Programozás 1

Blokkstruktúra
Változók láthatósága
Függvények végrehajtása

Blokkstruktúra a C nyelvben

- A C nyelvben blokkon **{ }** zárójelpárba zárt blokkot értünk.

- Egy C program blokkjai mellérendeltségi és alárendeltségi viszonyban vannak.
- Ezt a viszonyt az ide vonatkozó szabályokkal együtt blokkstruktúrának nevezzük.

Blokkstruktúra a C nyelvben

1. Sorrendiségi szabály
   - A program egy adott helyén csak olyan azonosítóra hivatkozhatunk, amely a hivatkozás helyét megelőzően deklarálva lett. Változó azonosító, függvény azonosító és típus azonosító megjelenése helyén deklaráltnak minősül.
2. Egyediségi szabály
   - Egy adott blokkban deklarált minden azonosító csak egyszer deklarálható, nem számítva az alárendelt blokkokat.

Blokkstruktúra a C nyelvben

```
main() {
  int b, b;
  a = 0;
  printf("? "); scanf("%d", &b);
  b = f(b);
  printf("%d (%d)\n", b, a);
}

int a;

int f(int n) {
  a++;
  return (n>1) ? f(n-1) + f(n-2) : 1;
}
```

Itt egyszerre két változót deklarálunk, és mindkettőnek a **b** nevet adjuk. Ha ezt megtehetnénk, hogy döntené el a fordító, hogy melyik esetben melyik változóval kellene dolgoznia?

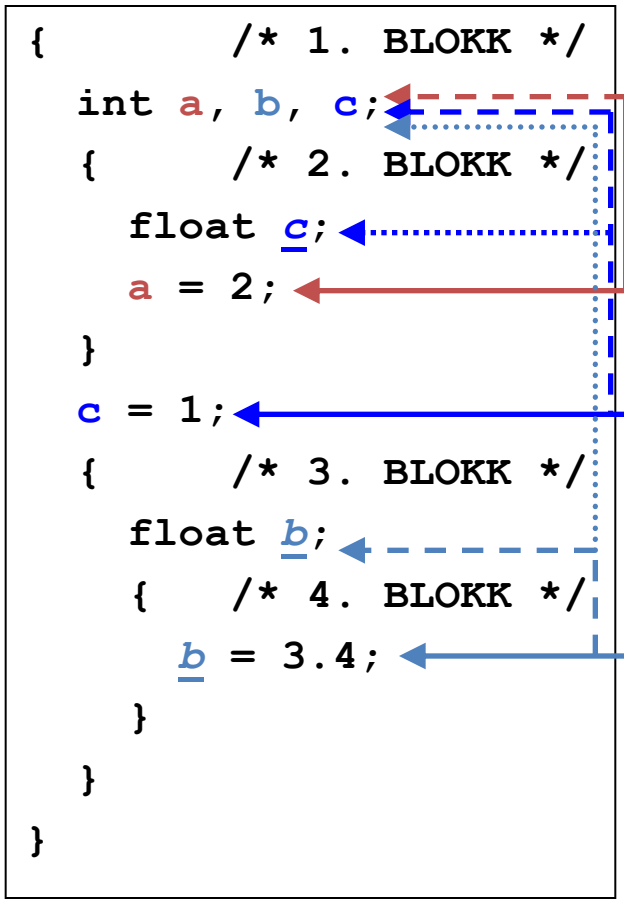
Itt pedig használjuk az **f** és az **a** azonosítókat, holott még azt sem tudjuk, hogy melyik micsoda.

Itt már jogos az **f** és az **a** használata is, mert ezeken a pontokon már mindkettő deklarálva van, még ha **f** definíciója még nincs is teljesen befejezve.

Blokkstruktúra a C nyelvben

3. Láthatósági szabály
   - Egy B1 blokkban deklarált A azonosító akkor és csak akkor látható (hivatkozható) egy B2 blokkban, ha teljesül az alábbi két feltétel:
     a.) B1 megegyezik B2-vel, vagy B2 alárendeltje B1-nek és az A azonosító előbb van deklarálva, mint B2
     b.) Az A azonosító nincs deklarálva egyetlen olyan C blokkban sem, amely alárendeltje B1-nek és amelynek B2 alárendeltje (beleértve azt, hogy B2 vagy megegyezik C-vel)

Blokkstruktúra a C nyelvben



A fent deklarált **a** változó használható ezen a ponton, hiszen:
- a 2. blokk alárendeltje az 1. blokknak,
- **a** előbb van deklarálva mint a 2. blokk,
- nincs olyan blokk a kettő között, amelyben **a** deklarálva lenne.

Ezen a ponton az 1. blokkban deklarált **c** változó használható. A 2. blokkban deklarált **c** ezen a ponton NEM használható, hiszen az 1. blokk nem alárendeltje a 2. blokknak. Így az itt leírt **c** az 1. blokkban deklarált **c** változót jelenti.

Az 1. blokkban deklarált **b** ezen a ponton NEM használható, mert a 4. blokk ugyan alárendeltje az 1. blokknak, és később fordul elő, mint **b** 1. blokkbeli deklarációja, de van közben egy 3. blokk, amelynek a 4. blokk szintén alárendeltje, és amiben **b** szintén deklarálva van.
A 3. blokkban deklarált **b** viszont használható, így azután a 4. blokkban leírt **b** a 3. blokk változóját jelenti.

Blokkstruktúra a C nyelvben

- Azon blokkok összességét, amelyből egy A azonosító látható, az A azonosító hatáskörének nevezzük.
- Egy azonosítót lokálisnak nevezünk egy blokkra nézve, ha az azonosító az adott blokkban van deklarálva.
- Azt mondjuk, hogy egy A azonosító globális egy B blokkra nézve, ha nem B-ben van deklarálva, de látható B-ben.

Blokkstruktúra a C nyelvben

- A blokkstruktúra alapján látható, hogy a C nyelvben vannak úgynevezett lokális változók, sőt általában ezeket használjuk.
- Látható azonban az is, hogy a programfájlban deklarált programegységek globálisak az összes függvénydeklarációra nézve, vagyis ezek minden blokkban láthatóak a deklarálásuktól kezdve az újradeklarálásukig.
- Ezeket csak nagyon indokolt esetben szoktuk használni.

Tárolási osztályok

- auto
  - Az auto az automatikus memóriafoglalásra utal. Ezt az alapszót nem szoktuk kiírni.
- static
  - A lokális változó értéke megmarad a blokk végrehajtása után is és az újabb függvényművelet végrehajtásakor a megőrzött érték felhasználható.
- extern
  - A programegységet csak deklaráljuk, de nem ebben a fájlban kerül definiálásra. A szerkesztőprogram feladata lesz a külső hivatkozás feloldása.

Tárolási osztályok

```
/* A static változót mutatjuk be.
 * 1997. November 7.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

void stat();

main()
{
    int i;                      /* ciklusváltozó */
    for (i = 0; i < 5; i++) {
        stat();
    }
}
```

Tárolási osztályok

```
void stat()
{
    int ideiglenes = 1;
    static int allando = 1;
    printf("ideiglenes = %d allando = %d\n",
            ideiglenes,     allando);
    ideiglenes++;
    allando++;
}
```

Tárolási osztályok

- A példaprogram ezt írja ki:

```
ideiglenes = 1 allando = 1
ideiglenes = 1 allando = 2
ideiglenes = 1 allando = 3
ideiglenes = 1 allando = 4
ideiglenes = 1 allando = 5
```

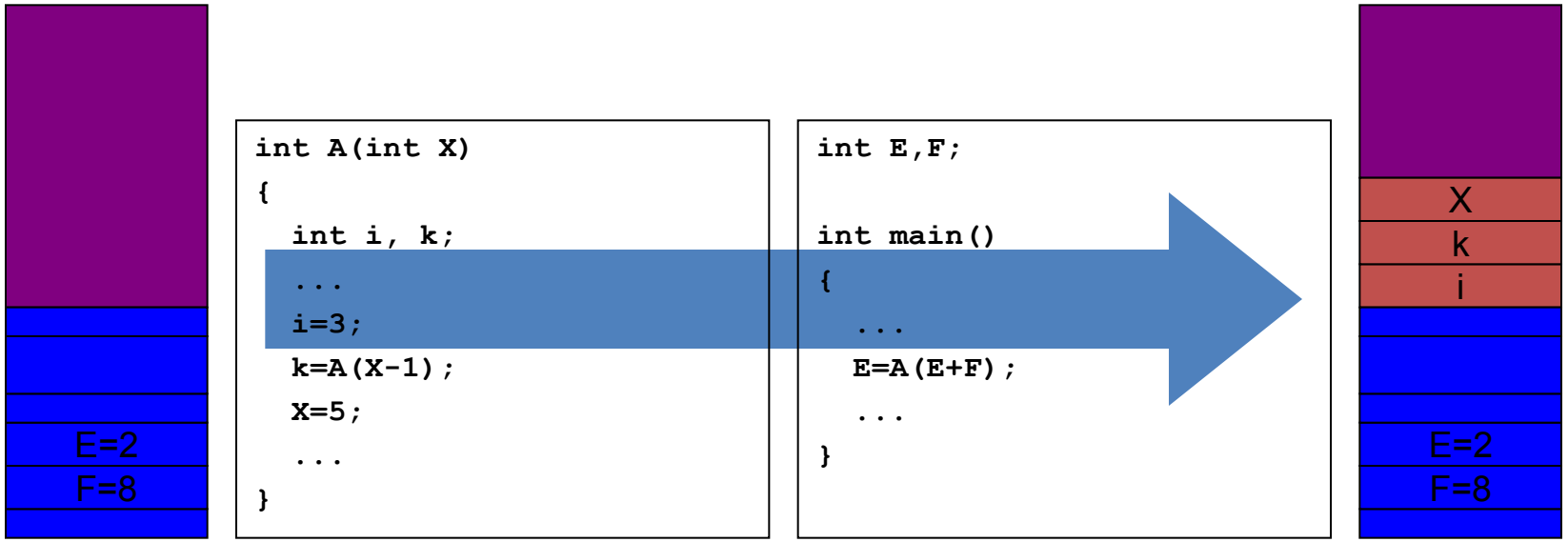
Végrehajtás

- A C nyelven blokknak hívjuk a program **{ }** zárójelek közötti részét is, és itt is lehet programelemeket deklarálni.
- Egy ilyen blokk végrehajtása a következő három tevékenységet jelenti:
  1.) Memória helyfoglalás a blokk lokális változói számára.
  2.) A blokk utasításrészének végrehajtása.
  3.) A blokk lokális változói számára foglalt memória felszabadítása.
- A memória azon részét, ahol a fenti tevékenységek lezajlanak, veremnek nevezzük.

Függvény végrehajtás

- Az F(A1, ... ,An) függvényművelet végrehajtása sorrendben a következő tevékenységeket jelenti

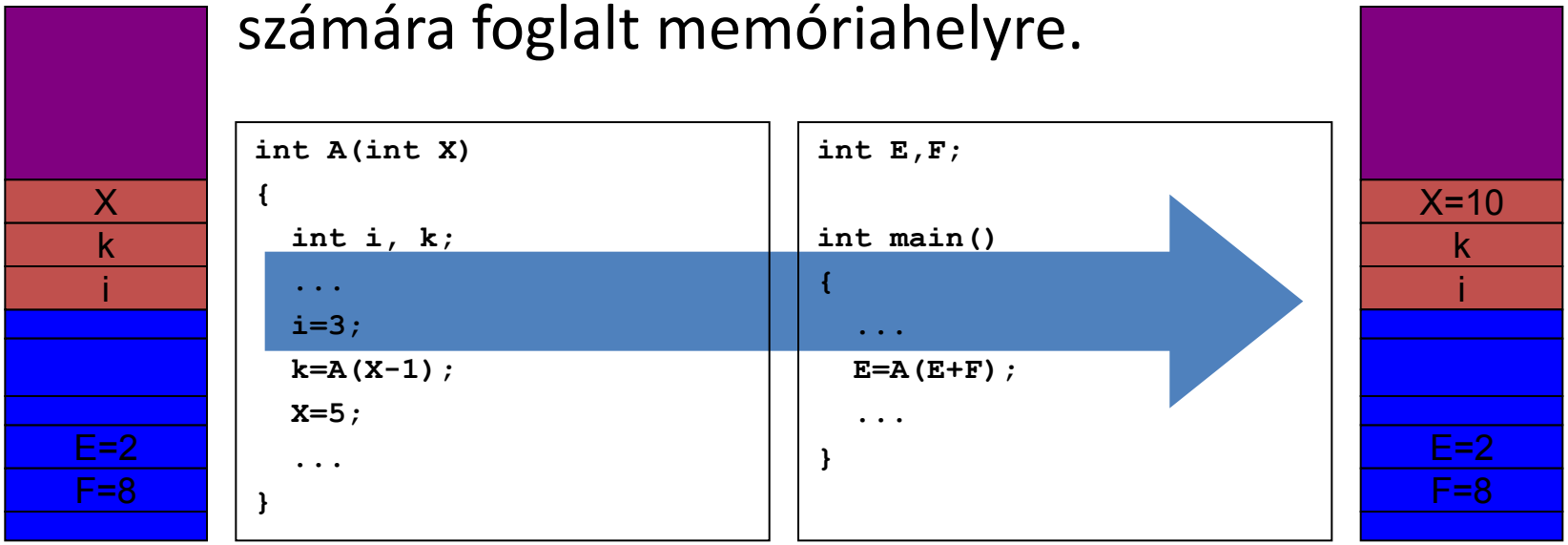
1.) Memória helyfoglalás a függvényblokk paraméterei és lokális változói számára.



Függvény végrehajtás

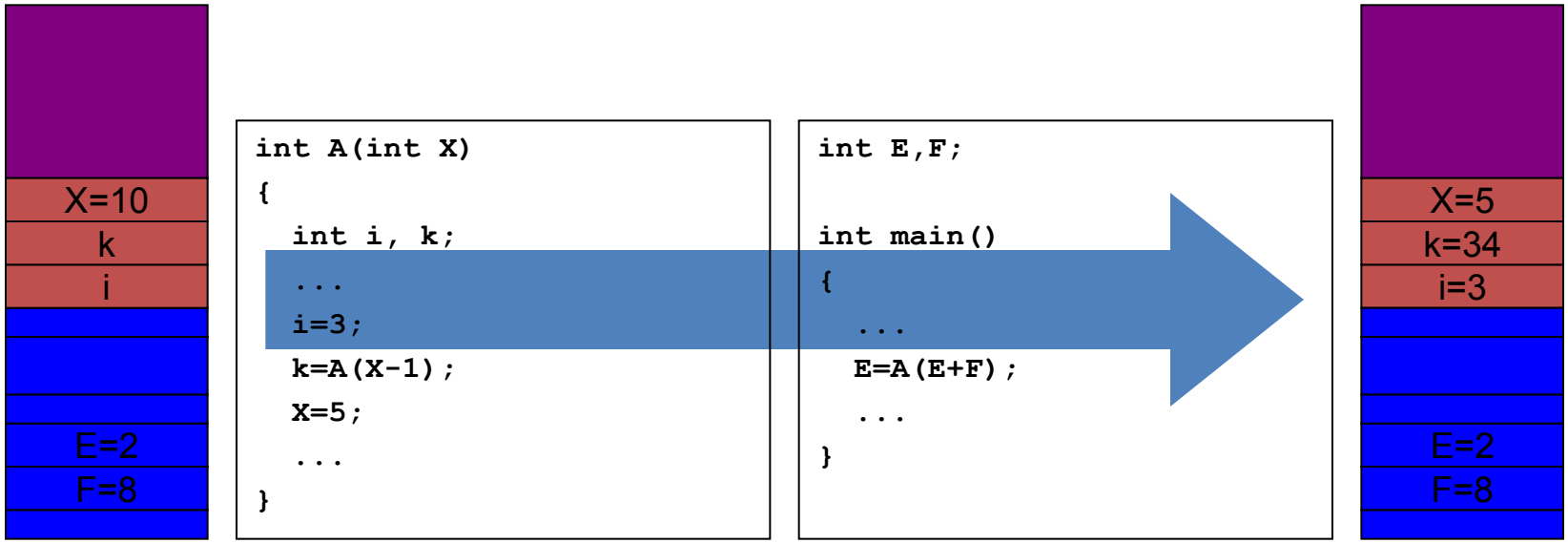
2.) Paraméterátadás.

– Először tetszőleges sorrendben kiértékelődnek az aktuális paraméterek. Mivel mind értékparaméter, az i-edik kiértékelt aktuális paraméter értéke átadódik az i-edik formális paraméternek, vagyis az aktuális paraméter értéke bemásolódik a formális paraméter számára foglalt memóriahelyre.



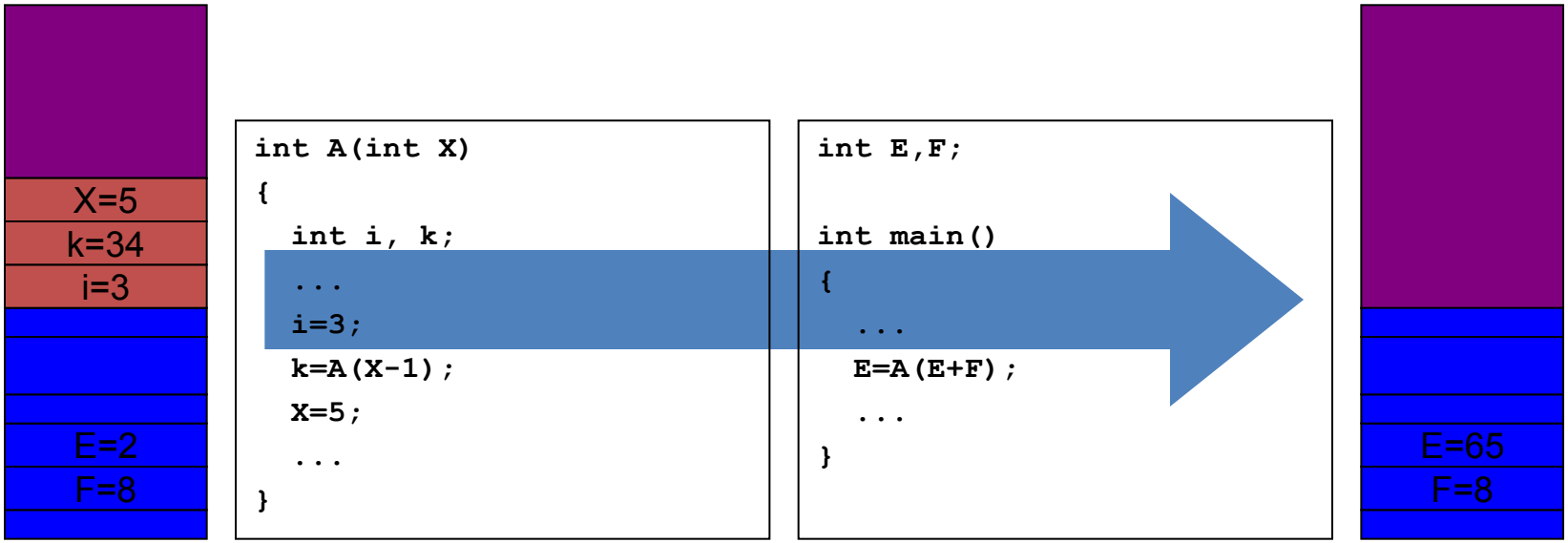
Függvény végrehajtás

3.) A függvényblokk utasításrészének végrehajtása.



Függvény végrehajtás

4.) A függvényblokk formális paraméterei és lokális változói számára foglalt memória felszabadítása.



Függvények mellékhatása

- Függvény mellékhatásán azt értjük, hogy a függvényhívás hatására nem csak a függvényérték számítódik ki, hanem megváltozhat egy globális változó értéke is.
- Mellékhatás következménye, hogy az összeadás kommutativitása nem feltétlenül teljesül, ha a tagok függvényhívások.

Függvények mellékhatása

- Vegyük az alábbi példát

```
int A,B,Z;
int f(int x) {
  int r;
  r = x + A;
  A = x + 1;
  return r;
}
main () {
  A = 1; B = 2;
  Z = f(A) + f(B);  /* f(A)+f(B) == 6 */
                    /* f(B)+f(A) == 9 */
}
```

Függvények mellékhatása

- A mellékhatást kerülni kell!
- Egy változót tehát vagy globális változóként használjunk, vagy aktuális paraméterként, de egyszerre mindkét céllal ne!

- A C-ben nincs meghatározva, hogy két részkifejezés közül melyiket kell előbb kiértékelni, tehát az sem világos, hogy ha mindkettőben van függvényhívás, melyik hajtódik végre előbb.