Unit IV
Run Time Environment and Intermediate Code Generation

- Run Time Environment / Run time Storage Management
  - *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.*
  - 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

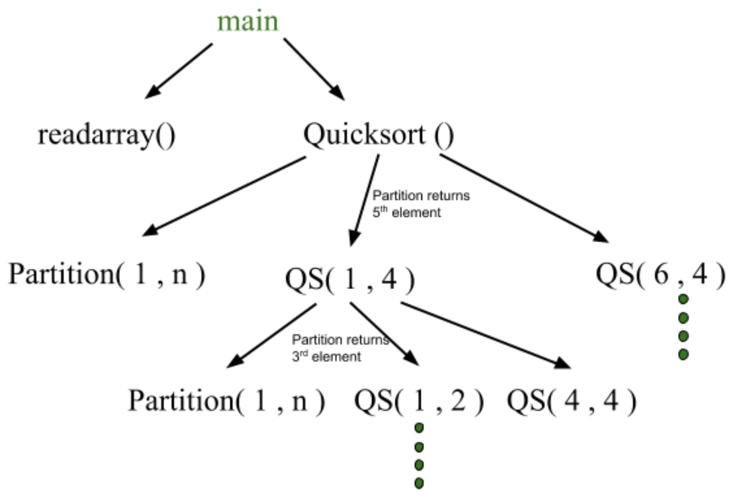
- Procedures
  - *A procedure definition is a declaration that associates an identifier with a statement. The identifier is procedure name, and statement is the procedure body.*
  - Identifier with a statement
  - Void add( )
  - { cout<<a+b;}
  - Identifier → function name → add
  - Statement → function body → cout statement
  - Function Execute → ***Activation***
  - Activation
    - Lifetime of Activation – steps in that function
    - Activation – Recursive

Activation Record
Add()
main()

Activation Tree

- 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: Quick Sort

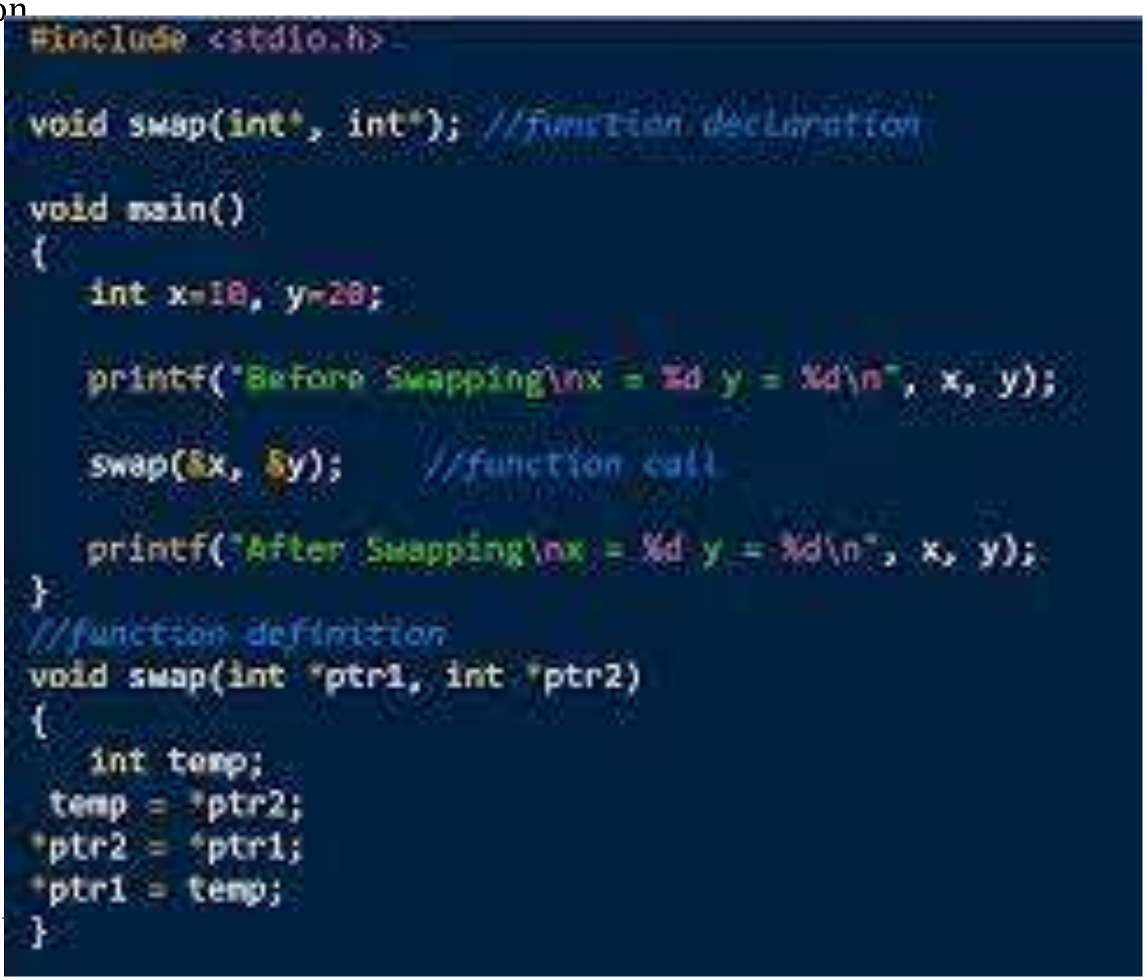
```
main() {

      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 Tree

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



```
#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;
}
```