Imperatív programozás

Dinamikus programszerkezet

Kozsik Tamás és mások

Eötvös Loránd Tudományegyetem

2022. augusztus 25.

Tartalomjegyzék

Dinamikus programszerkezet

- Hogyan működik a program?
- Információk a programvégrehajtás állapotáról
- Főprogramból induló alprogramhívások
- Változók tárolása a memóriában
- Adunk egy absztrakt modellt

Függvények

- Alprogramok
  - Függvények
  - Eljárások
  - Rutinok
  - Metódusok
- Nagyobb programok felbontása
- Paraméterezés
- Meghívás (kiértékelés)
- Eredmény visszaadás

Függvények

Egyszerű függvény

$$even : \mathbb{Z} \to \mathbb{L}$$

$$even(x) = \begin{cases} \uparrow & \text{ha } x \text{ páros} \\ \downarrow & \text{ha } x \text{ páratlan} \end{cases}$$

```
bool even(int x)
{
  return x % 2 == 0;
}

int main()
{
  if (even(42))
    printf("Páros\n");
  else
    printf("Páratlan\n");
}
```

Függvények

Visszatérési érték nélkül

```
bool goodArgs(int argc, char* argv[]) { ... }

void printUsage(char programName[]) {
  printf("This program can be used as follows:\n");
  printf("%s --in <input> --out <output>", programName);
}

int main(int argc, char* argv[]) {
  if (!goodArgs(argc, argv)) {
    printUsage(argv[0]);
    return 1;
  }

  doTheJob();

  return 0;
}
```

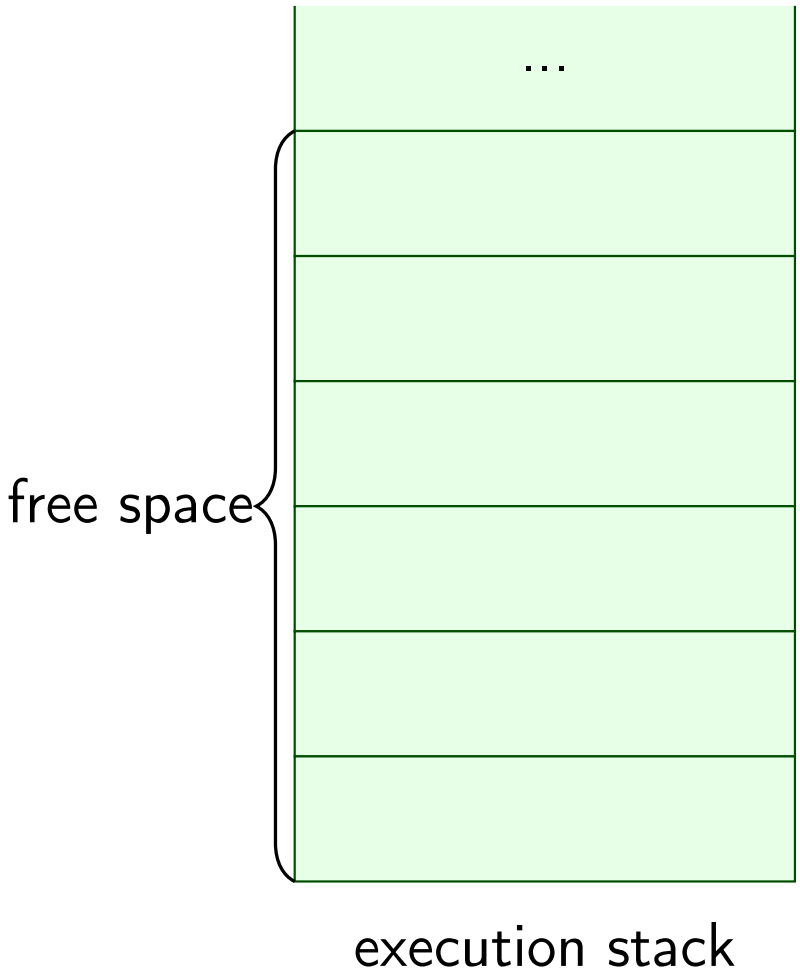
Végrehajtási verem

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

- 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

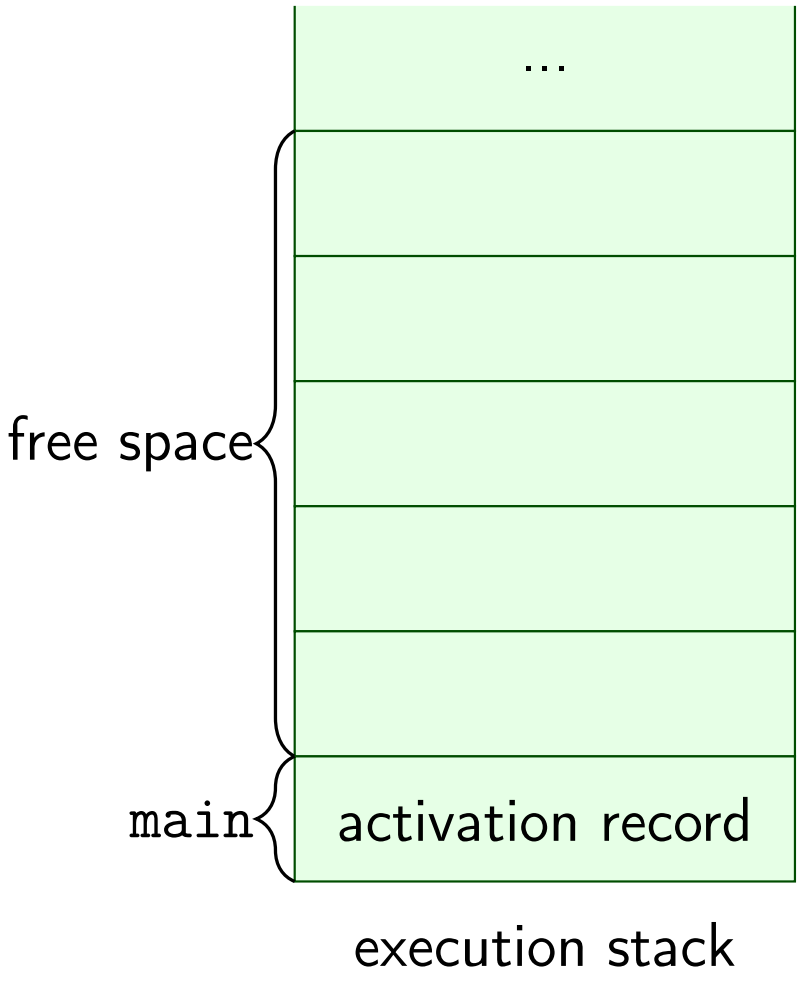
Alprogramhívások nyilvántartása

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



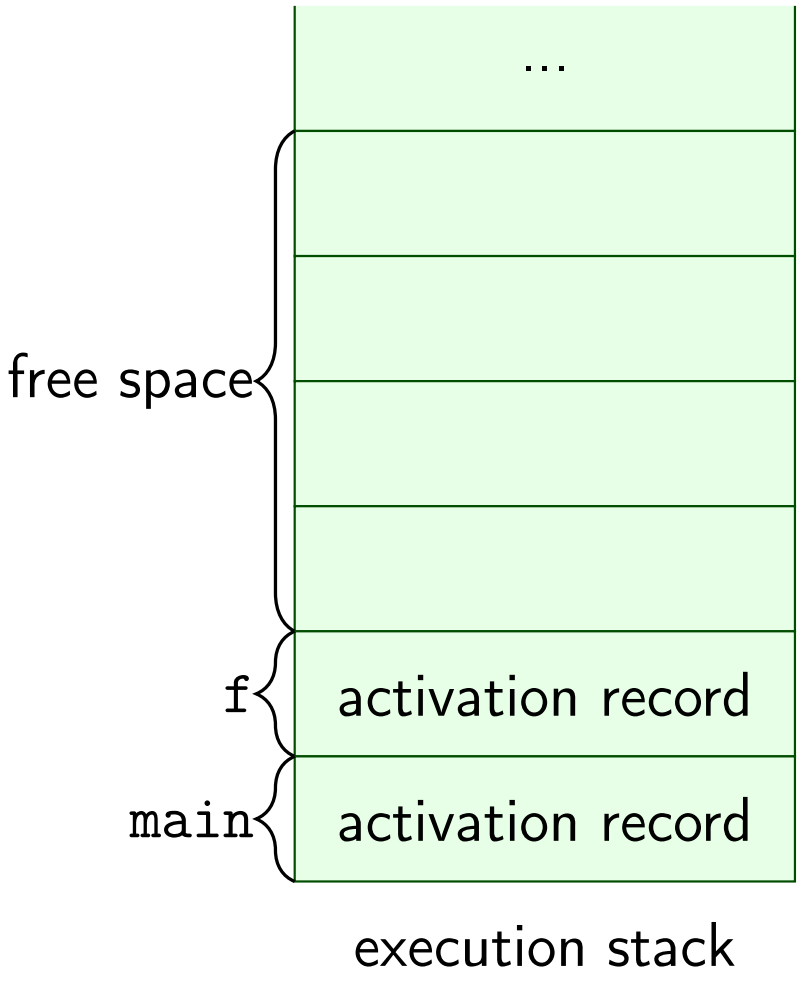
Alprogramhívások nyilvántartása

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



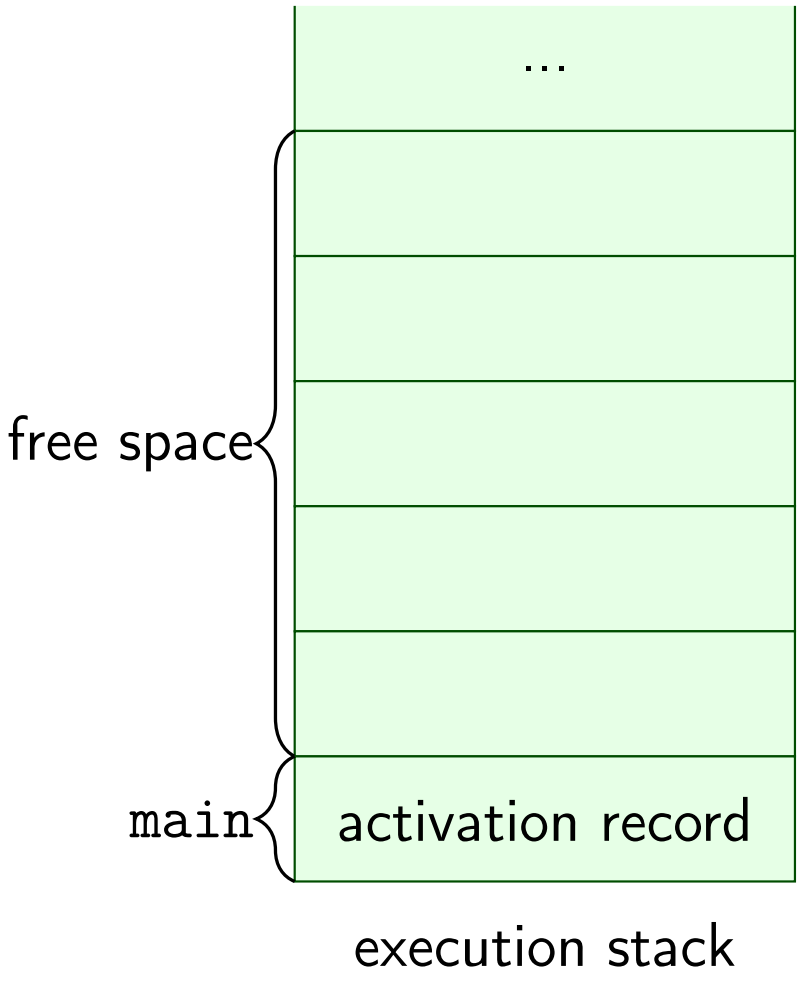
Alprogramhívások nyilvántartása

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



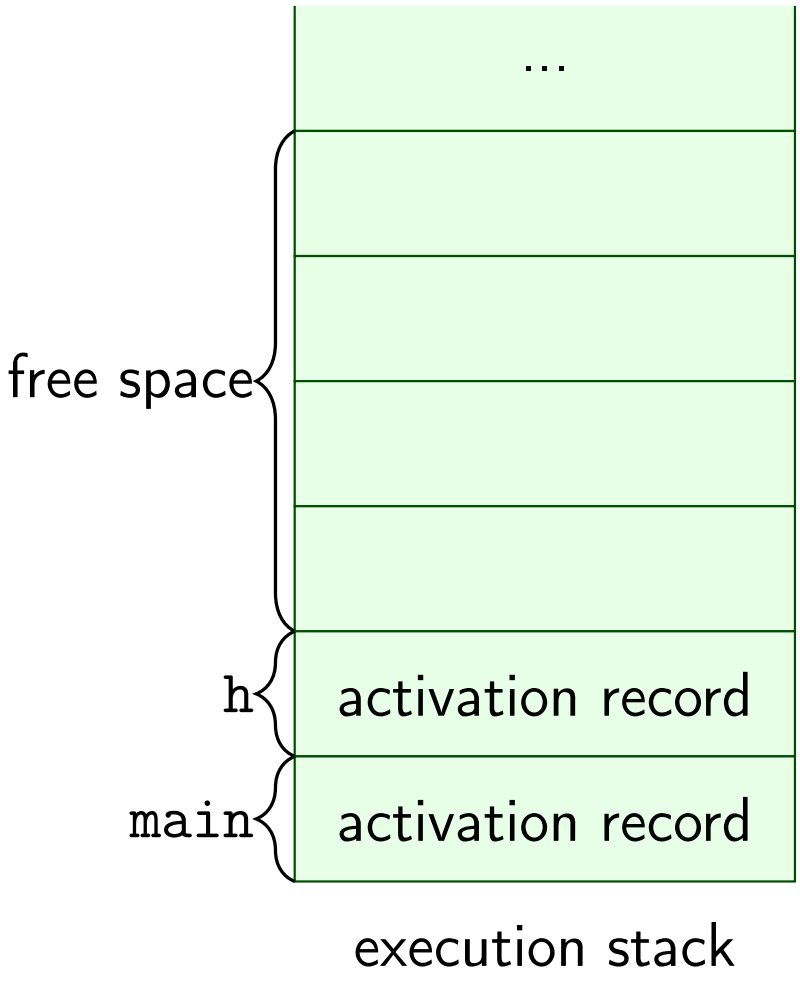
Alprogramhívások nyilvántartása

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



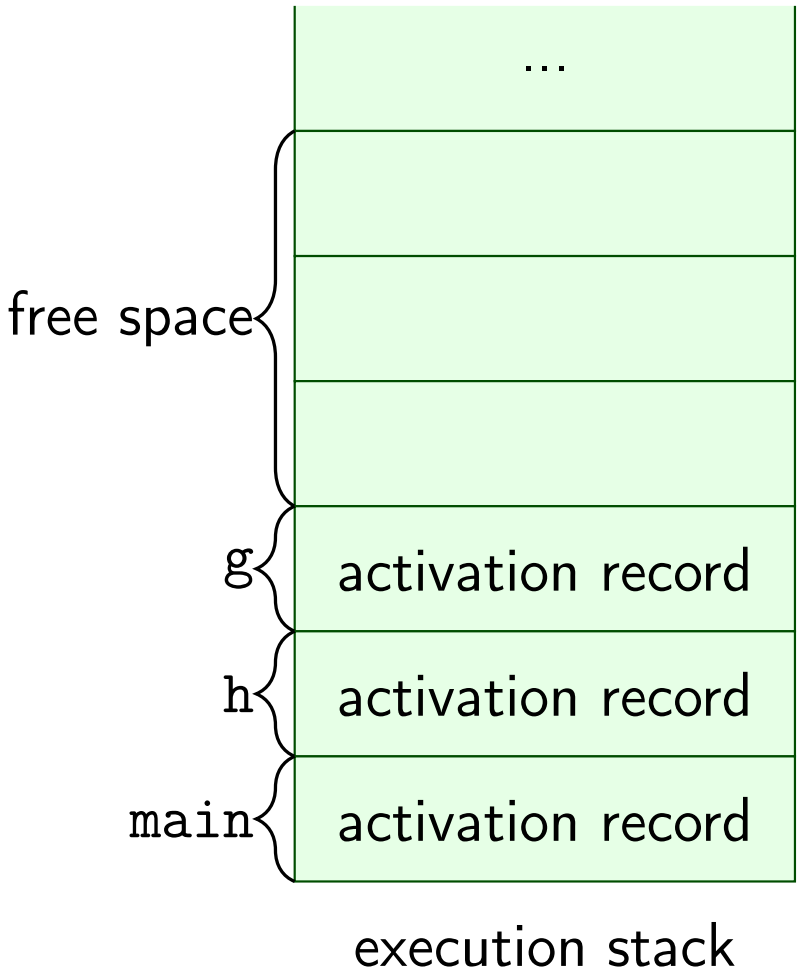
Alprogramhívások nyilvántartása

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



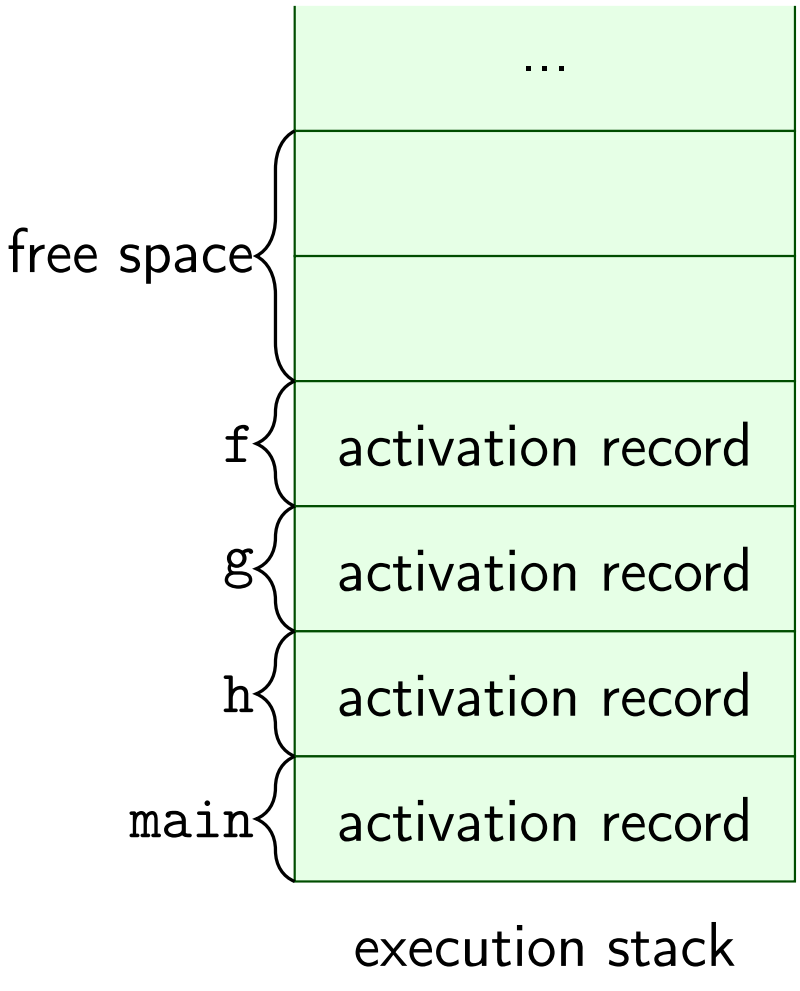
Alprogramhívások nyilvántartása

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



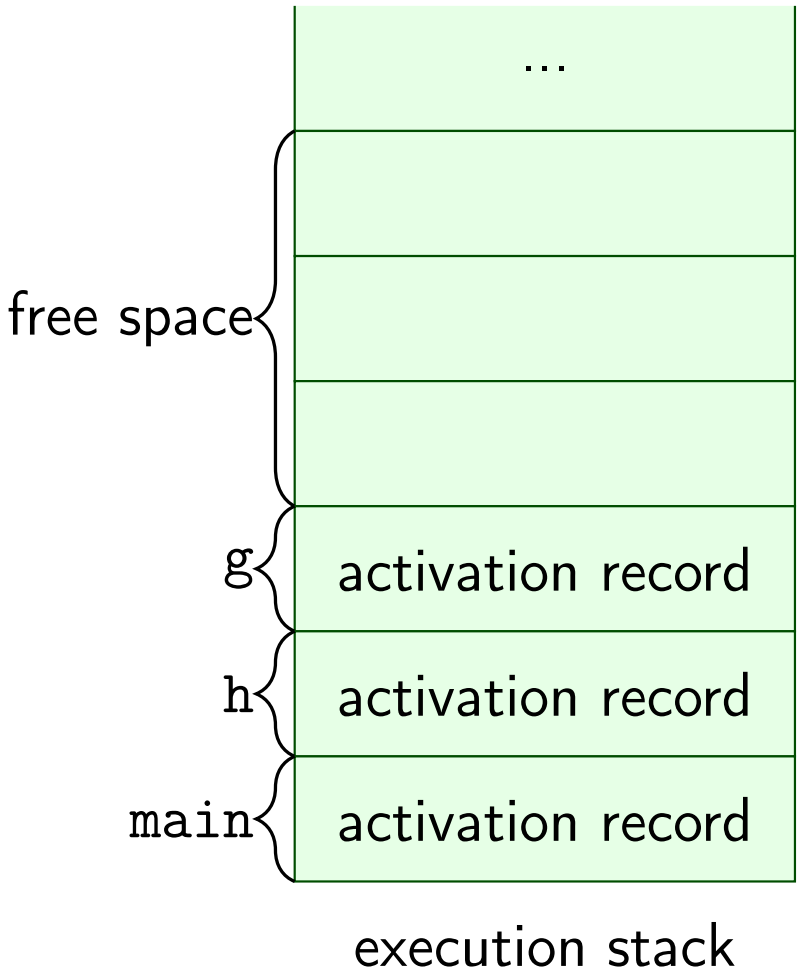
Alprogramhívások nyilvántartása

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



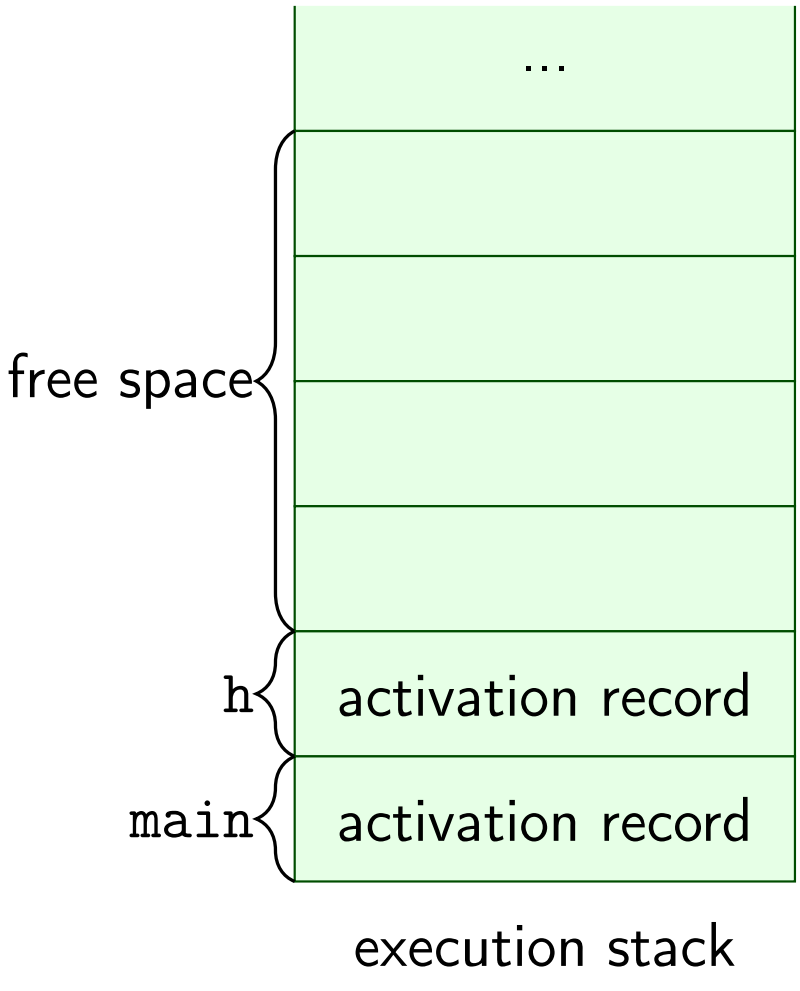
Alprogramhívások nyilvántartása

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



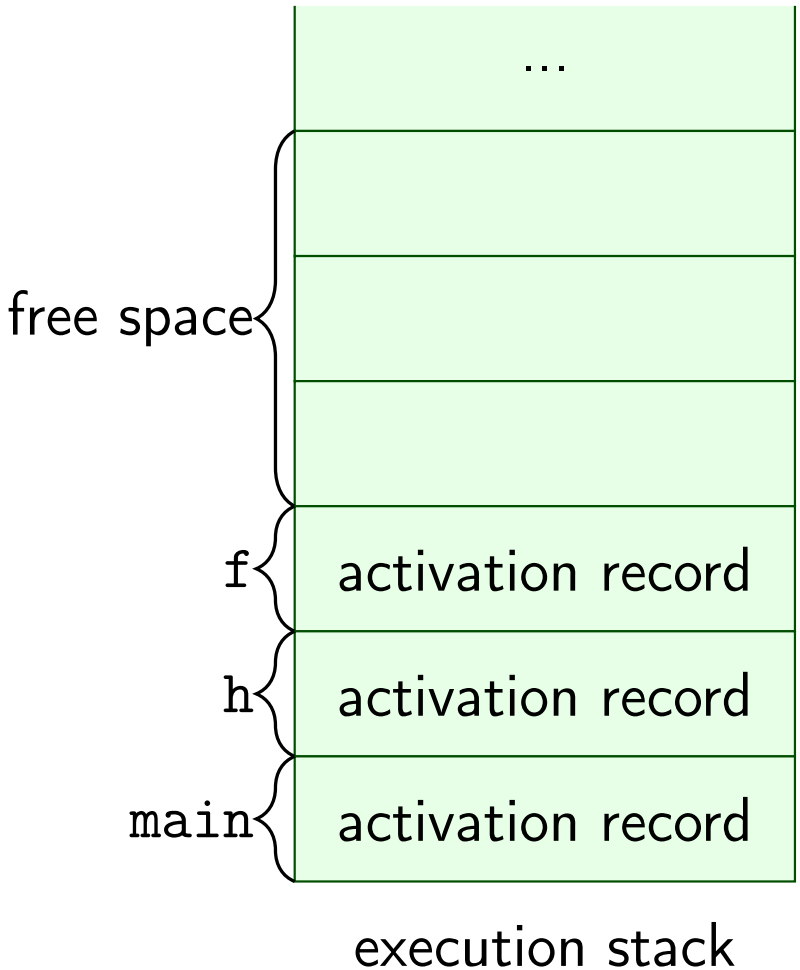
Alprogramhívások nyilvántartása

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



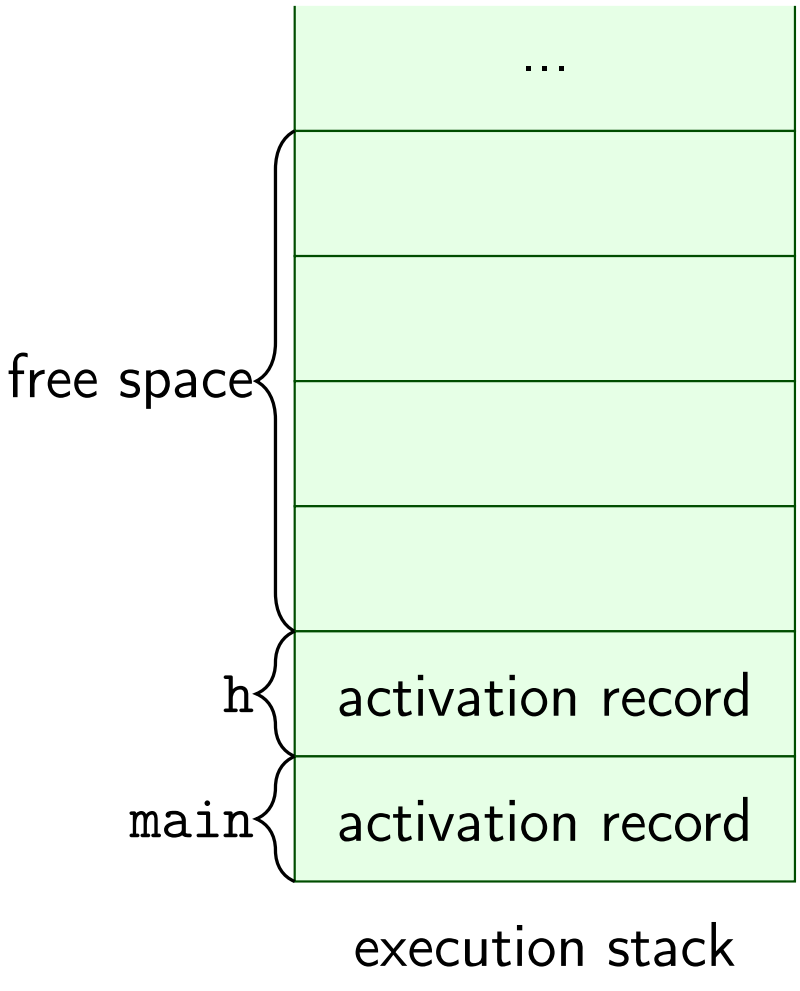
Alprogramhívások nyilvántartása

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



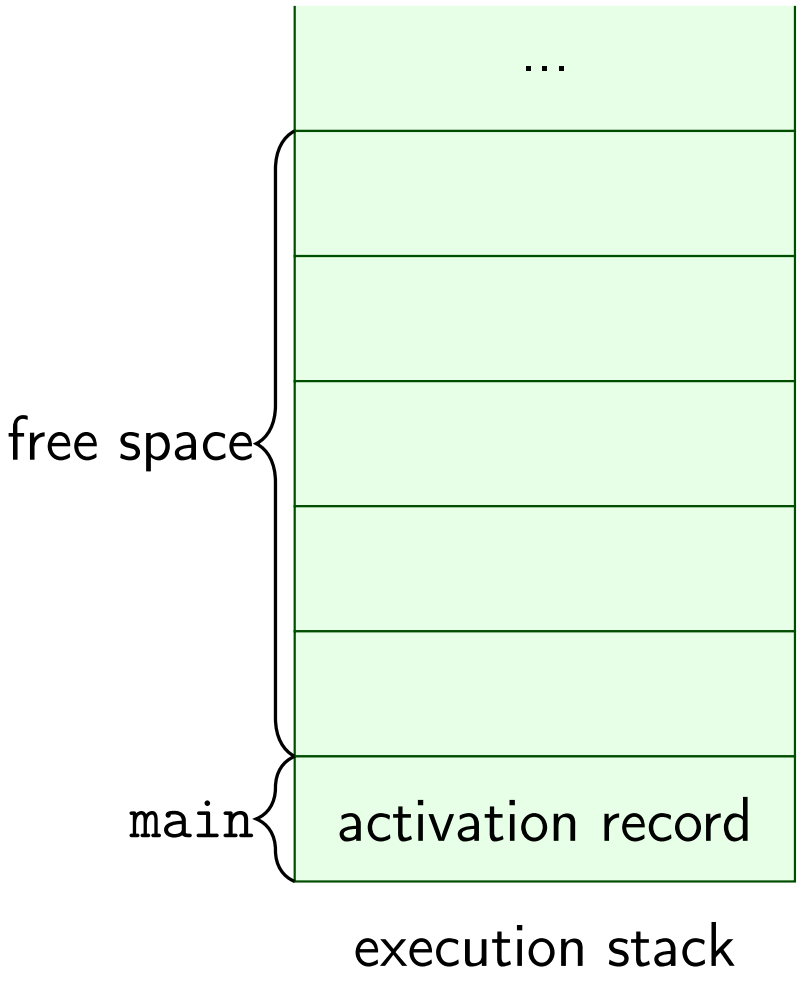
Alprogramhívások nyilvántartása

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



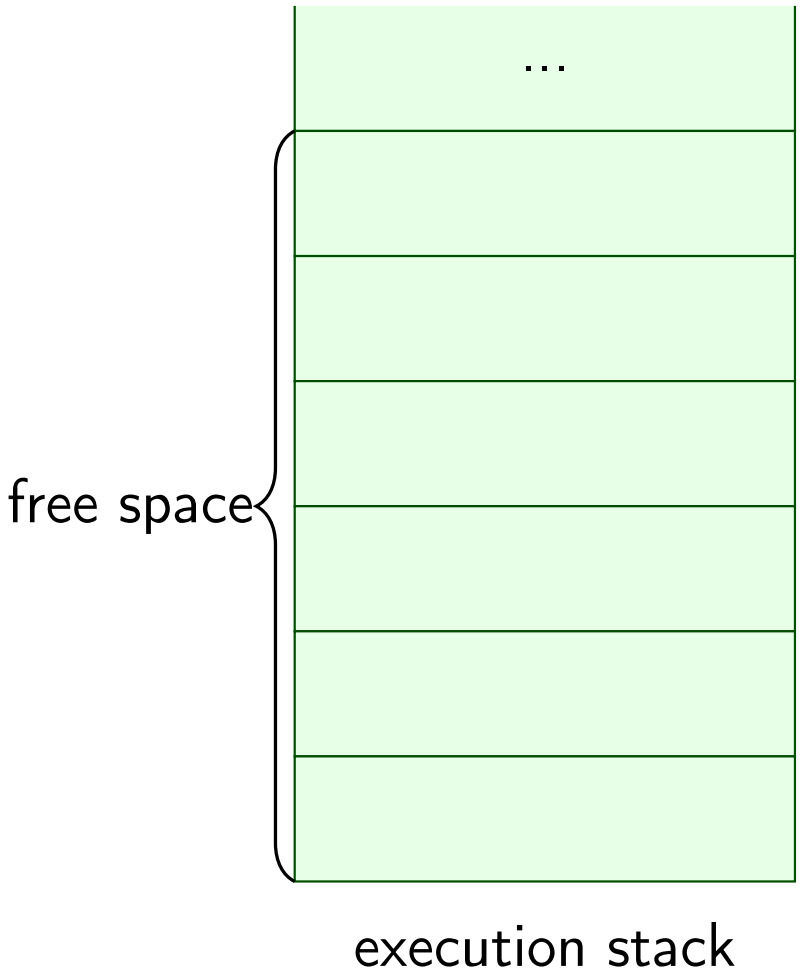
Alprogramhívások nyilvántartása

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



Alprogramhívások nyilvántartása

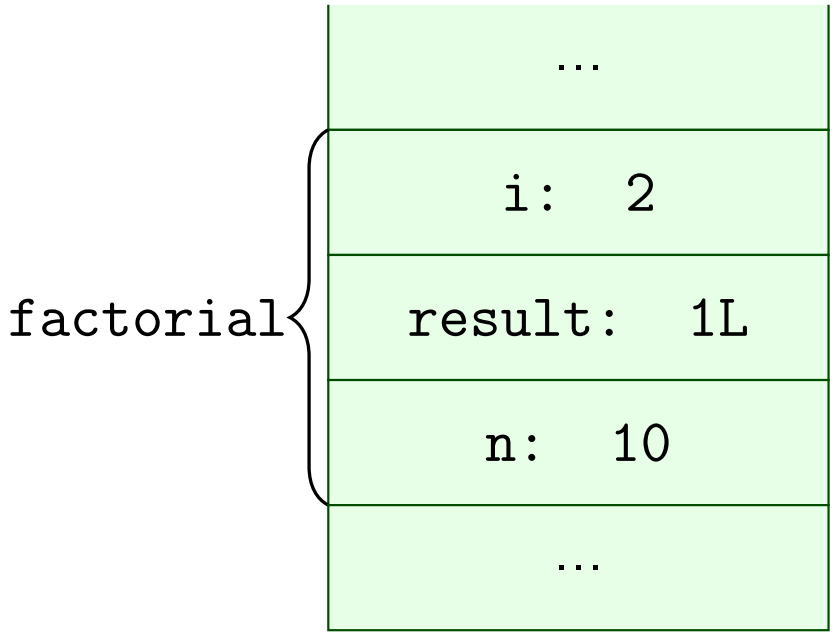
```
void f()
{
}
void g()
{
  f();
}
void h()
{
  g();
  f();
}
int main()
{
  f();
  h();
}
```



Végrehajtási verem

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

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



execution stack

Rekurzió

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

Faktoriális

```
int factorial(int n) {
  if (n < 2)
    return 1;
  else
    return n * factorial(n - 1);
}

int factorial(int n) {
  return n < 2 ? 1 : n * factorial(n - 1);
}

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

Végrekurzív függvény (Tail-recursion)

Kézenfekvő

```
int factorial(int n) {
  return n < 2 ? 1 : n * factorial(n - 1);
}
```

Végrekurzív

```
int fact_acc(int n, int acc) {
  return n < 2 ? acc : fact_acc(n - 1, n * acc);
}

int fact(int n) {
  return fact_acc(n, 1);
}
```

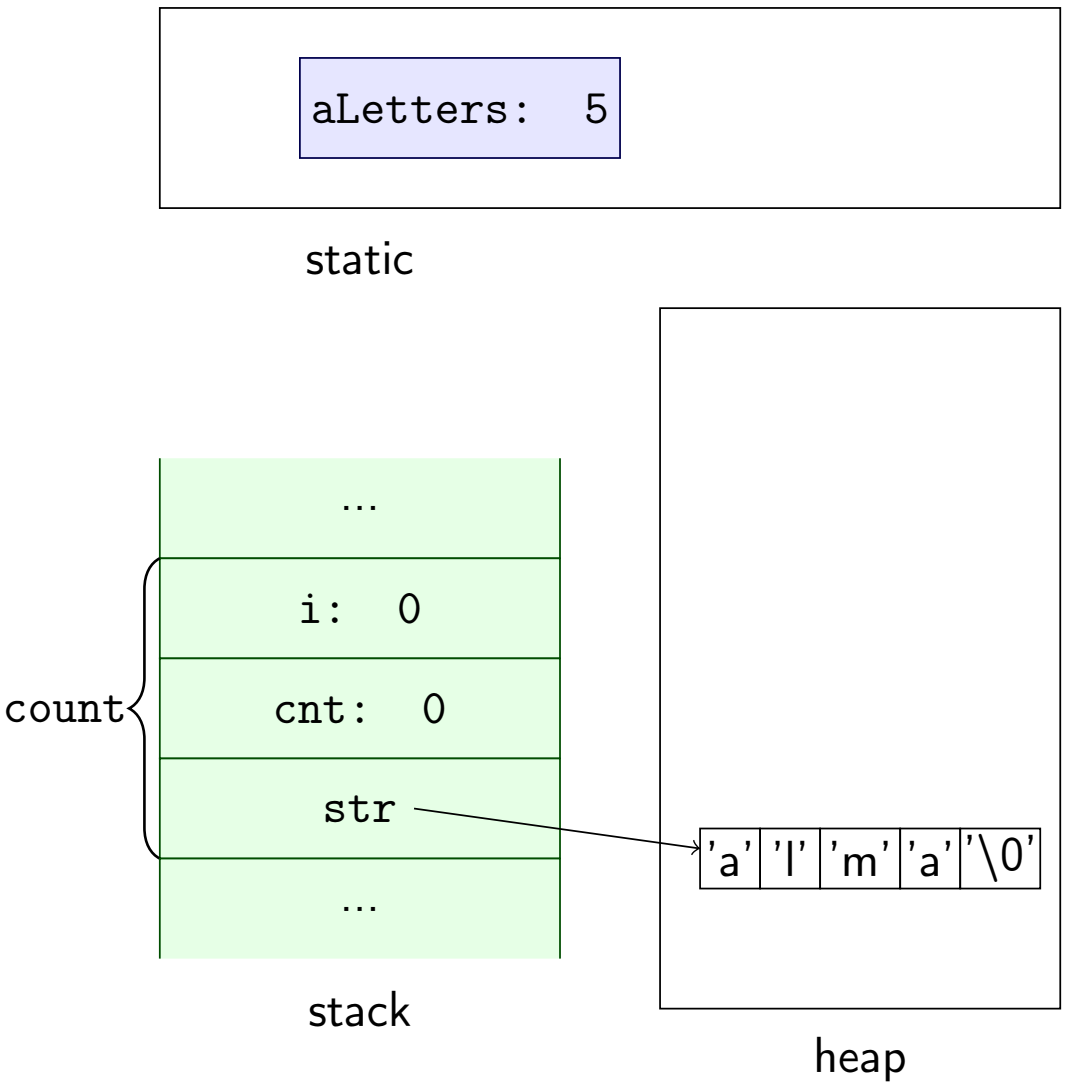
Változók tárolása a memóriában

- Végrehajtási verem → automatikus
- Statikus tárhely → statikus
- Dinamikus tárhely → dinamikus

Változók definiálása

- Deklarációval
  - Statikus és automatikus tárolású
    - Statikus tárhely
    - Végrehajtási verem
  - Élettartam: programszerkezetből
    - A hatókör
    - Kivéve lokális statikus
- Allokáló utasítással
  - Dinamikus tárolású
    - Heap (dinamikus tárhely)
  - Élettartam: programozható
  - Felszabadítás
    - Felszabadító utasítás (C, C++)
    - Szemétgyűjtés (Haskell, Python, Java)

stack - static - 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;
}
```

Automatikus tárolású változó

- Végrehajtási vermen (az aktivációs rekordokban)
- A lokális változók *általában* ilyenek
- Élettartam: blokk végrehajtása

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

Statikus tárolású változó

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

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

Statikus lokális változók

- `static` kulcsszó
- Hatókör: lokális változó (információ elrejtés elve)
- Élettartam: mint a globális változónál

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

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

Alprogram paraméterei

- Definícióban: formális paraméterlista
- Hívásnál: aktuális paraméterlista

```
void f(int x) {
  return 2 * x;
}

int main() {
  printf("42 kétszerese: %d\n", f(42));
}
```

Paraméterátadási technikák

- 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)

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

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