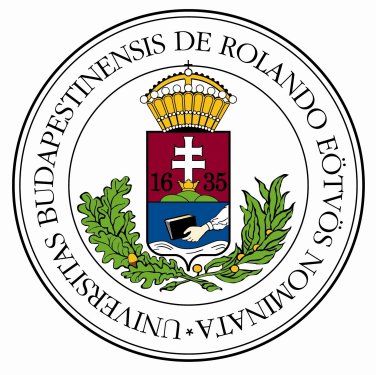
Imperatív programozás

Dinamikus programszerkezet



Kozsik Tamás és mások

ELTE Eötvös Loránd Tudományegyetem

Dinamikus programszerkezet

Hogyan működik a program?

- A változók tárolása a memóriában
- Információk a programvégrehajtás állapotáról
  - A főprogramból induló alprogramhívások

Absztrakt modell, implementációs módszer

Outline

Végrehajtási verem

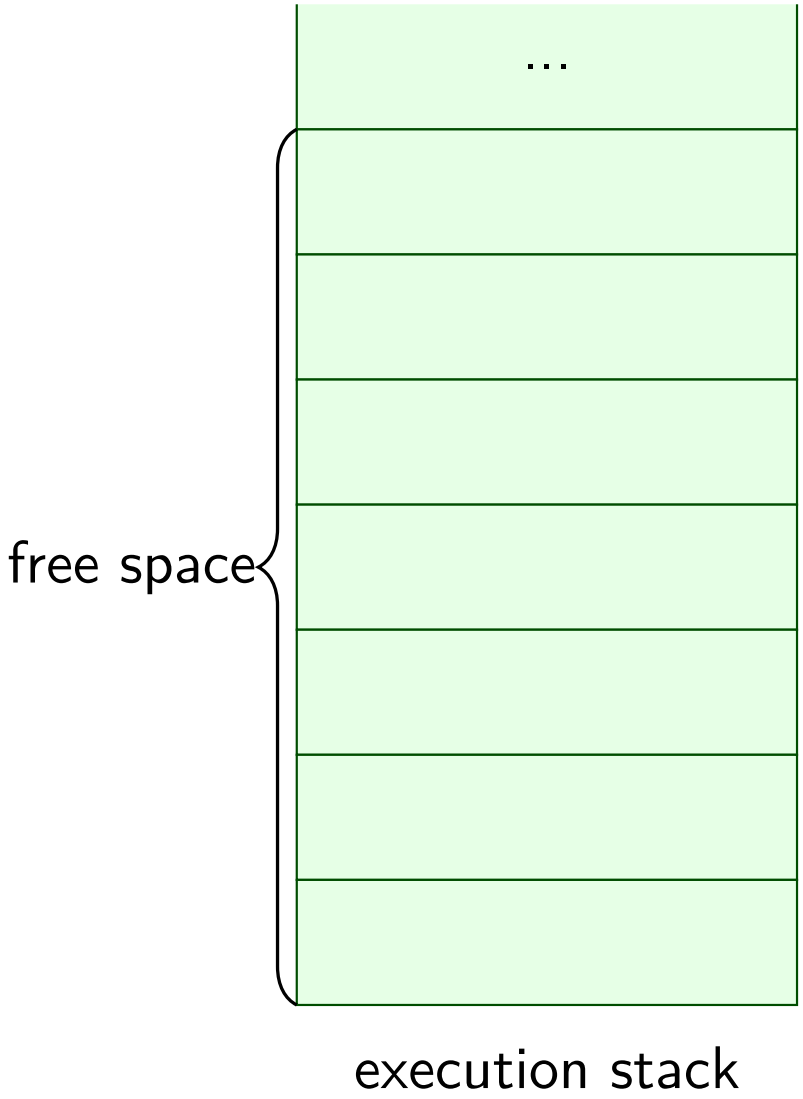
```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```

Execution stack

- Alprogramhívások logikája
  - LIFO: Last-in-First-Out
  - Verem adatszerkezet
- Minden alprogramhívásról egy bejegyzés
  - Aktivációs rekord
  - Például információ arról, hova kell visszatérni
- Verem alja: főprogram aktivációs rekordja
- Verem teteje: ahol tart a programvégrehajtás

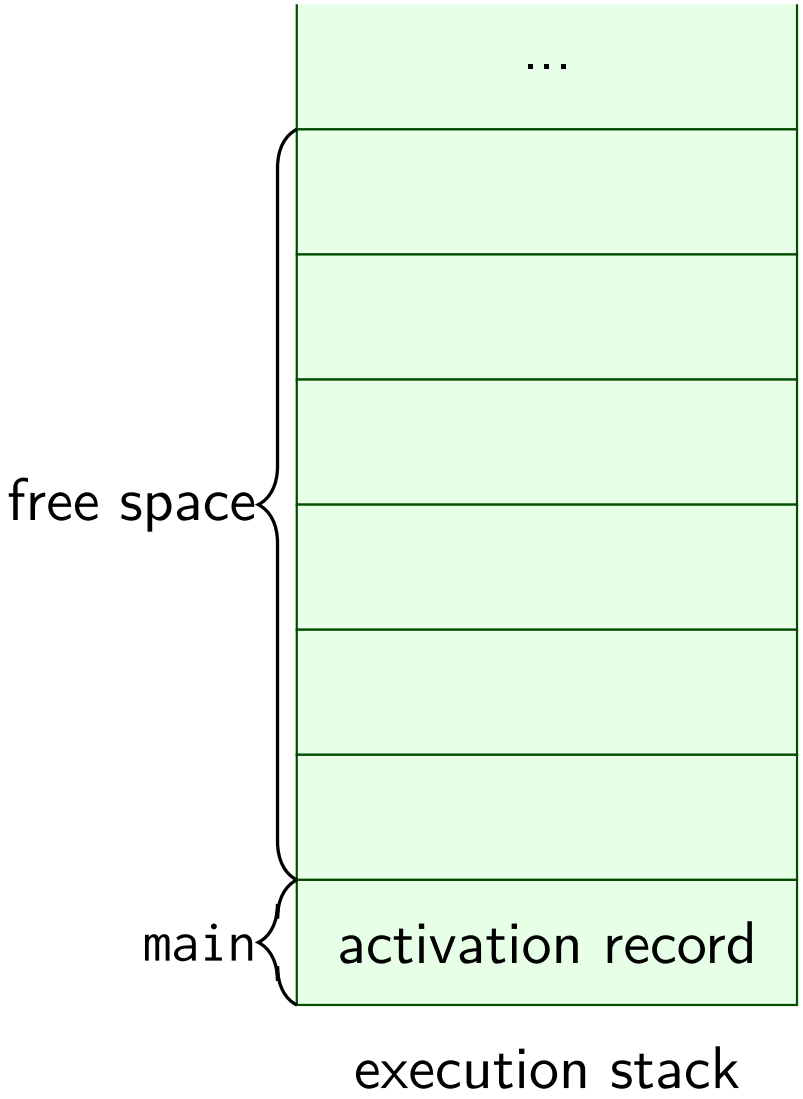
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



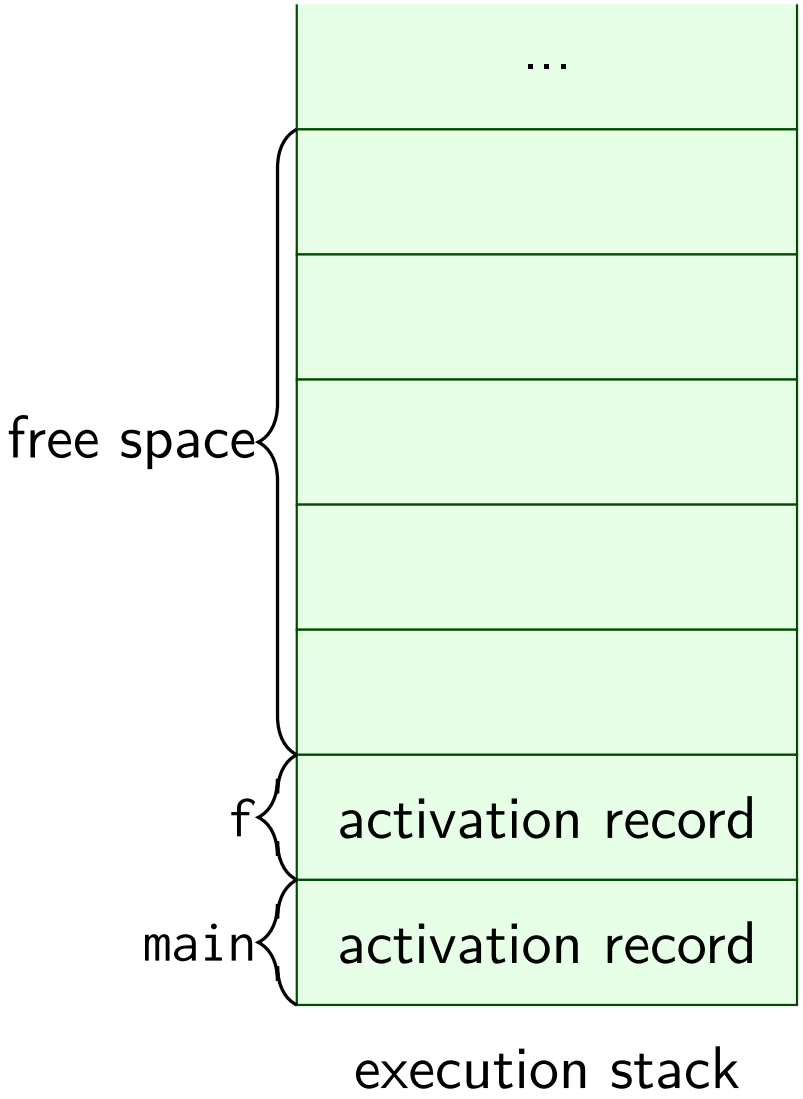
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



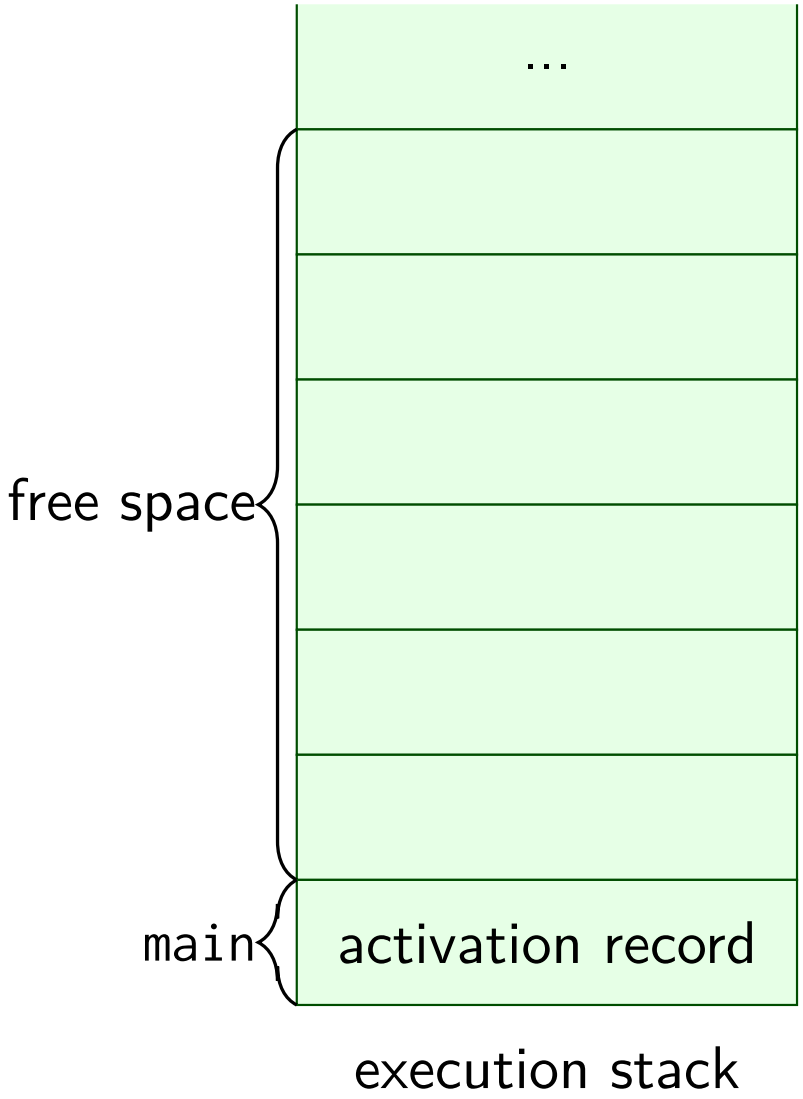
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



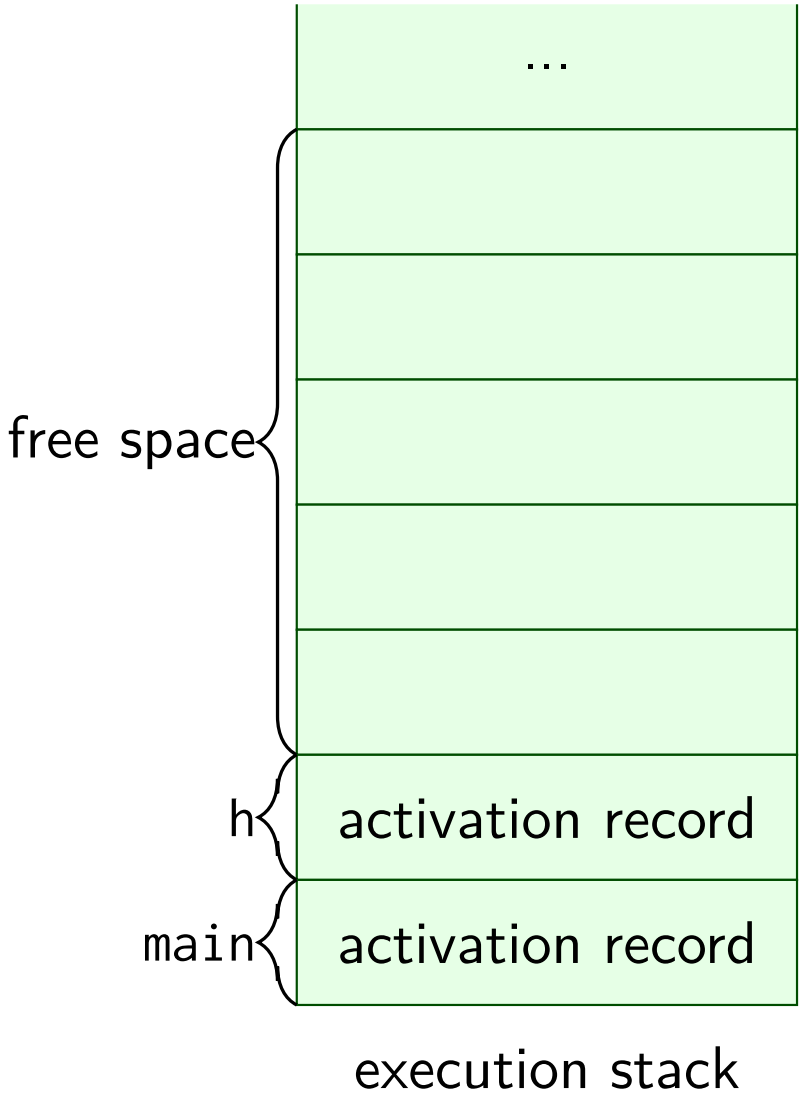
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



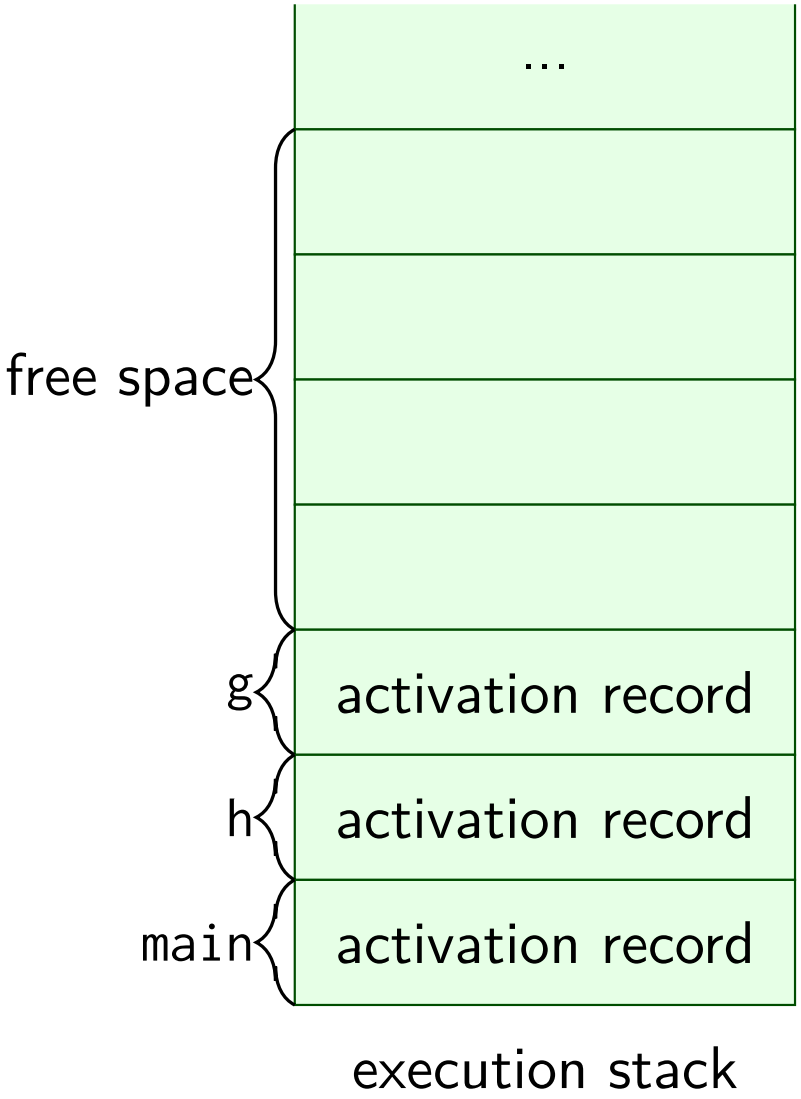
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



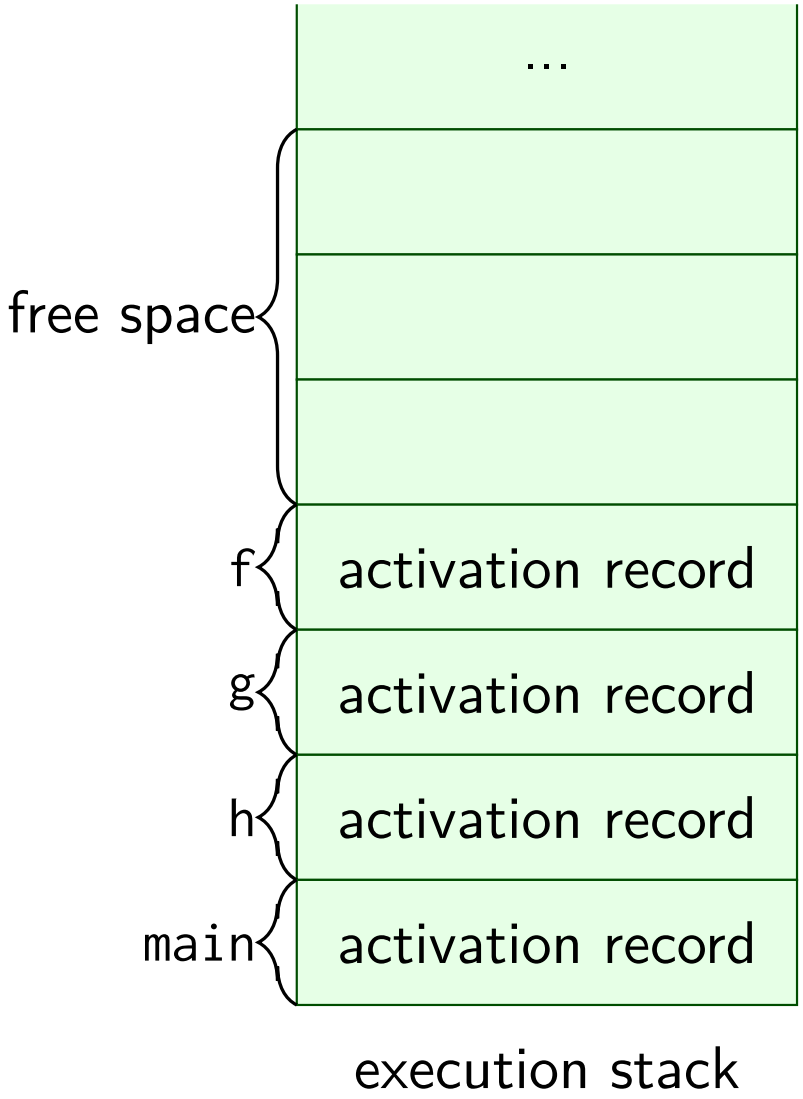
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



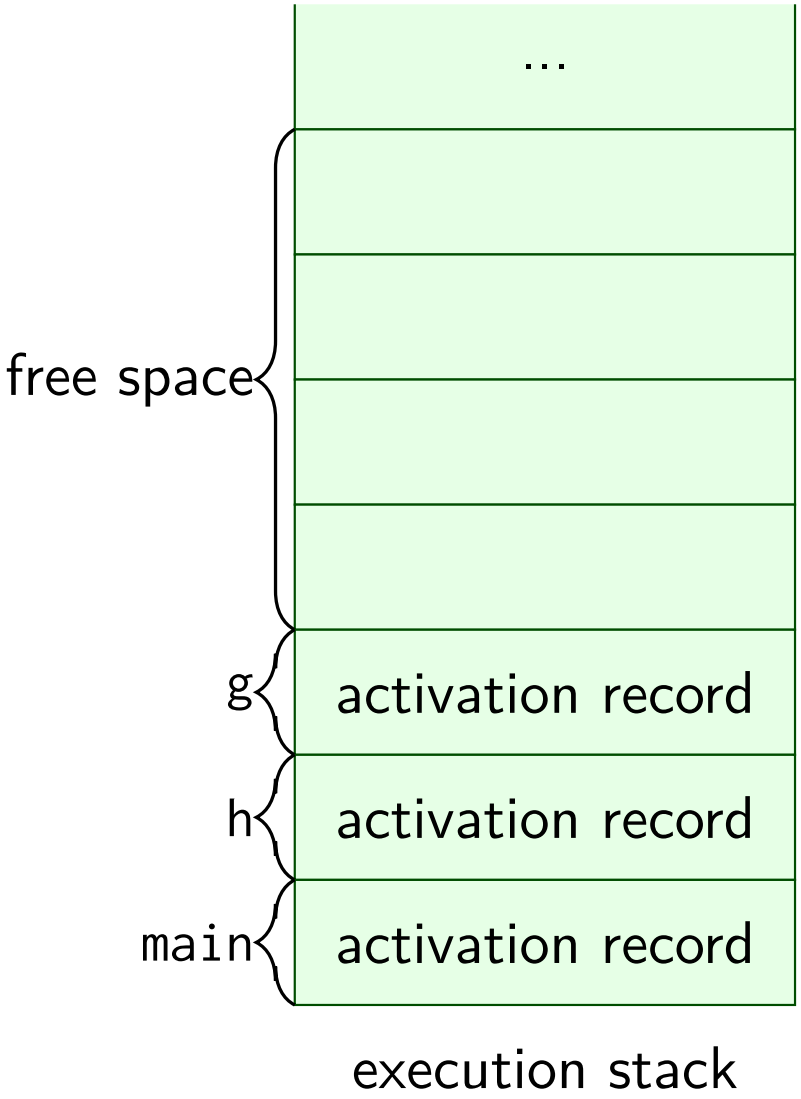
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



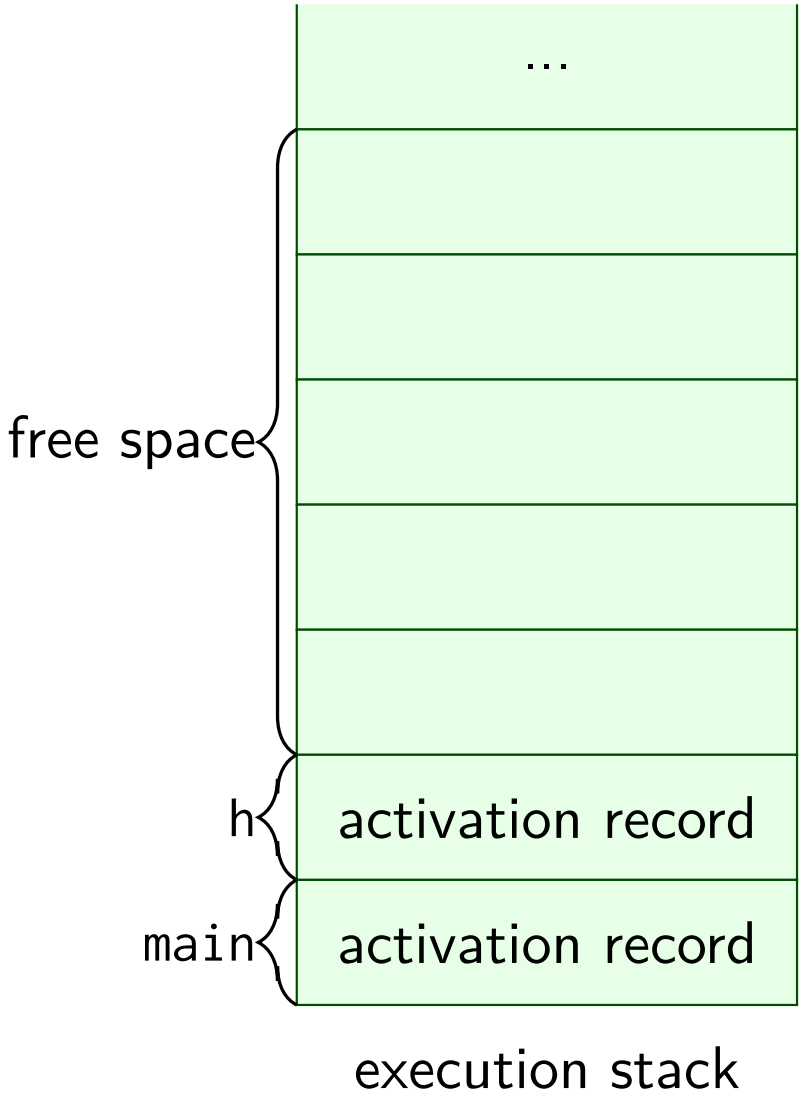
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



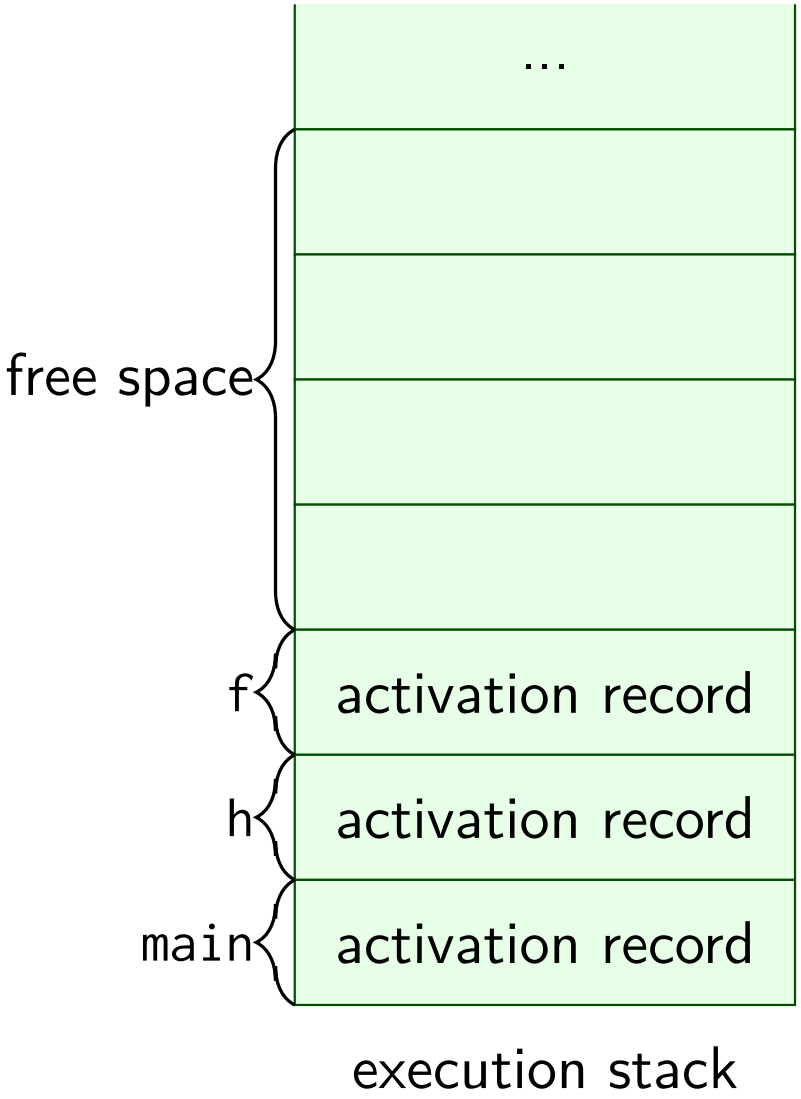
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



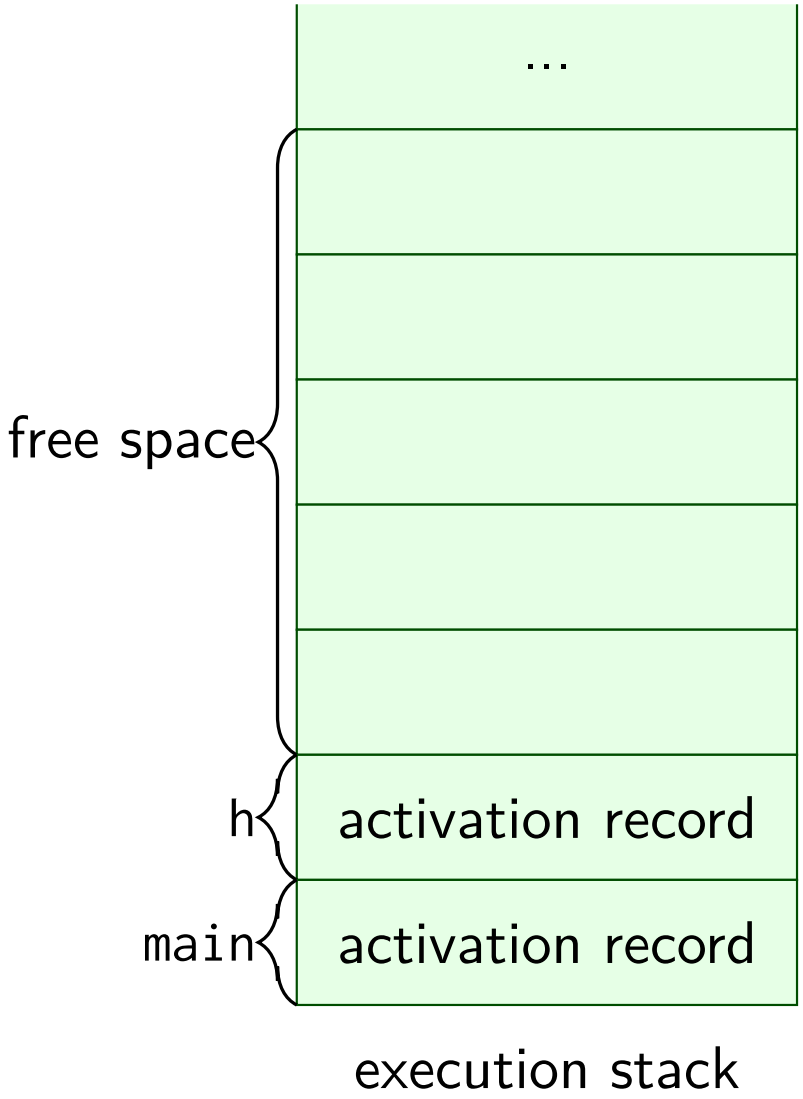
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



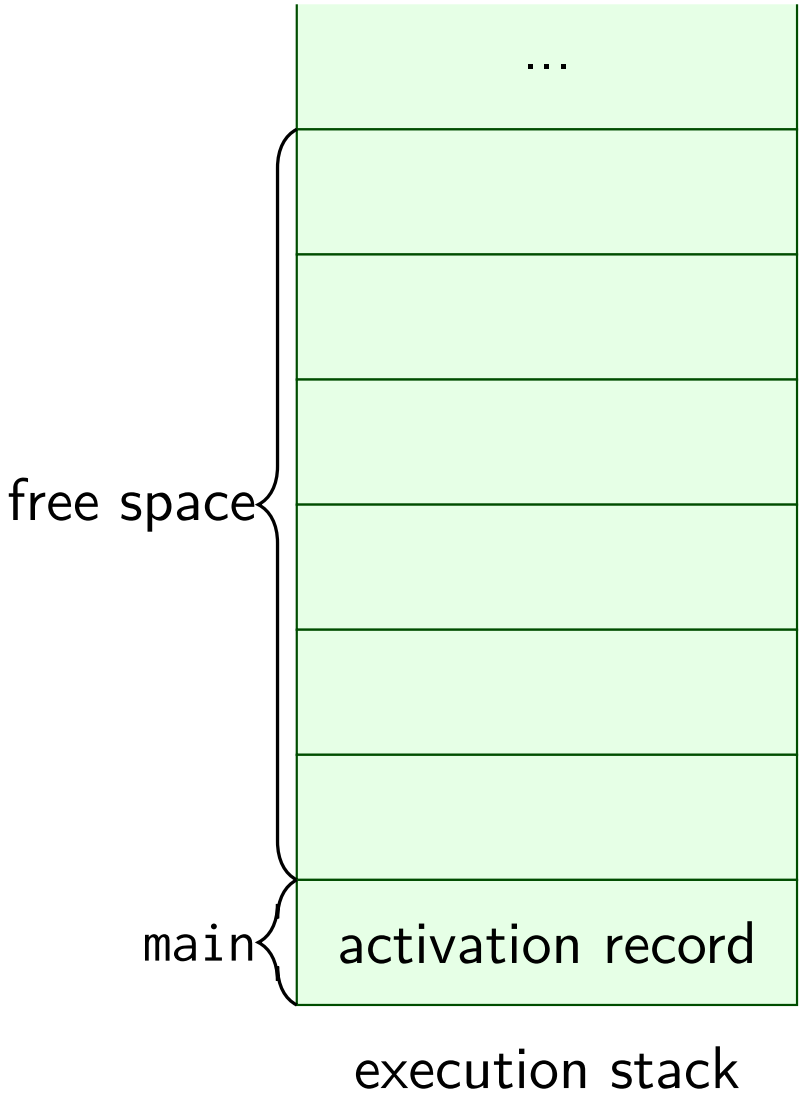
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



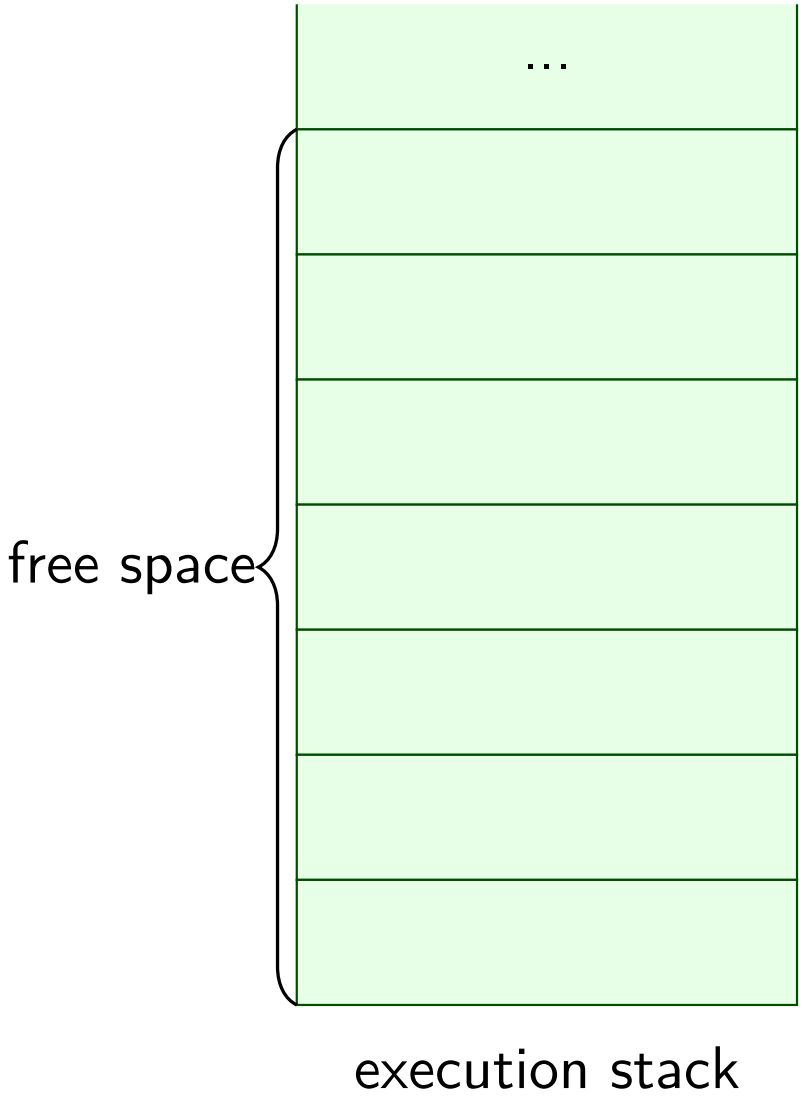
Alprogramhívások nyilvántartása

```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



Alprogramhívások nyilvántartása

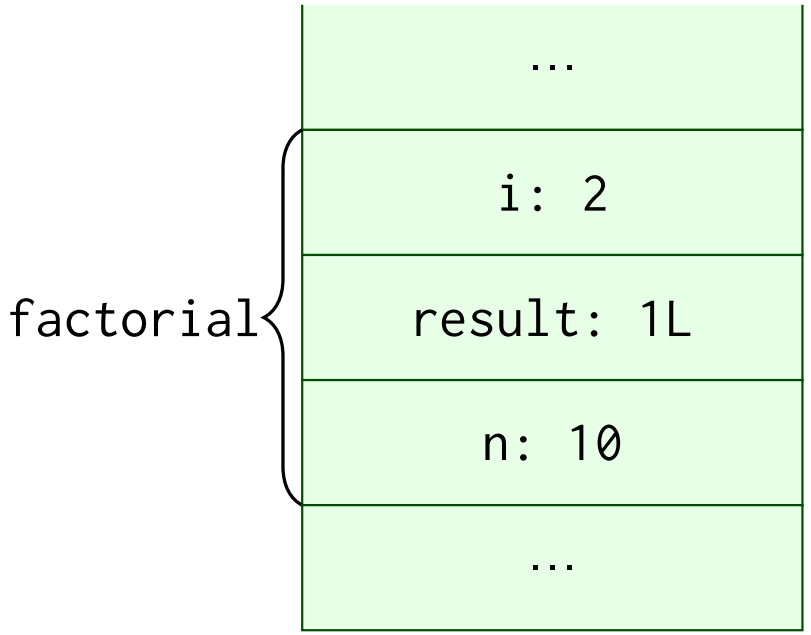
```
void f(void)
{
}
void g(void){
    f();
}
void h(void){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



Aktivációs rekord

- Mindenféle technikai dolgok
- Alprogram paraméterei
- Alprogram (egyes) lokális változói

```
long factorial( int n )
{
    long result = 1L;
    int i = 2;
    for( ; i<=n; ++n )
        result *= i;
    return result;
}
```



Rekurzió

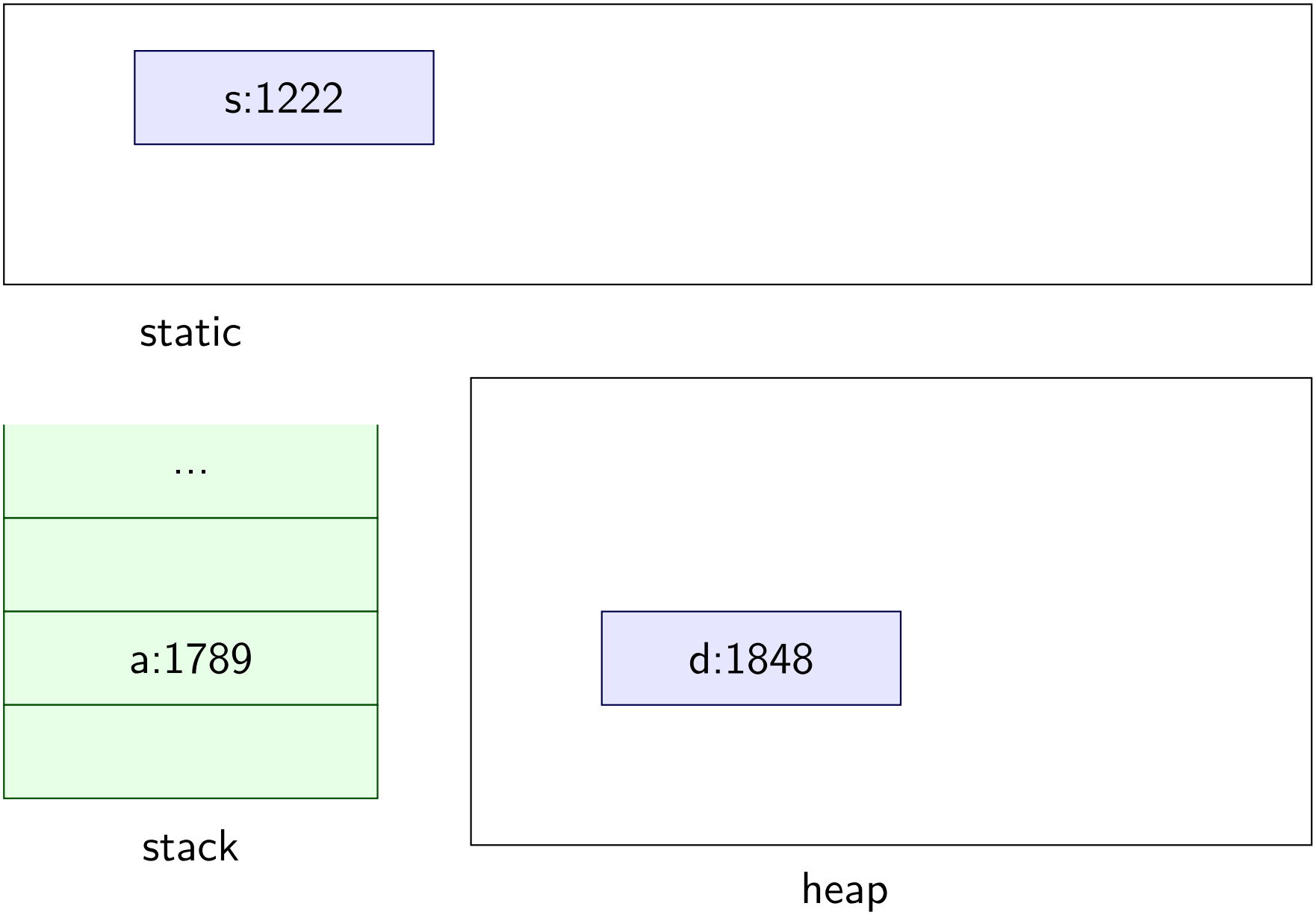
- Egy alprogram saját magát hívja
  - Közvetlenül
  - Közvetve
- Minden hívásról új aktivációs rekord
- Túl mély rekurzió: Stack Overflow
- Költség: aktivációs rekord építése/lebontása

Outline

„Változók'' tárolása a memóriában

- statikus tárhely $\rightarrow$ statikus
- végrehajtási verem $\rightarrow$ automatikus
- dinamikus tárhely (heap) $\rightarrow$ dinamikus

static - stack - heap



Statikus tárolású változó

- Statikus tárhely
  - Statikus deklarációkiértékelés
  - A fordító tudja, mekkora tár kell
- Pl. globális változók
- Élettartam: a program elejétől a végéig

```
int counter = 0;
void signal(void)
{
    ++ counter;
}
```

```
counter = 0
def signal():
    global counter
    counter += 1
```

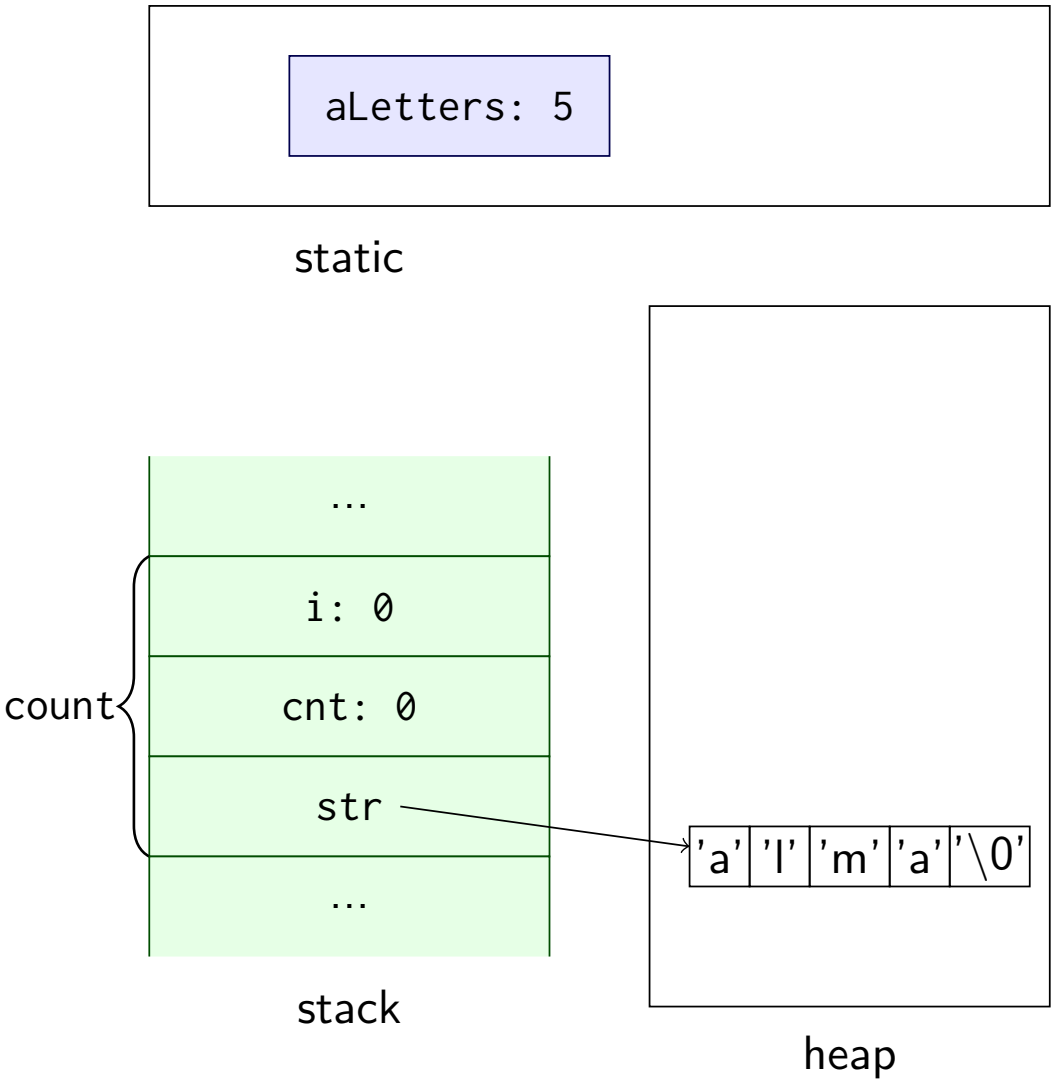
Automatikus tárolású változó

- Végrehajtási vermen
  - Az aktivációs rekordokban
- A lokális változók *általában* ilyenek
- Élettartam: blokk végrehajtása
  - Automatikusan jön létre és szűnik meg

```
int lnko( int a, int b ){
    int c;
    while( b != 0 ){
        c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

```
def lnko(a,b):
    while b!=0:
        c = a % b
        a = b
        b = c
    return a
```

static - stack - heap




```
int aLetters = 0;
int count( char *str )
{
    int cnt=0, i=0;
    while (str[i]!='\0')
    {
        if (str[i]=='a')
            ++cnt;
        ++i;
    }
    a_letters += cnt;
    return cnt;
}
```

C - statikus lokális változók

- `static` kulcsszó
- Hatókör: lokális változó
  - Információelrejtés elve
- Élettartam: mint globális változónál

```
int counter = 0;
void signal(void)
{
    ++ counter;

}
```

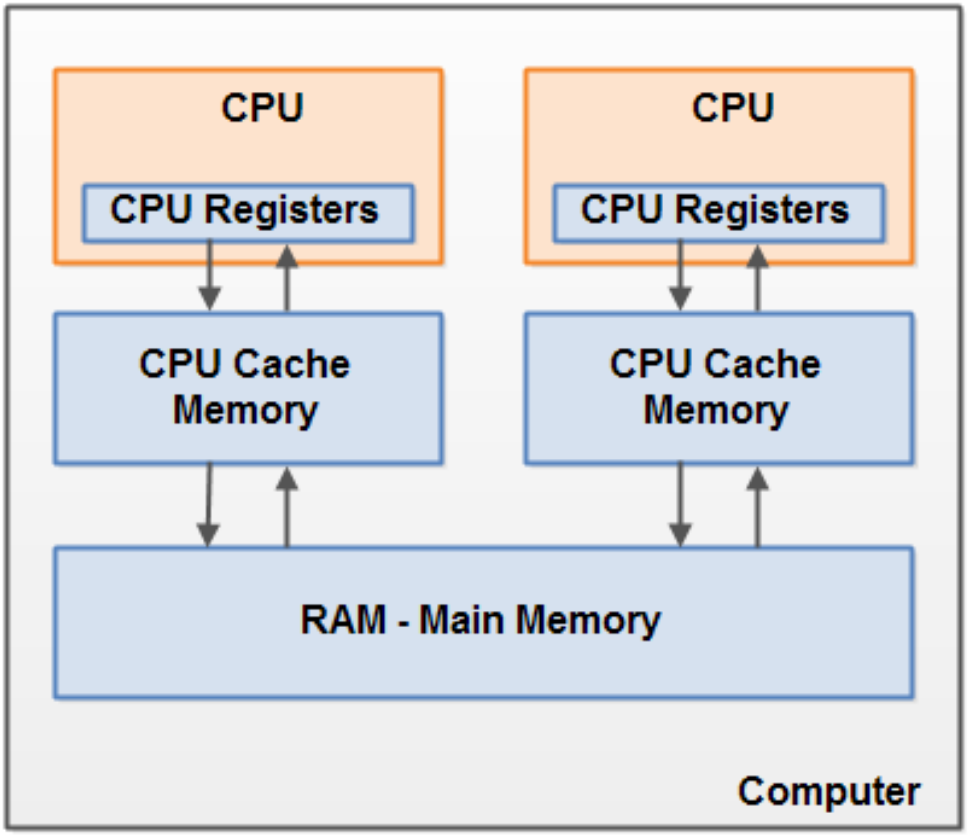
```
int signal(void)
{
    static int counter = 0;
    ++ counter;
    return counter;
}
```

Tárolási mód kifejezése C-ben

- `static`
  - lokális
  - globális
- `auto` (nem használjuk)
  - C++ nyelvben az `auto` kulcsszó mást jelent
- `register` (nem használjuk)
  - optimalizáció

```
int lnko( int a, int b ){
    auto int c;
    while( b != 0 ){
        c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

Számítógép memóriája



Optimalizáció: memóriaműveletek emberi skálán

Forrás: David Jeppesen

órajel	0.4 ns	1 sec
L1 cache	0.9 ns	2 sec
L2 cache	2.8 ns	7 sec
L3 cache	28 ns	1 min
DDR memória	~100 ns	4 min
SSD I/O	50-150 microsec	1,5-4 nap
HDD I/O	1-10 ms	1-9 hónap
Internet	65 ms	5-10 év

Globális változók használata

Kerülendő!

Változók definiálása

Deklarációval

- Statikus és automatikus tárolású
  - Statikus tárhely
  - Végrehajtási verem
- Élettartam: programszerkezetből
  - A hatókör
  - Kivéve lokális statikus (C)

Allokáló utasítással

- Dinamikus tárolású
  - Heap (dinamikus tárhely)
- Élettartam: programozható
- Felszabadítás
  - Felszabadító utasítás (C)
  - Szemétgyűjtés (Python)

Blokk utasítás

- Új hatókör, lokális deklarációkkal
  - Névtér szennyeződése elkerülhető
- Automatikus tárolású változók
  - Élettartam lerövidíthető

Outline

Alprogram paraméterei

- Definícióban: formális paraméterlista
- Hívásnál: aktuális paraméterlista

Paraméterátadási technikák

- Többféle paraméterátadás van a különféle nyelvekben
  - Érték szerinti (pass-by-value, call-by-value)
  - Érték-eredmény szerinti (call-by-value-result)
  - Eredmény szerinti (call-by-result)
  - Cím szerinti (call-by-reference)
  - Megosztás szerinti (call-by-sharing)
  - Igény szerinti (call-by-need)
  - Név szerinti (call-by-name)
- Végrehajtási verem!

Érték szerinti paraméterátadás

- Formális paraméter: automatikus tárolású lokális változó
- Aktuális paraméter: kezdőérték
- Hívás: az aktuális paraméter értéke bemásolódik a formális paraméterbe
- Visszatérés: a formális paraméter megszűnik

Érték szerinti paraméterátadás – példa

```c
int lnko( int a, int b )
{
    int c;
    while( b != 0 ){
        c = a % b;
        a = b;
        b = c;
    }
    return a;
}
int main()
{
    int n = 1984, m = 356;
    int r = lnko(n,m);
    printf("%d %d %d\n",n,m,r);
}
```

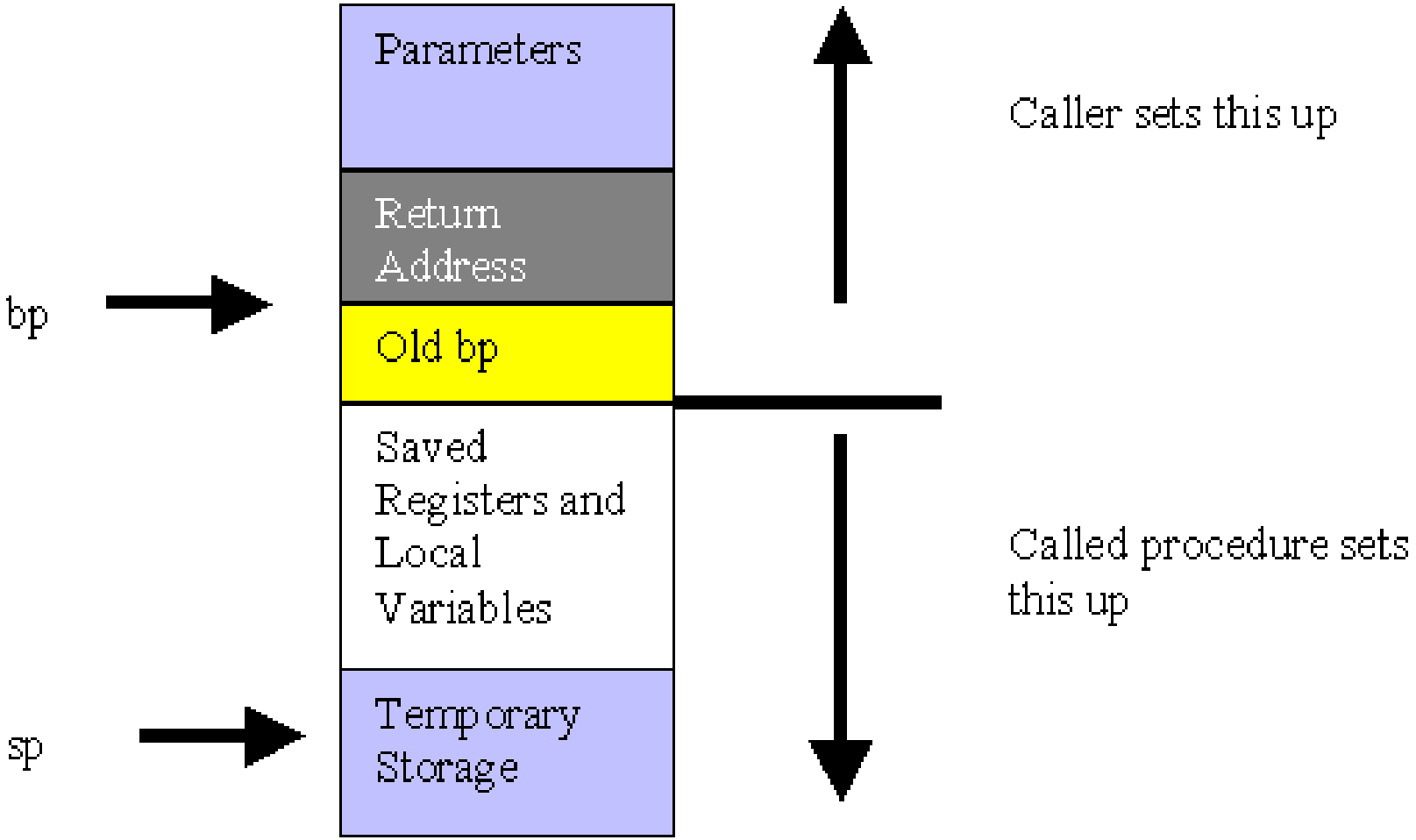
```python
def lnko(a,b):
    while b != 0:
        a, b = b, a%b
    return a

n = 1984
m = 356
r = lnko(n,m)
print(n,m,r)
```

Aktivációs rekord

- Mindenféle technikai dolgok
- Alprogram automatikus tárolású változói
  - PI. az alprogram formális paraméterei
  - Kivéve a regiszterekben átadott paramétereket

Precízebben



C programok címtere