Sept. 17

Binding: associating a name to the thing it represents.
e.g. variable: type, value, memory location, etc.

Binding Times: Time at which a binding is created.

- Language definition time
- Language implementation time
- Program writing time
- Compile Time
- Link Time
- Load Time
- Run Time

static bindings ↑ ↓ Dynamic bindings.

~~Referencing Environment~~

Program (c)

```
#include <stdio.h>
int gcd(int a, int b){
    while (a != b) {
        if (a > b)
            a = a - b;
        else
            b = b - a;
    }
    return a;
} //gcd()
```

```
int main() {
    int x, y;
    //input x, y
    int g = gcd(x, y);
    //print g
    return 0;
} //main()
```

Bindings: Program Writing Time

gcd: function, 2 int params, returns int
a, b: int, parameters
main: function, no params, returns int
x, y: int
g: int

Run Time

//Input x, y
a = x, b = y
values of a + b
value of g

Lifetime : when + how long (i.e. time) during which a name has a binding.

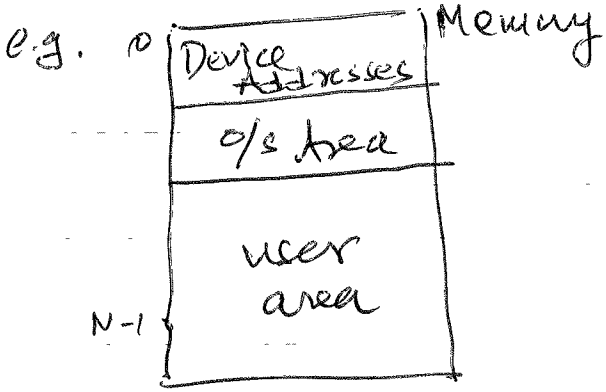
Object Lifetime e.g. int a;

name 'a' has a storage (object) associated with it to store an int value.

Time between creation + destruction of an object is object's lifetime.

Object lifetime depends on storage allocation scheme used by language implementation.

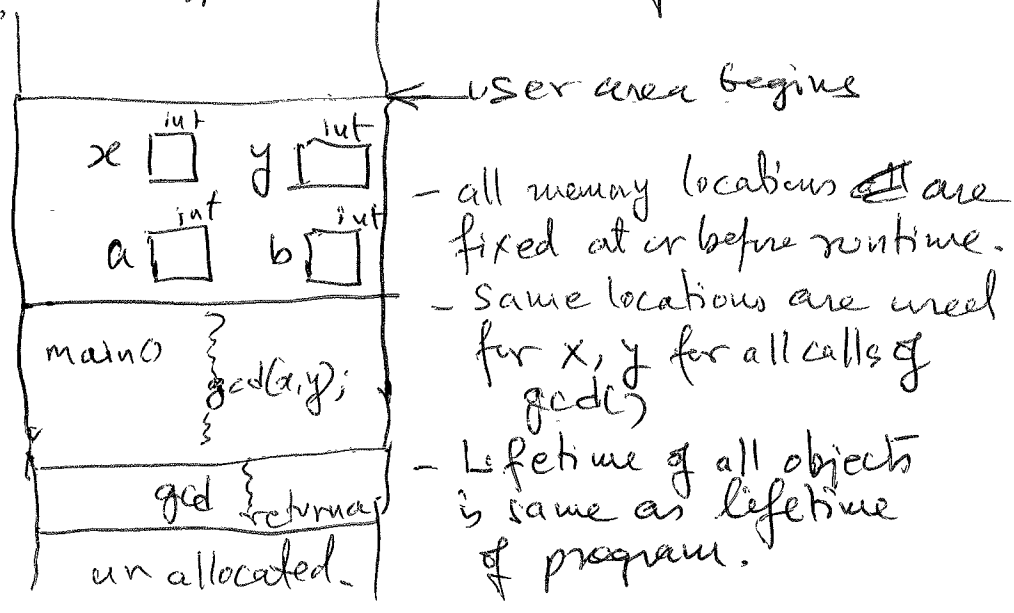
e.g.



Static Allocation

when all objects are allocated before runtime.

e.g.



Consequences:

- Could be potential source of errors

```
gcd() {
  int w;
  // do something with w
}
```

upon return + reinvocation, last value of w is retained.

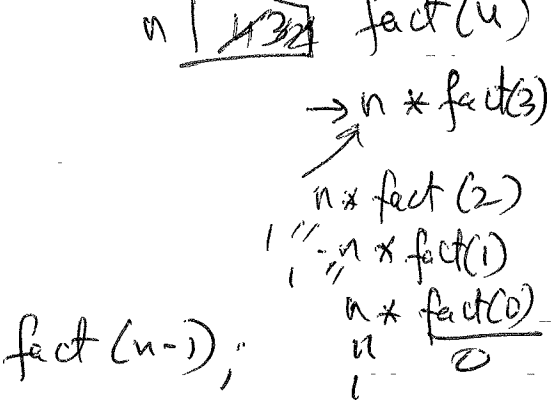
- Does not allow recursion!!

e.g.

```
int gcd(int a, b){
  if (a==b)
    return a;
  if (a > b)
    return gcd(a-b, b);
  else
    return gcd(a, b-a);
}
```

e.g.

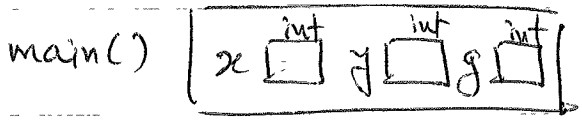
```
int fact(int n) {
  if (n==0)
    return 1;
  else return n * fact(n-1);
}
```

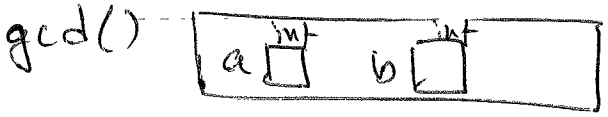


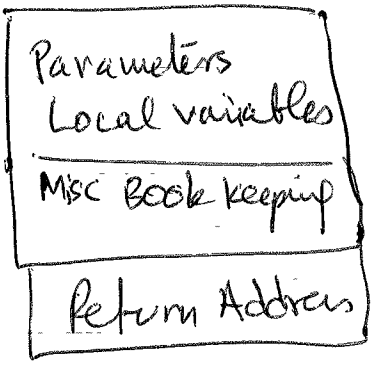
how to save the recursive calls!

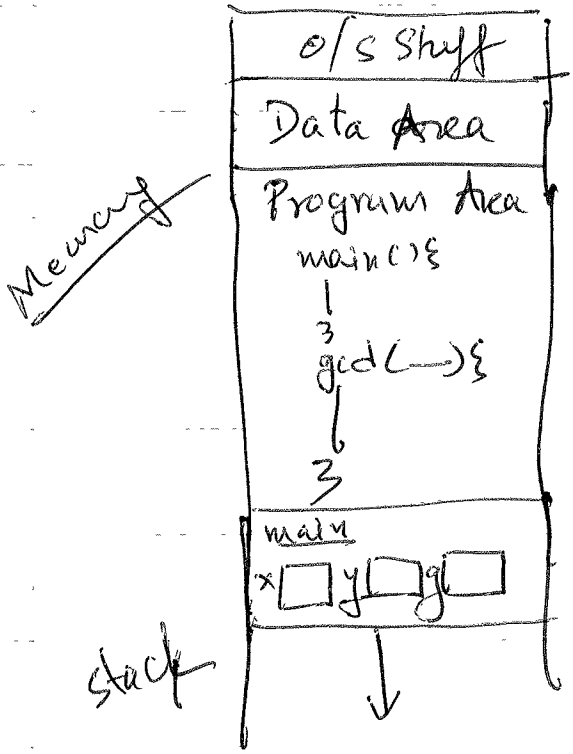
Stack-based Allocation

Every function call has a Stack Frame. (aka Activation Record)



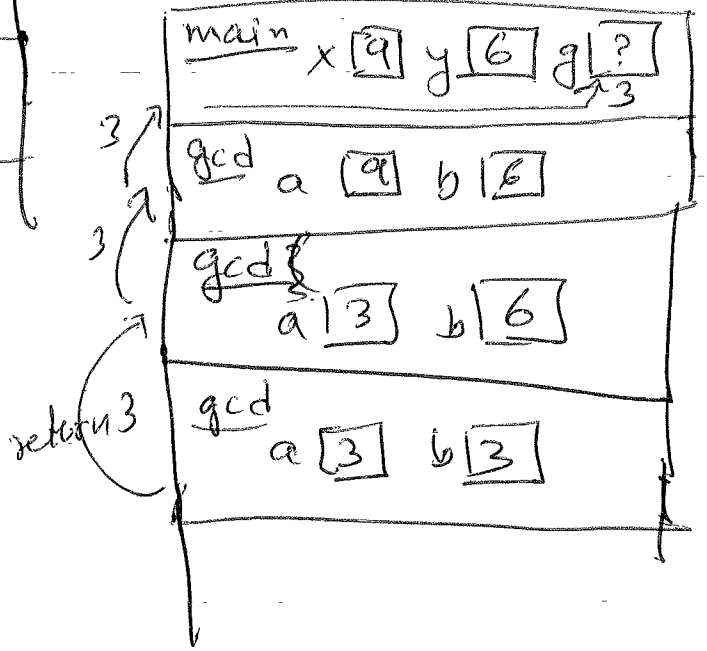






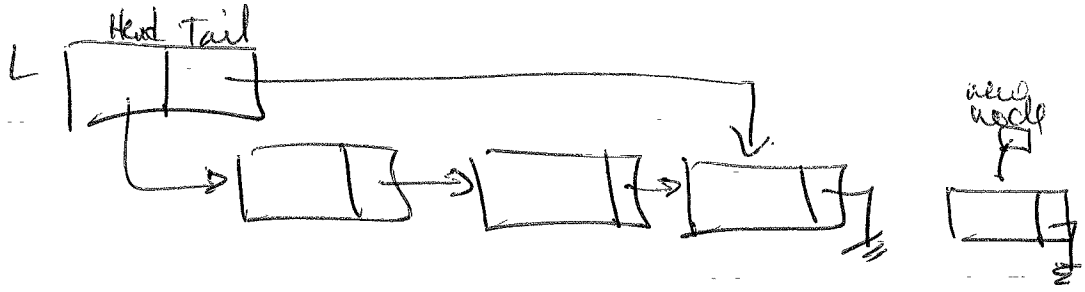
A new stack frame is created during runtime for each function call.

e.g. when computing gcd(9,6)



Heap-based Allocation

For dynamically allocated data



JAVA

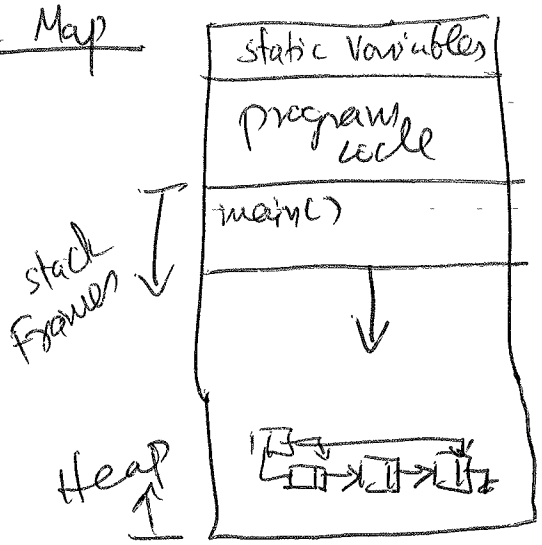
```
Node * newNode = new Node(__);
```

C

```
Node *newNode = (Node) malloc(sizeof(Node));
```

Q: Where is newNode allocated?
A: On a [Heap] of bytes in Memory.

Memory Map



when does stack meet heap? stack smash!!

PL decisions

- Who/when memory allocation is done
  - statically - no recursion!
  - dynamically
- How?
  - explicit
    - Java: new
    - C: malloc, free, cfree
    - C++: dispose, new
  - implicit
    - Python

- What runtime support is needed?
  - When allocating any checks done?
    - NO — C
    - YES — JAVA, PYTHON, etc
  - Can used space be reused?
    - NO — C
    - YES — Java, Python

Needs: Runtime Garbage Collection! ← (less efficient)