

# Bevezetés a programozásba

11. Előadás  
Esettanulmány

# PLanG → C++

- Nézzünk meg egy példát
- Feladat: szöveges labirintusos játék
  - A játékban helyszínek vannak, amelyekről 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

# 1.0 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;

    return 0;
}
```

# 1.0 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 boltba." <<endl;
        }
        if( h==2 ) {
            cout << "A parkban vagy. Mehetsz a suliba, vagy haza." <<endl;
        }
        if( h==3 ) {
            cout << "A boltban 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;

    return 0;
}
```

**Gyanúsan  
repetitív kód**



**Lehet csalni**



# 1.1 verzió: ne lehessen csalni

```
#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 >> 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;
    return 0;
}
```


# 1.1 verzió

```
#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 >> 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;
    return 0;
}
```

**Ez még mindig  
repetitív**



**Nehéz  
változtatni a  
térképet, ez túl  
bonyolult**



## 1.2 Használjunk tárolót

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

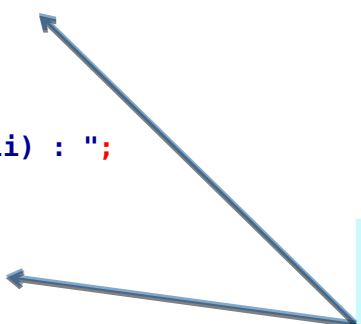
int main() {
    vector<string> terkep(4);
    terkep[0] = "Otthon vagy. Mehetsz a parkba, vagy a bolthoz.";
    terkep[1] = "A parkban vagy. Mehetsz a suliba, vagy haza.";
    terkep[2] = "A boltnál vagy. Mehetsz a suliba, vagy haza.";
    terkep[3] = "Megérkeztél a suliba, éljen.";
    int h = 0;
    cout << terkep[h] <<endl;
    while( h!=3 ) {
        cout << "Hova mész? (1:haza, 2:park, 3:bolt, 4: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;
    }
    return 0;
}
```

## 1.2 Használjunk tárolót

```
#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? (1:haza, 2:park, 3:bolt, 4: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;
    }
    return 0;
}
```

**Ez még mindig  
túl merev  
szerkezet**





## „Bedrótozott” adat

- Azt nevezzük „bedrótozott”-nak, ami a programkódban fixen le van írva
  - Például jelen esetben a program a térképet inkább fájlból lenne jó beolvasni
- Általában az a cél, hogy a végleges programban már ne legyen olyan bedrótozott adat, ami a feladatban megváltozhat
- Ez viszont sokszor nem túl egyszerű, át kell gondolni a dolgokat

## 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

## A helyszín reprezentációja

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

- A név és a leírás szöveges változóval megoldható
- Az összeköttetéseket tömbben reprezentáljuk, amelynek az elemei az innen elérhető helyszínek indexei lesznek

# 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 = "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 választás." <<endl;
    }
    cout << terkep[h].leiras <<endl;
}

return 0;
}
```

# 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 választás." <<endl;  
    }  
    cout << terkep[h].leiras <<endl;  
}  
  
return 0;  
}
```

Menü

## 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

2 1 2

park

A parkban vagy. Mehetsz

2 0 3

boltnál

A boltnál vagy. Mehetsz

2 0 3

suliba

Megérkeztél a suliba,

0

program

bolthoz.

```
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 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 választás." <<endl;
        }
        cout << terkep[h].leiras <<endl;
    }
    return 0;
}
```

# 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/bérlet...
    - „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
  - Problématér -- Megoldástér
- Ebben segít a struct és a függvények...
- ...vagyis a saját típusaink, és azok műveletei