List , Tuples , Set

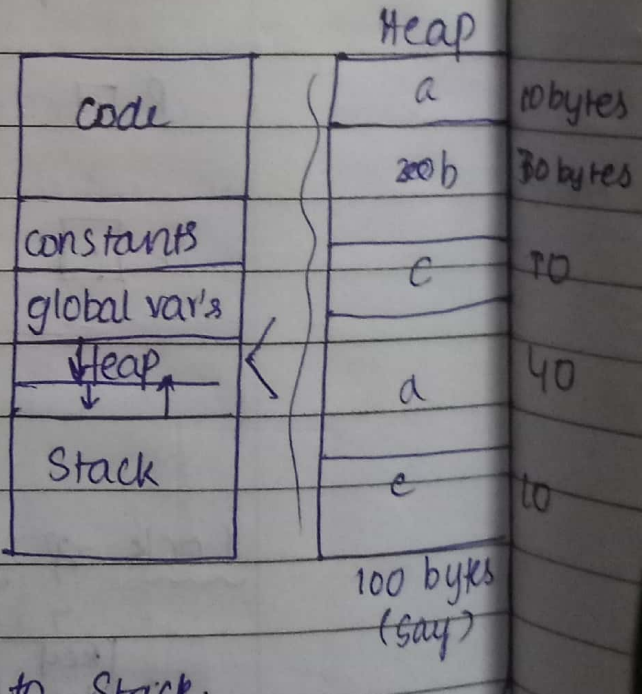
Set – 1) List   no repetitions are allowed in sets
2) Array
3) Bit vector
4) Hash Table

Files – resides on disk, survives on power off.

Static Variables
1) Internal Static var's :-
Inside a function
Retains values across function calls.
Generation of random no's.

9/2/18

<u>Heap management</u>:
Remaining part consists of 2 parts
1) Stack (Call Stack):
Activation record being pushed when a f^n is called.
→ Dynamically growing & shrinking of mem.
→ Heap grows opposite to Stack.
→ Sometimes we have a problem of space available for activation record.



malloc(10)
on this instⁿ, it checks whether 10 bytes of ~~sp~~ contiguous space in heap (memory).
free(c)
free(e)

☆ Now if user claims 5 bytes, where would the memory be allocated c/e?

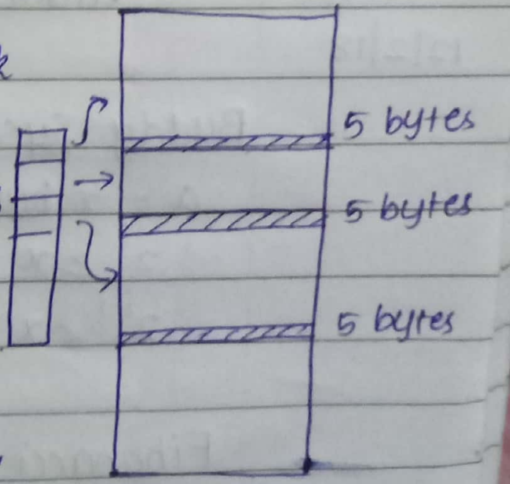
<u>First fit algo</u>: whenever request for malloc is triggered, find a hole ~~row~~ (which comes first).

<u>Best fit algo</u>: Scan entire heap and find a hole which exactly fits / min. of all available holes.

10 bytes
30 bytes
70
40
10

<u>Worst fit algo</u>: Scan all free areas & return the max ~~si~~ hole size.
$\hookrightarrow$ 1, 2

If we want to keep track of how much free size is available, it would cross the available free space.



① External Fragmentation

2) Internal Fragmentation:
depends on granularity i.e., min. size available.

After all this survey, it was decided that simply allocate the space which comes first

→ To handle external fragmentation, we move all the holes to one side, this process is known as Compaction.

☆ → How will program work now?

Now we'll not return heap pointers to the programmer but the indices that are introduced to point that allocated space.

Adv: 1) Compaction is easy

Disadv: 1) Storing space inc.

2) Every access gets slowed down.

Instead of wasting time, waste space but keep track of all free info.

Q. Which data structure would likely be store that?

linked list.

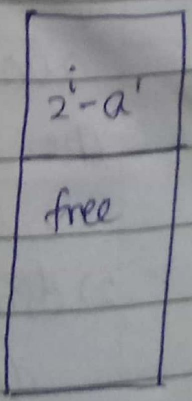
2|2|18

Buddy System:

a - x bytes

$2^i \geq x$

$2^{i-1} < x$



Fibonacci Heap:

Garbage Collection:

<u>Type</u> <u>Descriptor</u> → keeps track of pointers in all types ←

Every array, f^n, structure, typedefs, ~~global~~ global & local vars

Addresses at which pointers to these were stored?

Record, offsets store the locations of each type

Mark & Sweep:

1. Mark all heap elements as inactive.
2. Starting from pointers outside the heap. mark the pointed heap elements as active.
3. Collect out the inactive elements.

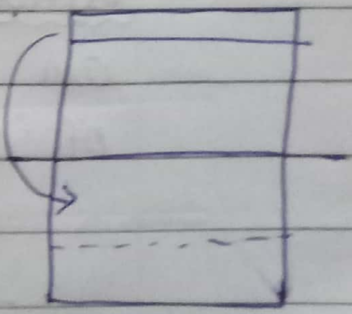
when heap meets stack, space is filled then we go for garbage collection, ~~that~~ which itself requires stack for that which is not available anymore with us.

→ Mark itself with 2 pointers.

19/2/18 **Stop & Copy:**

while marking all active heap elements they are copied into 'passive'.

&



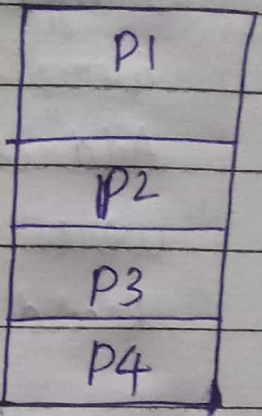
Now they become contiguous.

Later we'll assume passive to be in virtual mem & active to be in Physical mem where when referring to individual swaps i.e., passive ↔ active.

Generational Collection:
Divide the heap into multiple small parts.

→ We start the allocation from part1.

→ If some element in P1 is active for long time ~~and~~ then it is promoted to P2.



→ Here garbage collection goes through parts

Conservative Collection: We can't do compaction.

3/2 Names & Scopes:

Static Scoping: Every ref to a var can be mapped to one particular def^n. in the program.

Block structured languages:
Every f^n def^n is block.

Dynamic Scoping:- In a f^n, if we make an access to var 'a', but definition of 'a' is not found in 'f', 26/2

Alaises:- 2 or more names referring to same object.

$*p = 10;$

$a = *p;$

$*q = 3$

$b = *p;$

Var - value model of Variables:-

$a = 2$	a $\boxed{2}$	
$b = 2$	b: $\boxed{2}$	
$c = a+b$.	c: $\boxed{4}$	
	value model	var model.

$$\begin{tikzcd} a \arrow[r] & \boxed{2} \\ b \arrow[ur] & \\ c \arrow[r] & \boxed{4} \end{tikzcd}$$

Sequence Control

Expressions

$d = a + b - c$

① 3-add codes

$R_1 \leftarrow a$

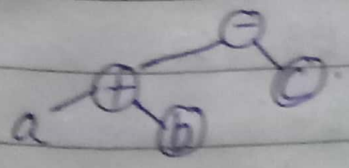
$R_2 \leftarrow b$.

$R_1 \leftarrow R_1 + R_2$

$R_2 \leftarrow c$

$R_1 \leftarrow R_1 - R_2$

② Expression Tree



③ Post fix form / Prefix

abc * +

Order of Evaluation

$a - f(b) - c * d.$

1/3 Boolean Expressions

```
if ((a>b) and (b>c))
    then   x = x+1
    else   x = 0.
```

short-circuiting :- If a>b is false, there is no need of checking b>c, this is short-circuiting.

if(((a>b) and (c>d)) or (e != f)) then S1 else S2;

```
        CMP   R1, R2
        JLE   L1
        CMP   R3, R4
        JLE   L2.
    L1: CMP   R5, R6.
        JE    L3
    L2: S1
        JMP  L4.
    L3: S2
    L4:
```

with short-circuiti

Scanned by CamScanner

```
Case x of
        1 : Stmt A
      2,7 : stmt B
      3,5 : stmt C
       10 : stmt D
     ELSE : stmt E
END.
```

```
            r1 <- x.
        CMP   r1, #1          L1: Stmt 1
        JZ    L1                  JMP L7
        CMP   r1, #2          L2: Stmt 2
        JZ    L2                  JMP L7
        CMP   r1, #7          L3: Stmt 3
        JZ    L2                  JMP L7
        CMP   r1, #3          L4: Stmt 4
        JLT   L6                  JMP. L7
        CMP   r1, #5          L5: Stmt 5
        JLE   L3.
    L6: CMP   r1, #10
        JZ    L4.
        JMP   L5
```

H/w write a program using switch & less no. of Labels.

Scanned by CamScanner

Loops

```
for i ← 1 to 10 step1 do
        S1;
```

$r_3 \leftarrow 1$

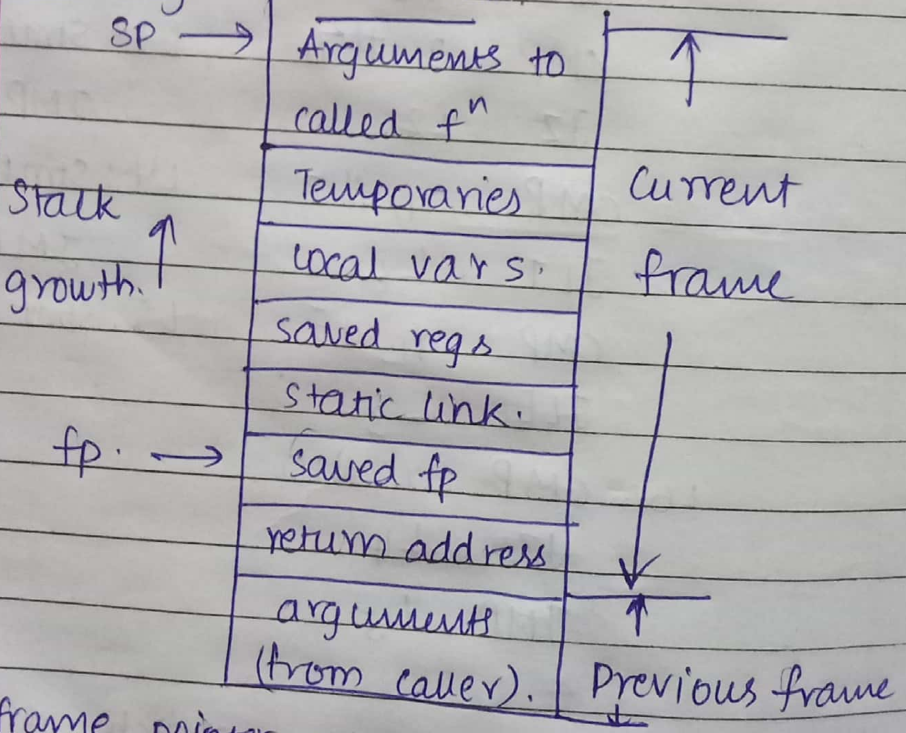
$r_1 \leftarrow 1$ lower limit

$r_2 \leftarrow 1$ step

$r_4 \leftarrow 10$ upper limit

while, for

9/3. **Subprogram Control:**



→ frame pointer comes onto screen at runtime.
fp - environment in which f^n is working.

```
f(int b){
    int a,c
      a=b+c;        fp[-dc]   local var.
}
         fp[db]
```

we know offset of 'b', the moment f^n is defined. (no. of fields req for activation record ~~can~~ are determined).

```
        R1 ← fp[d_b]
        R2 ← fp[-dc]
    Mov  R1, fp[d_b]
    Mov  R2, fp[-dc]
    ADD  R1, R2.
    Mov  fp[-da], R1
```

Things to be done during f^n call

1. Establishment of parameters (by caller)
2. Store return address
3. Store old fp.

caller - saved registers   transient values.

Callee . saved registers.

local var's.

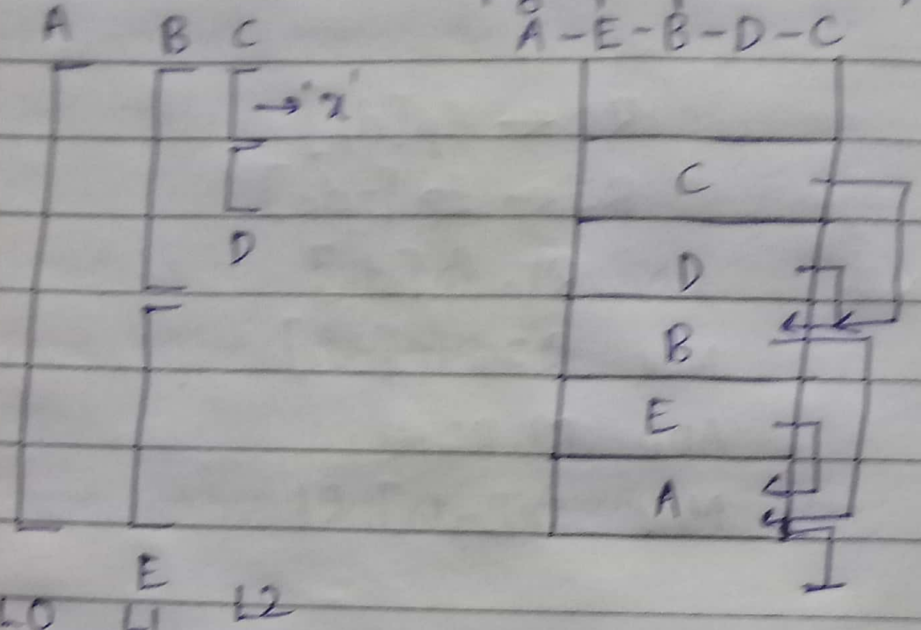
4. Store registers
5. Update value of fp
6. Update PC.

Inverse of these have to be done at return.

Scanned by CamScanner

13 Implementation of static scoping:

Using 'static link' which is passed by caller as an implicit addition parameter



Ea level is associated here

Callee's level = 1+ caller level, then it is directly in the scope of caller.

then Static link$_{callee}$ = fp$_{caller}$

If

$l_{callee} = l_{caller} + 1$

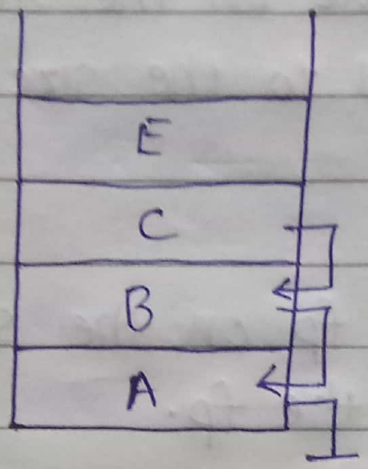
Static link$_{callee}$ = fp$_{caller}$

Else

Dereference$(l_{callee} - l_{caller} + 1)$ times the static link of the caller.

Use this value as static link off callee.

A - B - C - E
0   1   2   1



$(2-1+1) \rightarrow 2$ times deref. C.

$C \rightarrow E$ non-local ref

At compilation time itself, we have to check locality of 'x'.

Prologue of the Caller:

Seq. of steps by caller before it calling callee.

1) saves any caller-saves registers on the stack.
2) Computes values of arguments & moves them on stack.
3) Compute static link & pass it as an extra hidden parameters.
4) Use a spl. subroutine 'call' instruction to jump to the callee & simultaenously store the return values.

1. Allocates a frame on the stack by subtracting ~~subroutin~~ a value equal to the size of AR callee from the sp.

2. Saves the old fp on the stack & assign a new value to the fp.

3. Saves any callee saved registers on stack.

Epilogue of the Callee

1. Moves the return value to a register/specified location on the stack.
2. Restore callee-saves registers
3. Restores fp & sp.
4. Jump back to return address.

Epilogue of the caller:
1. Move to return value to correct place.
2. Restore caller-saves registers.

13

1. Leaf routine - that doesn't return anything.
   a) Not save return address if hardware stores in a register
   b) Static link computation & its passing is not required.

This doesn't call anyother function.

Scanned by CamScanner

c) can avoid saving caller saved registers.

activation record, push, pop @ runtime
piece of code on which stack works must be
emitted out @ compile time.

How can compiler identify whether it is a
local / global variable?
All data about vars is ~~[illegible]~~ stored in a symbol table.
Implementing it as a Hash Table is efficient.

11^{ly} we keep a track of functions in that table
and search in that.
If it is of type 'function', then a f^n call
will be there so it is not leaf.

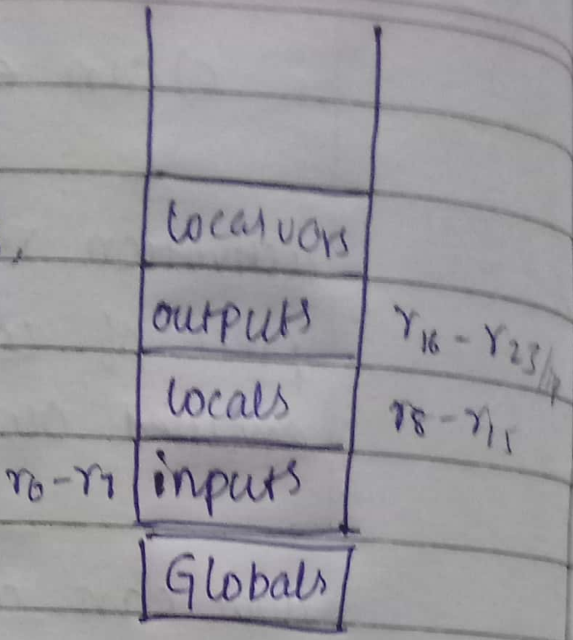
2) Inline function:
Similar to macro.

```
#define    DIVIDES (a, n)  (!(n % a))
           DIVIDES (y+Z, x)
                          y+Z % a
```

```
#define  SWAP(a,b) {int t=a; a=b; b=t;}
```

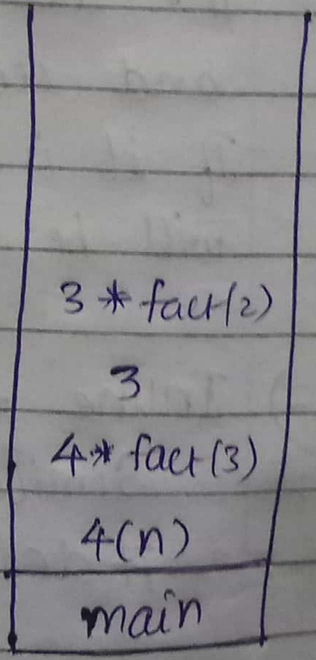
- Expand function in place.

3). Register windows
when we are exhausted
with all sets of registers,
it flushes out registers
and now use them.
Assume this as
circular Window



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

fact(4)



```
int fact (int n){
    factorial (n,1);
  }
```

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

Scanned by CamScanner

DATE :

```
int fib(int n){
    if(n==1) return 0
   else if (n==2) return 1
   else return (fib(n-1) + fib(n-2)):
}
```

```
int fib(int n){
  if (n==1) { f[1]=0 ; return 0; }
  else if (n==2){ f[2]=1, return 1;}
  else  if ( fib[n]!=-1)
                 return (fib[n]);
  else { int t = fib(n-1)+ fib(n-2);
                 fib[n]=t;
                return t;
       }
}.
```

```
int get_accum_num (FILE *s){
      char buf[100];
      char *p=buf;
      do{
            *p=getc(S);
       }while (*p++!='\n');
      *p='\0';
      return (atoi(buf));
 }
```

Scanned by CamScanner