Python

No predeclaration of variable names, types

Mutable / immutable values

Memory reuse - garbage collection

C, without the gory details

Variables have types int, float

Must be declared in advance

Program is a collection of functions

if, while, for etc

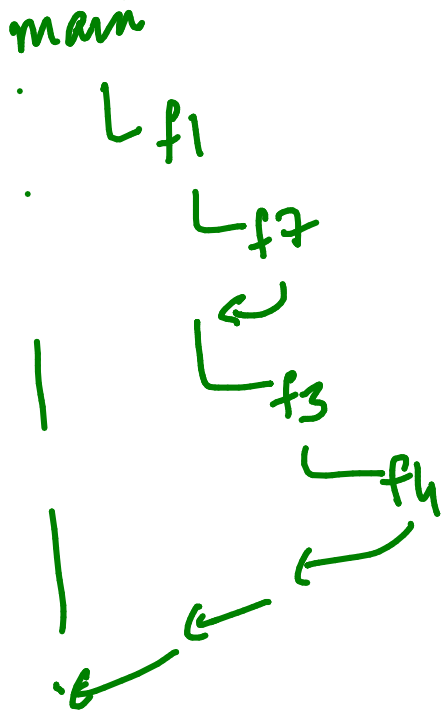
Typical C program

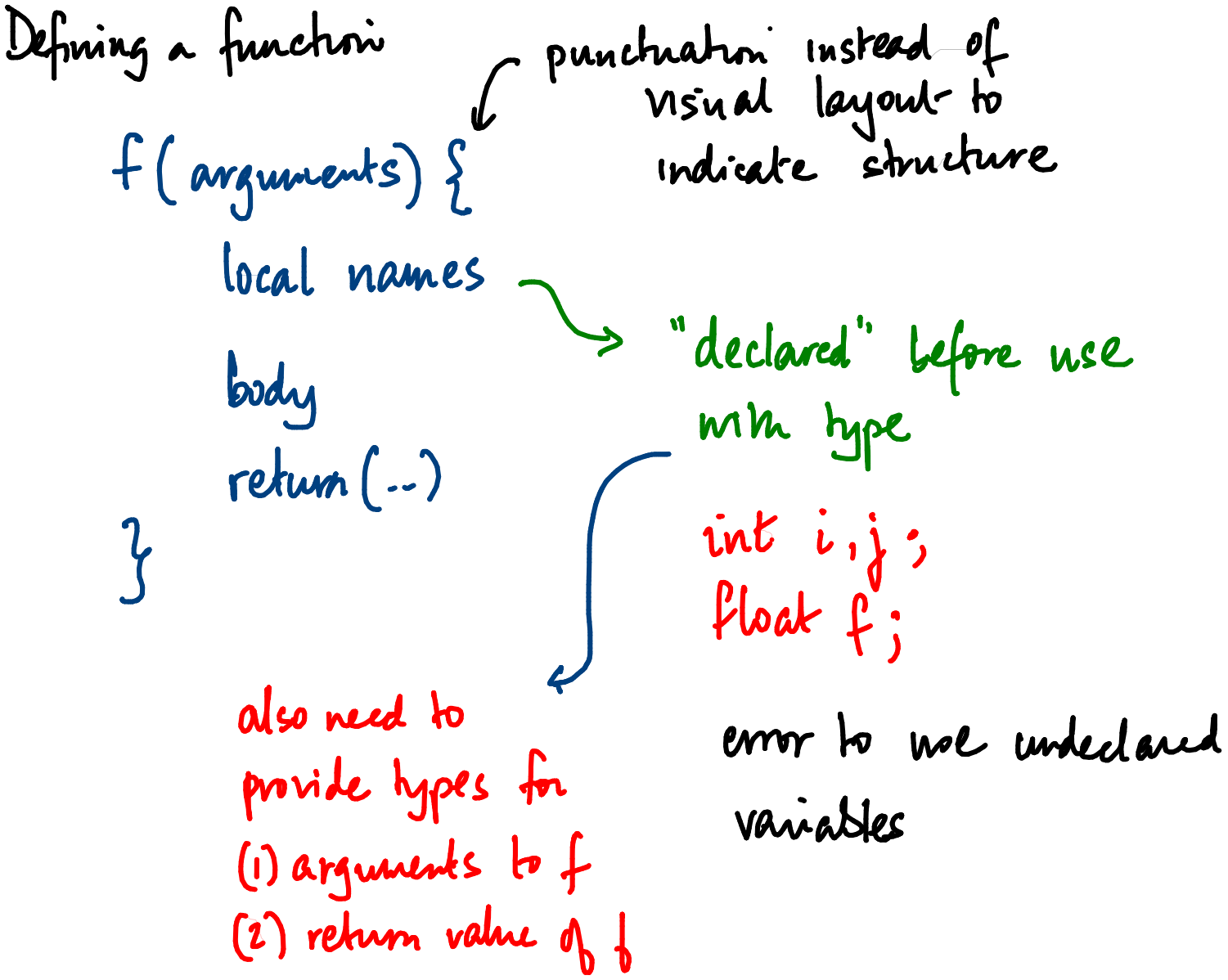
```
f1(..) {
   :
}

f2(..){
   =
}
:
fn(..){
..
}
```

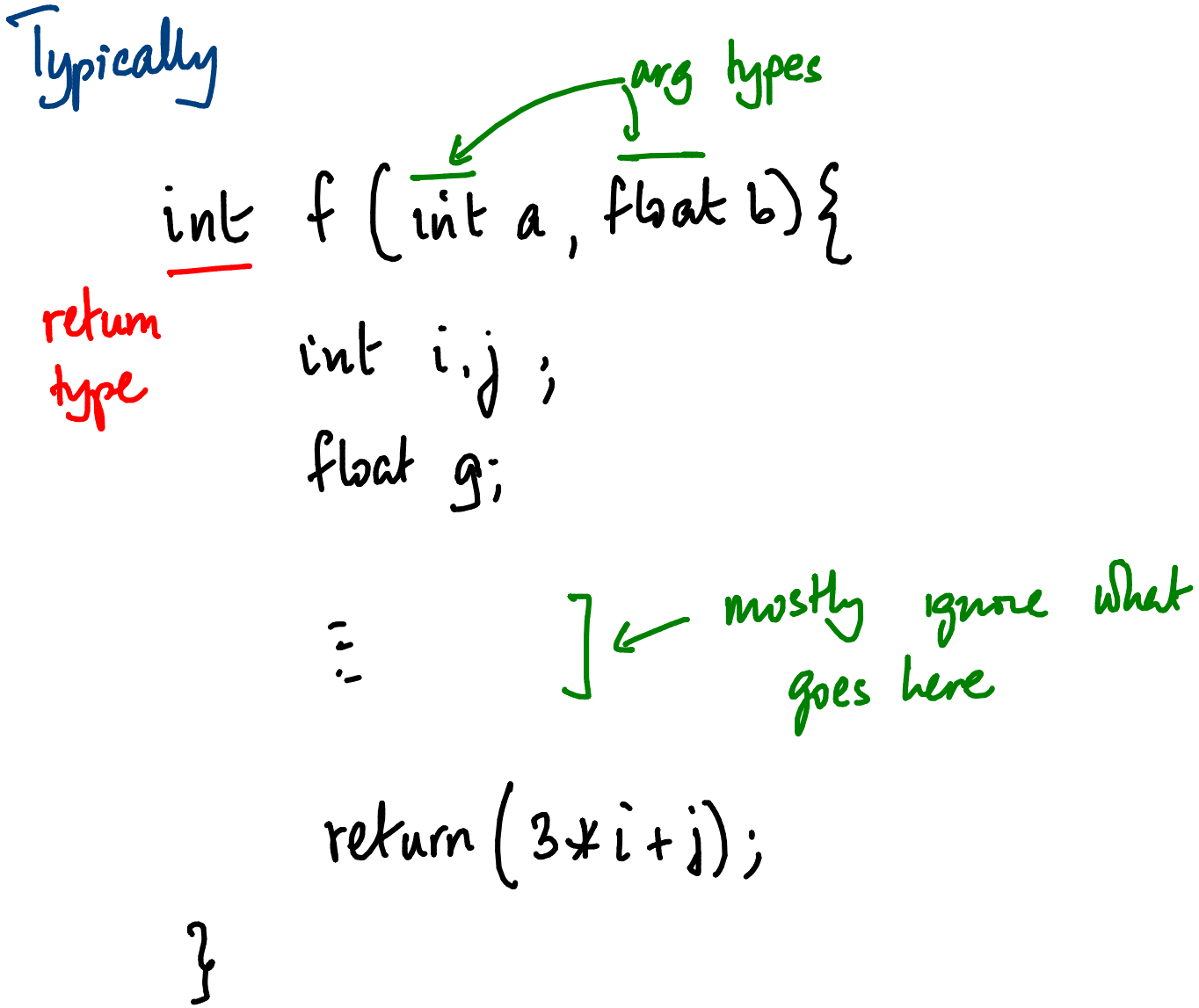
Special function name main()

where execution starts

choice of name is arbitrary




Defining a function
punctuation instead of visual layout to indicate structure
f(arguments) {
local names
body
return(--)
}
"declared" before use with type
int i,j;
float f;
error to use undeclared variables
also need to provide types for
(1) arguments to f
(2) return value of f


Typically
arg types
int f (int a, float b) {
return type
int i,j ;
float g;
:
mostly ignore what goes here
return (3*i+j);
}

Memory model

Recall that we read/write from/to disk in blocks

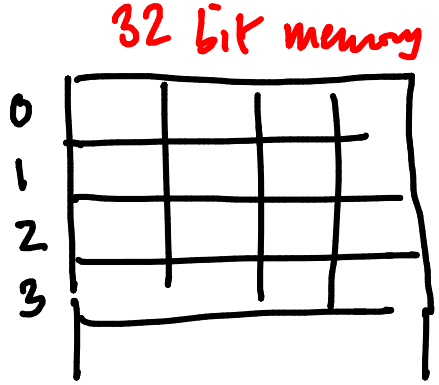
≈ 4KB

Memory is organized in words

1 byte = 8 bits

Typical word is 4 bytes "32 bits"

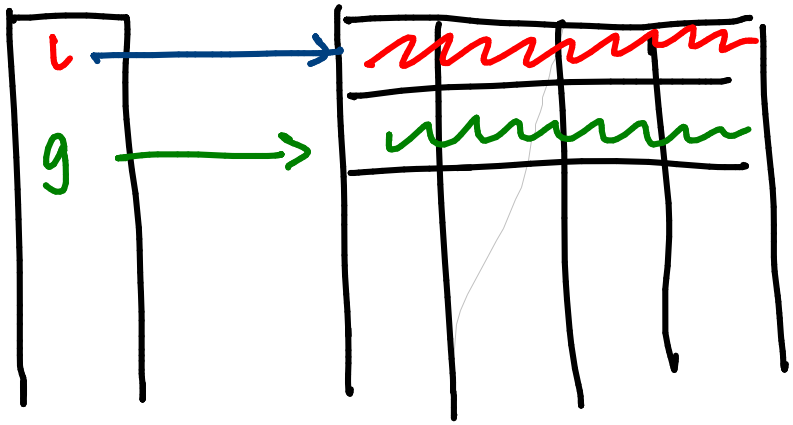
8 bytes "64 bits"



C like language

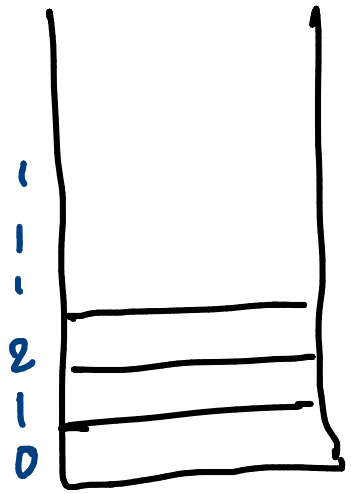
```
int i;
float g;
```

→ allocate 1 or more words per variable



Static assignment of memory words to variables, based on declarations

Better to think of memory growing upwards



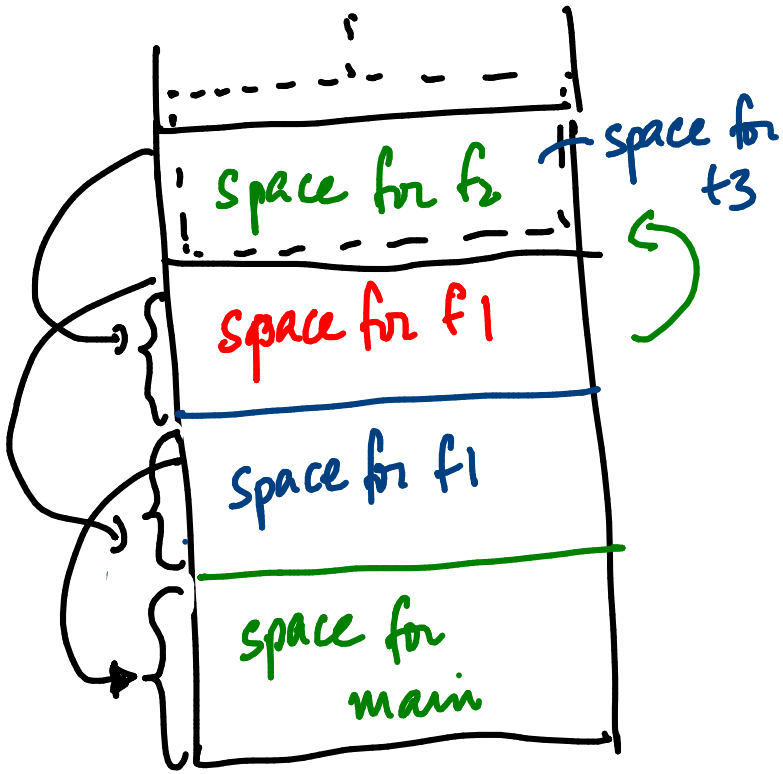
When is memory allocated?

Statically, when reading function definitions?

Recursive calls create multiple simultaneously active copies of function

∴ Dynamically allocate memory

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



"STACK"

main()

f1()

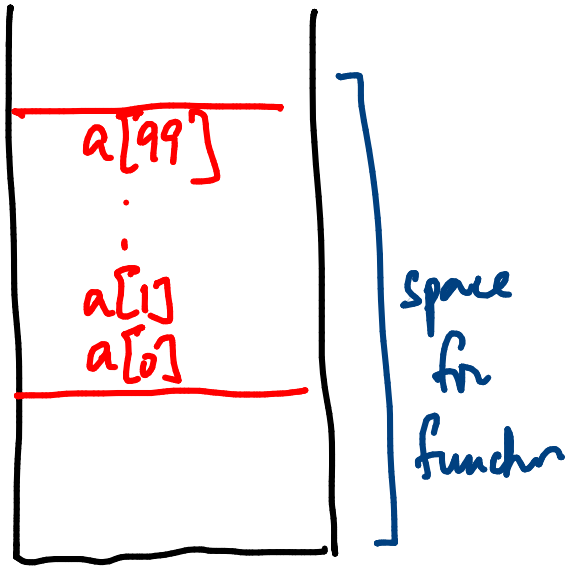
f2()

main
└f1
└f1
└f2
└f3
...

Arrays

```
int a[100];
```

Fix a block of 100 words, called a[0], a[1], .. , a[99]



Know where a[0] lies

a[j] position can be computed

a[0] + j words

Hence random access