket. Számítsuk ki a mérési adatok átlagát, és számláljuk meg azokat az adatokat, melyek a megadott *szazalek*-nél nem térnek el jobban az átlagtól:

```

MertErtek[i] >= atlag - atlag*szazalek/100  &&
MertErtek[i] <= atlag + atlag*szazalek/100

```

(GYAK5_3)

A program futásának eredménye:

Meresi adatok szama: 5	Szazalek: 20
0. meresi adat: 2	Atlag: 4.42
1. meresi adat: 3.4	
2. meresi adat: 4	3.536 es 5.304 kozotti ertekek:
3. meresi adat: 5.2	2.elem: 4
4. meresi adat: 7.5	3.elem: 5.2
	Atlag +- szazalek eltereseK szama: 2

4. Használjuk az alábbi definíciót és deklarációt!

```

struct szovizs{
    char szo[30];
    int a_db;
    int egyeb;
};

int main()
{
    int i = 0;
    szovizs v;
    v.a_db= 0; v.egyeb = 0;
}

```

Olvassunk be egy szót és számláljuk meg, hogy mennyi 'a' betű, illetve egyéb karakter van benne! A vizsgálathoz használjunk **if** utasítást! (GYAK5_4)

A program futásának eredménye:

```

A vizsgalando szo: almafa
Az 'a' karakterek szama: 3
Egyeb karakterek szama : 3

```

5. Használjuk az alábbi definíciót és deklarációt!

```

struct szovizs{
    char szo[30];
    int a_db;
    int egyeb;
};

int main()
{
    int i = 0;
    szovizs *pv;
    pv = new szovizs;
    pv->a_db= 0; pv->egyeb = 0;
    . . .
}

```

Módosítsuk a 4. feladatot, használjunk dinamikus adatstruktúrát! A vizsgálathoz használjuk a **swicth** utasítást! (GYAK5_5)

A program futásának eredménye:

```

A vizsgalando szo: valamifele
Az 'a' karakterek szama: 2
Egyeb karakterek szama : 8

```