Parul®
University

Run time environments

Nisha Panchal, Asst. Professor
Computer Science & Engineering

Parul® University

DIGITAL LEARNING CONTENT

CHAPTER-7

Run time environments

Runtime environment

A translation of the program is focused on two things, first is the relation of the source program to dynamic actions that will happen during the runtime implementation of the program. And the program has various names of identifiers, functions, etc., that names are required to map with the actual memory location at runtime.

The runtime environment is a state of the target machine, which includes software libraries, environment variables, etc., to provide services to the processes running in the system.

Activation Record

- A program may have different functions, A function has a function name and the body of the function. Each execution of the function is referred to as an activation of the function. The lifetime of the function depends on the number of statements available in the function.
- When we have two functions and execution of that function perform one after another or after completion of first function second function start then that is called non-overlapping function.
- If during the execution of the current function second function starts or before completion of the first function (Activation) second function starts then the function is recursive.

Parul® University

DIGITAL LEARNING CONTENT

Activation Record

Properties of activation trees are:-

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

Let’s take one example of an activation record for quicksort

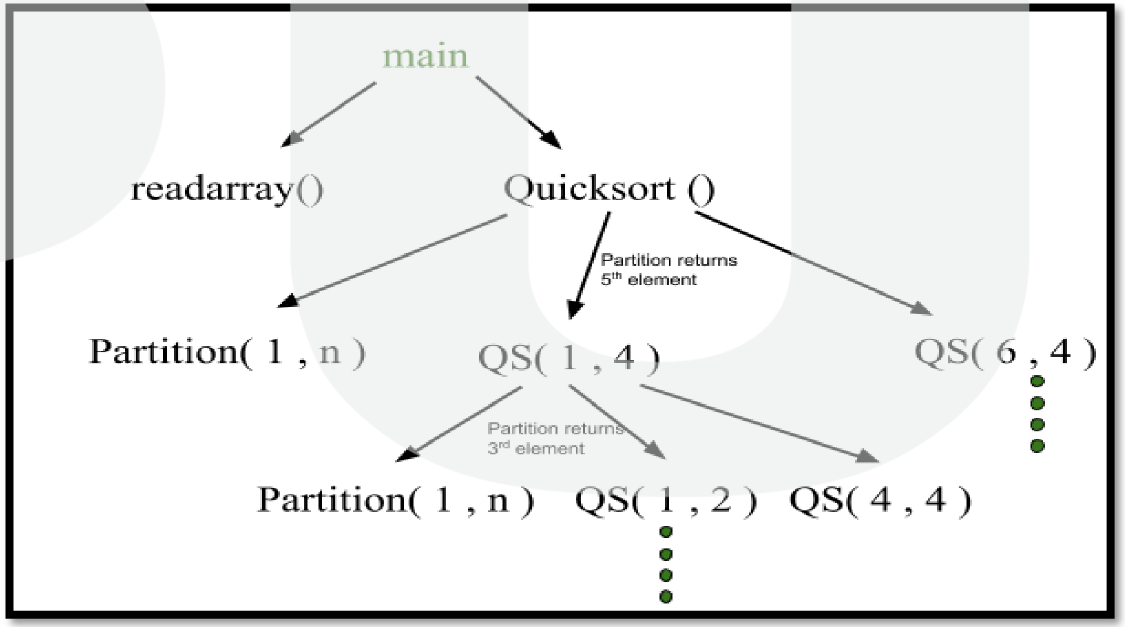
Activation Record

```
main() {

     Int n;
     readarray();
     quicksort(1,n);
}

quicksort(int m, int n) {

    Int i= partition(m,n);
    quicksort(m,i-1);
    quicksort(i+1,n);
}
```

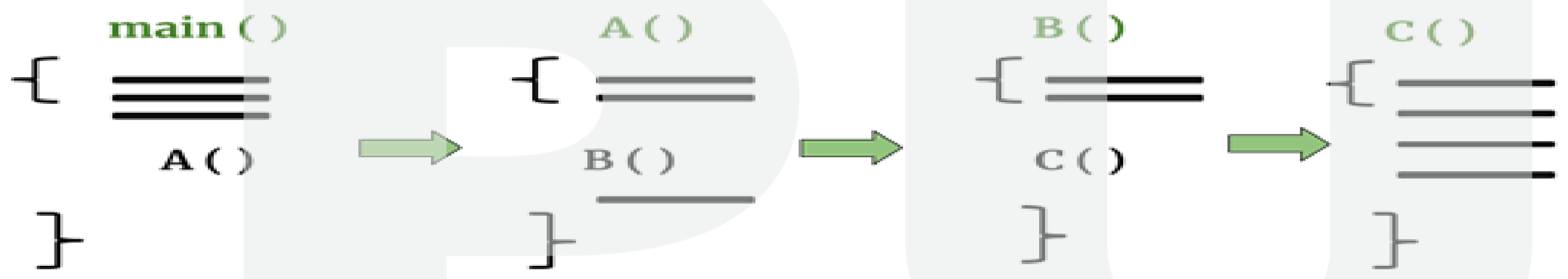


Activation Record

The first main function is whenever a function calls, the activation record will push into the stack, and when the function ends, the activation record is popped out from the stack.

Information needed by a single execution of a function is managed using an activation record or frame. When a function is called, an activation record is pushed into the stack, and as soon as the control returns to the caller function the activation record is popped.

Activation Record



Before going to A() activation record of main is created when A() is called activation record of A() is created.

Before going to B() activation record of main and A() in stack.

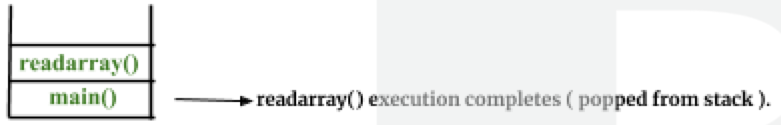
Before going to C() activation record of main and B() is pushed in stack.

Activation Record

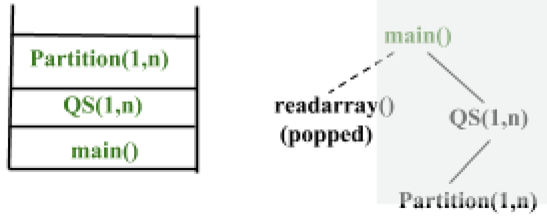
Activation record has the fisted things:

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

Activation Record

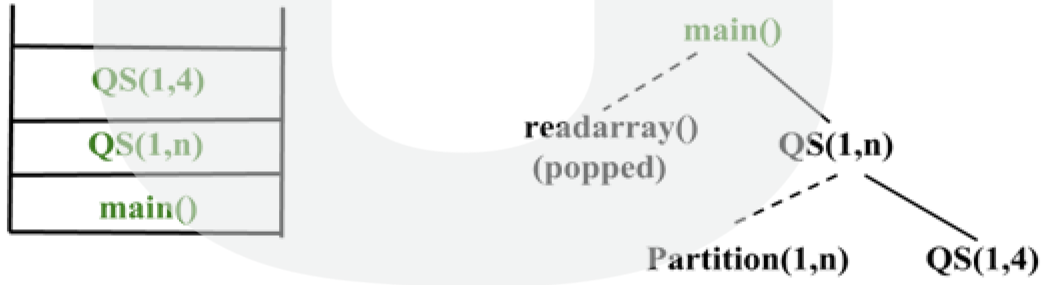


QS is called so it Enters the Stack.



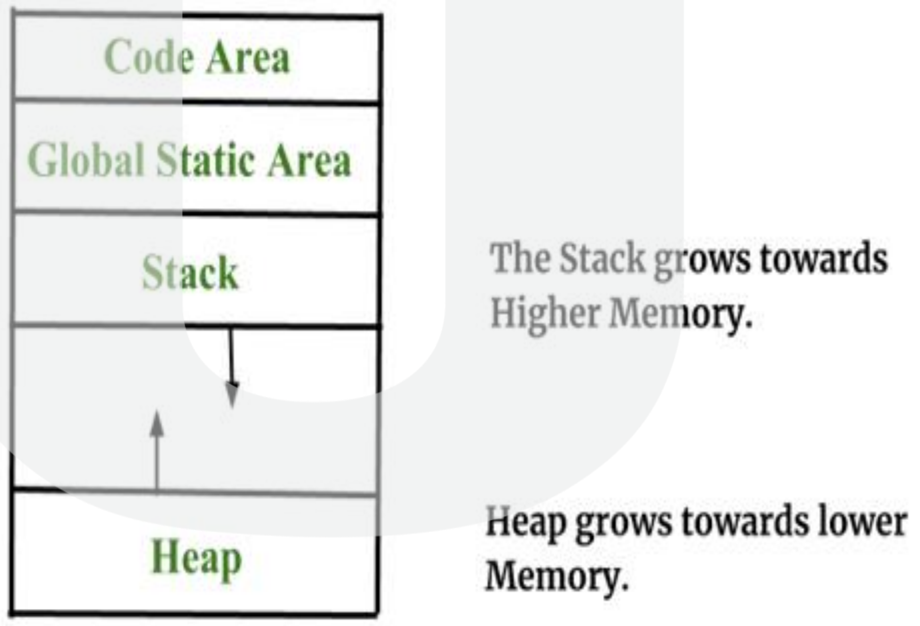
Partition Execution completed (popped out of stack)

Now QS is called again so it enters the Stack.



SUBDIVISION OF RUNTIME MEMORY

Runtime storage has a target code, static data objects, dynamic data objects, and automatic data objects. Target code contains the source code and its size can determine at compile time so it is static. Dynamic data objects store in heap memory and automatic data objects are stores in the stack.


Code Area
Global Static Area
Stack
Heap
The Stack grows towards Higher Memory.
Heap grows towards lower Memory.

STORAGE ALLOCATION TECHNIQUES

I. **Static Storage Allocation**

Static memory allocation is for any program which creates a memory at compile time so it will create in a static area and will deallocate after completion of the program. There are some drawbacks of static memory allocation, first, is recursion will not be supported and size of the data allocated should be known at compile time.

STORAGE ALLOCATION TECHNIQUES

II. Stack Storage Allocation

Stack storage allocation work based on push and pop operations. If the function is activated then the function will push into the stack and after completion, it will pop. So, at the all-time top, the stack contains the current activated function. Recursion of the function is allowed in stack allocation.

STORAGE ALLOCATION TECHNIQUES

III. Heap Storage Allocation

Memory allotment and deallocation should be possible whenever and at any spot contingent upon the necessity of the user. heap allocation is utilized to dynamically allocate memory to the variables and guarantee it back when the variables are not anymore required. Recursion of function is allowed in heap memory allocation.

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in