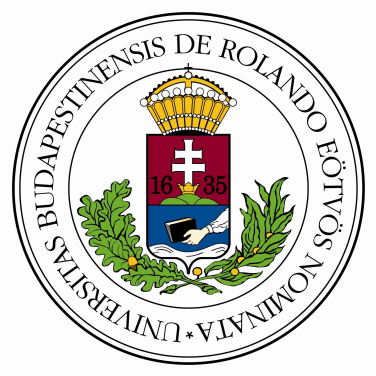
Imperative programming

7th Lecture



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

General Concepts

- Declaration, Definition
- Block
- Scope
- Static Scoping
- Local, Non-local, Global declarations
- Hiding

Loop Variables

C99: Variable Local to Loop

```
for( int i=0; i<10; ++i )
{
    printf("%d",i);
}
printf("%d",i); /* compilation error */
```

C: 012345678910

```
int i;
for( i=0; i<10; ++i ){
    printf("%d",i);
}
printf("%d",i);
```

C: Infinite Loop

```
signed char i;
for( i=0; i<=127; ++i )
{
    printf("%c",i);
}
```

Definition without Declaration

```
double x = x + x
six = double 3
zoo = dooble 10.0
```

```
six = (\x -> x+x) 3
```

```
double = \x -> x+x
six = double 3
```

Higher-order Functions

Functional Programming Paradigm

```
filter predicate (x:xs)
  | predicate x
    = x : filter predicate xs
  | otherwise
    = filter predicate xs
filter _ [] = []

filter even [1..10]
filter (\x -> x > 4) [1..10]
filter ( > 4 ) [1..10]
```

Dynamic Scoping Rules

Bash

```
#!/bin/bash
x=1
function g()
{
    echo $x;
    x=2;
}
function f() {
    local x=3;
    g;
}

f
echo $x
```

C

```
#include <stdio.h>
int x = 1;
void g()
{
    printf("%d\n",x);
    x = 2;
}
void f(){
    int x = 3;
    g();
}
void main(){
    f();
    printf("%d\n",x);
}
```

Outline

Dynamic Program Structure

How is the program working?

- Information about the state of program execution
  - Subprogram calls started from main program
- Storing variables in memory.

We give an abstract model.

Execution Stack

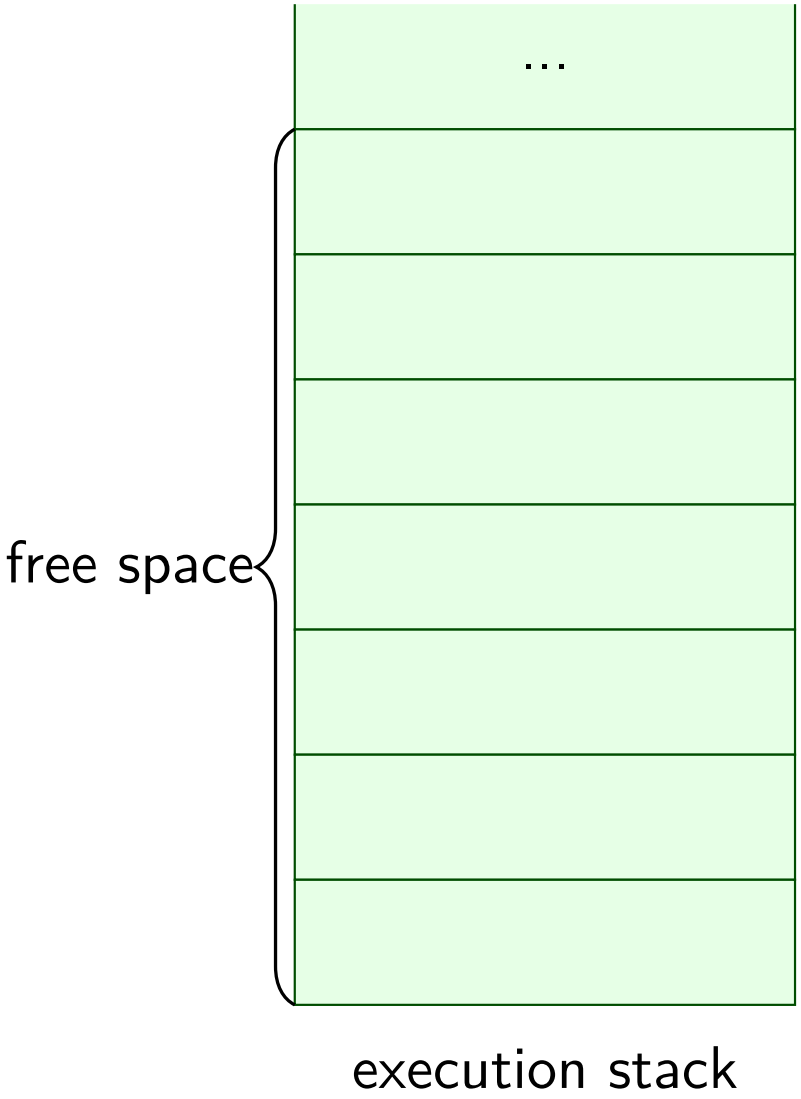
```
void f(void)
{
}
void g(void){
  f();
}
void h(void){
  g();
  f();
}
int main()
{
  f();
  h();
  return 0;
}
```

Execution stack

- Logic of subprogram calls
  - LIFO: Last-in-First-Out
  - Stack data structure
- One record of all subprogram calls
  - Activation record
  - e.g. information on where to return
- Bottom of stack: activation record of main program
- Top of stack: where current execution is

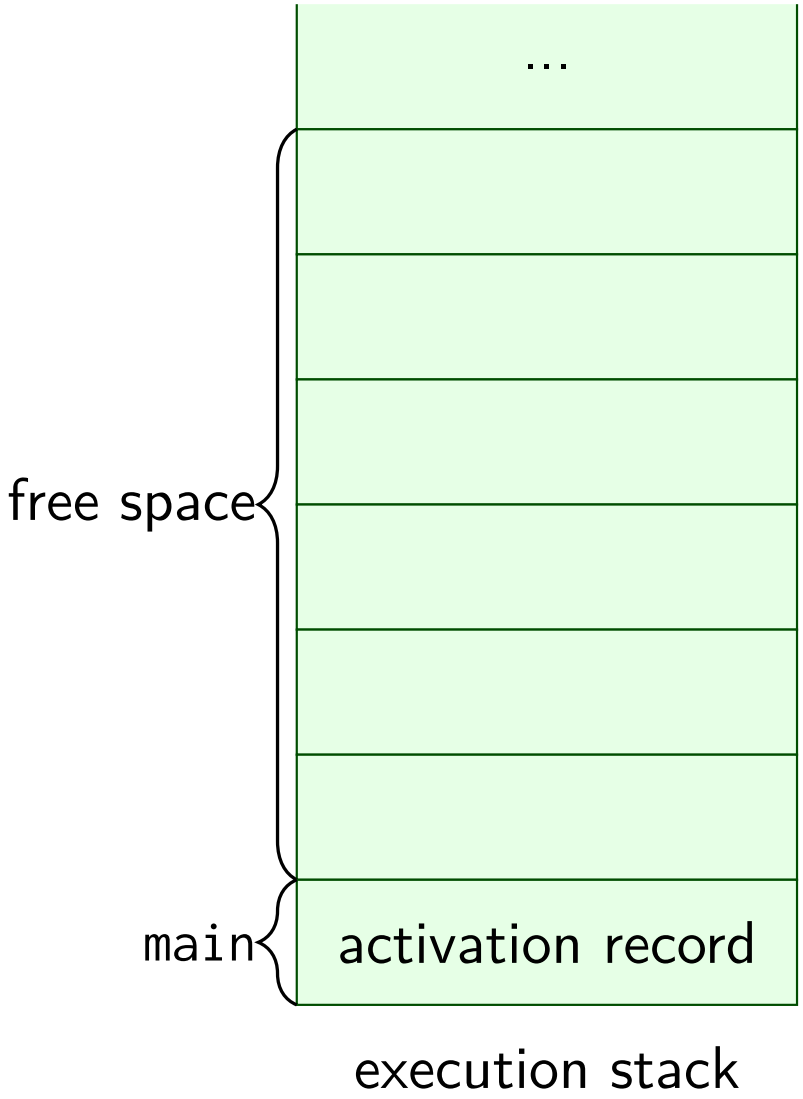
Management of subprogram calls

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



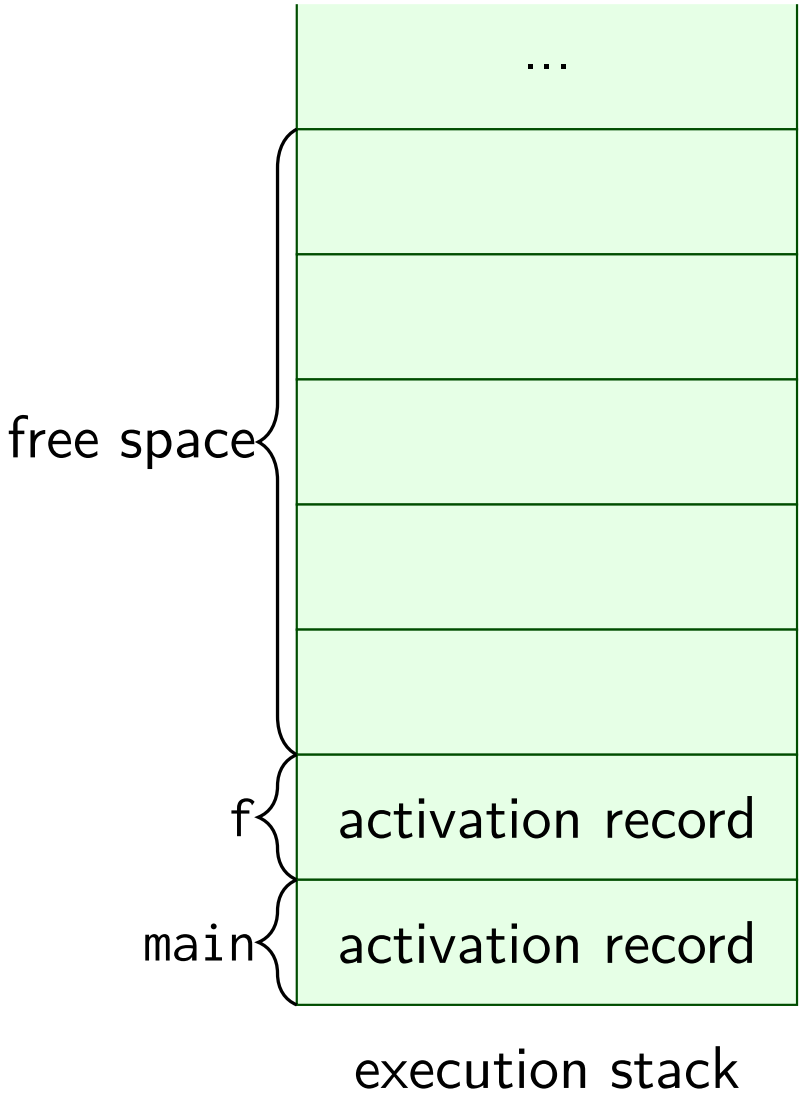
Management of subprogram calls

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



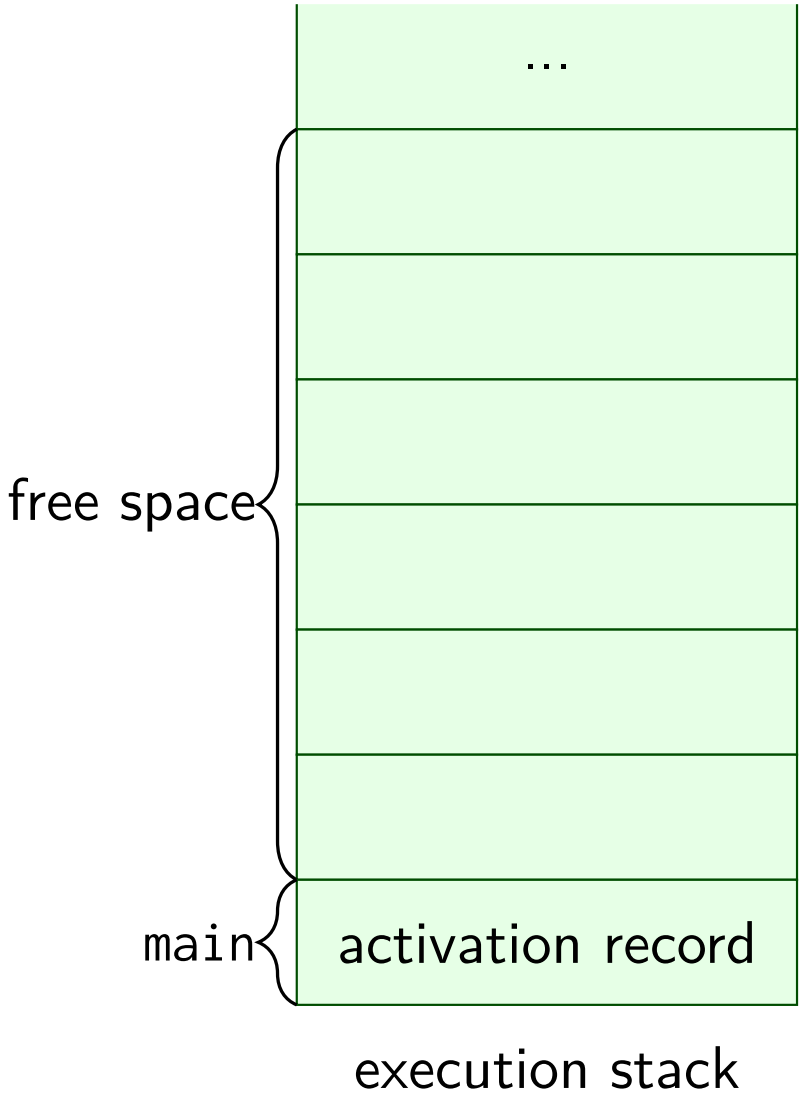
Management of subprogram calls

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



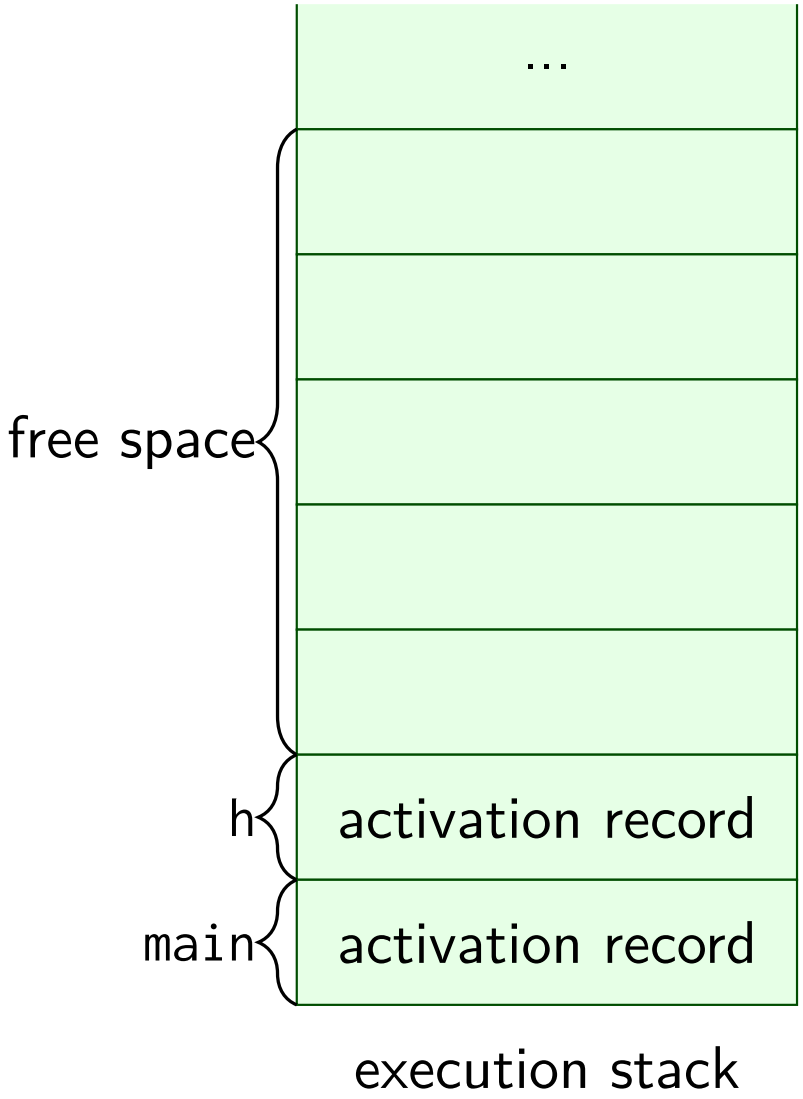
Management of subprogram calls

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



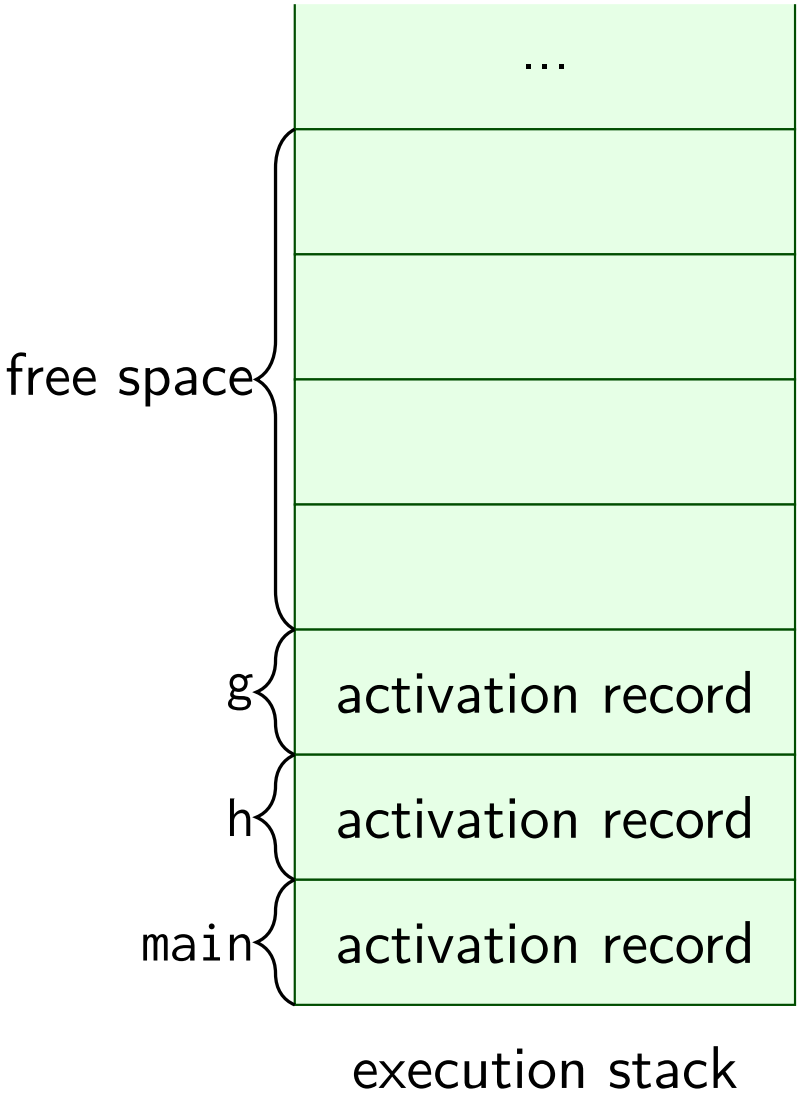
Management of subprogram calls

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



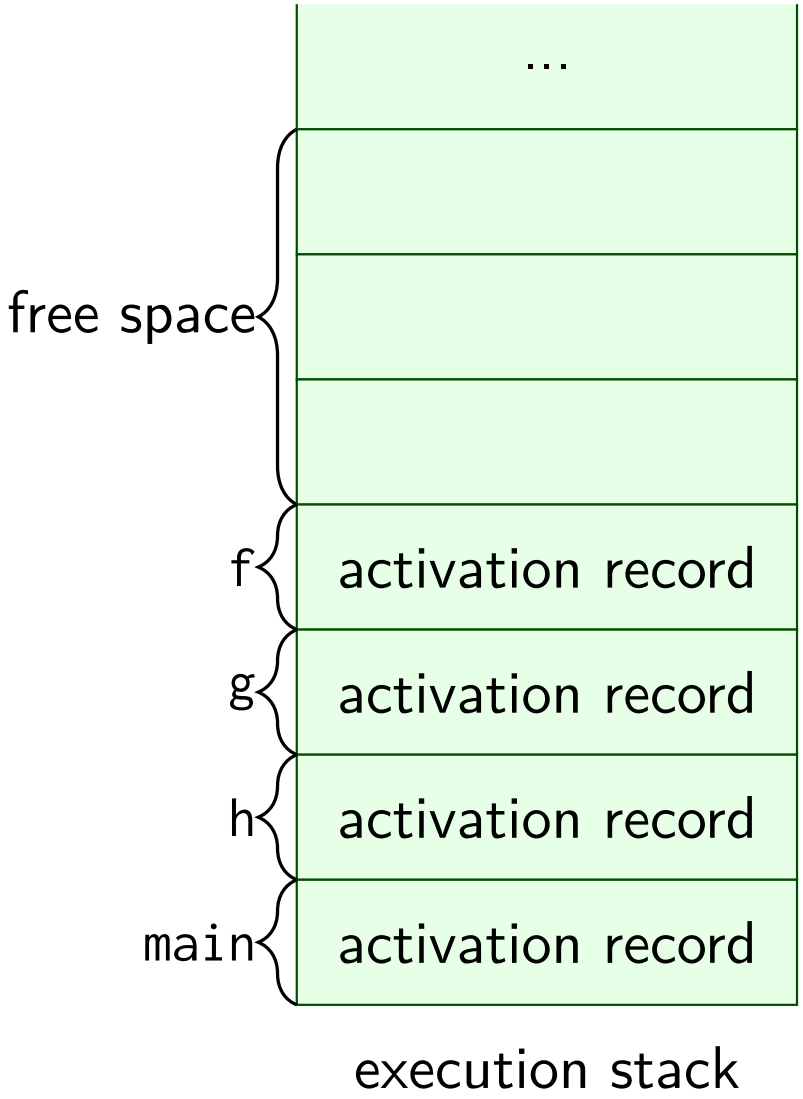
Management of subprogram calls

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



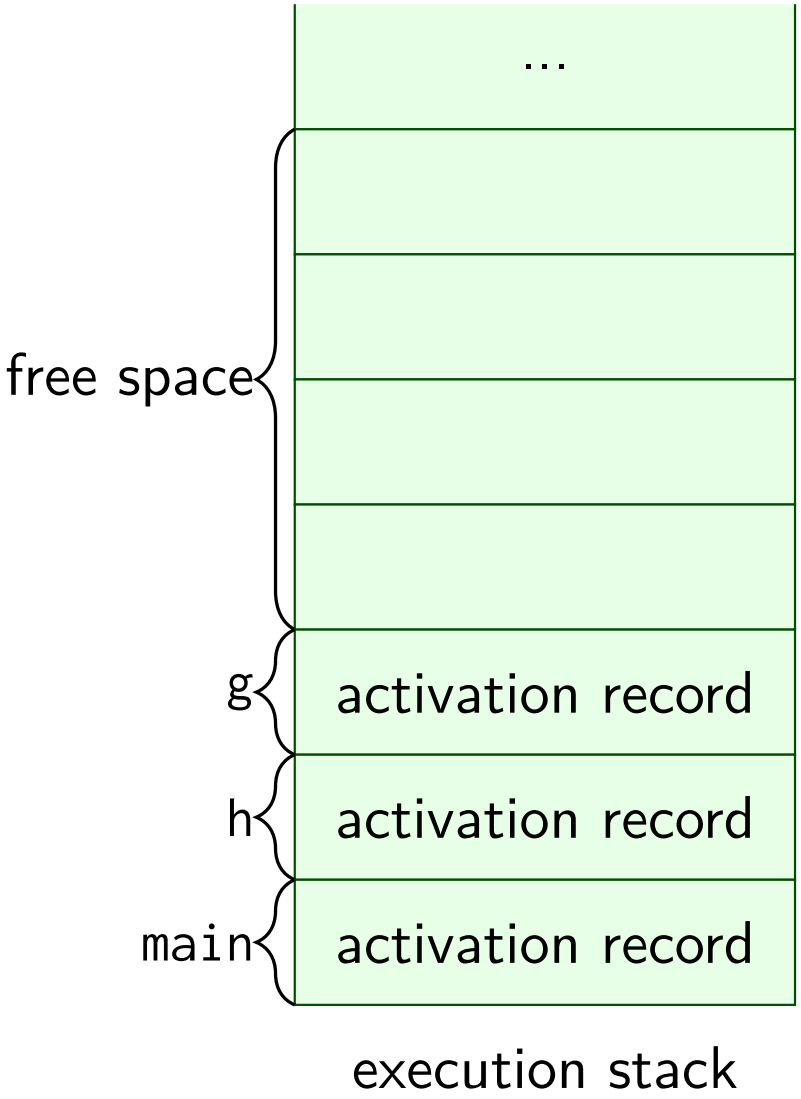
Management of subprogram calls

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



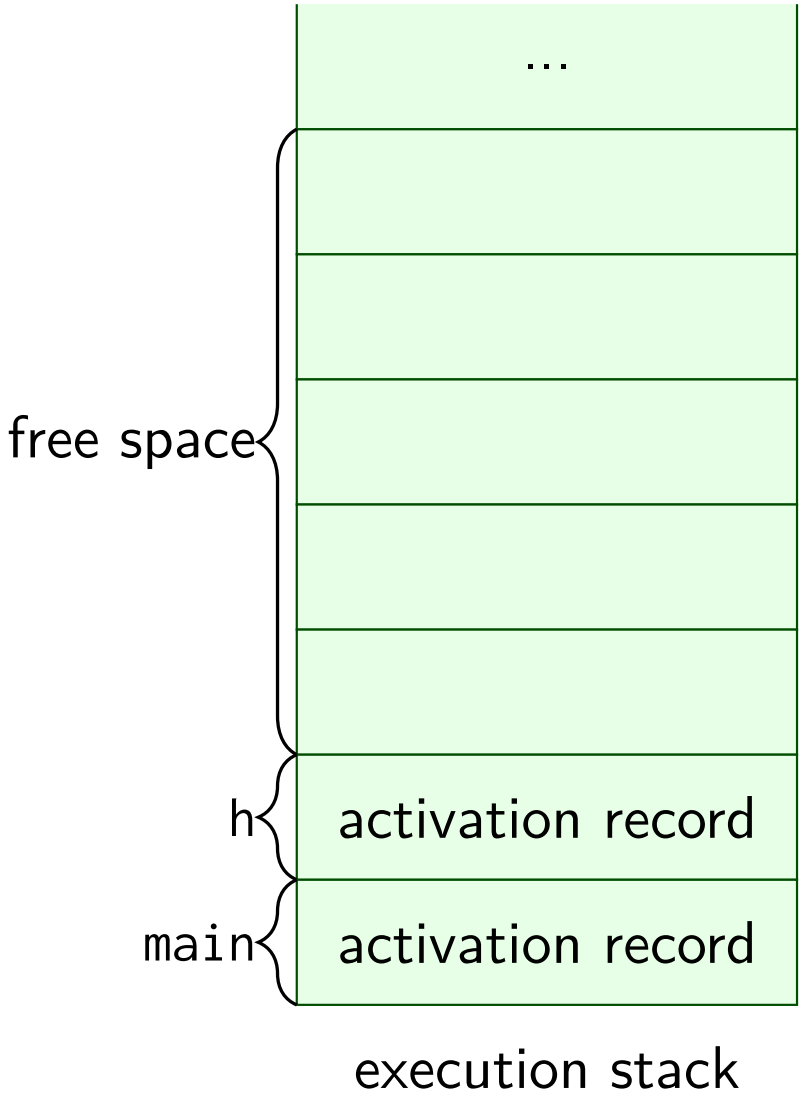
Management of subprogram calls

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



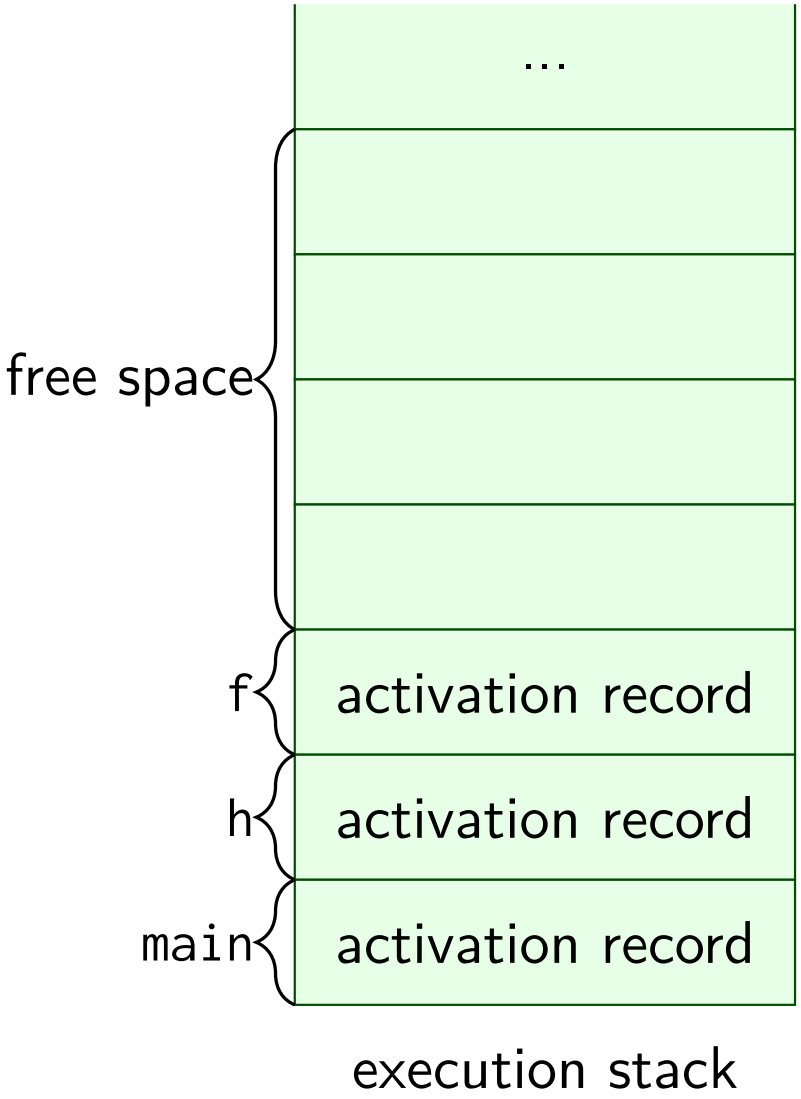
Management of subprogram calls

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



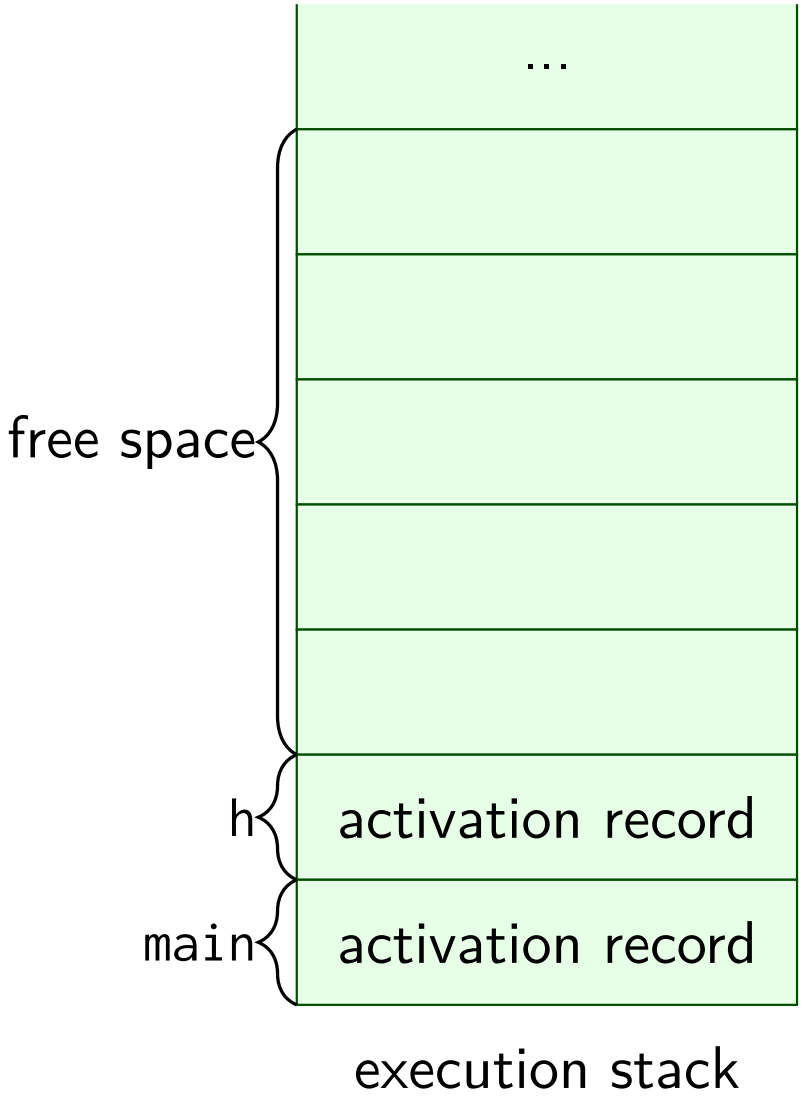
Management of subprogram calls

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



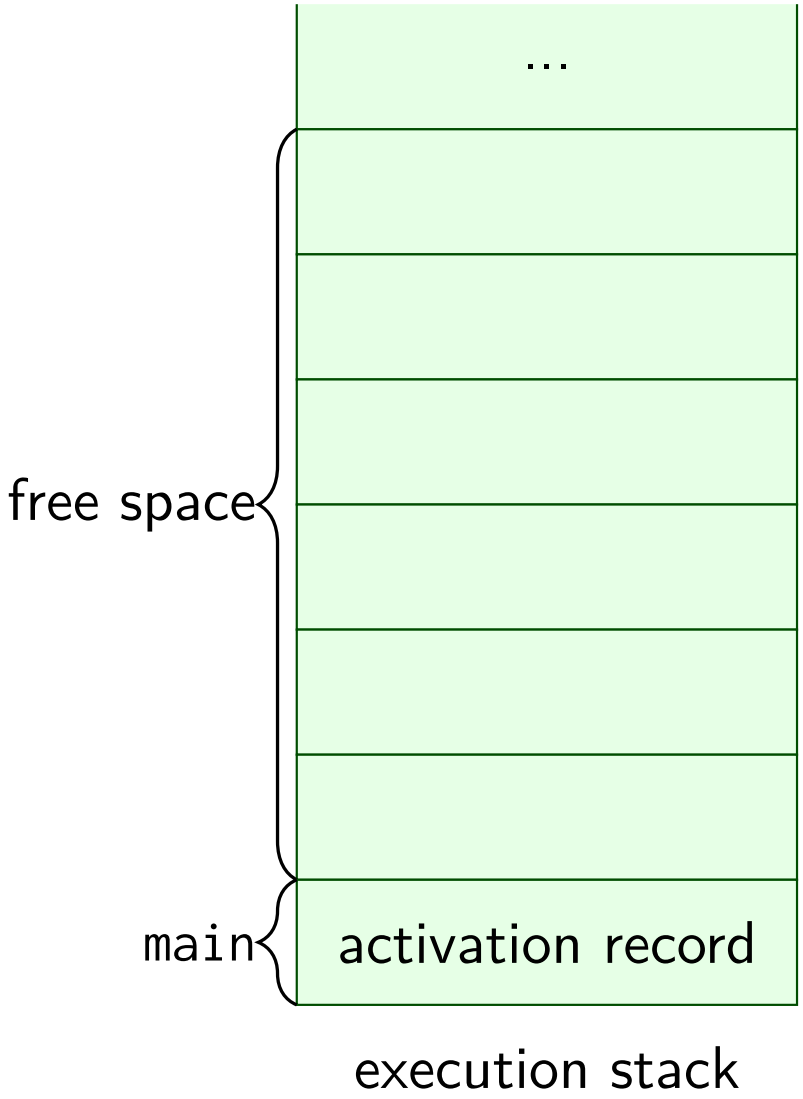
Management of subprogram calls

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



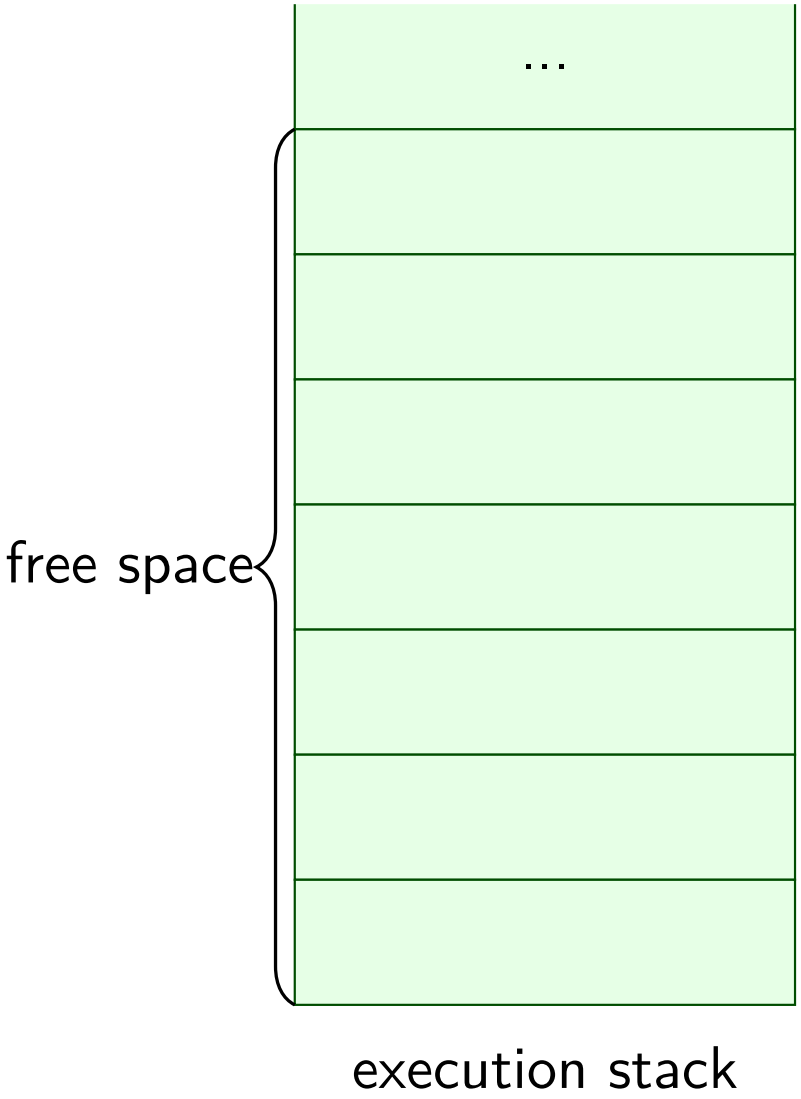
Management of subprogram calls

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



Management of subprogram calls

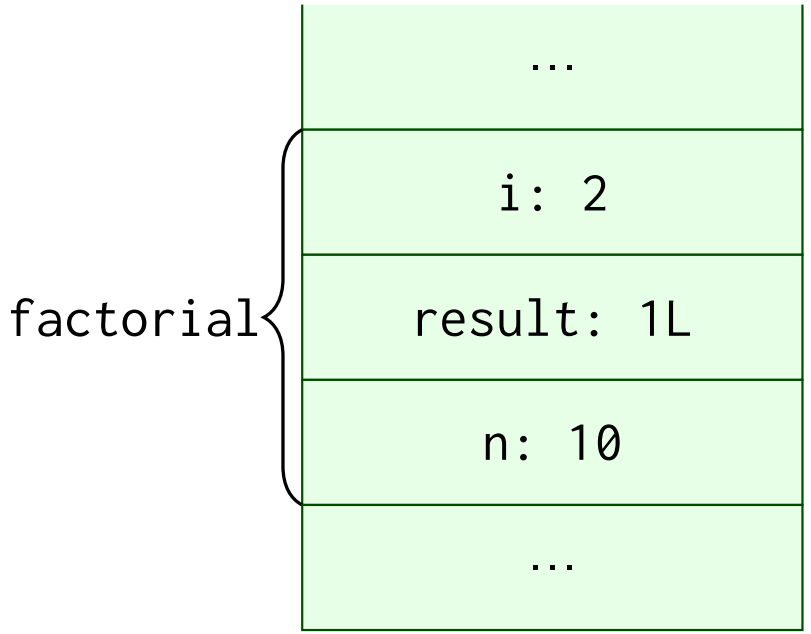
```
void f()
{
}
void g(){
    f();
}
void h(){
    g();
    f();
}
int main()
{
    f();
    h();
    return 0;
}
```



Activation Record

- All sorts of technical things
- Parameters of subprogram
- Local (some) variables of subprogram

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



execution stack

Recursion

- A subprogram calls itself
  - Directly
  - Indirectly
- New activation record of every call
- Too deep recursion: Stack Overflow
- Cost: building/destroying activation record