Run Time Storage & Symbol Table Management

* Source language issues -

There are various language features that affect the organization of memory. The organization of data is determined by answering the following questions. The source language issues are.

a) does the source language allow recursion?
While handling the recursive calls there may be several instances of recursive procedures that are active simultaneously. Memory allocation must be needed to store each instance with its copy of local variables & parameters passed to that recursive procedures. But the number of active instances is determined by run time.

b) how the parameters are passed to the procedure?

c) does the procedure refer nonlocal names? How?

d) does the language support the memory allocation & deallocation dynamically?

* Storage Organization

- The compiler demands for a block of memory to operating system. The compiler utilizes this block of memory for running the compiled program. This block of memory is called run time storage.
- The run time storage is subdivided to hold code & data such as
  i) the generated target code
  ii) data objects
  iii) Information which keeps track of procedure activations.

i) The generated target code -
- The size of generated code is fixed.
- hence, the target code occupies the statically determined area of memory.
- Compiler places the target code at the lower end of the memory.

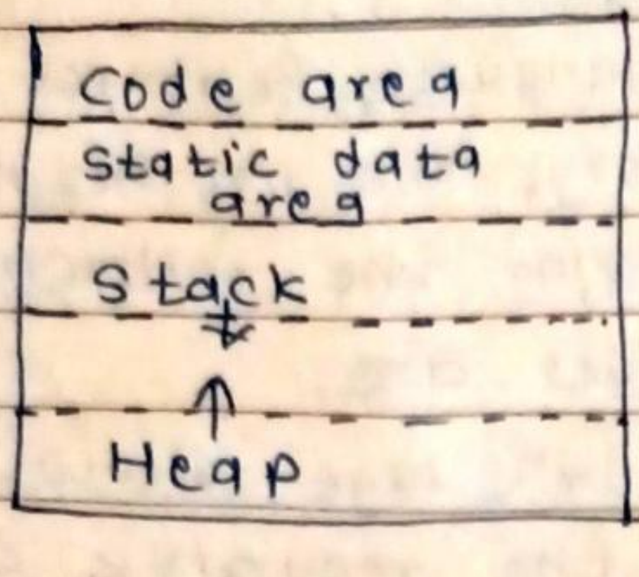


fig. Run time storage organization.

Data objects -

- The amount of memory required by the data objects is known at the compiled time & hence data objects also can be placed at the statically determined area of the memory.
- Compiler prefers to place the data objects in th statically determined area because these data obje then can be compiled into target code.

ex. In fortran all the data objects are allocated statically. Hence, the static data area is on the top of code area.

- The counterpart of control stack is used to manage the active procedures. Managing of acti procedures means that when a call occurs ther execution of activation is interrupted & informati about status of the stack is saved on the stack. When the control returns from the call this suspended activation is resumed after storing the values of relevant registers. Also, the progra counter is set to the point immediately afte the call. This information is stored in the stack area of runtime storage.

- The heap area is the area of run time storage in which the other information is stored.
ex. memory for some data items is allocated under the program control. Memory required for these data items is ~~allocated under the~~ program ~~ee~~ obtained from this heap area. Memory for some activation is also allocated from heap area.
- The size of stack & heap is not fixed it may grow or shrink interchangeably during the program execution.
- Pascal & C need the runtime stack.

* Storage Allocation Strategies -
There are three different storage allocation strategies based on this division of run time storage.
The strategies are
1) static allocation
2) stack allocation
3) heap allocation

1) static allocation -
- The static allocation is for all the data objects at compile time.
- The size of data objects is known at compile time. The names of these objects are bound to storage ~~at~~ compile time only & such an allocation of data objects is done by static allocation.
- The binding of name with the amount of storage allocated do not change at run time hence, the name of this allocation is static allocation
- In static allocation the compiler can determine the amount of storage required by each data object. And therefore it becomes easy for compiler to find the address of these data in the activation record.

- At compile time compiler can fill the address at which the target code can find the data it operats on.
ex. Fortran uses static allocation.

Limitations of static allocation -
- The static allocation can be done only if the size of data objects is known at compile time
- The data structure can't be created dynamically, In the sense that, the static allocation can't manage the allocation of memory at run time
- Recursive procedures are not supported by this type of allocation.

Stack Allocation -
- Stack allocation is a strategy in which the storage is organized as stack. This stack is also called control stack.
- As activation begins the activation records are pushed onto the stack & on completion of this activation. The corresponding activation records can be popped.
- The locals are stored in each activation record hence, locals are bound to corresponding activation record on each fresh activation.
- The data structures can be created dynamically for stack allocation.

Limitations of stack allocation -
The memory addressing can be done using pointer & index registers. Hence, this type of allocation is slower than static allocation.

* Heap Allocation -

- In heap allocatron, heap is used to manage dynamic memory allocation.
- If the values of non local variables must be retained even after the activation record then such a retaining is not possible by stack allocation. This limitation of stack allocation is because of its Last in first out nature. For retaining of such local variables heap allocation strategy is used.
- The heap allocation allocates the continuous block of memory. When required for storage of activation records or other data objects. This allocated memory can be deallocated when activation ends. This deallocated space can be further reused by heap manager.
- The efficient heap management can be done by a) creating a linked list for ~~free~~ free blocks & when any memory is deallocated the block of memory is appended in the linked list.
  b) allocate the most suitable block of memory from the linked list. i.e. use best fit technique for allocation of block.

* Activation Record -

- The activation record is block of memory used for managing information needed by a single execution of a procedure.
- ex. FORTRAN uses the static data area to store the activation record where as in PASCAL & C the activation record is situated in stack area.

The contents of activation record are as follows.

Return value
Actual parameters
Control link (Dynamic link)
Access link (Static link)
Saved machine status
Local variables
temporaries

Fig. model of activation record.

1) temporary values -
Temporary variables are needed during the evalu- of expression. Such variables are stored in the temporary field of activation record.

2) Local variables -
The local data is a data that is local to the execution of procedures is stored in this field of activation record.

3) Saved machine registers -
This field holds the information regarding the status of machine just before the procedure is called. This field contains the machine register & program counter.

4) Control link -
This field is optional. It points to the activation record of the calling procedure. This link is also called dynamic link.

5) Access link -
This field is also optional. It refers to the non local data in other activation record. This field is also called static link field.

6) Actual parameters -
This field holds the information about the actu- parameters. These actual parameters are passed to the called procedure.

7) Return value -
This field is used to store the result of a function call.

The size of each field of activation record is determined at the time when procedure is called.

1) By taking the example of factorial program explain how activation record will look like for every recursive call in case of factorial (3)

→ /* source code for factorial program */

```
main()
{
  int f;
  f = factorial (3);
}
  int factorial (int n)
{
  if (n == 1)
      return 1;
  else
      return (n * factorial (n-1));
}
```

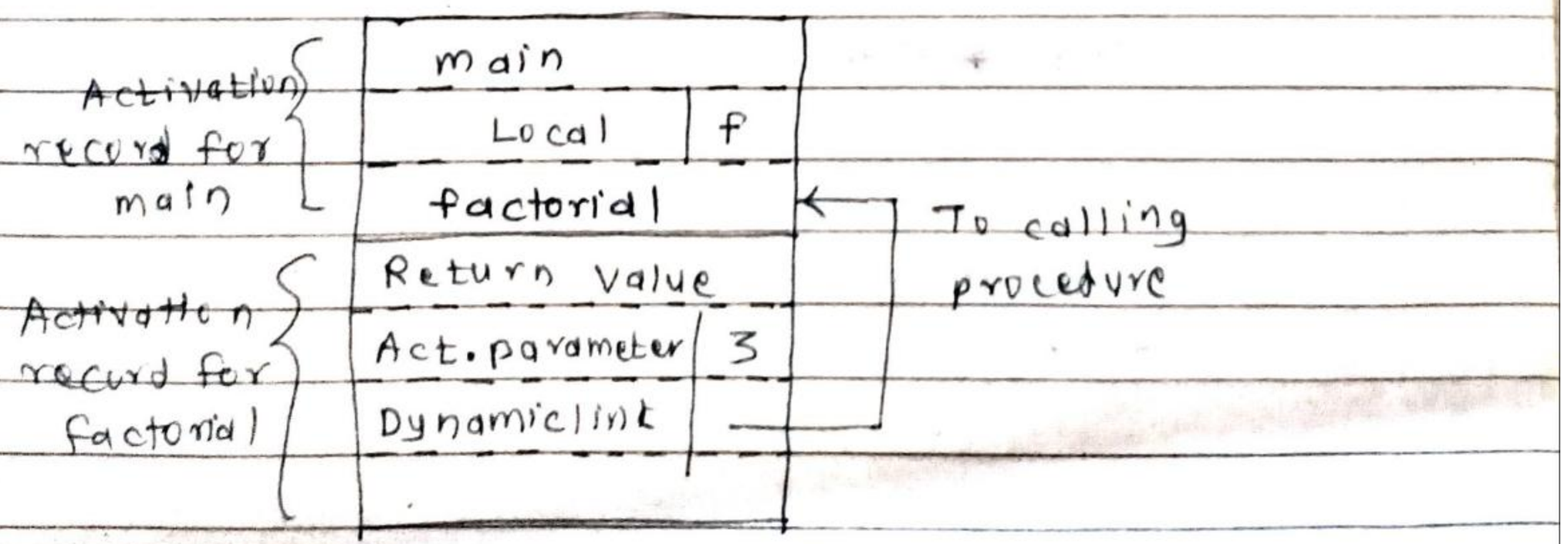
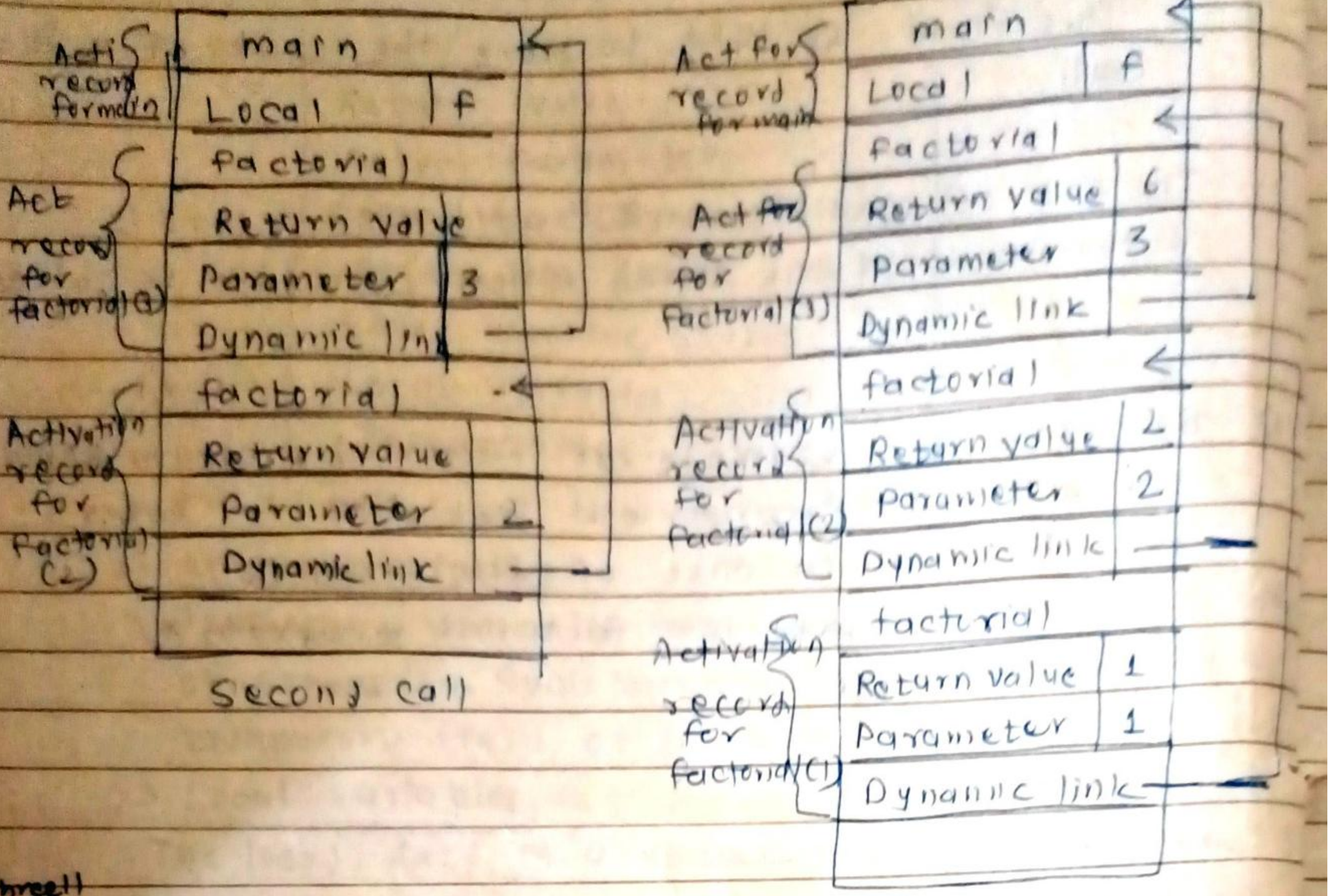


fig. FIRST call

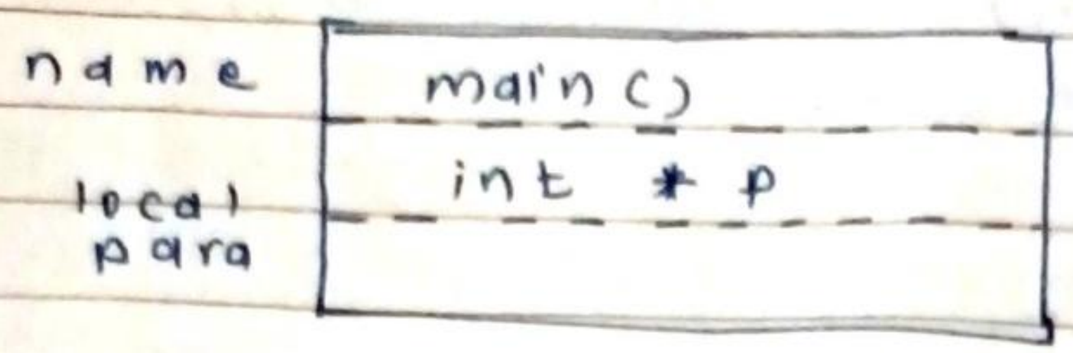


2) For the following C program, draw the details of the activation records if i) stack allocation is used
ii) Heap allocation is used

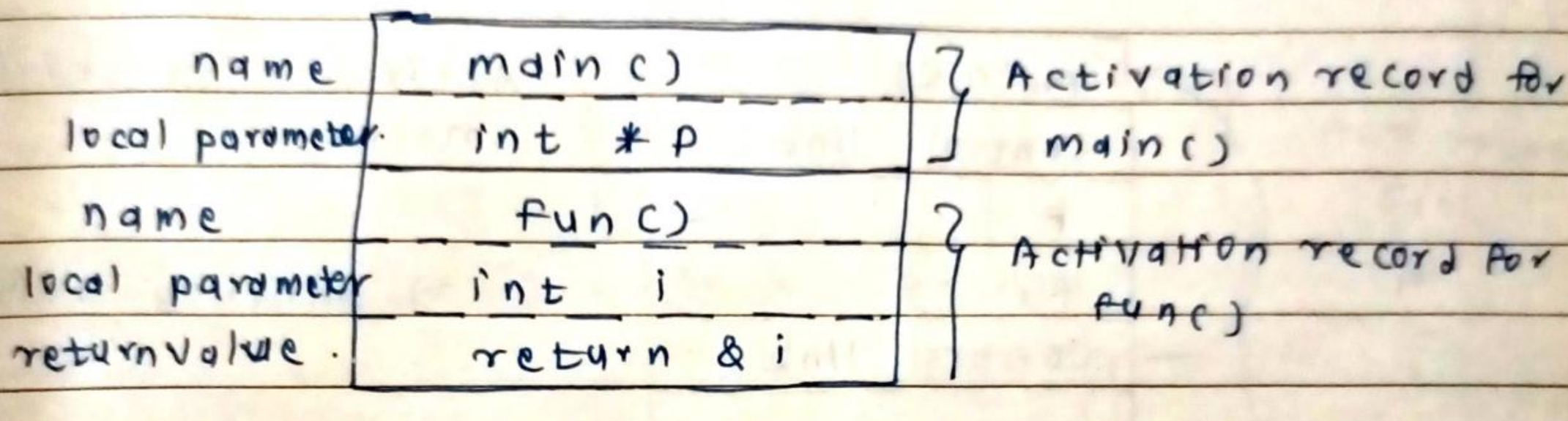
```
main()
{
int *p;
p = fun();
}
int *fun()
{
int i=23;
return &i;
}
```

→ Stack allocation

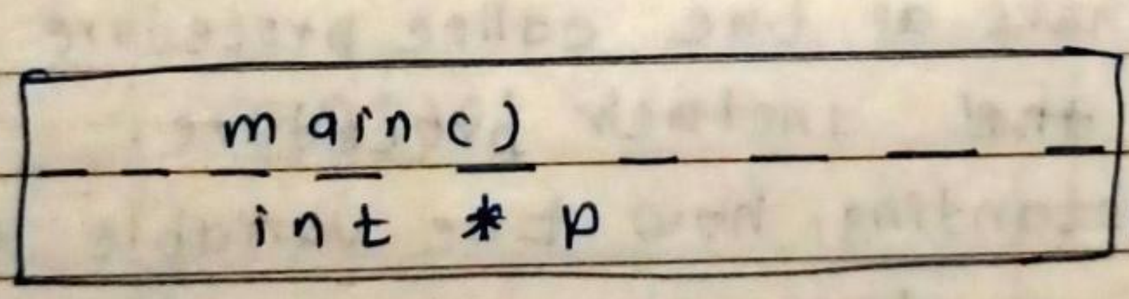
first of all the main() will be executed.



In the main function the function fun() is called when such call is made, record of fun() is pushed onto the stack.



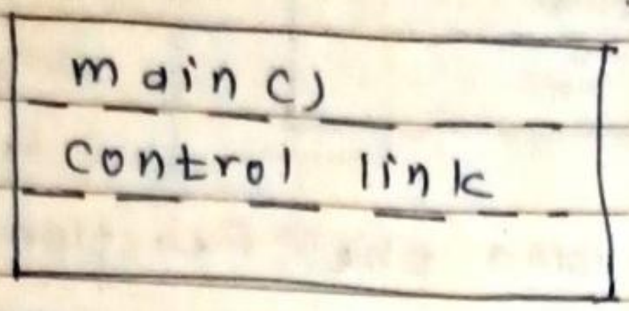
After execution of fun() the control returns to main. Hence, function fun() is popped off from the stack. Hence memory for local variable will be deallocated.



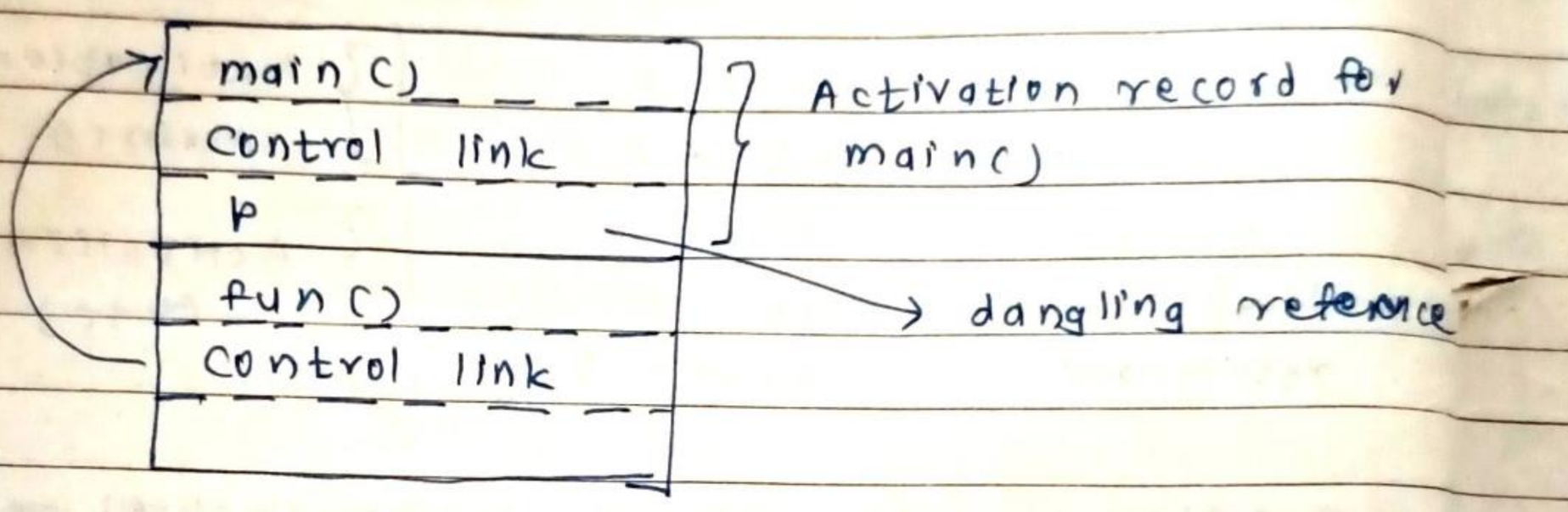
When control returns the main value of fun() will get stored in variable p. As local variable in fun() can be free & used for some other purpose later on the variable p in main refers to that storage in fun() p is called dangling reference.

2) Heap allocation -

In heap allocation the control flow of activation record can be represented. Initially, main() will be pushed onto the stack.



When a call to function fun() is made then it can be represented as.



* Variable Length data

- The caller procedure is a procedure which calls another procedure where as the callee procedure which gets called by ~~the~~ another procedure.
- For understanding, how the variable length data is handled by runtime storage consider following scenario. Suppose, a procedure A calls the procedure B. Procedure A has two arrays x & y. Storage to these arrays is not the part of activation record of A. In the activation record of A only the pointers to the begining of x & y are appearing. We can obtain the relative addresses of these arrays at compile time.

The activation record for procedure B begins after the array of A. Suppose, the procedure B ha

Variable length arrays p & q. Then after the activation record of B the arrays for procedure B can be placed.

Two pointers can be maintained top & top-sp to keep track of these records. The top points to the actual top of the stack & top_sp points to the end of some field in the activation record. By this the length of the activation record of callee procedures is known to caller at compile time.

Fig. storing variable length data

* **Block structure & Non Block structure Storage Allocation**

The storage allocation can be done for two types of data variables.

1) local data 2) Non local data.

1) **Local data** -

- The local data can be handled using activation record
- The offset relative to base pointer of an activation record points to local data variable within an activation record.

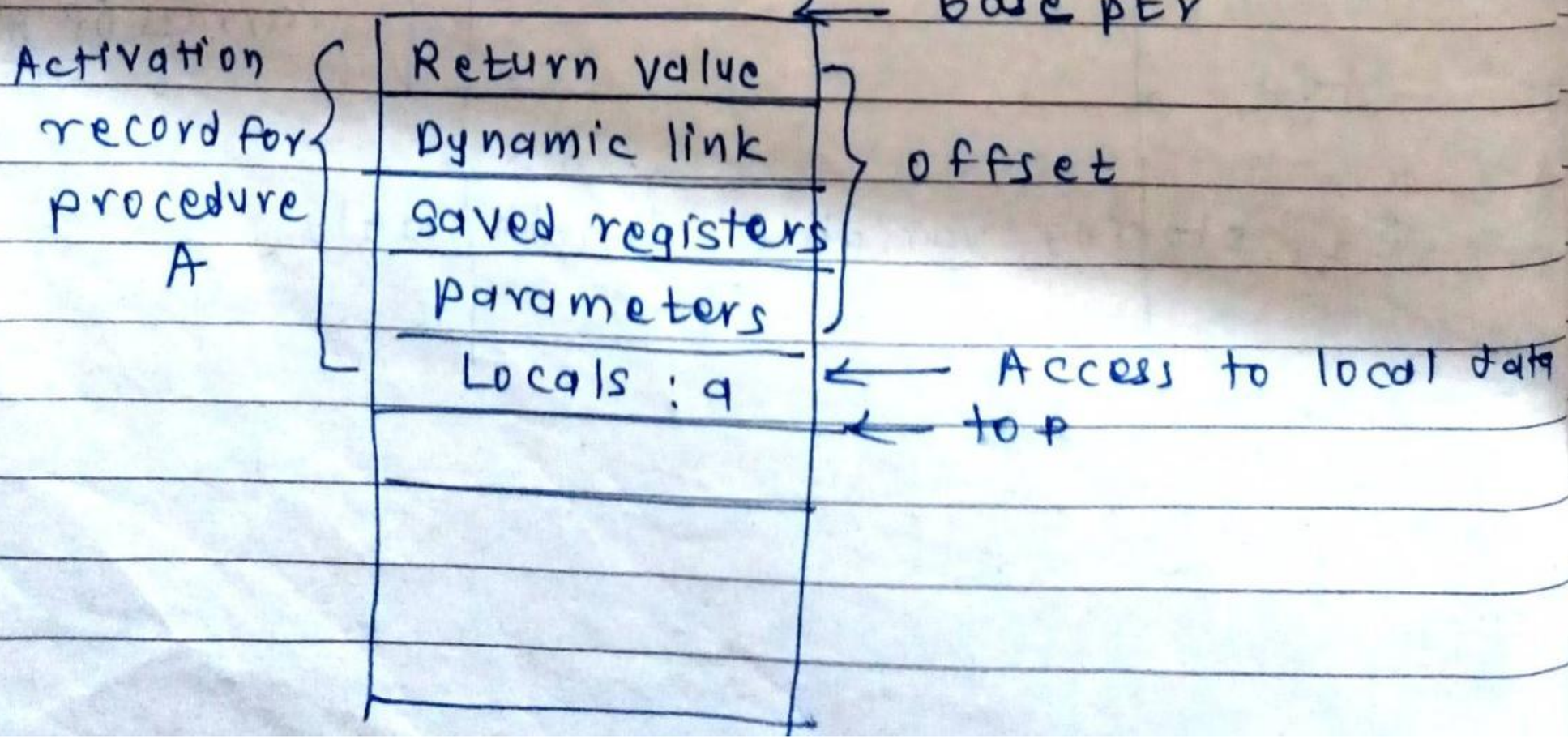
hence,

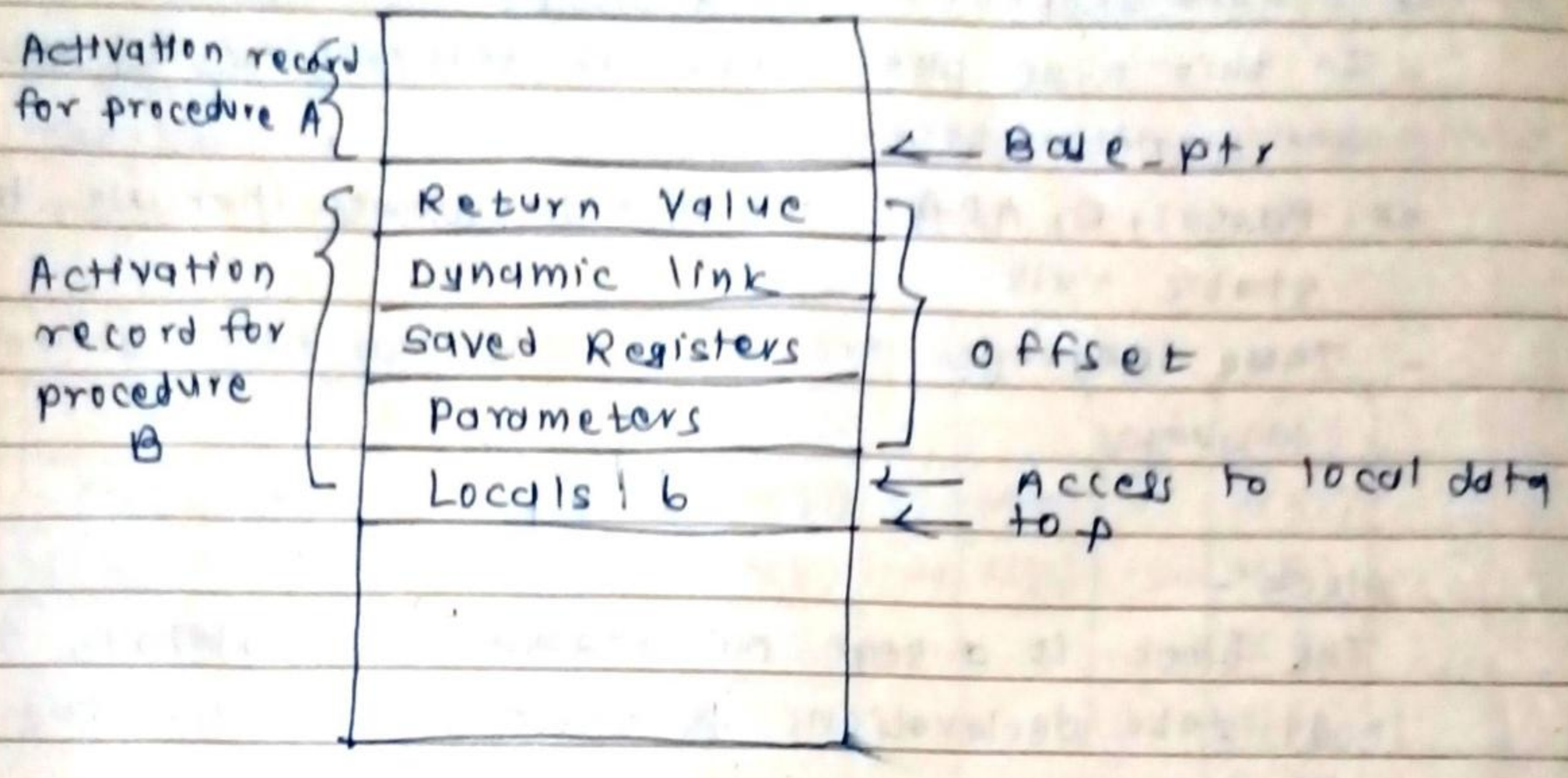
reference to any variable x in procedure = Base pointer pointing to start of procedure + offset of variable x from base pointer.

ex. Consider the following program

```
procedure A
    int a;
procedure B
    int b;
    body of B;
    body of A;
```

The contents of stack along with base pointer & offset are as follows





* Non local data

- This can be handled using scope information.
- For non local data items there are two types of types of scople rules.

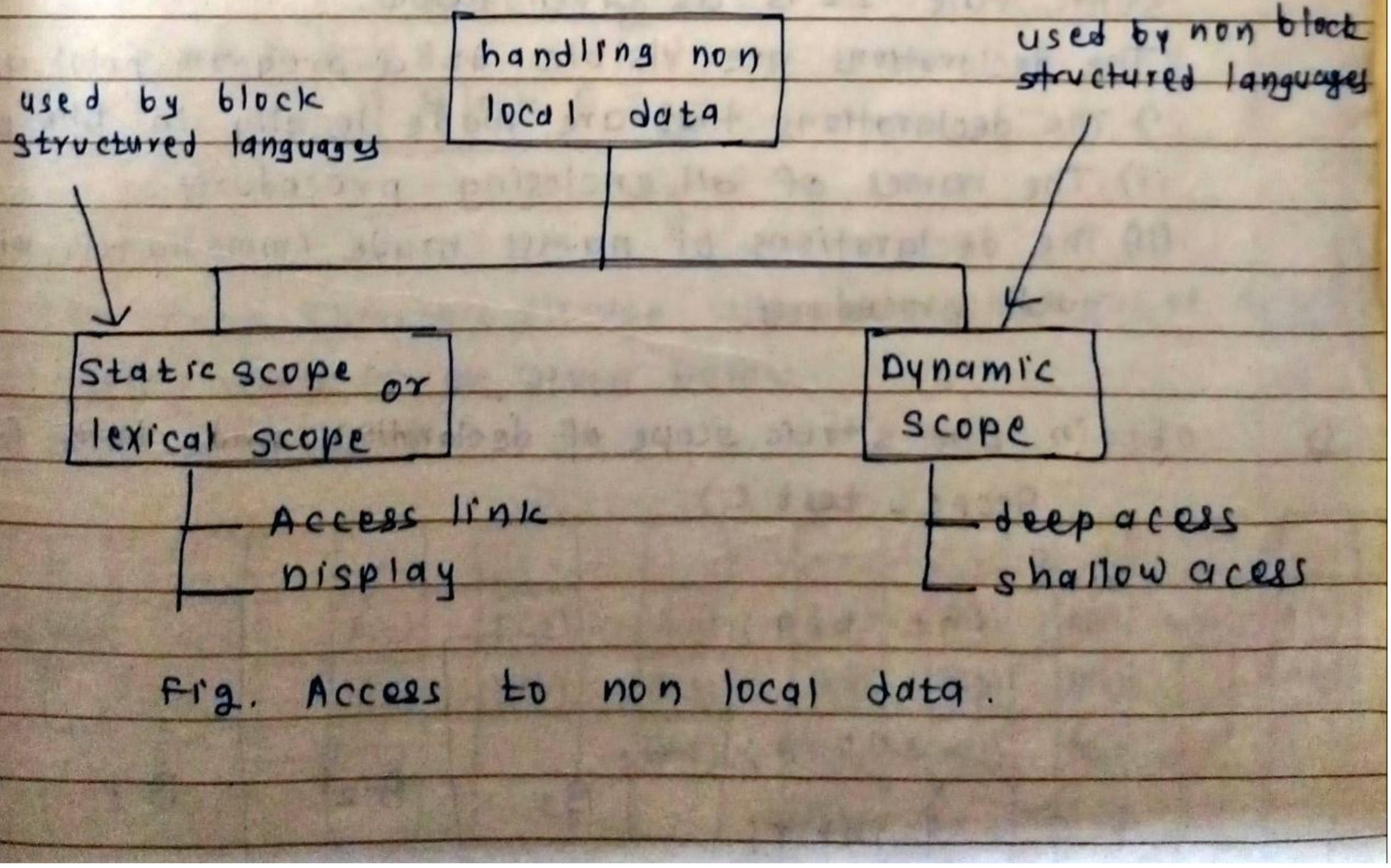


Fig. Access to non local data.

1) Static scope or lexical scope -

- In this type the scope is determined by examining the program text.

ex. Pascal, C, ADA are the languages that use the static rule.

- These languages are also called as block structured languages.

Block -

The block is a segn of statements containing the local data declarations & enclosed within the delimeters.

ex. {
declaration statements
}

- The scope of declaration in a block structured language is given by most closely nested loop or static rule. It is as given below.

The declarations are visible at a program point are.

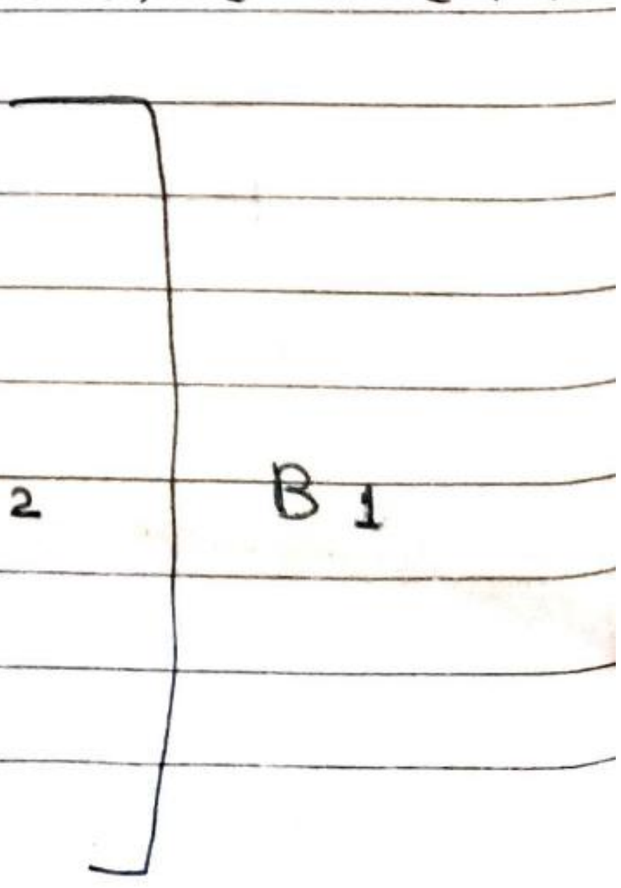
i) The declarations that are made locally in the proced

ii) The names of all enclosing procedures.

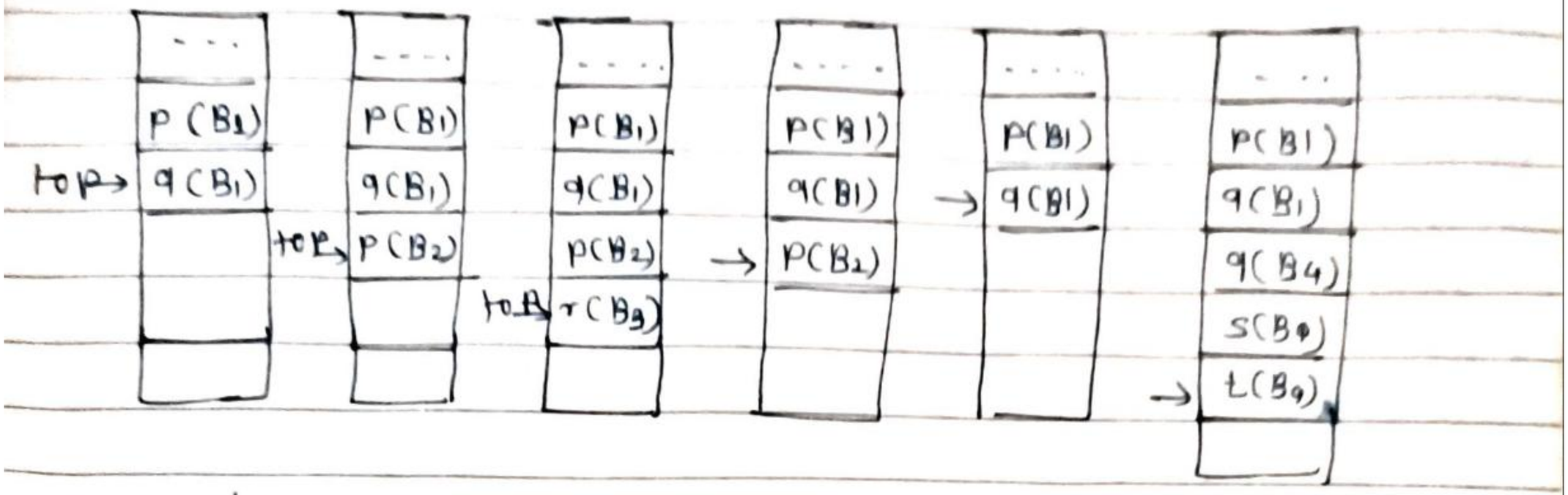
iii) The declarations of names made immediately withi such procedures.

1) Obtain the static scope of declarations made in the follow

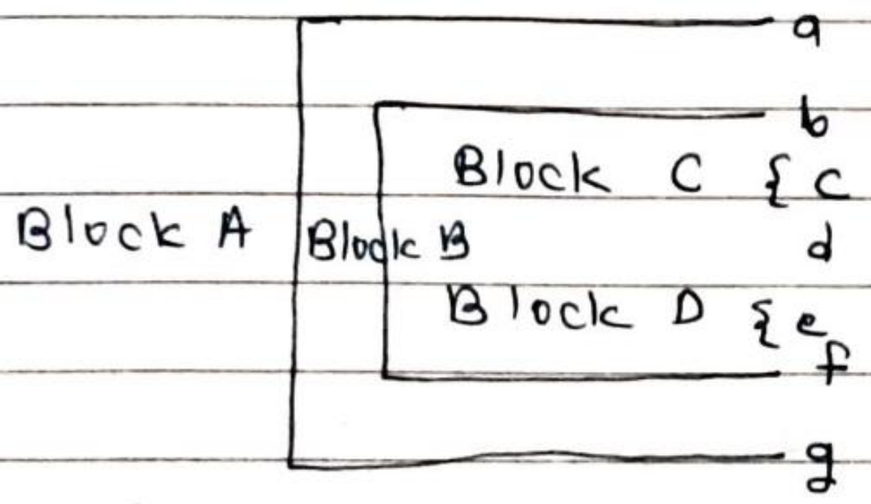
```
Scope_test ()
{
  int p,q;
  {
    int p;
    {
      int r;
    }
  }
  {
    int q,s,t;
  }
}
```



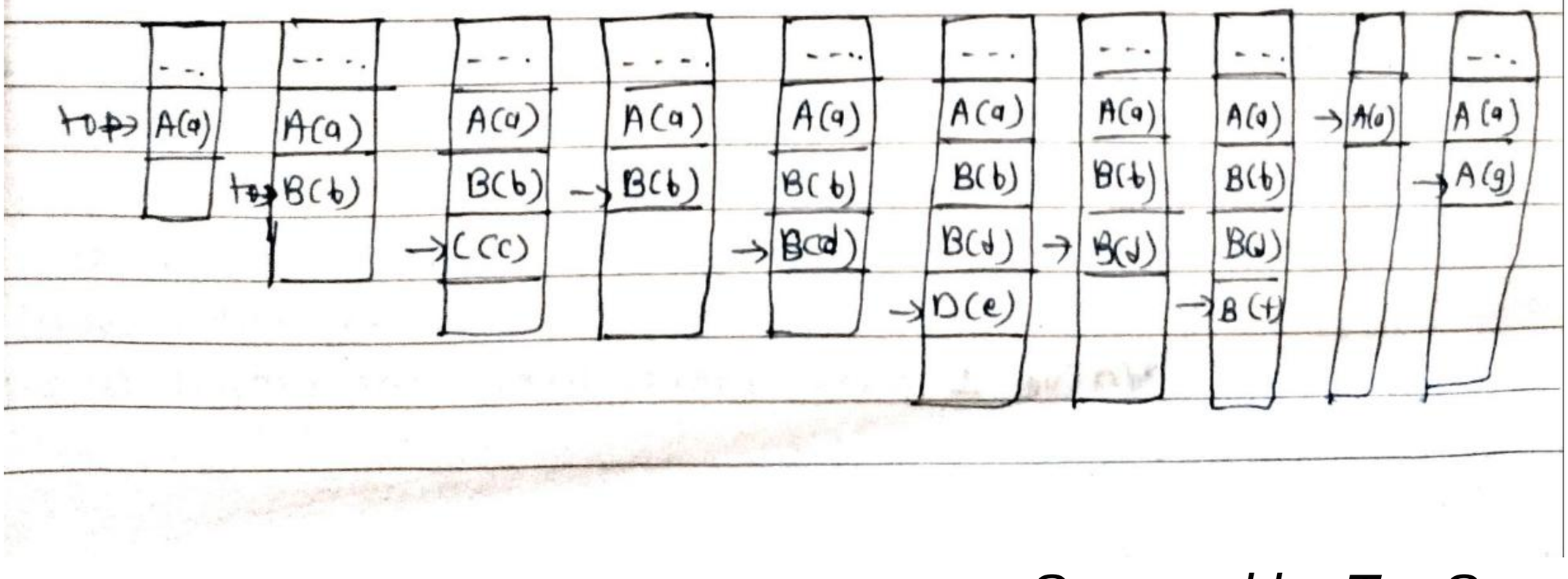
→ The storage can be allocated for a complete procedure body at one time.
The storage for the names corresponding to particular block can be as shown below.



e) Explain the behaviour of stack, used for allocation of storage, for the execution of each statement of the block structured program.



→ The block structure storage allocation which can be done using stack is as given below.



* **Dynamic Scope** –

In dynamic scoping a use of non local variable refers to the non local data declared in most recently called & still active procedure. Therefore each time new bindings are set up for local names & procedure.

- In dynamic scoping symbol table can be acquired at run time.

eg.

```
procedure main()
    value : int;
procedure first()
    begin
        value : int;
        value := 1;
        print (value);
    end
    begin
        value := 2
        print (value);
        first ();
    end;
```

In pascal like language static or lexical scope is used. Hence for a static scope the output will be

2 2

however, LISP like languages use dynamic scope & under dynamic scope output will be

2 1

This is because local variable value is given value as 1 & global variable value is given value as 2.

In dynamic scoping most recent activation record is always referred. The procedure 'first' is the most recent activation record. And the variable 'value' gets value 1 over there, hence the output is 2 1

There are two ways
i) Deep access ii) Shallow access.

1) Deep access -
The idea is keep a stack of active variables, use control links instead of access links & when you want to find a variable then search the stack from top to bottom looking for most recent activation record that contains the space for desired variables. This method of accessing non local variables is called deep access.
- In this method a symbol table needs to be used at run time.

2) Shallow access -
The idea is to keep a central storage with one slot for every variable name. If the names are not created at run time then that storage layout can be fixed at compile time otherwise when new activation of procedure occurs, then that procedure changes the storage entries for its locals at entry & exit.

Comparison of deep & shallow access
- Deep access takes longer time to access the non locals while shallow access allows fast access.
- Shallow access has a overload of handling procedure entry & exit.
- Deep access needs a symbol table at run time.

* Procedure Parameters -

There are two types of parameters.

i) formal parameters ii) Actual parameters.

Based on these parameters there are various parameter passing methods, the most common methods are.

1) call by value -

- This is the simplest method of parameter passing
- The actual parameters are evaluated & their r-value are passed to called procedure.
- The operations on formal parameters do not change the values of actual parameter.

ex. C, C++ use actual parameter passing.

2) call by reference -

- This method is also called as call by address or call by location
- The L-value, the address of actual parameter is passed to the called routines activation record.
- The values of actual parameters can be changed.
- The actual parameters should have an L-value.

3) Copy restore -

- This method is a hybrid beth call by value & call by reference.
- This method is also known as copy-in-copy out or values result.
- The calling procedure calculates the value of actual parameter & it is then copied to activation record for the called procedure.
- During execution of called procedure, the actual value is not affected.
- If the actual parameter has L-value then at return the value of formal parameter is copied actual parameter.

ex. ADA

4) Call by name -

- This is less popular method of parameter passing.
- procedure is treated like macro. The procedure body is substituted for call in caller with actual parameters substituted for formals.
- The actual parameters can be surrounded by parenthesis to preserve their integrity.
- The local names of called procedure & names of calling procedure are distinct
  eX. ALGOL

* Symbol Tables

- Symbol table is a data structure used by compil to keep track of semantics of variable.
- Symbol table stores the information about scope & binding information about name.
- Symbol table is built in lexical & syntax analysis phase.
- The symbol table used by various phases. Semantic analysis phase refers symbol table for type conflict issue.
  Code generation refers symbol table knowing how much runtime space is allocated.
- To achieve compile time efficiency compiler use symbol ta

* l-value & r-value –

The 'l' & 'r' prefixes comes from left & right s assignment.

ex.

a: = i+1

↑ ↑

l-value r-value.

* Symbol-Table Entries –

The items to be stored in symbol table are

i) variable names
ii) constants
iii) procedure names
iv) function names
v) literal constants & strings
vi) compiler generated temporaries
vii) labels in source languages.

Compiler uses following types of information fr symbol table

i) data type
ii) Name
iii) declaring procedure
iv) offset in storage
v) if structure or record then pointer to structure table
vi) for parameters, whether parameter passing is by value or reference
vii) number & type of arguments passed to the function
viii) base address

Attributes of symbol tables

i) variable names
ii) constants
iii) data types
iv) compiler generated temporaries
v) function names
vi) parameter names
vii) scope information

* Types of symbol table

The organization of symbol tables for non block structured languages is by ordered or unordered manner

Ordered Symbol Table -

- In ordered symbol table the entries of varriables is made in alphabetical manner.
- The searching of ordered symbol table can be done using linear & binary search.

Advantages -

- The searching of particular variable is efficient.
- The relationship of particular variable with another can be established very easily.

Disadvantage -

Insertion of element in orderd list is costly. If there are large number of entries in symbol table.

2) Unordered symbol table –

As variable is encountered, then its entry is mad in symbol table. These enteries are not in stored manner. Each time, before inserting any variable in the symbol table. A look up is made to check wheth it is already present in the symbol table or not

Advantages –

Insertion of variable in symbol table is very efficient.

Disadvantages –

- Searching must be done using linear search.
- for large table size, this method turns to be inefficient, because before insertion a lookup is made.

* How to store names in Symbol Table?

There are two types of name representation

1) fixed length name

A fixed space for each name is allocated in symbol table. In this type of storage name is too small then then there is wastage of spac

ex. The name can be referred by pointer to symb table entry.

ex.

Name									Attribute
c	a	l	c	u	l	a	t	e	
s	u	m							
a									
b									

2) Variable length name -

- The amount of space required by string is used to store the names.
- The name can be stored with the help of starting index & length of each name.

ex.

Name	
Starting index	length
0	10
10	14
24	2
16	2

Attribute

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c	a	l	c	u	l	a	t	e	$	s	u	m	$	a	$	b	$

* Symbol table Management -

Requirements for symbol table management

i) for quick insertion of identifier & related information
ii) for quick searching of identifier.

following are commonly used data structures for symbol table construction.

1) List data structure for symbol table -

- Linear list is a simplest kind of mechanism to implement the symbol table.
- In this method an array is used to store names & associated information.
- New names can be added in the order as they arrive.
- The pointer 'available' is maintained at the end of all stored records. The for list data structure using array is given below.

Name 1	info 1
Name 2	info 2
⋮	⋮
Name n	info n

← available (start of f

- To retrive the information about some name we sh from begining of array & go on searching upto available pointer. If we reach at pointer availa without finding a name we get an error "use of undeclared name".
- While inserting a new name we should ensure th it should not be already there. If it is there another error occures. i.e. "multiple defined name
- This takes minimum amount of space.

3)

2) Self organizing list -

This symbol table implementation is using linked l A link field is added to each record.

- We search the records in the order pointed by th link of link field.
- A pointer "First" is maintained to point first record of the symbol table.

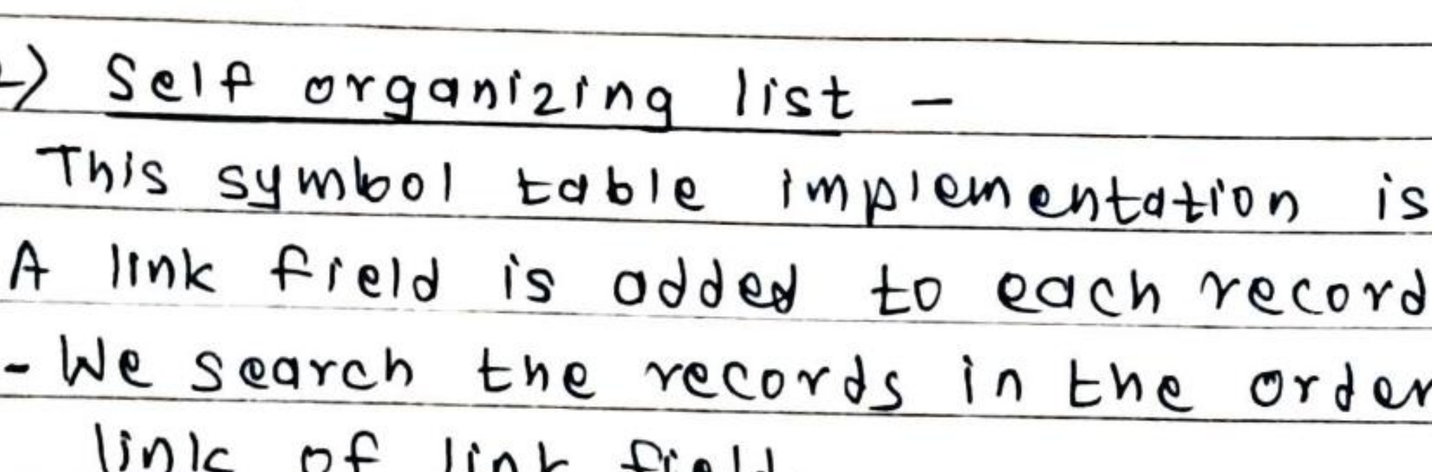


fig. symbol table.

The reference to these names can be Name 3, Name 1, Name 4, Name 2.

- When the name is referenced or created it is moved to the front of the list.
- The most frequently referred names will tend to be front of the list. Hence, access time to most frequ referred names will be the least.

Advantages of list -

- The list organization takes less space.
- Insertion of new symbol is easy.

3) Binary trees -

When the organization symbol table is by means of binary tree, the node structure will be as follows.

Left child	Symbols	information	right child

The left child field stores the address of previous symbol & right child field stores the address of next symbol. The symbol field is used to store the name of the symbol. And information field is used to gives information about the symbol.

Advantages -

- ~~Advan~~ Insertion of any symbol is efficient.
- Any symbol can be searched efficiently using binary search method.

Disadvantages -

This structure consumes lot of space in storing left pointer, right pointer & null pointers.

4) Hash Tables -

- Hashing is an important technique used to se the records of symbol table.
- this method is superior to list organization.
- In hashing scheme two tables are maintained a hash table & symbol table.
- The hash table consists of k entries from 0 k-1. These entries are basically pointers to symb table pointing to the names of symbol table.
- To determine whether the Name is in symbol tab we use a hash function 'h' such that h(na will result any integer betn 0 to k-1. We c search any name by.

position = h(name)

- Using this position we can obtain the exact locations of name in symbol table.
- The hash function should result in uniform distribution of names in symbol table.
- The hash function should be such that there will be minimum number of collision.

Advantage -

- Quick search is possible

Dissadvantage -

Complicated to implement.

Extra space is required.

1) Create tree structure using symbol table organization for following code.

```
int m,n,p;
int compute(int a, int b, int c)
{
    t = a+b*c;
    return(t);
}
main()
{
int k;
k = compute(10,20,30);
}
```

→

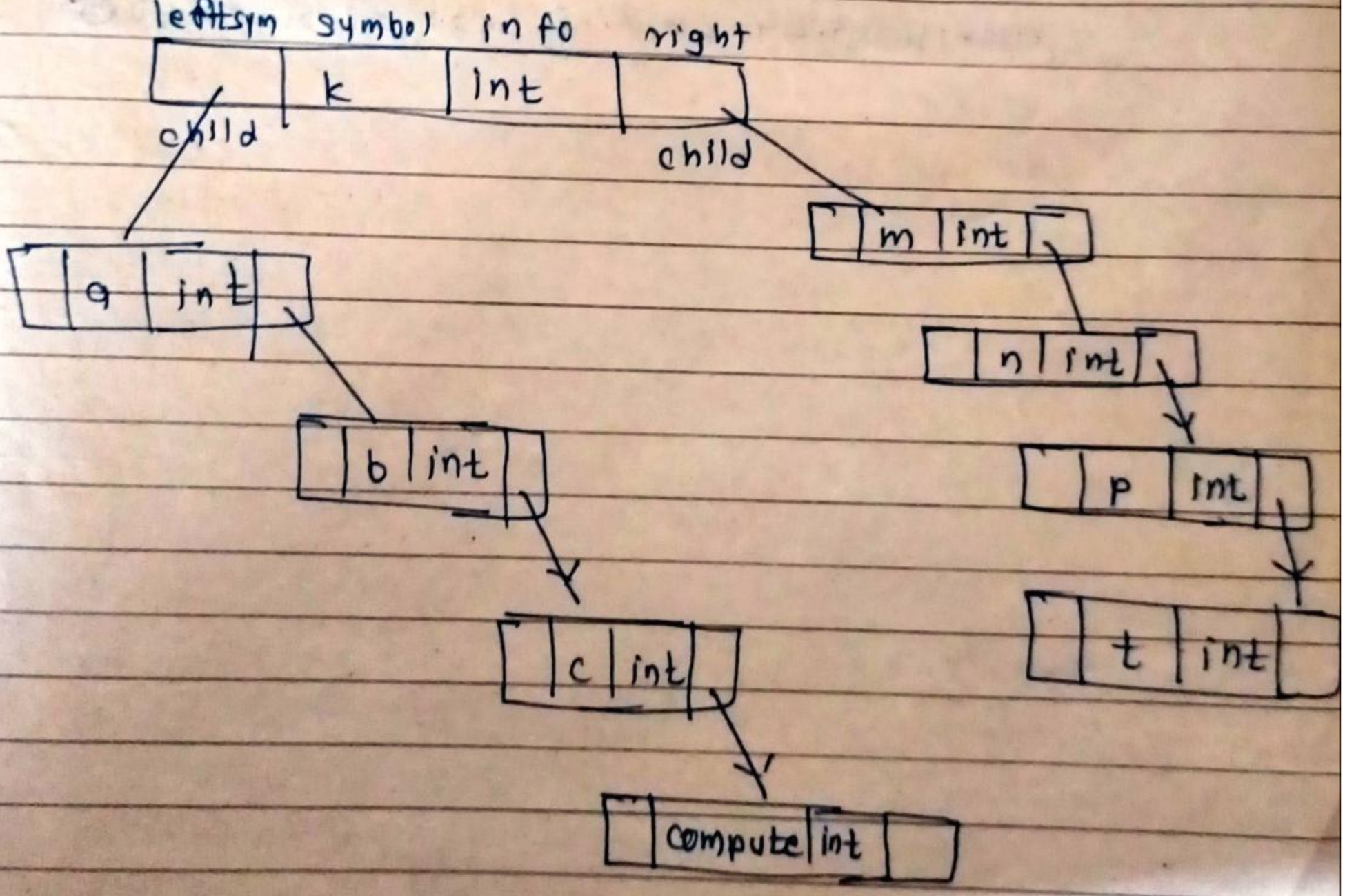


fig. tree structure organization