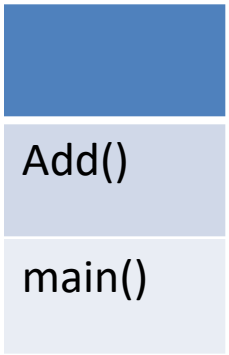
Unit IV
Run Time Environment and Intermediate Code Generation

- Run Time Environment / Run time Storage Management
  - Where the Application is executed
  - Resources should be correctly assigned for runtime environment to be successful
  - Source code – name of identifiers/functions should be mapped to actual memory @runtime
  - Program during execution
    - How the memory is assigned for variables
    - Dynamic Memory Management

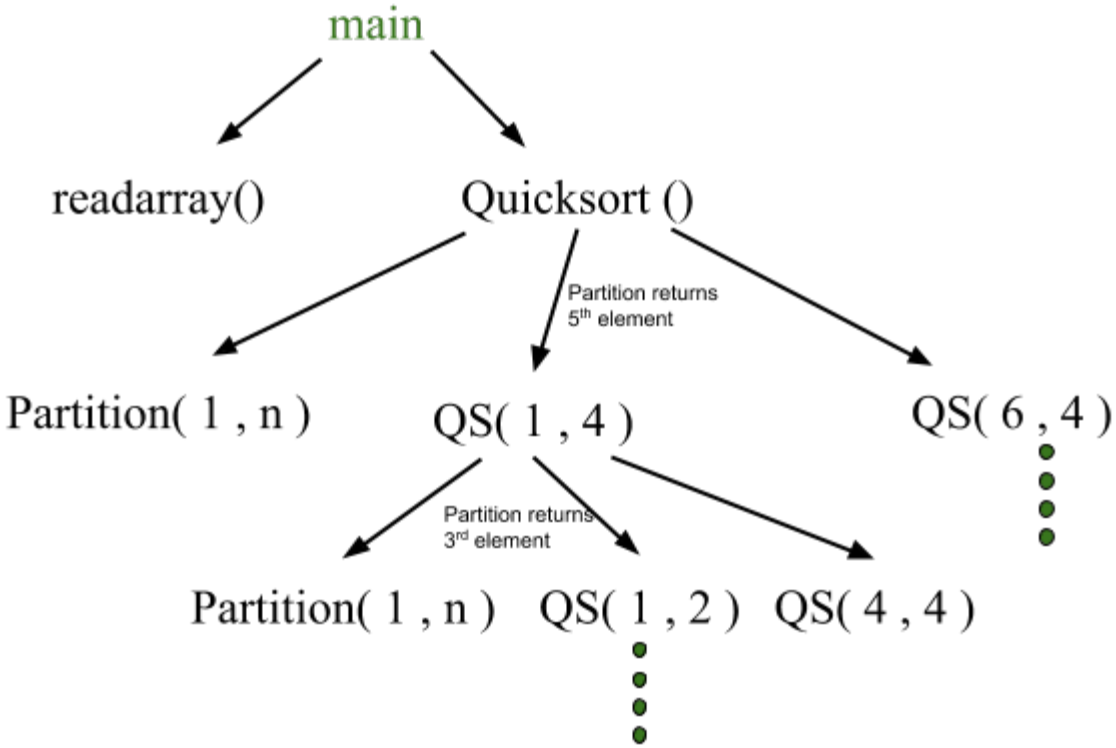
Source language Issues

- Procedures
  - Identifier with a statement
  - Void add( )
  - { cout<<a+b;}
  - Identifier → function name → add
  - Statement → function body → cout statement
  - Function Execute → ***Activation***
  - Activation
    - Lifetime of Activation – steps in that function
    - Activation – Recursive



Activation Tree

- Node – Activation
- Root node- main function



Activation Record

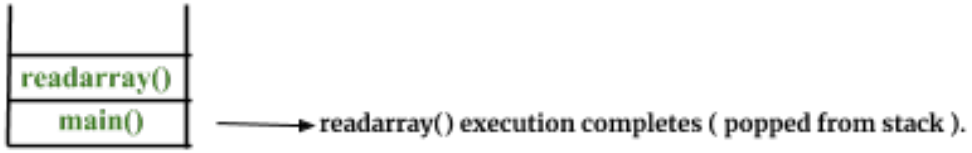
Local Variable : local to that function

- Temporary values : evaluation of expression
- Machine Status : status before the function call
- Access Link: variables outside local scope
- Control link: Activation record of caller
- Return Value: called to calling function
- Actual Parameter: Function call

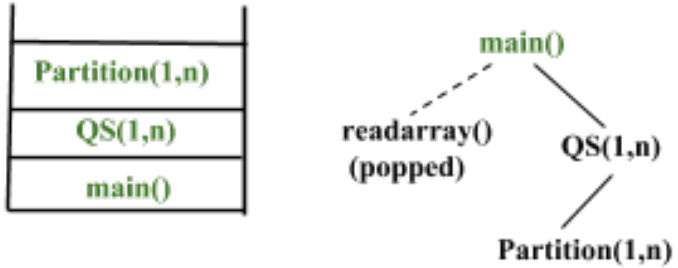
```c
#include <stdio.h>

void swap(int*, int*); //function declaration

void main()
{
   int x=10, y=20;

   printf("Before Swapping\nx = %d y = %d\n", x, y);

   swap(&x, &y);     //function call

   printf("After Swapping\nx = %d y = %d\n", x, y);
}
//function definition
void swap(int *ptr1, int *ptr2)
{
   int temp;
 temp = *ptr2;
*ptr2 = *ptr1;
*ptr1 = temp;
}
```

Control Stack

- Control stack / Run time stack – track the live execution
- ***Control Stack***



QS is called so it Enters the Stack.

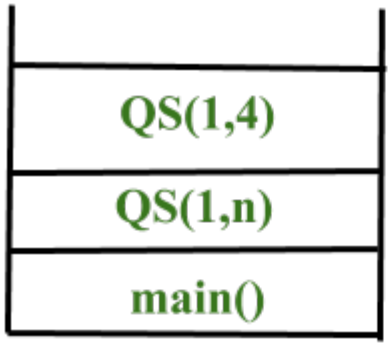


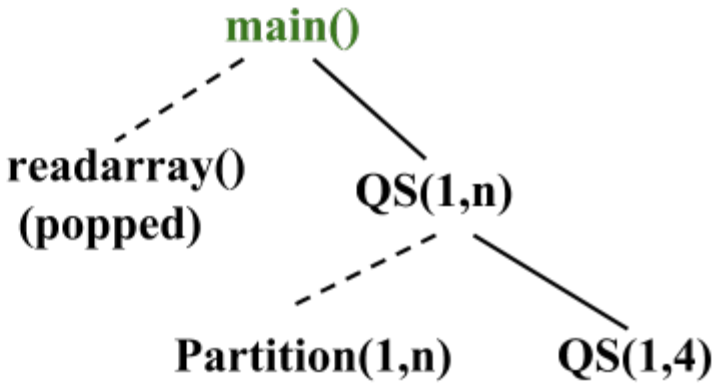
Partition Execution completed (popped out of stack)

Control Stack

- ***Control Stack***

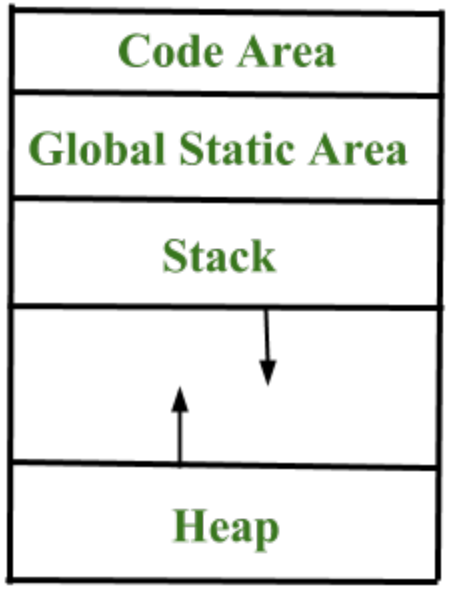
Now QS is called again so it enters the Stack.





Storage Organization

- Runtime storage
  - Code – Text part (Memory req – compile time) – doesn't change @runtime
  - Global static Area
  - Stack – procedure call random manner *[stack – procedure]*
  - *Heap* – managing memory allocation of memory for *variables* @ runtime



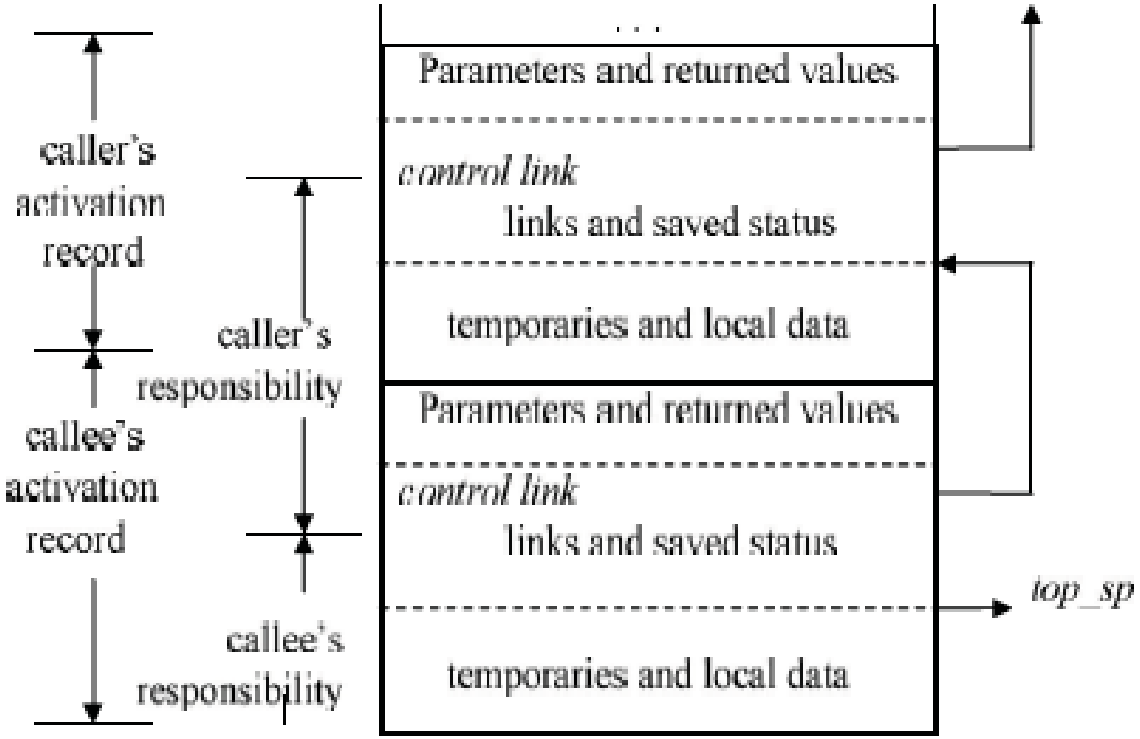
The Stack grows towards Higher Memory.

Heap grows towards lower Memory.

Storage Allocation Strategies

- *Static Storage Allocation*
  - Recursion is not supported
  - Should know the value @compile time
  - Ex: FORTRAN
- *Stack Storage Allocation*
  - Recursion is supported
  - Stack activation are pushed and popped
- *Heap Storage Allocation*
  - Recursion is supported
  - Dynamic Allocation of memory to variables

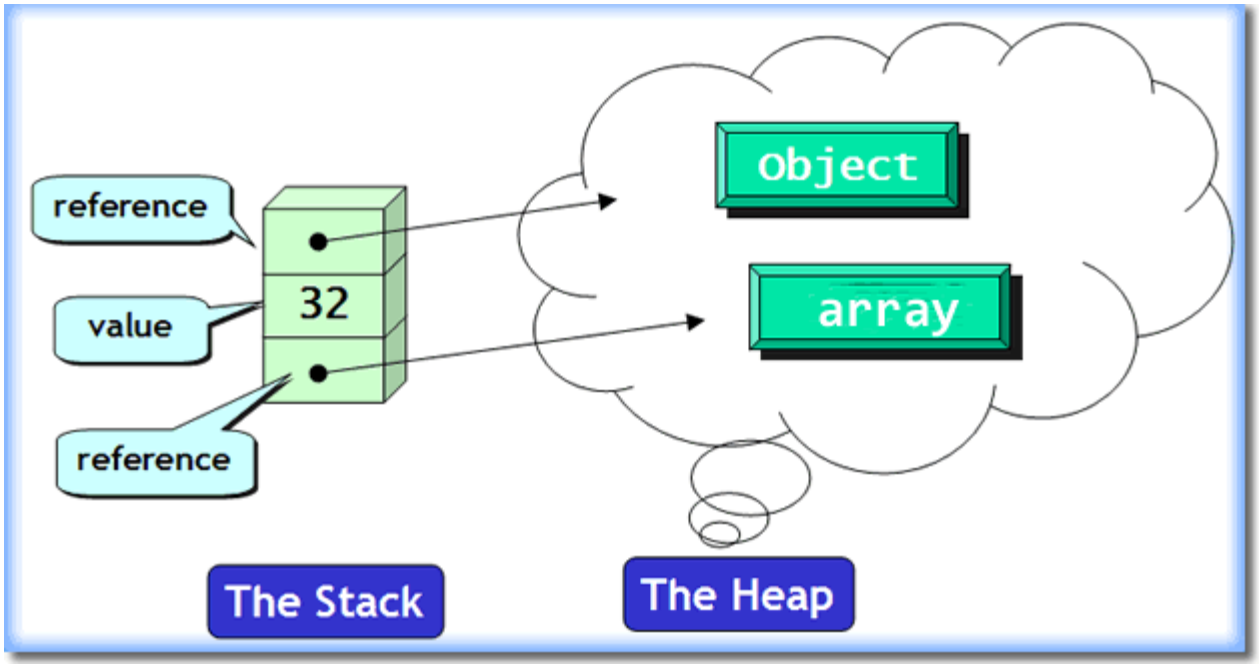
Stack Allocation of Space /Temporary memory allocation



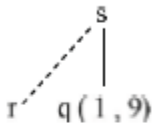
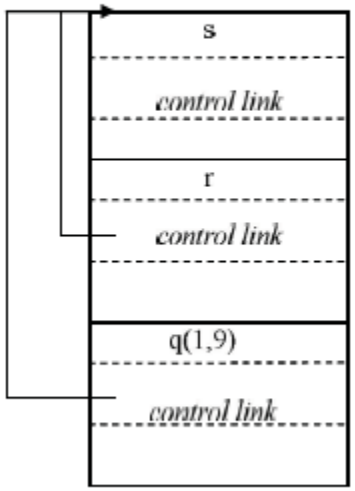
Division of tasks between caller and callee

Heap Allocation of Space

- Value – local name – retained – activation ends
- Global variables
- Objects – Heap
- Referencing variables of object - stack



Heap Allocation of Space

Position in the activation tree	Activation records in the heap	Remarks
		Retained activation record for r