Date .....

Example 6.22 → Improvised version

```
if (x<100 || x>200 && x!=y) x=0;

    if (x<100) goto L1
    goto L2
L2: if x>200 goto L3
    goto L4
L3: if x!=y goto L1
    goto L4
L1: x=0
L4:
```

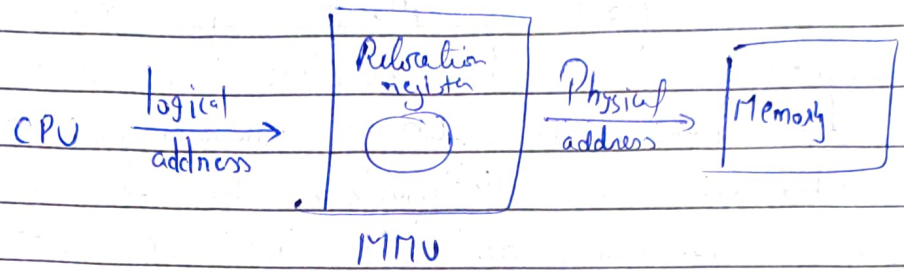
chapter - 7 Runtime Environment

Issues →

i) Layout of address locations
ii) Allocation of address to objects
iii) Mechanism to access variables
iv) Linkages between procedures
v) Passing parameters

Date .....

Storage Organization



The global constants & the data generated by the compiler is placed in statically determined called "static" which may be known at compile time. The size of the generated target code is fixed at compile time so that the compiler can place the executable target code in a statically determined area "code" which is usually placed in the low end of the memory.

Heap & Stack

To maximize the utilization of space at runtime, the other two areas called stack & heap are at the opposite ends of the remainder of the address space. These areas are dynamic i.e., their size can change as the program executes. The stack is used to store data structures called activation records generated during procedure calls.

Date .....

Static vs Dynamic Storage Allocation

The layout & allocation of data to memory locations in the runtime environment are the key issues in storage management. The storage allocation is distinguished by two i.e. static & dynamic and the time at which the it is decided is called static & dynamic time.

Dynamic Storage Allocation

Stack & Heap → Stack stores the names local to a procedure whereas heap allocates the memory to the object where they are created like in C, malloc & ~~return~~ that storage when they are invalidated.

Garbage Collection :- It enables the runtime system to detect

Stack Allocation → whenever a procedure/function/subroutine/method, space for its local variables are pushed onto the stack. When the procedure terminates, that space is popped off the stack.

Spiral

Date .....

Activation Trees

Stack allocation would not be feasible if procedure calls, or activations of procedures, did not nest in time.

If an activation of procedure p calls procedure q, then that activation of q must end before the activation of p can end. There are three common cases:-

→ The activation of q terminates normally. Then in essentially any language, control resumes just after the point p at which the call to q was made.

→ The activation of q, or some procedure q called, either directly or indirectly aborts, i.e. it becomes impossible for execution to continue. In that case, p ends simultaneously with q.

→ The activation of q terminates because of an exception that q can't handle. Procedure p may handle the exception, in which case the activation of q has terminated while activation of p continues, although not necessarily from the point at which the call to q was made. If p can't handle the exception, then this activation of p terminates at the same time as the activation of q.

Sheral

Date .....

Quicksort

```
int a[11];
void readArray(){
    int i;
}
int partition(int m, int n){
/* picks a separation value (pivot) v & partitions a[m...n]
so that a[m...p-1] <= v >= a[p+1...n]. Returns p */
void quicksort(int m, int n){
int i;
if (n > m){
    i = partition(m,n);
    quicksort(m, i-1);
    quicksort(i+1, n);
}
}
main(){
    readArray();
    a[0] = -9999;
    a[10] = 9999;
    quicksort(1,9);
}
```

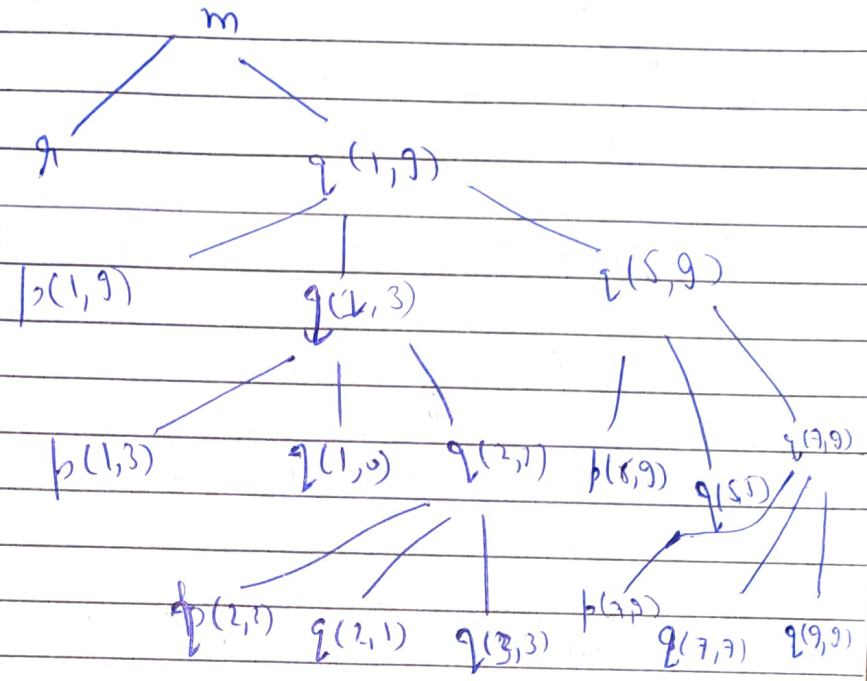
Activations →

enter main()
enter readArray()
leave readArray()
enter quicksort(1,9)
enter Partition(1,9)
leave Partition(1,9)
enter quicksort(1,3)
leave quicksort(1,3)
enter quicksort(5,9)
leave quicksort(5,9)
leave main()

Spiral

Date .....

Activation Tree



The use of run-time stacks is enabled by several useful relationships between the activation tree & the behaviour of the program

→ The sequence of procedure calls corresponds to a preorder traversal of activation tree.

→ The sequence of returns corresponds to a post order traversal of the activation tree

→ Suppose that control lies within particular activation of some procedure, corresponding to a Node N of activation tree. Then the activations that are currently open are those that correspond to node N & its ancestors. The order in which these activations were called is the order in which they appear along the path to N, starting at the root & they will return in the reverse of that order.

Spiral

Date .....

Control Stack / Run-time Stack

Procedure calls & returns are usually managed by a run-time stack called the control stack.

Activation Record → Each live activation has an activation records (sometimes called frame) on the control block.

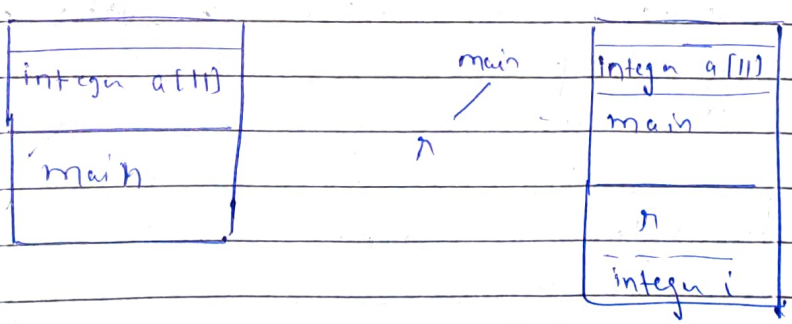
→ Root of activation trees at the bottom content of activation record.

Content of Activation Records

1) Actual Parameters → used by calling procedures
2) Returned values → if procedure returns a value
3) Control link → pointing to activation of caller
4) Access link → used to locate data by called procedure
5) Saved Machine Status → status is stored just before calling another procedure.
6) Local data → belong to procedure
7) Temporaries → arising from evaluating expression.

Stack →

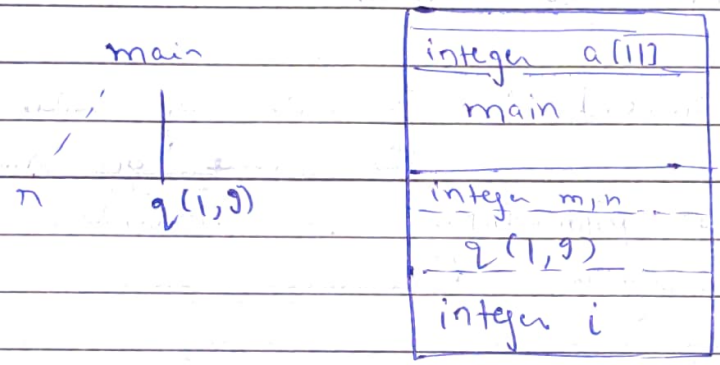
frame for main



Spiral

Date .....

c) n has been popped & q(1,9) pushed



Downward growing stack of activation records

Example → find the Output

```
int f(int x, int *y, int **z) {
    **z += 1;
    *y += 2;
    x += 3;
    return x + *y + **z;
}
int main(){
    int x, c, *b, **a;
    c = 4, b = &c, a = &b;
    x = f(c, b, a);
    printf("%d", x);
    return 0;
}
```

Ans → 21

Spiral

Date .....

Calling Sequence → Consists of code that allocates an activation record on the stack.

Return stack → Consists of code to restore the state of machine so the calling procedure can continue its execution after call.