**Runtime Environments in Compiler Design**

A translation needs to relate the static source text of a program to the dynamic actions that must occur at runtime to implement the program. The program consists of names for procedures, identifiers, etc., that require mapping with the actual memory location at runtime. Runtime environment is a state of the target machine, which may include software libraries, environment variables, etc., to provide services to the processes running in the system.

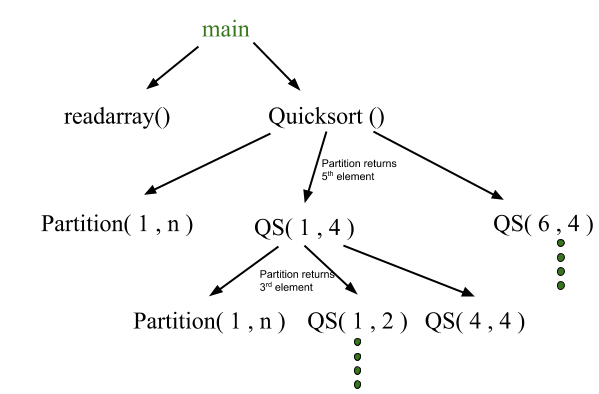
### ****SOURCE LANGUAGE ISSUES:****

**Activation Tree**

A program consist of procedures, Each execution of the procedure is referred to as an activation of the procedure. Lifetime of an activation is the sequence of steps present in the execution of the procedure An activation tree shows the way control enters and leaves activations. Properties of activation trees are :-

* Each node represents an activation of a procedure.
* The root shows the activation of the main function.
* The node for procedure ‘x’ is the parent of node for procedure ‘y’ if and only if the control flows from procedure x to procedure y.

**Example –** Consider the following program of Quicksort



**CONTROL STACK AND ACTIVATION RECORDS**

Control stack or runtime stack is used to keep track of the live procedure activations i.e the procedures whose execution have not been completed. A procedure name is pushed on to the stack when it is called (activation begins) and it is popped when it returns (activation ends). Information needed by a single execution of a procedure is managed using an activation record or frame.

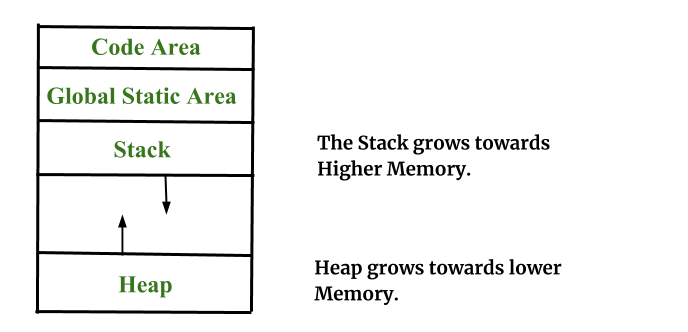
A general activation record consists of the following things:

* **Local variables:** hold the data that is local to the execution of the procedure.
* **Temporary values**: stores the values that arise in the evaluation of an expression.
* **Machine status:** holds the information about the status of the machine just before the function call.
* **Access link (optional):** refers to non-local data held in other activation records.
* **Control link (optional):** points to activation record of caller.
* **Return value:** used by the called procedure to return a value to calling procedure
* Actual parameters

**SUBDIVISION OF RUNTIME MEMORY**

Runtime storage can be subdivided to hold :

* Target code- the program code, is static as its size can be determined at compile time
* Static data objects
* Dynamic data objects- heap
* Automatic data objects- stack



**STORAGE ALLOCATION TECHNIQUES**

1.**Static Storage Allocation**

* For any program, if we create a memory at compile time, memory will be created in the static area.
* For any program, if we create a memory at compile-time only, memory is created only once.
* It doesn’t support dynamic data structure i.e memory is created at compile-time and deallocated after program completion.
* The drawback with static storage allocation is recursion is not supported.
* Another drawback is the size of data should be known at compile time

**2 Stack Storage Allocation**

* Storage is organized as a stack and activation records are pushed and popped as activation begins and end respectively. Locals are contained in activation records so they are bound to fresh storage in each activation.
* Recursion is supported in stack allocation

**3.  Heap Storage Allocation**

* Memory allocation and deallocation can be done at any time and at any place depending on the requirement of the user.
* Heap allocation is used to dynamically allocate memory to the variables and claim it back when the variables are no more required.
* Recursion is supported.

**PARAMETER PASSING:**

The communication medium among procedures is known as parameter passing

**Basic terminology :**

* **R- value:**The value of an expression is called its r-value
* **L-value:**The location of the memory(address) where the expression is stored is known as the l-value of that expression
* **i.Formal Parameter:**Variables that take the information passed by the caller procedure are called formal parameters.

**ii.Actual Parameter:**Variables whose values and functions are passed to the called function are called actual parameters.

**Different ways of passing the parameters to the procedure:**

* **Call by Value**  In call by value the calling procedure passes the r-value of the actual parameters and the compiler puts that into called procedure’s activation record. Formal parameters hold the values passed by the calling procedure, thus any changes made in the formal parameters do not affect the actual parameters.
* **Call by Reference** In call by reference the formal and actual parameters refers to same memory location. The l-value of actual parameters is copied to the activation record of the called function. Thus the called function has the address of the actual parameters. If the actual parameters does not have a l-value (eg- i+3) then it is evaluated in a new temporary location and the address of the location is passed. Any changes made in the formal parameter is reflected in the actual parameters
* **Call by Copy Restore** In call by copy restore compiler copies the value in formal parameters when the procedure is called and copy them back in actual parameters when control returns to the called function. The r-values are passed and on return r-value of formals are copied into l-value of actuals.
* **Call by Name** In call by name the actual parameters are substituted for formals in all the places formals occur in the procedure. It is also referred as lazy evaluation because evaluation is done on parameters only when needed.

### Advantages:

**Portability:**A runtime environment can provide a layer of abstraction between the compiled code and the operating system, making it easier to port the program to different platforms.

**Resource management:** A runtime environment can manage system resources, such as memory and CPU time, making it easier to avoid memory leaks and other resource-related issues.

**Dynamic memory allocation:** A runtime environment can provide dynamic memory allocation, allowing memory to be allocated and freed as needed during program execution.

**Garbage collection:** A runtime environment can perform garbage collection, automatically freeing memory that is no longer being used by the program.

**Exception handling:** A runtime environment can provide exception handling, allowing the program to gracefully handle errors and prevent crashes.

### Disadvantages:

**Performance overhead:** A runtime environment can add performance overhead, as it requires additional processing and memory usage.

**Platform dependency:** Some runtime environments may be specific to certain platforms, making it difficult to port programs to other platforms.

**Debugging:**Debugging can be more difficult in a runtime environment, as the additional layer of abstraction can make it harder to trace program execution.

**Compatibility issues:** Some runtime environments may not be compatible with certain operating systems or hardware architectures, which can limit their usefulness.

**Versioning:**Different versions of a runtime environment may have different features or APIs, which can lead to versioning issues when running programs compiled with different versions of the same runtime environment.