1) **Source Language Issues**

source language issues are:

1. **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 and parameters passed to that recursive procedure. But the number of active instances is determined by run time.

2. **How the parameters are passed to the procedure?**
   - There are two methods of parameter passing: Call by value and call by reference. The allocation strategies for each of these methods are different.

3. **Does the procedure refer nonlocal names? How?**
   - Any procedure has access to its local names. But a language should support the method of accessing non local names by procedures.

4. **Does the language support the memory allocation and deallocation dynamically?**
   - The dynamic allocation and deallocation of memory brings the effective utilization of memory.
   - There is a great effect of these source language issues on run time environment. Hence storage management is very important.

2) **Storage Organization**

   - The compiler demands for a block of memory to operating system. The compiler utilizes this block of memory executing the compiled program. This block of memory is called **run time storage**.
   - The run time storage is subdivided to hold code and data such as The generated target code and Data objects
   - The size of generated code is fixed. Hence the target code occupies the determined area of the memory. Compiler places the target code at end of the memory.
   - The amount of memory required by the data objects is known at the time and hence data objects also can be placed at the statically determined area of the memory.
Stack is used to manage the active procedure. Managing of active procedures means when a call occurs then execution of activation is interrupted and information about status of the stack is saved on the stack. When the control returns from the call this suspended activation resumed after storing the values of relevant registers.

Heap area is the area of run time storage in which the other information is stored. For example memory for some data items is allocated under the program control. Memory required for these data items is obtained from this heap area. Memory for some activation is also allocated from heap area.

3) **Storage allocation strategies**

1. **Static allocation**
   ✓ The size of data objects is known at compile time. The names of these objects are bound to storage at compile time only and 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 a compiler to find the addresses of these data in the activation record

2. **Stack allocation**
   ✓ Stack allocation strategy 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 completion of this activation the corresponding activation records can be popped.

3. **Heap allocation**
   ✓ The heap allocation allocates the continuous block of memory when required for storage of activation records or other data object. This allocated memory deallocated when activation ends. This deallocated (free) space can be reused by heap manager.
   ✓ The efficient heap management can be done by:
     I. Creating a linked list for the free blocks and when any memory deallocated that block of memory is appended in the linked list.
     II. Allocate the most suitable block of memory from the linked list.

4) **Activation Record (IMP)**

Various field of activation record are as follows:

1. Temporary values: The temporary variables are needed during the evaluation of expressions. 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 procedure 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 registers and 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 actual parameters. These actual parameters are passed to the called procedure.

7. Return values: This field is used to store the result of a function call.

### 5) Block Structure and Non Block Structure Storage Allocation

- The storage allocation can be done for two types of data variables.
  - 1. Local data
  - 2. Non local data.

- The local data can be handled using activation record whereas non local data can be handled using scope information.

- The block structured storage allocation can using static scope or lexical scope and the non block structured storage allocation done using dynamic scope.

1. **Local Data**

- The local data can be accessed with the help of activation record.
  - The offset relative to base pointer of an activation record points to local data variables within an a record, Hence
  - Reference to any variable \( x \) in procedure = Base pointer pointing to start of procedure + Offset of variable \( x \) from base pointer

2. **Access to non local names**

- A procedure may sometimes refer to variables which are not local to it. Such variable are called as non local variables. For the non local names there are two types of rules that can be defined: static and dynamic
The static scope rule is also called as lexical scope. In this type the scope is determined by examining the program text. PASCAL, C and ADA are the languages use the static scope rule. These languages are also called as block structured language.

Dynamic scope rule

For non block structured languages this dynamic scope allocation rules are used.

The dynamic scope rule determines the scope of declaration of the names at run time by considering the current activation.

LISP and SNOBOL are the languages which use dynamic scope rule.

6) Parameter passing methods

There are two types of parameters.

Formal parameters & Actual parameters

And based on these parameters there are various parameter passing methods, the common methods are,

1. Call by value:

This is the simplest method of parameter passing.

The actual parameters are evaluated and their r-values are passed to caller procedure.

The operations on formal parameters do not change the values of a parameter.

Example : Languages like C, C++ use actual parameter passing method

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.

---

```c
void main()
{
    int x, y;
    printf("Enter the value of X & Y:");
    int x,y;
}
```

```c
void swap(int *, int *);
void main()
{
}
```
```
scanf("%d%d", &x, &y);
swap(x, y);
printf("\n Values inside the main function");
printf("\n x=%d, y=%d", x, y);
getch();
}
void swap(int x, int y)
{
    int temp;
temp=x;
x=y;
y=temp;
printf("\n Values inside the swap function");
printf("\n x=%d y=%d", x, y);
}
```

3. **Copy restore:**
   - This method is a hybrid between call by value and 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 and it then copied to activation record for the called procedure.
   - During execution of called procedure, the actual parameters value is not affected.
   - If the actual parameter has L-value then at return the value of formal parameter is copied to actual parameter.

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 and names of calling procedure are distinct.

7) **Symbol tables (Most IMP)**
   - Definition: symbol table is a data structure used by compiler to keep track of semantics of variable. That means symbol table stores the information about scope and binding information about names.
   - Symbol table is built in lexical and syntax analysis phases.
   - L-value and r-value: The l and r prefixes come from left and right side assignment.
Symbol table entries

✓ The items to be stored in symbol table are:
   1) Variable names
   2) Constants
   3) Procedure names
   4) Function names
   5) Literal constants and strings
   6) Compiler generated temporaries
   7) Labels in source language

✓ Compiler use following types of information from symbol table
   1) Data type
   2) Name
   3) Declaring procedure
   4) Offset in storage
   5) If structure or record then pointer to structure table
   6) For parameters, whether parameter passing is by value or reference?
   7) Number and type of arguments passed to the function
   8) Base address

Attributes of Symbol Table

Variable names

✓ When a variable is identified, it is stored in symbol table by its name, the name must be unique.

Constants

✓ The constants are stored in symbol table. These constants can be accessed by the compiler with the help of pointers.

Data types

✓ The data types of associated variables are stored in symbol table.

Compiler generated temporaries

✓ The intermediate code is generated by compiler. During this process many temporaries may get generated which are stored in symbol table.

Function names

✓ The names of the functions can be stored in symbol table.

Parameter names

✓ The parameters that are passed to the function are stored in the symbol table. The information such as call by value or call by reference is also stored in table.

Scope information

✓ The scope of variable, where it can be used, such type of information can be stored in
symbol table.

**Ordered and unordered symbol table**

**Ordered Symbol Table**

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

**Advantages**

- Searching of particular variable is efficient.
- Relationship of particular variable with another can be established very easily.

**Disadvantages**

- Insertion of element in ordered list is costly if there are large number of entries in symbol table.

**Unordered Symbol Table**

- As variable is encountered, then its entry is made in symbol table. These entries are not in sorted manner. Each time, before inserting any variable in the symbol table, a lookup is made to check whether 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.

8) **How to store names in symbol table? (IMP)**

- 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 if name is too small then there is wastage of space

<table>
<thead>
<tr>
<th>Name</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>s</td>
<td>u</td>
</tr>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

- The name can be referred by pointer to symbol table entry.

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 and length of each name.
9) Data structures for symbol table

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 and 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 list data structure using arrays is as given below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name 1</td>
<td>Info 1</td>
</tr>
<tr>
<td>Name 2</td>
<td>Info 2</td>
</tr>
<tr>
<td>Name 3</td>
<td>Info 3</td>
</tr>
<tr>
<td>Name n</td>
<td>Info n</td>
</tr>
</tbody>
</table>

   ✓ To retrieve the information about some name we start from beginning of array and go on searching up to available pointer. If we reach at pointer available without finding a name we get an error "use of undeclared name".

   ✓ While inserting a new name we should ensure that it should not be already there. If it is there another error occurs i.e. "Multiple defined Name".

   ✓ The advantage of list organization is that it takes minimum amount of space.

2. Self organizing list
   ✓ This symbol table implementation is using linked list. A link field is added to each record.
   ✓ We search the records in the order pointed by the link of link field.
   ✓ A pointer "First" is maintained to point to first record of the symbol table

   | Name 1 | Info 1 |
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 time to most frequently referred names will be the least.

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

<table>
<thead>
<tr>
<th>Left child</th>
<th>Symbols</th>
<th>Information</th>
<th>Right child</th>
</tr>
</thead>
</table>

   - 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.
   - Information field is used to give information about the symbol.
   - Binary tree structure is basically a binary search tree in which the value of left is always less than the value of parent node. Similarly the value of right node is always more or greater than the parent node.

   **Advantages**
   - 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 and null pointers.

4. **Hash table**
   - Hashing is an important technique used to search the records of symbol table. This method is superior to list organization.
   - In hashing scheme two tables are maintained-a hash table and symbol table.
   - The hash table consists of k entries from 0,1 to k-1. These entries are basically pointers to symbol table pointing to the names of symbol table.
   - To determine whether the 'Name' is in symbol table, we use a hash function 'h' such that h(name) will result any integer between 0 to k-1. We can search any name by position = h(name)
   - Using this position we can obtain the exact locations of name in symbol table.
   - Hash function should result in uniform distribution of names in symbol
   - Hash function should be such that there will be minimum number of collision. Collision is such a situation where hash function results in same location for storing the names.
   - Collision resolution techniques are open addressing, chaining, rehashing.
Advantage of hashing is quick search is possible and the disadvantage is that hashing is complicated to implement. Some extra space is required. Obtaining scope of variables is very difficult.

10) **Language facilities for Dynamic storage allocation**

- There are two approaches used to allocate data - the explicit allocation and implicit allocation.
- **Explicit allocation** is the kind of memory allocation that can be done with the help of some procedures.
- Example - In PASCAL the memory allocation is done using new function and deallocation is done using dispose.
- **Implicit allocation** is a kind of allocation that can be done automatically for storing the results of expression evaluation. Various languages that handle implicit allocation are
  - LISP allocates a memory for the cell of a list when cons is used.
  - SNOBOL allows a string of varying length. The space required for varied length of the string is managed at run time. This space is taken from heap memory.
- **Garbage** is a substantial amount of memory which gets allocated dynamically but is unreachable memory. If this memory remains unreachable then it becomes a wastage of memory. Hence a new technique called garbage collection is introduced. Java and LISP are those languages which perform garbage collection.
- **Dangling reference** is a kind of complication that occurs due to explicit deallocation of memory. The dangling reference is a kind of reference that gets deallocated which is referred to.

11) **Dynamic Storage Allocation Techniques**

There are two techniques used in dynamic memory allocation and those are -

- Explicit allocation
- Implicit allocation

1. **Explicit Allocation**

- The explicit allocation can be done for fixed size and variable sized blocks.
  - **Explicit Allocation for Fixed Size Blocks**
    - This is the simplest technique of explicit allocation in which the size of the block for which memory is allocated is fixed.
    - In this technique a free list is used. Free list is a set of free blocks. This observed when we want to allocate memory. If some memory is de-allocated then the free list gets appended.
    - The blocks are linked to each other in a list structure. The memory allocation be done by pointing previous node to the newly allocated block. Memory deallocation can be done by de-referencing the previous link.
    - The pointer which points to first block of memory is called Available.
This memory allocation and deallocation is done using heap memory.

Available

![](image1)

The explicit allocation consists of taking a block off the list and deallocation consist of putting the block back on the list.

The advantage of this technique is that there is no space overhead.

**Explicit Allocation of Variable Sized Blocks**

Due to frequent memory allocation and deallocation the heap memory becomes fragmented. That means heap may consist of some blocks that are free and some that are allocated.

![Fragmented Heap Memory](image2)

In Fig. a fragmented heap memory is shown. Suppose a list of 7 blocks gets allocated and second, fourth and sixth block is deallocated then fragmentation occurs. Thus we get variable sized blocks that are available free.

For allocating variable sized blocks some strategies such as first fit, worst fit and best fit are used.

Sometimes all the free blocks are collected together to form a large free block. This ultimately avoids the problem of fragmentation.

**2. Implicit Allocation**

The implicit allocation is performed using user program and runtime packages.

The run time package is required to know when the storage block is not in use

The format of storage block is as shown in following Fig

<table>
<thead>
<tr>
<th>Block size</th>
<th>Reference count</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pointers to block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There are two approaches used for implicit allocation.

**Reference count:**

Reference count is a special counter used during implicit memory allocation. If any block is referred by some another block then its reference count incremented by one. That also means if the reference count of particular block drops down to 0 then, that means that block is not referenced one and hence it can be deallocated. Reference counts are best
used when pointers between blocks never appear in cycle.

Marking techniques:

✓ This is an alternative approach to determine whether the block is in use or not. In this method, the user program is suspended temporarily and **frozen pointers** are used to mark the blocks that are in use. Sometime bitmaps are used. These pointers are then placed in the heap memory. Again we go through hear memory and mark those blocks which are unused.

✓ There is one more technique called **compaction** in which all the used blocks are moved at the one end of heap memory, so that all the free blocks are available in one large free block.