Application of Stack & Queue

Stack Applications

Expression Evaluation

Stack is used to evaluate prefix, postfix and infix expressions.

Infix to Postfix or Infix to Prefix Conversion −

The stack can be used to convert some infix expression into its postfix equivalent, or prefix equivalent. These postfix or prefix notations are used in computers to express some expressions. These expressions are not so much familiar to the infix expression, but they have some great advantages also. We do not need to maintain operator ordering, and parenthesis.

Postfix or Prefix Evaluation −

After converting into prefix or postfix notations, we have to evaluate the expression to get the result. For that purpose, also we need the help of stack data structure.

Stack data structure is used for evaluating the given expression. For example, consider the following expression

5 * (6 + 2) - 12 / 4

Since parenthesis has the highest precedence among the arithmetic operators, **(6 +2) = 8** will be evaluated first. Now, the expression becomes

5 * 8 - 12 / 4

* and / have equal precedence and their associativity is from left-to-right.

So, start evaluating the expression from left-to-right.

5 * 8 = 40 and **12 / 4 = 3**

Now, the expression becomes

40 – 3

And the value returned after the subtraction operation is **37**.

Syntax Parsing

Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code.

String Reversal

Stack is used to reverse a string. We push the characters of string one by one into stack and then pop character from stack.

Parenthesis Checking

Stack is used to check the proper opening and closing of parenthesis.

Given an expression, you have to find if the parenthesis is either correctly matched or not. For example, consider the expression **(a + b) * (c + d).**

In the above expression, the opening and closing of the parenthesis are given properly and hence it is said to be a correctly matched parenthesis expression. Whereas, the expression, **(a + b * [c + d)** is not a valid expression as the parenthesis are incorrectly given.

BALANCED EXPRESSION	UNBALANCED EXPRESSION
(a + b)	(a + b
[(c – d) * e]	[(c – d * e]
{ ( ) } [ ]	{ [( ]) }

Here are some of the balanced and unbalanced expressions:

Consider the above mentioned unbalanced expressions:

•The first expression (a + b is unbalanced as there is no closing parenthesis given.

•The second expression [(c - d * e] is unbalanced as the closed round parenthesis is not given.

•The third expression { [( ]) } is unbalanced as the nesting of square parenthesis and the round parenthesis are incorrect.

Backtracking

Suppose we are finding a path for solving maze problem. We choose a path and after following it we realize that it is wrong. Now we need to go back to the beginning of the path to start with new path. This can be done with the help of stack.

Backtracking is one of the algorithm designing technique. For that purpose, we dive into some way, if that way is not efficient, we come back to the previous state and go into some other paths. To get back from current state, we need to store the previous state. For that purpose, we need stack. Some examples of backtracking is finding the solution for Knight Tour problem or N-Queen Problem etc.

Function Call

Stack is used to keep information about the active functions or subroutines.

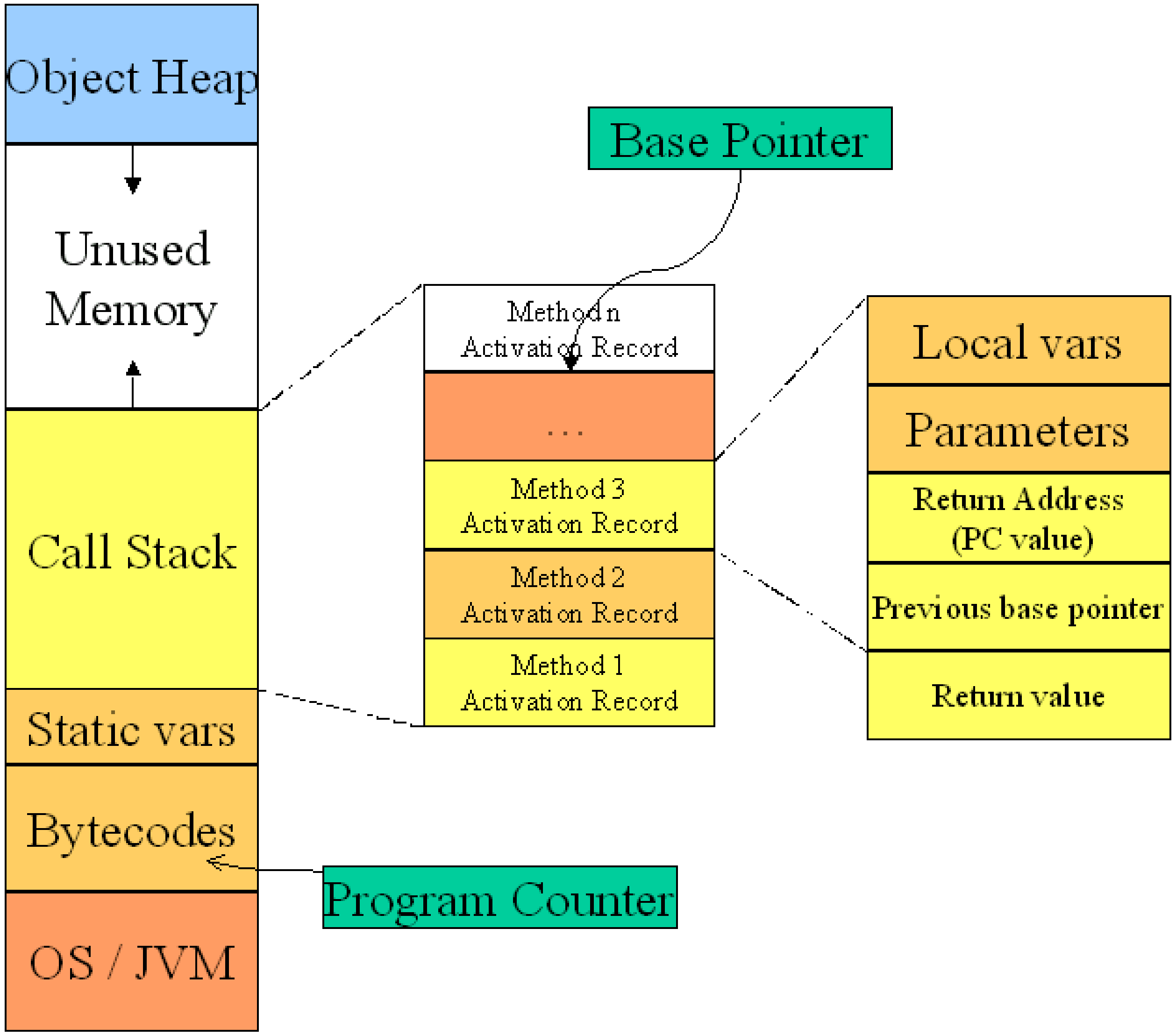
When we call a function from one other function, that function call statement may not be the first statement. After calling the function, we also have to come back from the function area to the place, where we have left our control. So we want to resume our task, not restart. For that reason, we store the address of the program counter into the stack, then go to the function body to execute it. After completion of the execution, it pops out the address from stack and assign it into the program counter to resume the task again.

Memory Management

Any modern computer environment uses a stack as the primary memory management model for a running program. Whether it's native code (x86, Sun, VAX) or JVM, a stack is at the center of the run-time environment for Java, C++, Ada, FORTRAN, etc.

The discussion of JVM in the text is consistent with older and modern OS such as Windows NT 8 or 10, Solaris, Unix/Linux runtime environments.

Each program that is running in a computer system has its own memory allocation containing the typical layout as shown below. [Thread management has a variant of this.]


Object Heap
Unused Memory
Call Stack
Static vars
Bytecodes
OS / JVM
Base Pointer
Program Counter
Method n Activation Record
. . .
Method 3 Activation Record
Method 2 Activation Record
Method 1 Activation Record
Local vars
Parameters
Return Address (PC value)
Previous base pointer
Return value

Call and return process

When a method/function is called

- An **Activation Record i**s created; its size depends on the number and size of the local variables and parameters.
- The **Base Pointer** "Register" value is saved in the special location reserved for it
- The **Program Counter** "Register" value is saved in the **Return Address** location
- The Base Pointer is now reset to the new base (top of the call stack prior to the creation of the AR)

- The Program Counter is set to the location of the first bytecode of the method being called
- Copies the calling parameters into the Parameter region
- Initializes local variables in the local variable region

While the method executes, the local variables and parameters are simply found by adding a constant associated with each variable/parameter to the Base Pointer.

When a method returns

- Get the program counter from the activation record and replace what's in the PC register
- Get the base pointer value from the AR and replace what's in the BP register
- Pop the Activation Record entirely from the stack.

  There is also a **Stack Pointer** that allow other temporary use of the stack above the top AR.

Application of Queues

- Data getting transferred between the IO Buffers (Input Output Buffers).
- CPU scheduling and Disk scheduling.
- Managing shared resources between various processes.
- Job scheduling algorithms.
- Round robin scheduling.
- Recognizing a palindrome.

Applications of circular queue

Traffic light functcolors in the traffic light follow a circular pattern.

In page replacement algorithms, a circular list of pages is maintained and when ioning is the best example for circular queues. The a page needs to be replaced, the page in the front of the queue will be chosen.

Applications of Dequeue

Pallindrome checker.

A-steal job scheduling algorithm

- The A-steal algorithm implements task scheduling for multiple processors (multiprocessor scheduling).
- The processor gets the first element from the double ended queue.
- When one of the processors completes execution of its own thread, it can steal a thread from other processors.
- It gets the last element from the deque of another processor and executes it.

Undo-redo operations in software applications.

Applications of Priority Queue

Prim's algorithm implementation can be done using priority queues.

Dijkstra's shortest path algorithm implementation can be done using priority queues.

A* Search algorithm implementation can be done using priority queues.

Priority queues are used to sort heaps.

Priority queues are used in operating system for **load balancing** and **interrupt handling**.

Priority queues are used in **huffman** codes for data compression.

In traffic light, depending upon the traffic, the colors will be given priority.