



Eötvös Loránd Tudományegyetem  
Informatikai Kar

## **Alkalmazott modul: Programozás**

---

### **5. fejezet**

### **Procedurális programozás: alprogramok, paraméterátadás**

---

**Giachetta Roberto**

A jegyzet az ELTE Informatikai Karának 2015. évi  
Jegyzetpályázatának támogatásával készült

# Alprogramok

## Szükségessége

---

- A program terjedelme a feladat bonyolultságával arányos
  - egy adott bonyolultságon túl a főprogram olyan méretűvé válik, hogy áttekinthetetlen lesz a programozó számára
  - előfordulhatnak benne ismétlődő szakaszok, amelyek feleslegesen növelik a kód hosszát
  - sok esetben ugyanazt az algoritmust alkalmazzuk, csak más változókkal, mégis több példányban írjuk le
- A problémákra a megoldás a *kódrészletek kiemelése*, és külön programegységben történő elhelyezése, amelyeket a főprogramban behivatkozunk, ezek az úgynevezett *alprogramok* (szubrutinok)

# Alprogramok

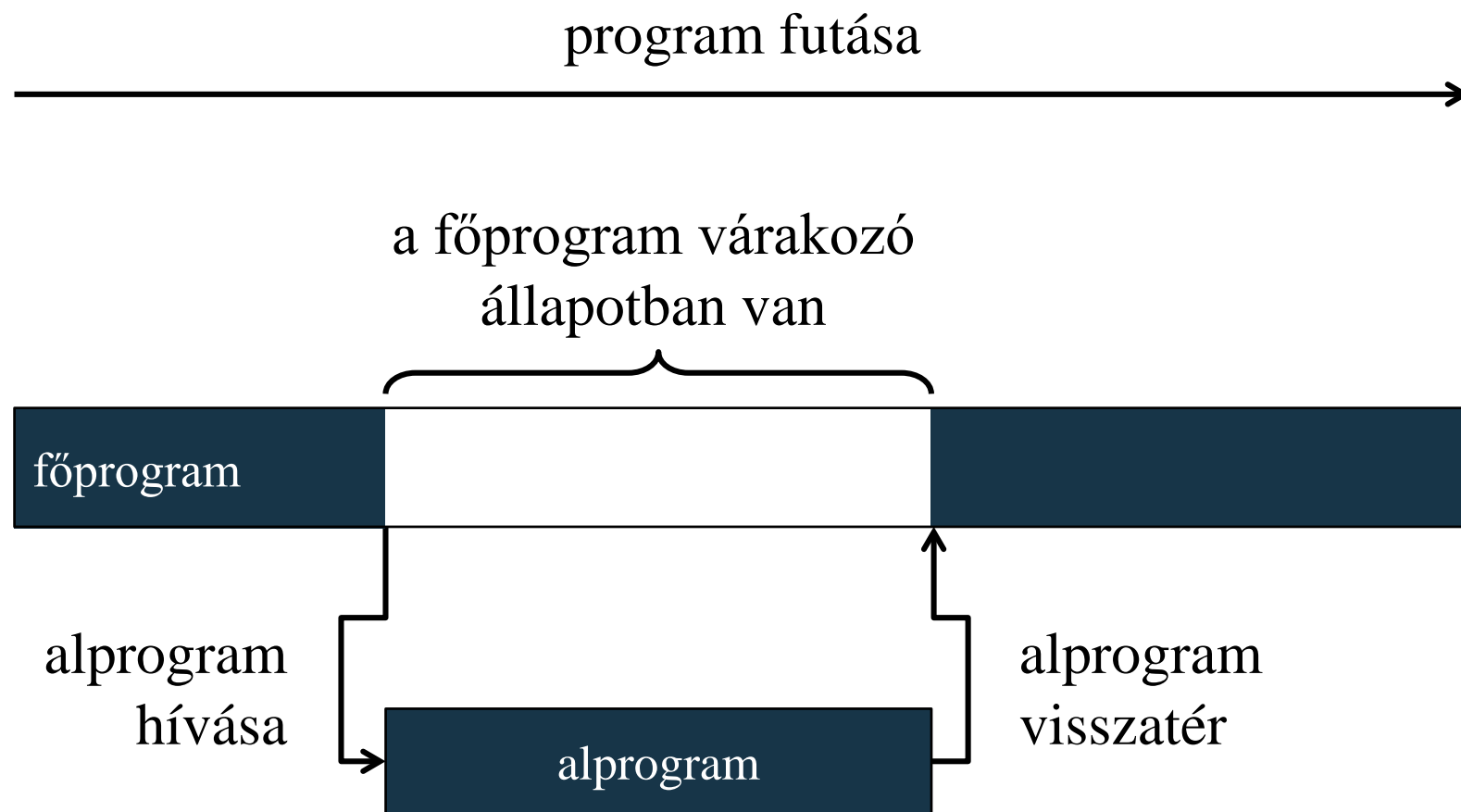
## Működése

---

- Az alprogramokat hívással futtatjuk
  - ekkor a főprogram átadja a vezérlést az alprogramnak, amely lefut, majd utána visszaadja a vezérlést
  - az alprogramokat tetszőleges sokszor futtathatóak
  - az alprogramok meghívhatnak további alprogramokat
- Az alprogramok *kommunikálhatnak egymással* (átadhatnak egymásnak értékeket):
  - *globális változók/konstansok*: bárhol elérhető értékek
  - *paraméterek*: direktben megadott értékek
  - *visszatérési érték*: alprogram által visszaadott érték

# Alprogramok

## Működése



# Alprogramok

## Működése

---

- Az alprogramok fajtái:
  - *eljárás*: egy utasítássorozatot hajt végre
  - *függvény*: egy számítást végez el, amelynek eredménye van, az eredményt pedig visszaadja a függvény meghívójának (ez a visszatérési érték)
- Az alprogram részei:
  - *deklarációs rész*: tartalmazza az alprogram nevét, függvény esetén a visszatérési érték típusát, illetve a paraméterek listáját
  - *alprogram törzse*: a hozzátartozó utasítássorozat (lehet külön a deklarációs résztől)

# Alprogramok

## C++-ban

- A C++-ban csak a függvények vannak implementálva, de lehet készíteni visszatérési érték nélküli függvényt is

- A C++ függvény szerkezete:

*<típus> <név>(<paraméterek>){ <utasítások> }*, ahol:

- a *típus* a visszatérési érték típusa (`void`, ha nincs)
  - a *név* az alprogram neve
  - a *paraméterek* a paraméterváltozók deklarációja
  - az *utasítások* az alprogram törzse
- Pl.:  
`void skip() {} // üres eljárás`  
`int one(){ return 1; } // 1-t visszaadó függvény`

# Alprogramok

## Hívás

- Alprogramokat meghívhatunk kifejezésben (amennyiben van visszatérési értéke), illetve végrehajthatunk utasításként, pl.:

...

```
skip(); // futtatás utasításként
```

```
one(); // a visszatérési érték elveszik
```

```
int a = one() + 10; // felhasználás kifejezésben
```

- A beépített függvények meghívása is ugyanígy történhet, pl.:

```
char c1 = toupper('a'); // nagy betűre alakítás
```

- Az alprogramban további, már deklarált alprogramokat is meghívhatunk, pl.:

```
void skip() {}
```

```
void bigSkip() { skip(); }
```

# Alprogramok

## Visszatérési érték

- Minden (visszatérési értékkel rendelkező) függvényben kell lennie egy érték visszaadásnak, erre a **return** utasítás szolgál
  - ez után már semmilyen utasítás nem fut le
  - többet is elhelyezhetünk a törzsben, de mindig pontosan egy fut le
  - a **return** után szerepelhet konstans érték, változó, kifejezés, amely típusának meg kell egyeznie a deklarációban megadott típussal
  - amennyiben nincs visszatérési érték, elhanyagolhatjuk, vagy nem írunk utána semmit, pl.:  

```
void skip() { return; }
```



# Alprogramok

## Példa

*Feladat:* Írjuk ki  $20 \times 20$  csillagot a képernyőre.

- használjunk egy alprogramot arra, hogy kiírjon egy sorban 20 csillagot, ez nyilván visszatérési érték nélküli lesz
- a főprogramban pedig meghívjuk ezt az alprogramot 20-szor

*Megoldás:*

```
void star(){ // star nevű alprogram
    for (int i = 0; i < 20; i++){
        cout << "*";
    }
    cout << endl;
} // alprogram vége
```

# Alprogramok

## Példa

*Megoldás:*

```
int main(){ // főprogram
    for (int i = 0; i < 20; i++){
        star(); // alprogram meghívása
    }
    return 0;
} // főprogram vége
```

# Alprogramok

## Példa

- A fenti alprogramot meg lehet úgy is valósítani, hogy adjon vissza 20 db csillagot **string** formájában, és a főprogram ezt írja ki a kimentre
  - a függvényen belül egy **string**-be beteszünk 20 csillagot, és a végén visszaadjuk annak értékét
  - ekkor a kiíratásnál kell meghívunk a függvény eredményét, amely a visszatérési értéként adott szöveget teszi ki a képernyőre

# Alprogramok

## Példa

*Megoldás:*

```
string star(){
    string stars = "";
    for (int i = 0; i < 20; i++)
        stars += "*";
    // hozzáveszünk 20 *-ot az üres szóhoz
    return stars; // visszaadjuk a csillagokat
}
int main(){
    for (int i = 0; i < 10; i++)
        cout << star() << endl;
    // a kiírásnál hívjuk meg a függvényt
    return 0;
}
```

# Alprogramok

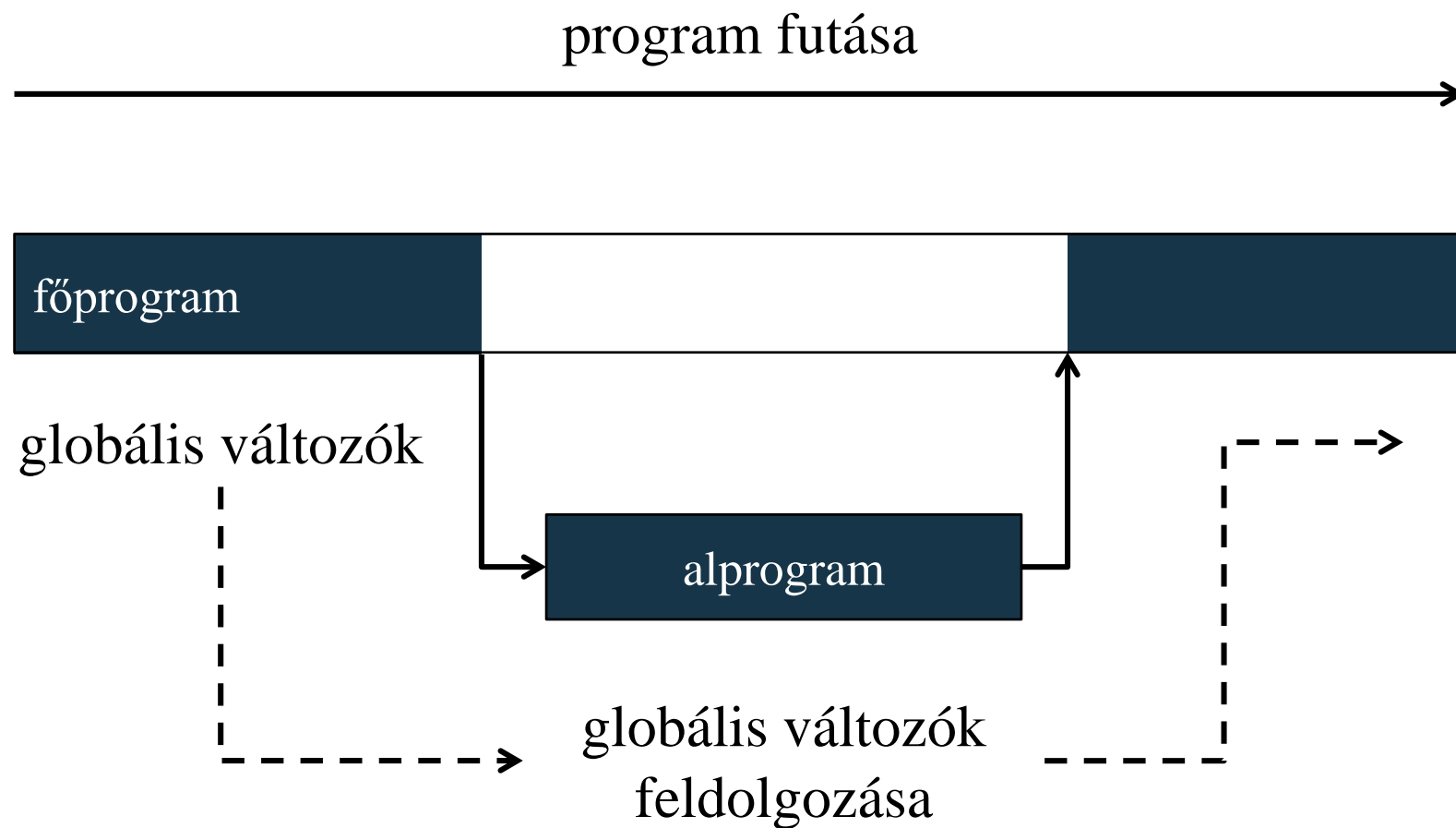
## Globális változók

---

- A *globális változók* olyan változók, amelyek minden programblokkon kívül vannak deklarálva (beleértve a főprogramét is), ezért értékük bárhol elérhető a programban
- Ezek a változók szintén a deklaráció pillanatától érvényesek, ezért célszerű őket az alprogramok előtt deklarálni, pl. a következő sorrend szerint:
  1. direktívák és névtér használat (**include**, **using**)
  2. globális változók
  3. alprogramok és főprogram
- Azokat a változókat, amelyek csak adott programblokkon belül érhetőek el, *lokális változóknak* nevezzük

# Alprogramok

## Globális változók



# Alprogramok

## Példa

*Feladat:* Adjunk össze alprogram segítségével két egész számot.

- használjunk két globális változót, amelyeket összegét a függvény visszaadja
- a főprogramban végezzük a bekérést és a kiíratást

*Megoldás:*

```
int a, b; // globális változók
```

```
int sum() // a és b összeadása  
{  
    return a + b;  
}
```

*Megoldás:*

```
int main(){
    cout << "Az első szám: ";
    cin >> a;
    cout << "A második szám: ";
    cin >> b;

    cout << "A számok összege: " << sum()
        << endl; // összeg kiíratása
    return 0;
}
```



# Alprogramok

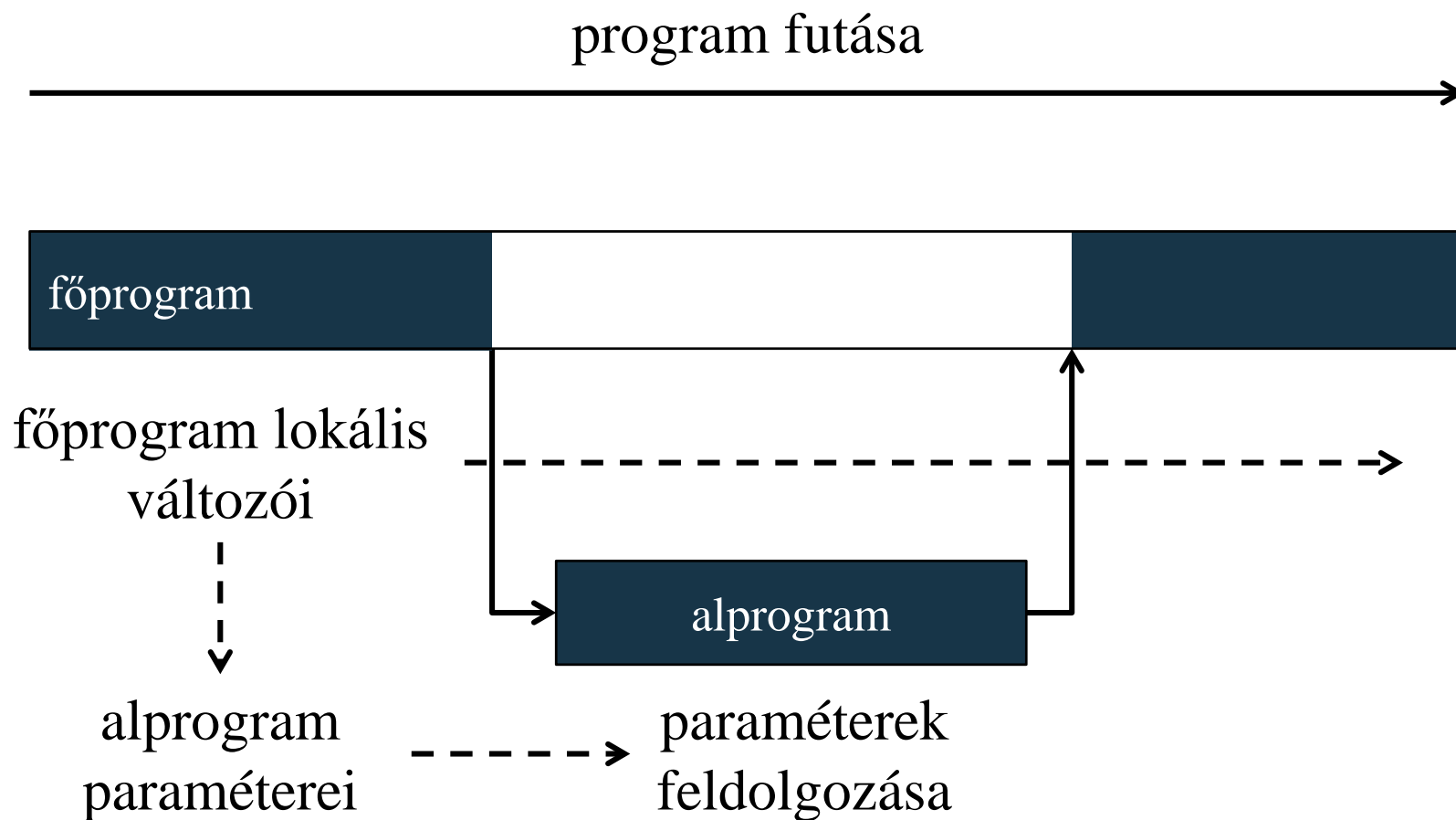
## Paraméterátadás

---

- A globális változók használata számos veszélyt hordoz magában, ezért célszerű őket elkerülni
  - a változókat bármely programrész elérheti, módosíthatja a másik tudta nélkül
  - a teljes programban láthatóak, ezért több forrásfájl esetén fellelphetnek névütközések
- A *paraméterátadás* lehetőséget ad arra, hogy lokális változókkal kommunikáljunk az alprogramok között
  - paraméterátadáskor az értékek a hívó lokális változóiból átkerülnek a paraméterekbe, amelyek tulajdonképpen a hívott alprogram lokális változói lesznek

# Alprogramok

## Paraméterátadás



# Alprogramok

## Paraméterátadás

---

- A paraméterátadásban két változó vesz részt:
  - *aktuális paraméter*: amit átadunk az alprogramnak a meghíváskor, lehet konstans, változó, kifejezés
  - *formális paraméter*: ami az alprogram deklarációs részében adunk meg, és lokális változóként használjuk az alprogramban
- Az aktuális és formális paraméterek típusának kompatibilisnek kell lennie
- Az aktuális paraméterek neve megegyezhet a formális paraméterekével, de ettől függetlenül mindenképpen más lokális változót jelentenek

# Alprogramok

## Példa

*Feladat:* Adjunk össze alprogram segítségével két egész számot.

- használjunk paraméterátadást, két paraméterre lesz szükségünk (x, y), amelyeknek két lokális változót (a, b) adunk át
- a főprogramban végezzük a bekérést és a kiíratást

*Megoldás:*

```
int sum(int x, int y)
    // formális paraméterek deklarálása
{
    return x + y; // használatuk
}
```

# Alprogramok

## Példa

*Megoldás:*

```
int main(){
    int a, b;
    cout << "Az első szám: ";
    cin >> a;
    cout << "A második szám: ";
    cin >> b;
    cout << "A számok összege: " << sum(a, b)
        << endl;
    // az aktuális paraméterek értéke átkerül a
    // formálisba, azaz:
    // x = a; y = b;
    return 0;
}
```

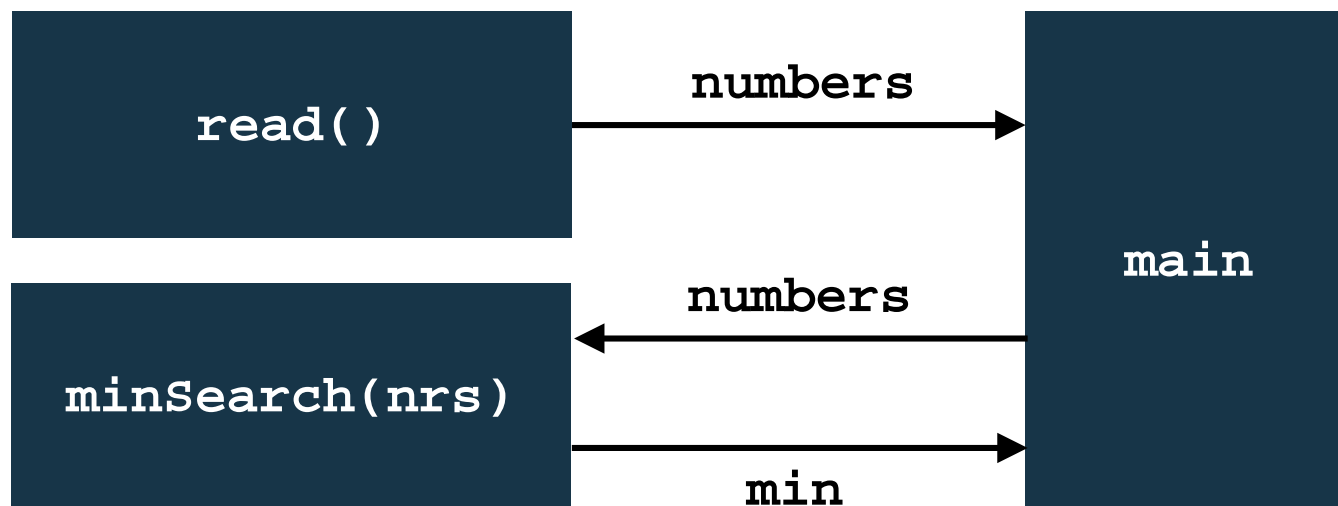
*Feladat:* Keressük meg egy 10 elemű egész számsorozat legkisebb elemét.

- tegyük a minimumkeresést, illetve a beolvasást alprogramba (függvénybe), a kiírást a főprogramba
- használjunk intelligens tömböt az adatok kezelésére
- a beolvasás (**read**) visszatérési értéként adja meg a beolvasott sortozatot (**numbers**)
- a minimumkeresés (**minSearch**) paraméterként kapja meg a sortozatot, és visszaadja a minimális elemet (**min**)

# Alprogramok

## Példa

*Tervezés:*



# Alprogramok

## Példa

*Megoldás:*

```
vector<int> read(){  
    // egész számok tömbje a visszatérési érték  
  
    vector<int> nrs(10); // 10 elemű tömb  
  
    for (int i = 0; i < nrs.size(); i++) {  
        cout << "A(z) " << i+1 << ". szám: ";  
        cin >> nrs[i];  
    }  
  
    return nrs;  
}
```



*Megoldás:*

```
int minSearch(vector<int> nrs){  
    // egész számok tömbje a paraméter  
  
    int min = nrs[0]; // maximumkeresés tétele  
    for (int i = 1; i < nrs.size(); i++)  
        if (nrs[i] < min)  
            // ha kisebb, mint az eddigi minimum,  
            // megváltoztatjuk  
            min = nrs[i];  
  
    return min;  
}
```

*Megoldás:*

```
int main(){
    vector<int> numbers = read();
    // lefuttatjuk a beolvasást

    cout << "A minimum: " << minSearch(numbers)
    << endl;
    // lefuttatjuk a minimumkeresést

    return 0;
}
```

# Alprogramok

## Alapértelmezett paraméterek

- Lehetőségünk van az alprogramoknak alapértelmezett paramétereket adni:

```
<típus> <név>(<paraméter> = <alapérték>,  
               <paraméter> = <alapérték>, ... ){  
    <utasítások>  
}
```

- ilyenkor híváskor nem kell megadnunk az összes paramétert, amit nem adunk meg, az alapértékre helyettesítődik be
- ha egy paraméternek adunk alapértéket, akkor az utána lévő összesnek kell

# Alprogramok

## Alapértelmezett paraméterek

---

- Pl.:

```
string first(string source, int nr = 1) {  
    // a szöveg elejének lekérdezése  
    return source.substr(0, nr);  
    // részszoveget veszünk, és azt adjuk vissza  
}  
...  
string s = "hello world";  
string sub1 = first(s, 5);  
    // ekkor sub1 == "hello"  
string sub2 = first(s, 1);  
    // ekkor sub2 == "h"  
string sub3 = first(s); // ugyanaz, mint az előző  
    // ekkor sub3 == "h"
```

- *Álneveknek (alias-knak)* nevezzük azokat az azonosítókat, amelyeket már létező változóhoz rendelünk, és így újra elnevezzük őket
  - a létrehozást követően a két azonosító ugyanazt a változót jelöli, módosítás hatására mindenhol változik az érték
  - az álnevek típusának meg kell egyeznie az eredeti változó típusával, a neve tetszőleges lehet
- A C++-ban az álneveket az & operátorral jelöljük meg:  
`<típus> <változó 1>;`  
`<típus> &<változó 2> = <változó 1>;`  
`// a változó 2 lesz a változó 1 álneve`

# Alprogramok

## Álnevek egyszerű változókra

- Egy változóra tetszőlegesen sok álnevet létrehozhatunk

- Pl.:

```
char x = 'a';
```

```
char &y = x;
```

```
// ráállítunk egy y álnevet az x-re
```

```
cout << y << endl;
```

```
// ez kiírja az 'a' értéket
```

```
x = 'b';
```

```
cout << y << endl; // ez kiírja a 'b' értéket
```

```
y = 'c';
```

```
cout << x << endl; // ez kiírja a 'c' értéket
```

# Alprogramok

## Álnevek tömbökre

- Lehetőségünk van tömbökre is ráállítani álneveket, de ekkor a \* operátort használunk:  
*<típus> <változó 1>[<méret>];*  
*<típus> \*<változó 2> = <változó 1>;*
- Ténylegesen a következő történik:
  - mivel minden változót a memóriában tárolunk, rendelkezik egy memóriabeli címmel, amit lekérdezhetünk
  - az & és \* operátorokkal létrehozott változók megkapják a már létező változók memóriabeli címét
  - értékmódosításkor ugyanarra a memóriaterületre fognak írni, ezért ugyanazt az értéket fogják módosítani

# Alprogramok

## A paraméterátadás iránya

---

- A paraméterátadás során nem csupán a hívott alprogram irányába, de a hívó irányába is adhatunk át értékeket, ezt a *paraméterátadás iránya* határozza meg, amely lehet:
  - *bemenő*: az aktuális paraméterek átadódnak az alprogram formális paramétereinek annak hívásakor
  - *kimenő*: az alprogram állítja be a formális paraméterei értékeit, majd a hívás végeztével (az alprogram futásának végén) az értékek átadódnak az aktuális paraméterekbe
  - *be- és kimenő paraméter*: az aktuális paraméterek átadódnak a formális paramétereknek a híváskor, majd a hívás végeztével a módosított értékek visszaadódnak az aktuális paraméterekbe



# Alprogramok

## A paraméterátadás módja

---

- Az irány mellett a paraméterátadás különböző módszerekkel történhet, amelyek hatással lehetnek az irányra is
- *Érték szerinti paraméterátadás:*
  - az aktuális paraméter értékét átadjuk a formális paraméternek a hívás pillanatában
  - a formális paraméter ezután azzal az értékkel fog dolgozni
  - az alprogramban hiába módosítjuk az aktuális, vagy a formális paramétert, az nincs hatással az aktuálisra, vagyis annak értéke megőrződik
  - eddig ezt használtuk

# Alprogramok

## A paraméterátadás módja

- pl.:

```
int GCD(int a, int b){ // érték szerint
    // euklideszi algoritmus
    while (a != b)
        if (a > b) a -= b;
        else b -= a;
    return a; // visszatérési érték
}
```

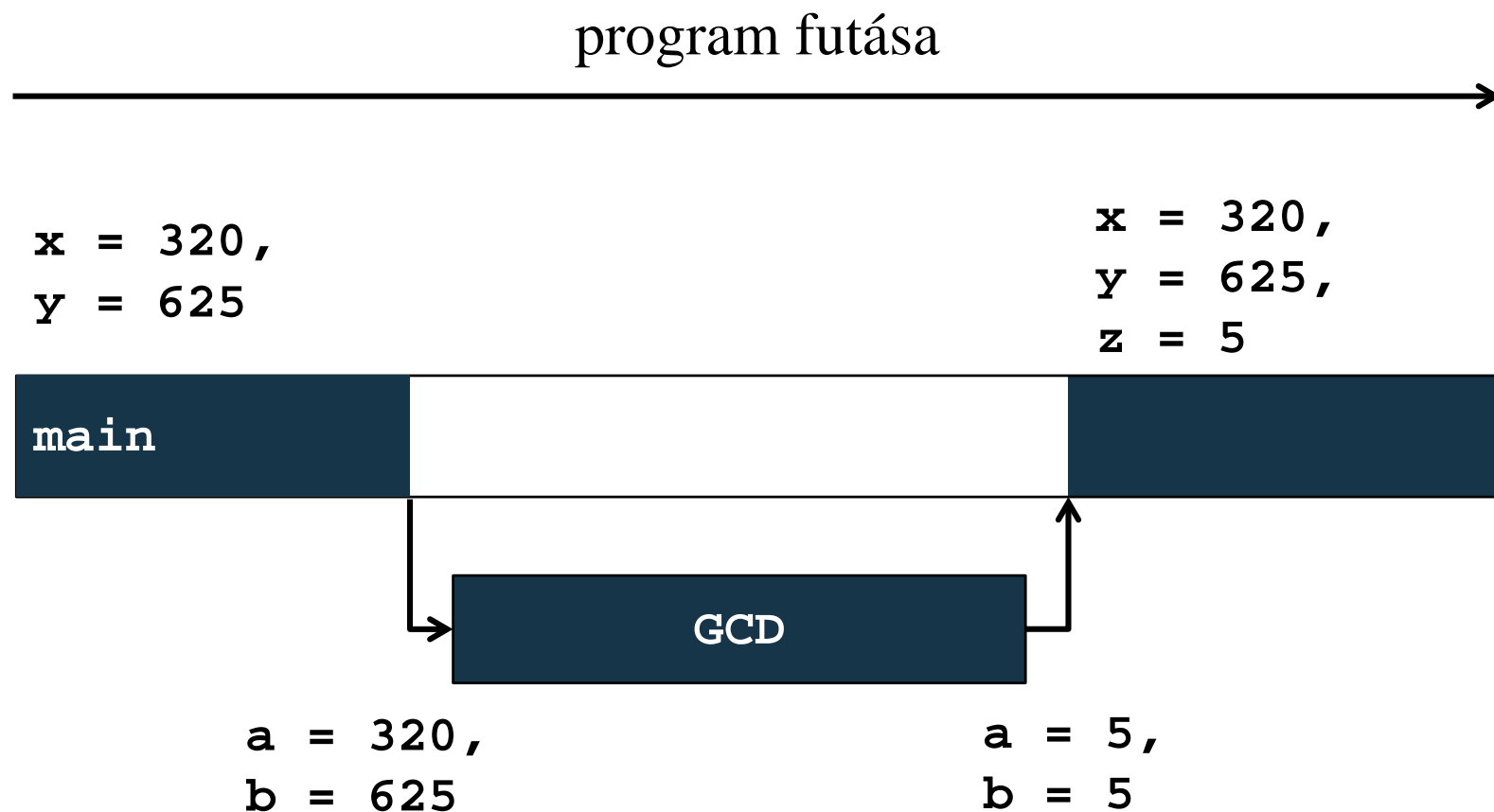
```
int x = 320, y = 625, z;
```

```
z = GCD(x, y);
```

```
// a bemenő paraméterek módosítása nem lesz
// hatással a és b értékére
```

# Alprogramok

## A paraméterátadás módja



- *Cím szerinti paraméterátadás:*
  - az aktuális paraméter memóriacímét adja át a formális paraméternek, a formális paraméter csak a változó memóriacímét tárolja, és az megváltoztathatatlan, viszont a memóriacímen található érték megváltoztatható
  - az alprogramban új értéket adhatunk a változónak, és az az érték visszakerül a főprogramba
  - a C++-ban ezt meg lehet valósítani álnevek használatával úgy, hogy formális paraméterként álneveket definiálunk, így az egyszerre módosul az aktuális paraméterrel
  - ezzel be- és kimenő paraméterátadást valósíthatunk meg

# Alprogramok

## A paraméterátadás módja

- pl.:

```
void GCD(int& a, int& b){ // cím szerint
    // euklideszi algoritmus
    while (a != b)
        if (a > b) a -= b;
        else b -= a;
} // nem kell visszatérési érték
```

```
int x = 320, y = 625;
```

```
GCD(x, y);
```

```
// a és b is módosítják az értéküket, így
// mindkettőben meglesz az eredmény, de a
// régi értékek elvesznek
```

# Alprogramok

## A paraméterátadás módja

program futása

```
graph LR; main[main] --> gcd[GCD]; gcd --> main;
```

**x** = 320,  
**y** = 625

**x** = 5,  
**y** = 5

**main**

**GCD**

**a** = 320,  
**b** = 625

**a** = 5,  
**b** = 5

# Alprogramok

## A paraméterátadás módja

---

- *Eredmény szerinti paraméterátadás*: megegyezik az érték szerinti átadással, de a formális paraméter értéke a hívás után visszakerül az aktuális paraméterbe
- *Név szerinti paraméterátadás*: egy kifejezés pontosan átadódik, és a formális paraméter hívásakor mindig kiértékelődik a benne használt változók értékének függvényében
- A C++ ez utóbbi kettőt nem támogatja, így a következő lehetőségeink vannak:
  - csak bemenő paraméter érték szerinti átadással
  - be- és kimenő paraméter cím szerinti átadással

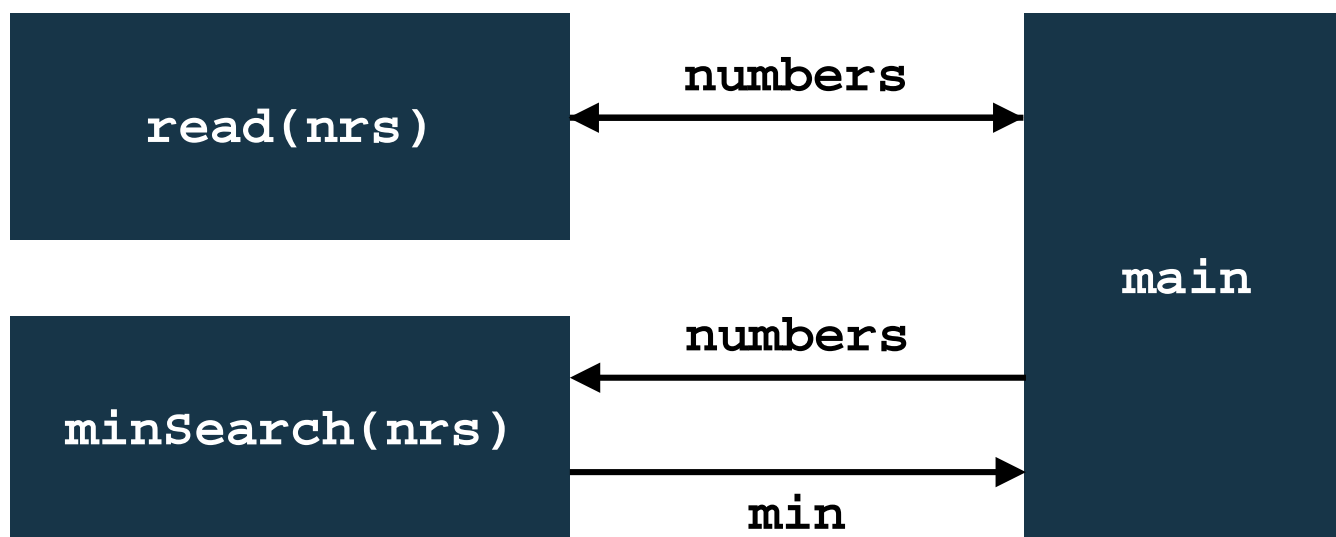
# Alprogramok

## Példa

*Feladat:* Módosítsuk a minimumkeresést úgy, hogy cím szerinti paraméterátadással kezeljük a tömböt.

- a tömböt kívül hozzuk létre, és paraméterben adjuk át

*Tervezés:*





# Alprogramok

## Példa

*Megoldás:*

```
void read(vector<int>& nrs){  
    // egész számok tömbje a paraméter cím szerint  
    // átadva  
  
    for (int i = 0; i < nrs.size(); i++) {  
        cout << "A(z) " << i+1 << ". szám: ";  
        cin >> nrs[i];  
    }  
}
```

# Alprogramok

## Példa

*Megoldás:*

```
int main(){
    vector<int> numbers(10);
    read(numbers); // lefuttatjuk a beolvasást

    cout << "A minimum: " << minSearch(numbers)
         << endl;
    // lefuttatjuk a minimumkeresést

    return 0;
}
```

*Feladat:* Írjuk ki egy számnak a rákövetkezőjét, de ellenőrizzük azt is, hogy számot adott-e meg a felhasználó.

- a beolvasást és az ellenőrzést végezzük külön alprogramban (**readInt**), amely megadja, hogy sikeres volt-e a beolvasás, illetve magát a számot
- az alprogram szövegfolyam segítségével alakítja át a bemenetet (közvetlenül a paraméterbe), és visszaadja, hogy sikeres volt-e az átalakítás
- mivel ez két visszaadott érték, a számot paraméterben adjuk vissza cím szerinti paraméterátadáással

# Alprogramok

## Példa

*Megoldás:*

```
bool readInt(int& numInp) // beolvasó függvény
    // visszatérési érték a beolvasás sikeressége
    // paraméter a beolvasott szám
{
    string textInp; cin >> textInp;
    stringstream sstr;
        // szövegfolyamot használunk a konverzióra
    sstr << textInp; // beírjuk a szöveget
    sstr >> numInp; // kiolvasunk egy számot
    return (!sstr.fail());
        // visszaadjuk, hogy sikeres volt-e az
        // átalakítás
}
```

# Alprogramok

## Példa

*Megoldás:*

```
int main()
{
    int numInp;

    cout << "Kérek egy számot: ";
    if (readInt(numInp))
        // ha sikeres volt a beolvasás
        cout << ++numInp << endl;
    else
        cout << "Nem szám!";
    return 0;
}
```

# Alprogramok

## Újrafelhasználhatóság

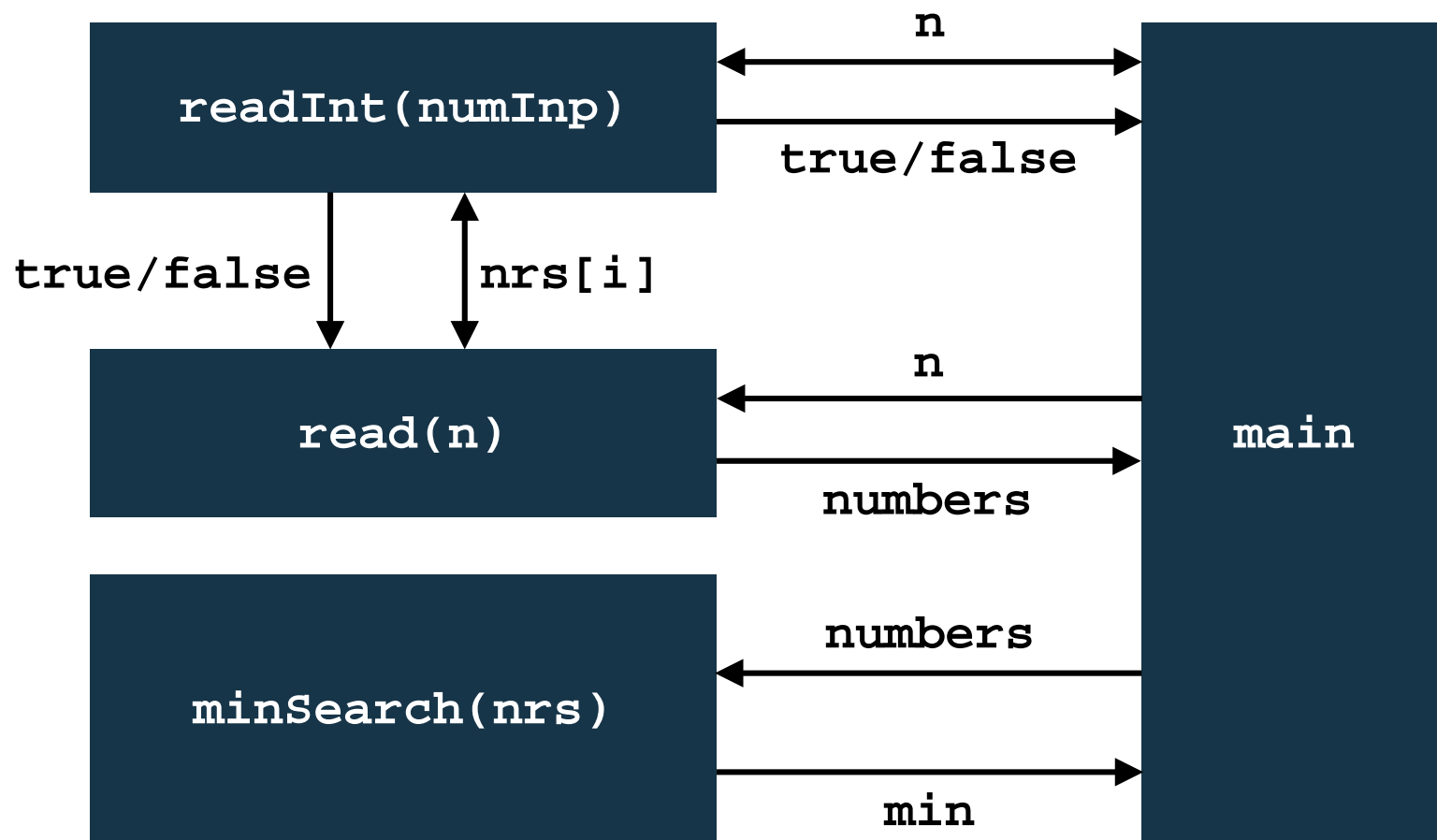
---

- Az alprogramok egyik jelentős előnye az *újrafelhasználhatóság*, hiszen tetszőleges sokszor futtathatjuk le őket
  - ugyanakkor az alprogramok törzse rögzített, így mindig ugyanaz az algoritmus fut le
- A paraméterek használata növelheti az újrafelhasználhatóságot, mivel nem csak a bemenő adatokat, de más, az algoritmus futását befolyásoló értékeket is átadhatunk paraméterben
  - így rugalmasabbá tehetjük az alprogram alkalmazhatóságát
  - fontos, hogy a paramétereket mindig ellenőrizzük
  - pl. bemenet hossza, vizsgálandó érték

*Feladat:* Módosítsuk a minimumkeresést úgy, hogy meg lehessen adni az elemek számát (ami így 0 is lehet).

- a beolvasó alprogram (**read**) megkapja az elemek számát, és ennek megfelelő méretű vektort ad vissza
- felhasználhatjuk az előző ellenőrző függvényt (**readInt**), és ezzel ellenőrizhetjük a beolvasott adatokat, azaz:
  - a tömb elemeit, amelyek csak egész számok lehetnek
  - a tömb méretét, amely csak pozitív egész szám lehet (bár 0 méretű vektor engedélyezett, azon nem lehet minimkeresést futtatni)

*Tervezés:*





*Megoldás:*

```
vector<float> read(int n){  
    // a számok száma a paraméter  
    // egész számok tömbje a visszatérési érték  
    vector<float> nrs(n); // n elemű tömb  
  
    for (int i = 0; i < nrs.size(); i++) {  
        cout << "A(z) " << i+1 << ". szám: ";  
        while (!readInt(nrs[i])) { // ellenőrzés  
            cout << "Nem egész szám! Kérem újra: ";  
        }  
    }  
    return nrs;  
}
```

# Alprogramok

## Példa

*Megoldás:*

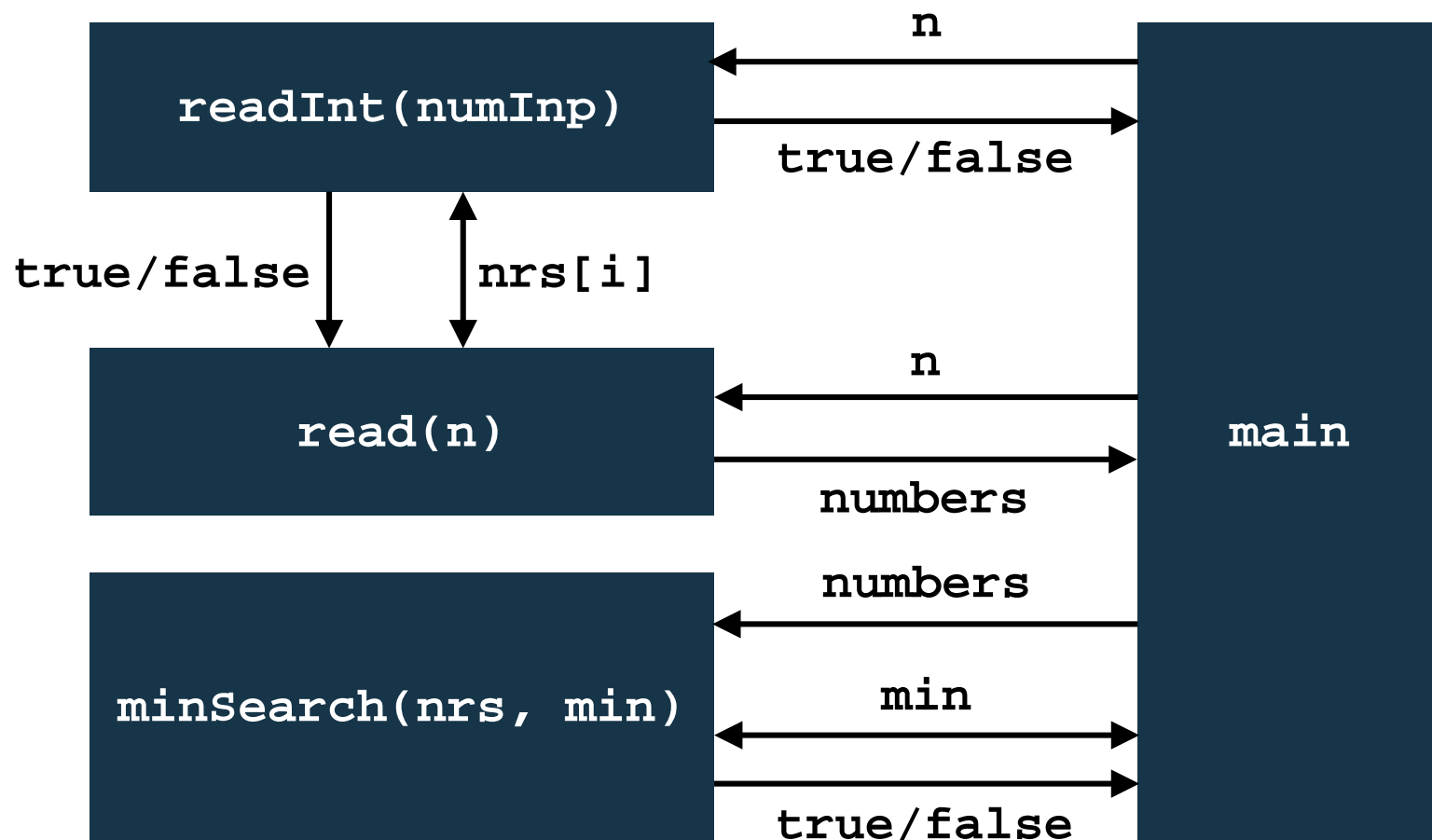
```
int main(){
    int n;
    cout << "Elemek száma: ";
    while (!readInt(n) || n <= 0){
        // ellenőrzés

        cout << "Nem pozitív egész szám!
                Kérem újra: ";
    }
    vector<int> numbers = read(n);
    // lefuttatjuk a beolvasást
    ...
}
```

*Feladat:* Módosítsuk a minimumkeresést úgy, hogy alprogramjaink teljes mértékben hibátűrők legyenek.

- előfordulhat, hogy a beolvasás (**read**) negatív értéket kap, amelyre nem hozható létre vektor, amennyiben ez előfordul, korrigáljuk a méretet 0-ra
- előfordulhat, hogy a minimumkeresés üres vektort kap paraméterben, ekkor ne végezzük el a keresést
  - ehhez be kell vennünk egy feltételt, amely jelzi, hogy sikeres volt-e a keresés
  - mivel így már két eredménye van a keresésnek, a minimumot adjuk vissza paraméterben

*Tervezés:*



# Alprogramok

## Példa

*Megoldás:*

```
vector<int> read(int n) {  
    vector<int> nrs(n); // n elemű tömb  
  
    if (n < 0)  
        // negatív méretre nem hozhatunk létre  
        // vektort  
        n = 0; // korrigáljuk a paramétert  
  
    ...  
  
    return nrs;  
}
```

# Alprogramok

## Példa

*Megoldás:*

```
bool minSearch(vector<int> nrs, int& min) {  
    // visszatérési értékben adjuk meg, sikeres  
    // volt-e a művelet, paraméterben adjuk vissza  
    // a minimumot  
  
    {  
        if (nrs.size() == 0)  
            // ha üres a vektor, nem tudunk minimumot  
            // keresni  
            return false;  
  
        ...  
        return true; // sikeres volt a keresés  
    }  
}
```

*Feladat:* Keressük meg egy sorban hány ‘a’ betű, illetve ‘b’ betű található, addig olvassuk be a sorokat, amíg üres sort nem írunk be.

- készítsünk függvényt, amely megszámolja, hány található az adott karakterből egy sorban (számlálás tétele), ezt hívjuk meg a főprogramban (paraméterben átadjuk az aktuális sort, valamint a karaktert), amennyiszer szükséges
- a soronként beolvasást a **getline** függvénnnyel végezzük, a ciklusfeltételben, és azonnal ellenőrizzük, hogy nem üres-e a sor

# Alprogramok

## Példa

*Megoldás:*

```
int countChars(string s, char ch)
    // az alprogramnak megadjuk a vizsgálandó
    // karaktert is
{
    int c = 0; // számlálás tétele
    for (int i = 0; i < s.length(); i++)
        if (s[i] == ch)
            c++;
    return c; // visszaadjuk az eredményt
}
```



# Alprogramok

## Példa

*Megoldás:*

```
int main(){
    string line;
    int nr_a, nr_b;
    while(getline(cin, line) && line.length() > 0){
        // amíg van bemenet és nem üres a sor
        nr_a = countChars(line, 'a');
        nr_b = countChars(line, 'b');
        // meghívás különböző paraméterekkel
        cout << "a-k száma: " << nr_a
              << ", b-k száma: " << nr_b << endl;
    }
    return 0;
}
```

# Alprogramok

## A főprogram paraméterei

---

- A főprogram (**main** függvény) is paraméterezhető
  - a főprogramnak átadott aktuális paraméterek a parancssorban megadott, fájlnev utáni szavak lesznek
  - bármennyi, bármilyen paramétert megadhatunk neki, amelyet a főprogrammal ki tudunk értékelteni
  - pl.: `./a.out ../szamok.txt 75 13 joprogi`
- A főprogram formális paraméterei:
  - `int argc`: mondja meg, hogy hány argumentummal indították el a programot
  - `char* argv[ ]`: tartalmazza az argumentukat egymás után

# Alprogramok

## A főprogram paraméterei

---

- A főprogramot ennek megfelelően a következőképpen írhatjuk:  
`int main(int argc, char* argv[]){ ... }`
- Az `argv[<index>]` elemet lekérdezve megtudhatjuk a valahányadik argumentumot (paramétert)
  - a tömb `argc` hosszú, tehát a tömb `0...argc-1` elemeit kérdezhetjük le
  - `argc` értéke minden esetben legalább egy, ugyanis az első argumentum maga a program neve (elérési úttal)
  - pl. ha a program futtatása `./a.out`, akkor  
`argv[0] = "home/groberto/a.out"` lehet

# Alprogramok

## Példa

*Feladat:* Olvassunk ki egy sort egy fájlból, amelyet paraméterként adtunk meg. Ha nem jó paramétert adunk meg, vagy egyáltalán nem adunk paramétert, akkor kérjük be billentyűzetről. Továbbá írjuk ki, hogy milyen néven fut a program.

- ha az első megnyitás nem sikerült, kérjük be billentyűzetről a fájlnevet, amíg sikertelen a megnyitás

*Megoldás:*

```
int main(int argc, char* argv[]){  
    cout << argv[0] << " program fut..." << endl;  
    // kiírjuk a programnevet
```

*Megoldás:*

```
ifstream f;
string name;
if (argc > 1) { // van paraméter
    cout << "Megnyitás: " << argv[1] << endl;
    f.open(argv[1]);
} else { // nincs paraméter, bekérjük
    cout << "File neve: ";
    cin >> name;
    f.open(name.c_str());
}
while (f.fail()){
    // ha nem sikerült az első megnyitás
    cout << "Hibás fájlnev!" << endl;
```

*Megoldás:*

```
f.clear();  
cout << "File neve: ";  
cin >> name;  
f.open(name.c_str());  
} // addig bekérjük, amíg nem sikerül megnyitni  
  
string line;  
getline(f, line);  
cout << "Az első sor: " << line << endl;  
f.close();  
return 0;  
}
```