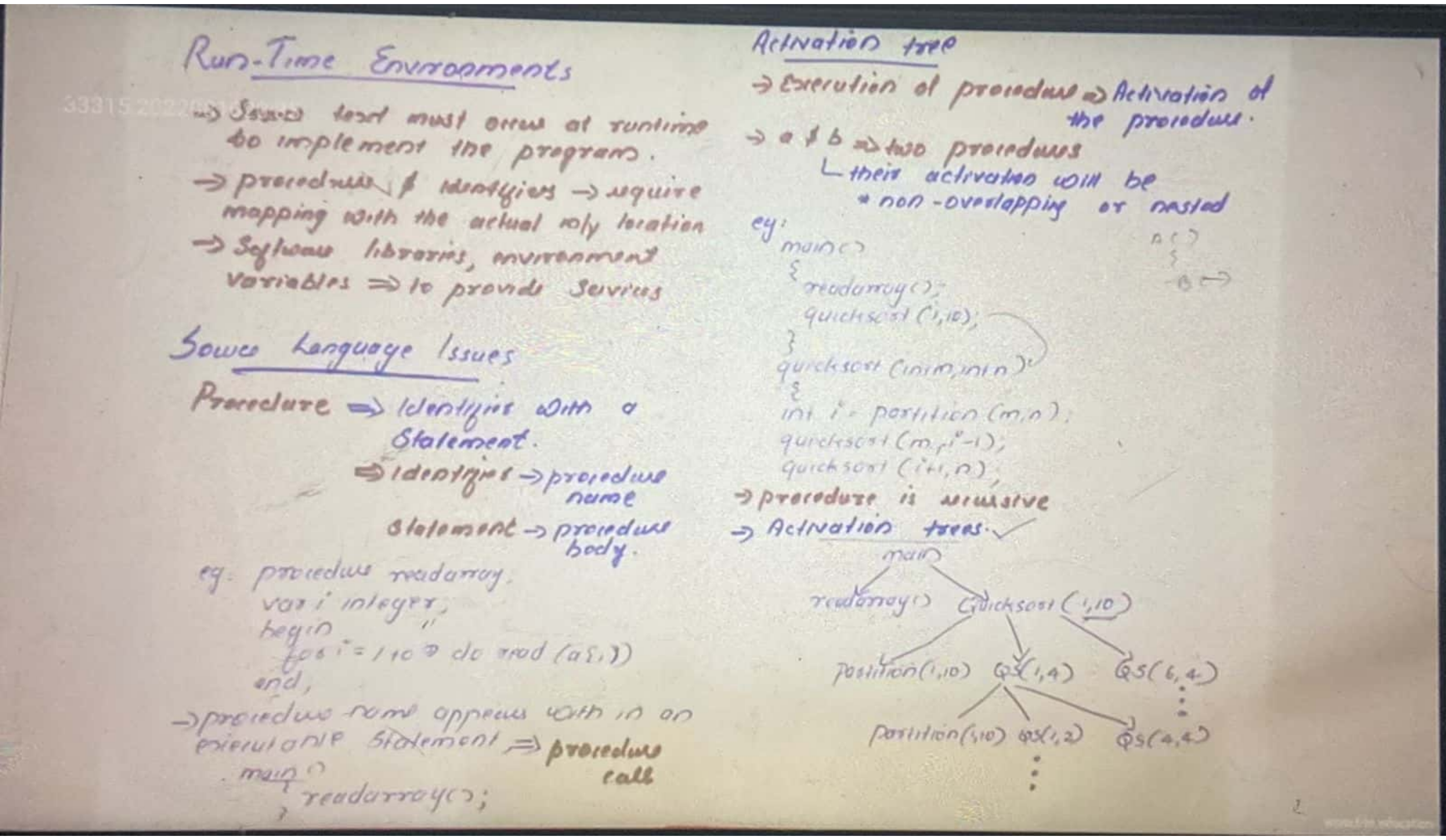


Run-Time Environments

- ⇒ Source text must occur at runtime to implement the program.
- ⇒ procedure & identifiers → require mapping with the actual mly location
- ⇒ Software libraries, environment variables ⇒ to provide Services

Source Language Issues

Procedure ⇒ Identifier with a Statement.

⇒ Identifier → procedure name

Statement → procedure body.

eg:

```
procedure readarray,
  var i integer;
  begin
    for i = 1 to 9 do read (a[i])
  end;
```

→ procedure name appears with in an executable Statement ⇒ procedure call

```
main()
{
  readarray();
}
```

Activation tree

⇒ Execution of procedure ⇒ Activation of the procedure.

⇒ a & b ⇒ two procedures

└ their activation will be

* non-overlapping or nested

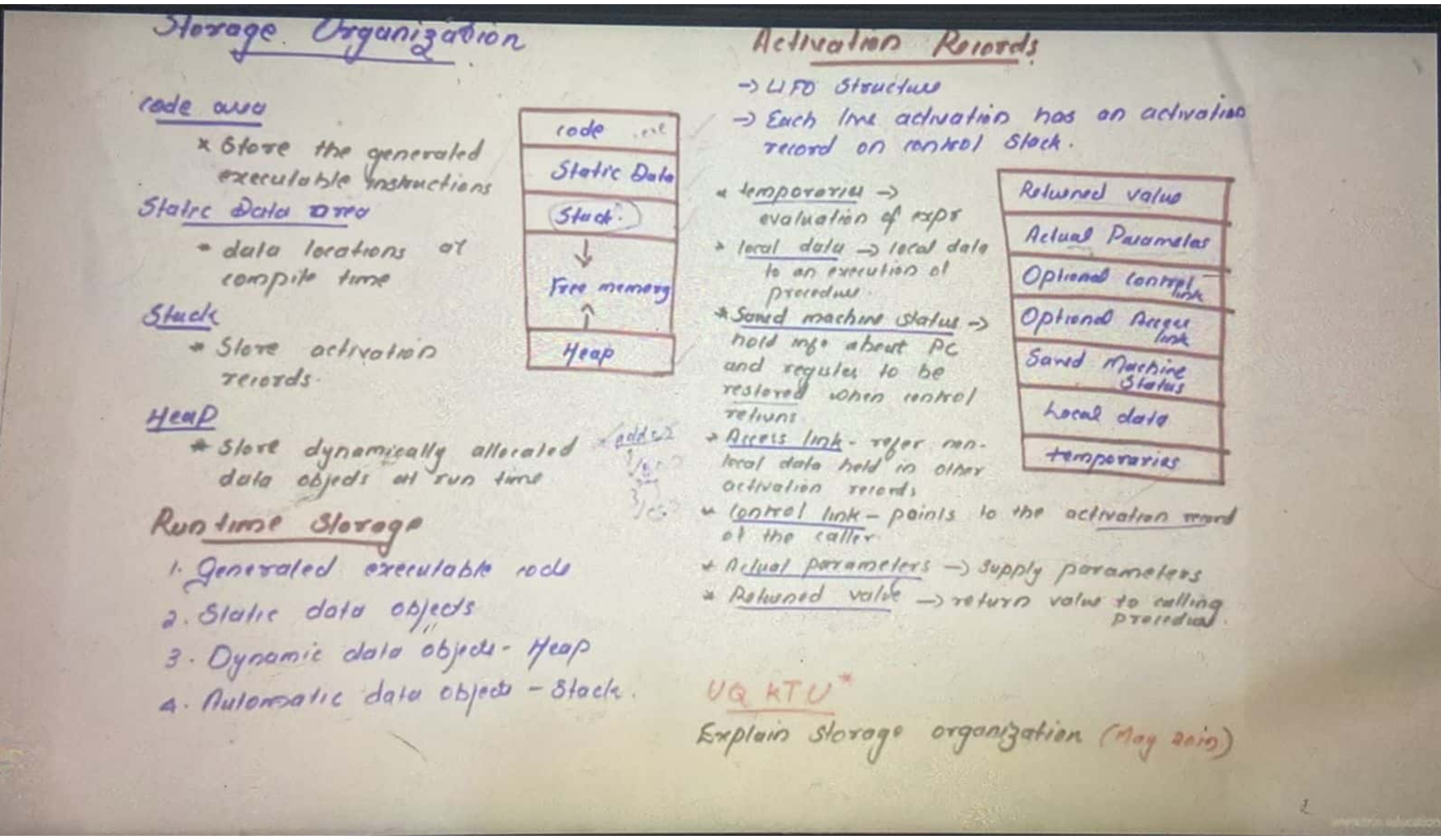
eg:

```
main()
{
  readarray();
  quicksort (1,10);
}
quicksort (int m, int n)
{
int i = partition (m,n);
quicksort (m, i-1);
quicksort (i+1, n);
```

```
A()
{
 B()
```

⇒ procedure is recursive

⇒ Activation trees. ✓



Storage Organization

code area

- Store the generated executable instructions

Static Data area

- data locations at compile time

Stack

- Store activation records.

Heap

- Store dynamically allocated data objects at run time

code .exe
Static Data
Stack.
Free memory
Heap

Run time Storage

1. Generated executable code
2. Static data objects
3. Dynamic data objects - Heap
4. Automatic data objects - Stack.

Activation Records

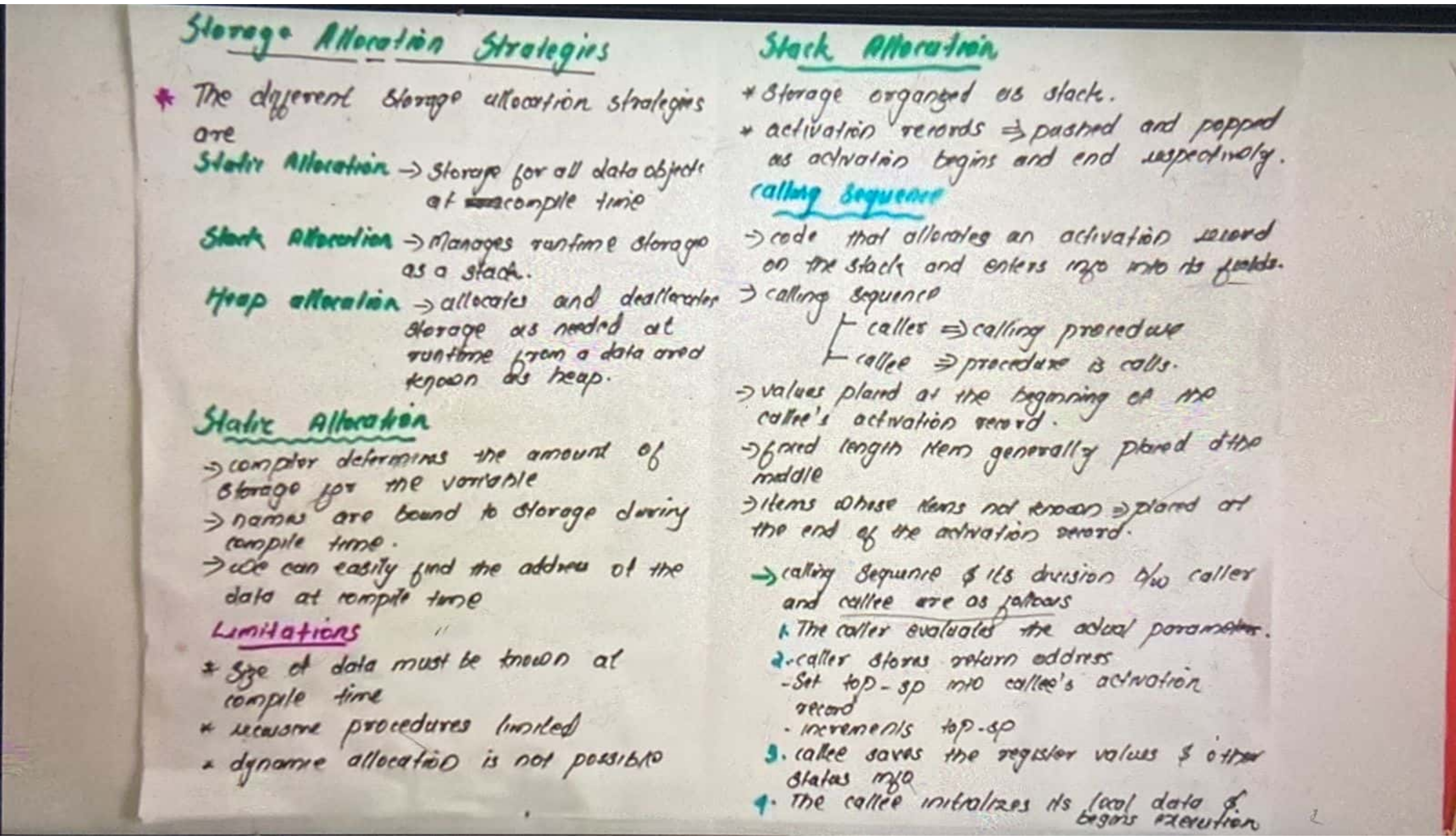
-> LIFO Structure

-> Each time activation has an activation record on control Stack.

- temporaries -> evaluation of expr
- local data -> local data to an execution of procedure.
- Saved machine status -> hold info about PC and register to be restored when control returns
- Access link - refer non-local data held in other activation records
- Control link - points to the activation record of the caller
- Actual parameters -> supply parameters
- Returned value -> return value to calling procedure.

UQ KTU*

Explain storage organization (May 2019)



Storage Allocation Strategies

- The different storage allocation strategies are

Static Allocation → Storage for all data objects at compile time

Stack Allocation → Manages runtime storage as a stack.

Heap allocation → allocates and deallocates storage as needed at runtime from a data area known as heap.

Static Allocation

→ compiler determines the amount of storage for the variable

→ names are bound to storage during compile time.

→ we can easily find the address of the data at compile time

Limitations

- Size of data must be known at compile time
- recursive procedures limited
- dynamic allocation is not possible

Stack Allocation

- Storage organized as stack.
- activation records ⇒ pushed and popped as activation begins and end respectively.

calling Sequence

→ code that allocates an activation record on the stack and enters info into its fields.

→ calling sequence
- caller ⇒ calling procedure
- callee ⇒ procedure is calls.

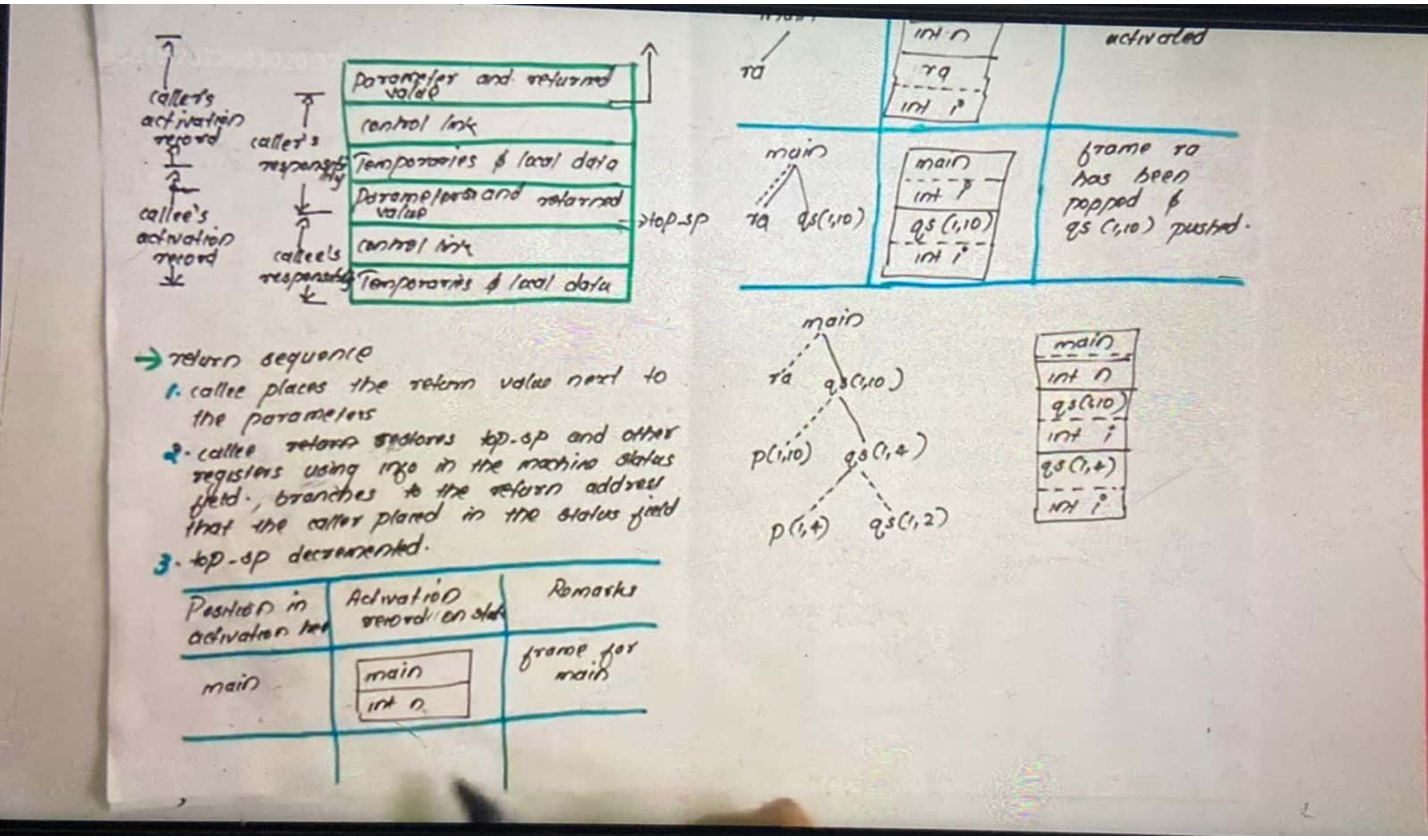
→ values placed at the beginning of the callee's activation record.

→ fixed length item generally placed at the middle

→ items whose items not known ⇒ placed at the end of the activation record.

→ calling Sequence & its division b/w caller and callee are as follows

1. The caller evaluates the actual parameter.
2. caller stores return address
   - Set top-sp into callee's activation record
   - increments top-sp
3. callee saves the register values & other status info
4. The callee initializes its local data & begins execution



caller's activation record
callee's activation record
caller's responsibility
callee's responsibility
top-sp

Parameter and returned value
control link
Temporaries & local data
Parameters and returned value
control link
Temporaries & local data

→ return sequence

1. callee places the return value next to the parameters
2. callee ~~return~~ restores top-sp and other registers using info in the machine status field, branches to the return address that the caller placed in the status field
3. top-sp decremented.

Position in activation tree	Activation record on stack	Remarks
main	main int n	frame for main

ra	int n ra int i	activated
main ra qs(1,10)	main int n qs (1,10) int i	frame ra has been popped & qs (1,10) pushed.

main
int n
qs(1,10)
int i
qs(1,4)
int i