heap allocation: for dynamically allocated things

heap A B C used D free (hole)

hole list

on dynamic allocation, search hole list for big enough space

L = first fit
L = best fit
L = worst fit

internal fragmentation: unused space inside allocated blocks
p = minlloc (100)

external fragmentation: space left from allocs/deallocs

compaction to unfragment (but need to find/repoint pointers)

A B C D
garbage collection: explicit deallocation - programmer uses
specific functions (e.g., free) to free space (delete/free)

fast, easy to implement (for language designers)

harder to program with

implicit deallocation - automatic freeing when
no more pointers to a block

garbage collection takes extra time
to track references, need to write garbage collector at runtime
scope: part of source code where a particular binding is active
(or a part of source code with same bindings in effect)

reference environment: sequence of scopes inner ➔ outermost

globals only (single scope): BASIC (also no declarations)

```java
int i;
for (int j = 0; j < 10; j++)
    int i;
    confusing?
};
```

```java
class A {
    public int score;
    public A(int score)
    {
        this.score = score; // scope resolution (C++ A::score=score)
    }
};
```
public class C {
    void foo() {
        x = 1;
        super.x = 2;
        super.super.y = 3;
        A.this.y = 3;
        ((A) this).y = 3;
    }
    class X {
        int i;
    }
    class Y {
    }
}