# Static Allocation

Static Obj → have an absoalute storage address that is retained throughout the execution of the pgm (Absolute add → are called real add)

- They are often allocated in protected, read-only mly.
- So that any attempt to write to them will cause a processor interrupt, allowing the OS to announce a run-time error.
- Adv :- fast access due to absolute addressing of the obj.

Eg of static obj

Global variables are static obj

- Pgm code is statically allocated in most implementations of imperative lang.
- Var that are local to single subroutine, but retain their values from one invocation to the next.
- Numeric and string valued const can be allocated statically.

- Static local var. in C
- Run time tables produced by compilers (these tables used for debugging, garbage collection)

## Stack Allocation

Static allocation does not work for local var. in potentially recursive subroutines.

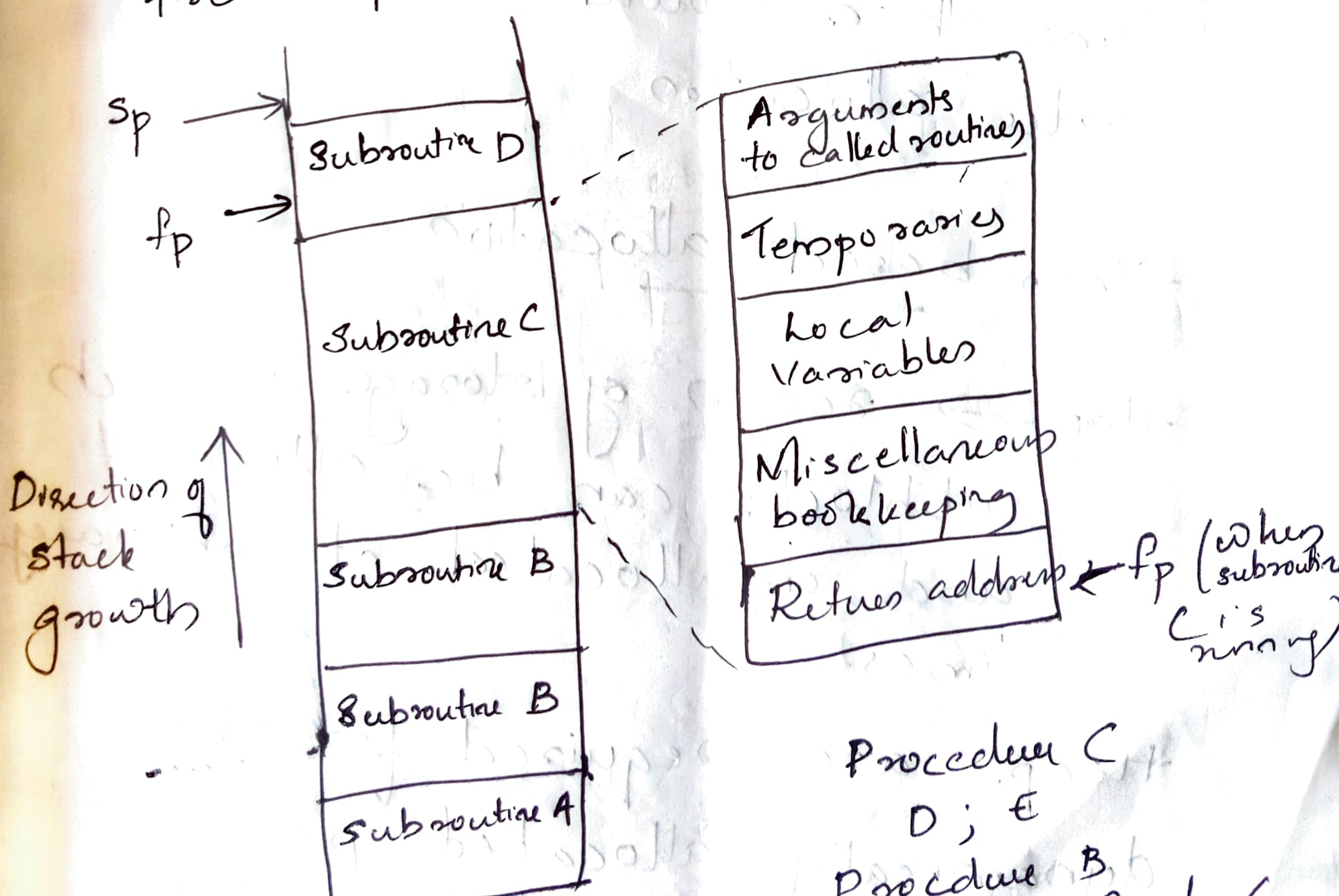
Every subroutine call have separate instances of local variables.

So we use stack allocation

Stack objects are allocated in LIFO order usually in conjunction with subroutine calls and returns.

- Each instances of a subroutine at run time has a frame on the run time stack called activation record containing arg & return values local var, temporaries + bookkeeping inf.
- Compiler generates subroutine calling sequence to setup frame, call the routine and to destroy the frame afterwards

- Frame layout vary b/w languages and implementations.
- A frame pointer (fp) points to the frame of the currently active subroutine at run time (always topmost frame on stack)
- Subroutine arg, local var, and return values are accessed by const. address offsets from fp
- The stack pointer (sp) points to free space on the stack.




Procedure C
D ; E
Procedure B
if .. then B else C
procedure A
B
main pgm

Assume → B has called itself once before calling C. If D returns and C calls E, E's frame will occupy the same space previously used for D's frame

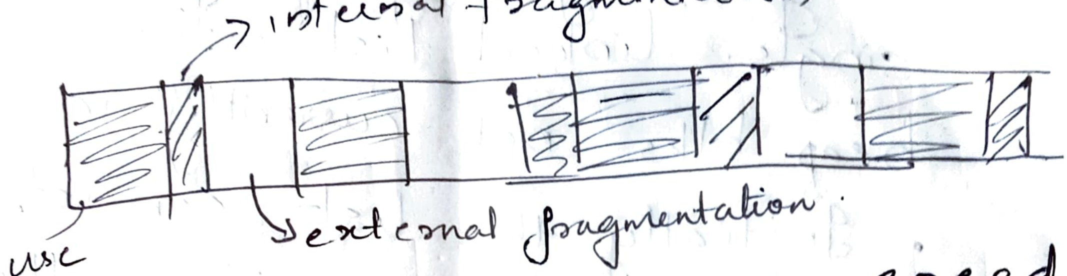
Sp → pts to 1st unused space
fp → pts to known loc. within the frame of the curr. subroutine

- A stack require less mly at run time than would be required for static allocation

Heap-based allocation

Heap → region of storage in which subblocks can be allocated and deallocated at arbitrary times.

Heaps are required for the dynamically allocated pieces of linked data structures, and for objects like fully general character strings, lists and sets, whose size may change as a result of an assignment statement or other update operation.




- The principal concern are speed and space
- space concern can be further subdivided into :- Internal and external fragmentation
- Internal fragmentation occurs when a storage-mgmt alg allocates a blk ie larger than required to hold a given object; the extra space is then unused
- External fragmentation occurs when the blks that have been assigned to active objects are scattered through the heap in such a way that the remaining, unused space is composed of multiple blks, there may however be quite a lot of free space

no space is large enough to satisfy future req.

- Many storage-mgmt alg maintain a single linked list - free list - of heap blks not currently in use.
  - Initially the list consists of a single blk comprising the entire heap.
  - At each allocation req. the alg searches the list for a blk of appropriate size.
  - With first fit alg select the first blk on the list ie large enough to satisfy the req.
  - With a best fit alg search the entire list to find the smallest blk ie large enough to satisfy the req.
  - In either case, if the chosen blk is larger than required, we divide it in two & return the unneeded portion to the free list as a smaller blk. If the

unneeded portion is below some min. threshold, leave it in the allocated blk as internal fragmentation.

. When a blk is de-allocated & returned to free list, we check to see whether either or both of the physically adjacent blk are free. if so combine.

- Best fit alg do better job of reserving large blks for large req. At the same time, it has higher allocation cost since it always search the entire list and tends to result in a larger number of very small "left-over" blocks.

- In any alg. that maintains a single free list, the cost of allocation is linear in the number of free blocks.
- To reduce this cost, some storage mgnt algs. maintain separate free lists for blocks of diff. sizes.
- In effect, heap is divided into pools, one for each standard size.

It may be static or dynamic..
- Two mechanisms for dynamic pool adjustment are known as the buddy system and the Fibonacci heap.

Buddy system :- std blk sizes are powers of two.
If a blk of size $2^k$ is needed, but none is available, a blk of size $2^{k+1}$ is split in two. One half is used to satisfy the req., other is placed on the $k^{th}$ free list.
When a blk is deallocated it is combined with its buddy, if it is free.

Fibonacci heap → use fibonacci numbers for std size. Alg is more complex but leads to lower internal fragmentation