



# Bevezetés a programozásba

11. Előadás: Esettanulmány

# Függvényhívás

```
#include <iostream>
#include <cmath>
using namespace std;

double terület(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha,hb,hc;
    cin >> ha >> hb >> hc;
    double t = terület(ha,hb,hc);
    cout << "terület: " << t << endl;
    return 0;
}
```

# Függvények

```
double terület(double a, double b, double c)
```

- Szignatúra:
  - Visszatérési típus (a függvény típusának is nevezik)
  - Függvény neve
  - Zárójelben
    - Paraméter 1, ha van
    - Paraméter 2, ha van
    - ..
- A szignatúrából minden kiderül arról, hogyan kell használni a függvényt

# Strukturált programozás

- Programtervezési elv: a feladatot osszuk részfeladatokra, és függvényekben csoportosítsuk azokat
  - „dekompozíció”, „redukció”
- ```
int nagyonnehézfeladat(P1 P2 P3 ..) {  
    egyikkicsitkönnyebb(P2 P4 ..);  
    másikkicsitkönnyebb(P1 P2 ..);  
    ...  
}
```
- Top-down vagy Bottom-up felépítés
- 80-as évekig nagy szoftvereknél egyeduralkodó

# STL vector

- PLanG:
- *[semmi]*
- VÁLTOZÓK :  
t:EGÉSZ[10]
- t[0] := 1
- *[ilyet PLanG-ban nem lehet csinálni]*
- C++
- `#include <vector>`
- `vector<int> t(10);` vagy  
`vector<int> t(10,0);`
- `t[0]=1`
- ```
int osszeg(vector<int> t){  
    int sum=0;  
    for (int i=0;i<t.size()  
        ;i++){  
        sum+=t[i];  
    }  
    return sum;  
}  
...osszeg(t)...
```

# Érték szerinti paraméterátadás

```
#include <iostream>
using namespace std;

void fv(double a)
{
    a=0;
}

int main()
{
    double d;
    d=1;
    fv(d);
    cout << "eredmeny: " << d << endl;
    return 0;
}
```

Az eredmény:  
1

# Referencia szerinti paraméterátadás

```
#include <iostream>
using namespace std;

void fv(double & a)
{
    a=0;
}

int main()
{
    double d;
    d=1;
    fv(d);
    cout << "eredmeny: " << d << endl;
    return 0;
}
```

Az eredmény:  
0

# struct

```
#include <iostream>
using namespace std;

struct koord {
    double x,y;
};

int main()
{
    koord k;
    k.x=1.0; k.y=1.0;
    cout << "[" << k.x << ", " << k.y
        << "]" << endl;
    return 0;
}
```



# struct structban

```
struct ember {  
    string nev;  
    string lakcim;  
    int szuletesi_ev;  
};  
  
struct diak {  
    ember e;  
    vector<int> jegyek;  
};
```

# struct és függvények

```
struct pont {  
    double x,y;  
    string nev;  
};  
  
void kiir(pont p) {  
    cout << "[" << p.x << "," << p.y << "," <<  
p.nev << "]" << endl;  
}  
  
int main(){  
    pont p;  
    ...  
    kiir(p);  
    return 0;  
}
```

# Operátorok

```
struct pont {  
    double x,y;  
};  
  
bool operator==(pont a, pont b) {  
    return a.x==b.x && a.y == b.y;  
}  
  
int main()  
{  
    pont a={1.0,1.0}, b={0.0,0.0};  
    cout << (a==b);  
    return 0;  
}
```

# Operátorok

- Megvalósítható operátorok:
  - @a alakú: +   -   \*   &   !   ~   ++   --
    - **operator@ (A) {..}**
  - a@ alakú: ++   --
    - **operator@ (A, int) {..}**
  - a@b alakú :   +   -   \*   /   %   ^   &   |   <
   
>   ==   !=   <=   >=   <<   >>   &&   ||   ,
    - **operator@ (A, B) {..}**
- Az operátorok szerepét illik a nevükhöz és a szokásos jelentésükhöz igazodva használni.
- Alaptípusokra nem bírálhatjuk felül az operátort

# Tervezési kérdések

```
struct pont {  
    double x,y;  
};  
  
double tav(pont a, pont b) {  
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));  
}  
  
pont legtavolabbi(pont a, vector<pont> v) {  
    pont b=v[0];  
    double max=tav(a,b);  
    for (int i=1;i<v.size();i++) {  
        if (tav(a,v[i])>max) {  
            b=v[i];  
            max=tav(a,b);  
        }  
    }  
    return b;  
}
```

# Tervezési kérdések

```
struct pont {  
    double x,y,z;  
};  
  
double tav(pont a, pont b) {  
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)+  
        (a.z-b.z)*(a.z-b.z));  
}  
  
pont legtavolabbi(pont a, vector<pont> v) {  
    pont b=v[0];  
    double max=tav(a,b);  
    for (int i=1;i<v.size();i++) {  
        if (tav(a,v[i])>max) {  
            b=v[i];  
            max=tav(a,b);  
        }  
    }  
    return b;  
}
```

# Tagfüggvény

```
struct koord {  
    double x,y;  
    void olvas(istream &be) {  
        be >> x >> y;  
    }  
};  
  
int main() {  
    koord a;  
    a.olvas(cin);  
    cout << a.x << "," << a.y <<endl;  
    return 0;  
}
```

# Gyakorlatban

- Nézzünk meg egy példát
- Feladat: szöveges labirintusos játék
  - A játékban helyszínek vannak, amikről a felhasználó rövid leírást kap
  - Minden helyszín összeköttetésben állhat más helyszínekkel, amerre tovább lehet menni
  - A felhasználó minden helyszínen megadhatja, hogy merre megy tovább
  - Ha megérkezik a célba, nyer



# Ahogy azt régen csináltuk volna

```
#include <iostream>
using namespace std;

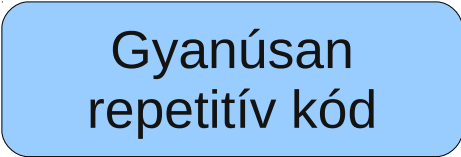
int main() {
    int h=1;
    while (h!=4) {
        if (h==1) {
            cout << "Otthon vagy. Mehetsz a parkba, vagy a bolthoz." << endl;
        }
        if (h==2) {
            cout << "A parkban vagy. Mehetsz a suliba, vagy haza." << endl;
        }
        if (h==3) {
            cout << "A boltnál vagy. Mehetsz a suliba, vagy haza." << endl;
        }
        cout << "Hova mész? (1:haza, 2:park, 3:bolt, 4:suli) : ";
        cin >> h;
    }
    cout << "Megérkeztél a suliba, éljen." << endl;
}
```

# Ahogy azt régen csináltuk volna

```
#include <iostream>
using namespace std;
```

```
int main() {
    int h=1;
    while (h!=4) {
        if (h==1) {
            cout << "Otthon vagy. Mehetsz a parkba, vagy a bolthoz." << endl;
        }
        if (h==2) {
            cout << "A parkban vagy. Mehetsz a suliba, vagy haza." << endl;
        }
        if (h==3) {
            cout << "A boltnál vagy. Mehetsz a suliba, vagy haza." << endl;
        }
        cout << "Hova mész? (1:haza, 2:park, 3:bolt, 4:suli) : ";
        cin >> h;
    }
    cout << "Megérkeztél a suliba, éljen." << endl;
}
```

Gyanúsan  
repetitív kód



```
graph TD
    A[Gyanúsan repetitív kód] --> B[if (h==1) { ... }
    A --> C[if (h==2) { ... }
    A --> D[if (h==3) { ... }
    E[Lehet csinálni] --> F[cin >> h;]
```

Lehet csinálni


# 1.1

```
#include <iostream>
using namespace std;
int main() {
    int h=1;
    while (h!=4) {
        if (h==1) {
            cout << "Otthon vagy. Mehetsz a parkba, vagy a bolthoz." << endl;
        }
        if (h==2) {
            cout << "A parkban vagy. Mehetsz a suliba, vagy haza." << endl;
        }
        if (h==3) {
            cout << "A boltnál vagy. Mehetsz a suliba, vagy haza." << endl;
        }
        cout << "Hova mész? (1:haza, 2:park, 3:bolt, 4:suli) : ";
        int hova;
        cin >> hova;
        if (h==1 && (hova==2 || hova==3) ||
            h==2 && (hova==1 || hova==4) ||
            h==3 && (hova==1 || hova==4)) {
            h=hova;
        } else {
            cout << "Arra nem lehet menni innen." << endl;
        }
    }
    cout << "Megérkeztél a suliba, éljen." << endl;
}
```


# 1.1

```
#include <iostream>
using namespace std;
int main() {
    int h=1;
    while (h!=4) {
        if (h==1) {
            cout << "Otthon vagy. Mehetsz a parkba, vagy a bolthoz." << endl;
        }
        if (h==2) {
            cout << "A parkban vagy. Mehetsz a suliba, vagy haza." << endl;
        }
        if (h==3) {
            cout << "A boltnál vagy. Mehetsz a suliba, vagy haza." << endl;
        }
        cout << "Hova mész? (1:haza, 2:park, 3:bolt, 4:suli) : ";
        int hova;
        cin >> hova;
        if (h==1 && (hova==2 || hova==3) ||
            h==2 && (hova==1 || hova==4) ||
            h==3 && (hova==1 || hova==4)) {
            h=hova;
        } else {
            cout << "Arra nem lehet menni innen." << endl;
        }
    }
    cout << "Megérkeztél a suliba, éljen." << endl;
}
```

Ez még mindig  
repetitív



Nehéz változtatni  
a térképet, ez túl  
bonyolult



# 1.2

```
#include <iostream>
#include <vector>
using namespace std;

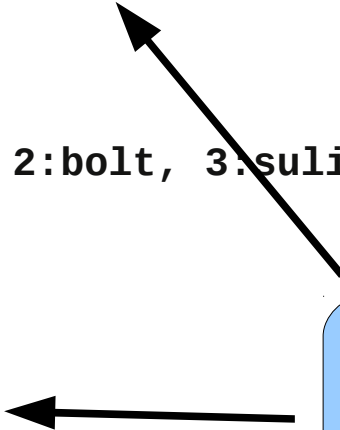
int main() {
    vector<string> terkep(4);
    terkep[0]="Otthon vagy. Mehetsz a parkba, vagy a bolthoz.";
    terkep[1]="A parkban vagy. Mehetsz a suliba, vagy haza.";
    terkep[2]="A boltnál vagy. Mehetsz a suliba, vagy haza.";
    terkep[3]="Megérkeztél a suliba, éljen.";
    int h=0;
    cout << terkep[h] << endl;
    while (h!=3) {
        cout << "Hova mész? (0:haza, 1:park, 2:bolt, 3:suli) : ";
        int hova;
        cin >> hova;
        if (
            h==0 && (hova==1 || hova==2) ||
            h==1 && (hova==0 || hova==3) ||
            h==2 && (hova==0 || hova==3)
        ) {
            h=hova;
        } else {
            cout << "Arra nem lehet menni innen." << endl;
        }
        cout << terkep[h] << endl;
    }
}
```

# 1.2

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<string> terkep(4);
    terkep[0]="Otthon vagy. Mehetsz a parkba, vagy a bolthoz.";
    terkep[1]="A parkban vagy. Mehetsz a suliba, vagy haza.";
    terkep[2]="A boltnál vagy. Mehetsz a suliba, vagy haza.";
    terkep[3]="Megérkeztél a suliba, éljen.";
    int h=0;
    cout << terkep[h] << endl;
    while (h!=3) {
        cout << "Hova mész? (0:haza, 1:park, 2:bolt, 3:suli) : ";
        int hova;
        cin >> hova;
        if (
            h==0 && (hova==1 || hova==2) ||
            h==1 && (hova==0 || hova==3) ||
            h==2 && (hova==0 || hova==3)
        ) {
            h=hova;
        } else {
            cout << "Arra nem lehet menni innen." << endl;
        }
        cout << terkep[h] << endl;
    }
}
```

Ez még  
mindig túl  
merev szerkezet



# „Bedrótozott” adat

- Azt nevezzük „bedrótozott”-nak, ami a programkódban fixen le van írva, pedig jobb lenne, ha nem így lenne
  - Adott esetben egy olyan program, ami a térképet fájlból olvasná be
- Általában cél, hogy a végleges programban már ne legyen olyan bedrótozott adat, ami a feladatban megváltozhat
- Ez viszont sokszor nem túl egyszerű, kicsit gondolkodni kell hozzá

# Jön a struct

- Azt láttuk, hogy ha szeretnénk kiemelni a bedrótozott adatot, akkor kezelni kell a következőket
  - tájékoztató szöveg kiírása az egyes helyszíneken
  - az összeköttetések a helyszínek között
  - esetleg a helyszínek neve
- Ezek közül az összeköttetések kezelése a legnehezebb



# std::vector

- Azon túl, hogy a primitív tömbnél jobb
  - paraméterátadásnál
  - kezdeti értékben
  - méretnek változót is kaphat, nem csak konstanst
- Néhány további hasznos szolgáltatás
  - push\_back(elem) : a tömb végére új elemet tehetünk
  - resize(újméret) : átméretezhető bármikor

# A helyszín reprezentációja

```
struct hely {  
    string nev;  
    string leiras;  
    vector<int> hova;  
};
```

- A név és a leírás szöveges változóval megoldható
- Az összeköttetések tömbben reprezentáljuk, aminek az elemei az innen elérhető helyszínek kódjai

# 2.0

```
#include <iostream>
#include <vector>
using namespace std;

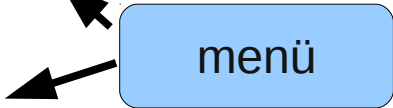
struct hely {
    string nev;
    string leiras;
    vector<int> hova;
};

int main() {
    vector<hely> terkep(4);
    terkep[0].nev="otthon";
    terkep[0].leiras="Otthon vagy. Mehetsz a parkba, vagy a bolthoz.";
    terkep[0].hova.push_back(1);
    terkep[0].hova.push_back(2);
    terkep[1].nev="park";
    terkep[1].leiras="A parkban vagy. Mehetsz a suliba, vagy haza.";
    terkep[1].hova.push_back(0);
    terkep[1].hova.push_back(3);
    terkep[2].nev="bolt";
    terkep[2].leiras="A boltnál vagy. Mehetsz a suliba, vagy haza.";
    terkep[2].hova.push_back(0);
    terkep[2].hova.push_back(3);
    terkep[3].nev="suli";
    terkep[3].leiras="Megérkeztél a suliba, éljen.";
    ... ..
```

# 2.0

```
... ..  
terkep[3].nev="suli";  
terkep[3].leiras="Megérkezteél a suliba, éljen.";  
int h=0;  
cout << terkep[h].leiras << endl;  
while (h!=3) {  
    cout << "Hova mész?" <<endl;  
    for (int i=0;i<terkep[h].hova.size();i++) {  
        cout << i+1 << ":" << terkep[terkep[h].hova[i]].nev << endl;  
    }  
    int melyik;  
    cin >> melyik;  
    if (melyik > 0 && melyik <= terkep[h].hova.size()) {  
        h=terkep[h].hova[melyik-1];  
    } else {  
        cout << "Hibás válasz." << endl;  
    }  
    cout << terkep[h].leiras << endl;  
}  
}
```

# 2.0

```
... ..  
terkep[3].nev="suli";  
terkep[3].leiras="Megérkeztél a suliba, éljen.";  
int h=0;  
cout << terkep[h].leiras << endl;  
while (h!=3) {  
    cout << "Hova mész?" <<endl;  
    for (int i=0;i<terkep[h].hova.size();i++) {  
        cout << i+1 << ":" << terkep[terkep[h].hova[i]].nev << endl;  
    }  
    int melyik;  
    cin >> melyik;   
    if (melyik > 0 && melyik <= terkep[h].hova.size()) {  
        h=terkep[h].hova[melyik-1];  
    } else {  
        cout << "Hibás válasz." << endl;  
    }  
    cout << terkep[h].leiras << endl;  
}  
}
```

## 2.0 értékelés

- Sikeresen szeparáltuk a konkrét adatokat az általános működéstől
- Felkészültünk arra, hogy fájlból töltsük be a konkrét adatokat
- Ezt azzal értük el, hogy
  - azonosítható helyszíneket használtunk
  - a helyszínek tulajdonságainak összetartozását is kifejeztük
- Így végül a főprogram lényegi része független lett az aktuális térképtől

# 2.1

adatok.txt

otthon

Otthon vagy. Mehetsz a parkba, vagy a bolthoz.

2 1 2

park

A parkban vagy. Hibba, vagy haza.

2 0 3

bolt

A boltnál vagy.

2 0 3

suli

Megérkeztél a su

0

program

```
void betolt(ifstream &be, hely & mit) {  
    be >> ws;  
    getline(be, mit.nev);  
    getline(be, mit.leiras);  
    int lsz,h;  
    be >> lsz;  
    for (int i=0;i<lsz;i++) {  
        be >> h;  
        mit.hova.push_back(h);  
    }  
}
```

# 2.1

adatok.txt

otthon

Otthon vagy. Mehetsz a parkba, vagy a bolthoz.

2 1 2

park

A parkban vagy. Hibba, vagy haza.

2 0 3

bolt

A boltban vagy. Hibba, vagy haza.

2 0 3

suli

Megérkeztél a suliba.

0

Ha esetleg még az előző sor végén állunk

program

```
void betolt(ifstream &be, hely & mit) {  
    be >> ws;  
    getline(be, mit.nev);  
    getline(be, mit.leiras);  
    int lsz, h;  
    be >> lsz;  
    for (int i=0; i<lsz; i++) {  
        be >> h;  
        mit.hova.push_back(h);  
    }  
}
```



# 2.1

```
int main() {
    ifstream f("adatok.txt");
    int hsz, cel;
    f >> hsz >> cel;
    vector<hely> terkep(hsz);
    for (int i=0;i<hsz;i++) betolt(f,terkep[i]);
    int h=0;
    cout << terkep[h].leiras << endl;
    while (h!=cel) {
        cout << "Hova méysz?" <<endl;
        for (int i=0;i<terkep[h].hova.size();i++) {
            cout << i+1 << ":" << terkep[terkep[h].hova[i]].nev <<
endl;
        }
        int melyik;
        cin >> melyik;
        if (melyik > 0 && melyik <= terkep[h].hova.size()) {
            h=terkep[h].hova[melyik-1];
        } else {
            cout << "Hibás válasz." << endl;
        }
        cout << terkep[h].leiras << endl;
    }
}
```

# További lehetőségek

- A „Térkép” fogalmának bevezetése
  - fájlnevből teljes térkép betöltés
  - összeköttetések kezelése
- Néhány lehetséges változás a feladatban
  - tárgyak kezelése, pl. csak akkor lehet egy összeköttetést használni, ha van nálunk kulcs
    - „Tárgy” és „Összeköttetés” típusok bevezetésével, a főprogram ciklusának érintése nélkül megoldható
  - grafikus megjelenítés
    - A „Hely” mezőinek kibővítésével megoldható egyes helyszínek eltérő képe

# Áttekintés

- Egy átlagos játékprogramon kevés programozó dolgozik, és nagyon sok designer
  - Nem lehet bedrótozni semmit
- Szét kell választani a konkrét adatokat az általános működéstől
- Az általános működés megfogalmazásához jó, ha magasabb szintű fogalmakat használhatunk
  - Megoldástér – Problématér
- Ebben segít a struct és a függvények...
- ...vagyis a saját típusaink, és azok műveletei