International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 03 | Mar-2018 www.irjet.net p-ISSN: 2395-0072

Dynamic Implementation of Stack Using Single Linked List

Dr. Mahesh K Kaluti[1], GOVINDARAJU Y[2], SHASHANK A R[3], Harshith K S[4]

[1]Assistant professor, CSE Dept. of Computer science and Engineering PESCE, Mandya
[2,3,4] M tech, CSE Dept. of Computer science and Engineering PESCE, Mandya

---***---

Abstract - *This paper gives the general overview of linked list and its various types. This research papers covers a brief history of the stacks and various operations of stack that is insertion at the top, deletion from the top, display of stack elements. In this we focus on how to insert an element in to stack, delete an item from the stack and display stack elements by using single linked list with program code.*

Key Words: **Node; stack; linked list; top; data structure**

1. INTRODUCTION

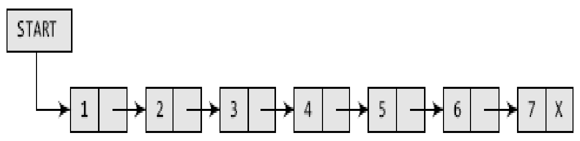
A linked list, in simple terms, is a linear collection of data elements. These data elements are called *nodes*. Linked list is a data structure which in turn can be used to implement other data structures. Thus, it acts as a building block to implement data structures such as stacks, queues, and their variations. A linked list can be perceived as a train or a sequence of nodes in which each node contains one or more data fields and a pointer to the next node.

2. TYPES OF LINKED LIST

Linked lists are classified into following categories depending upon the number of pointers on the basis of requirement and usage.

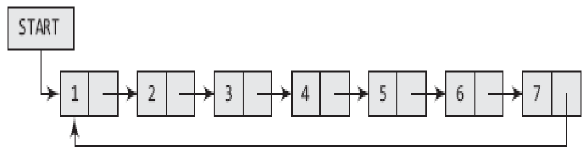
2.1 SINGLE LINKED LISTS

A singly linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. By saying that the node contains a pointer to the next node, we mean that the node stores the address of the next node in sequence. A singly linked list allows traversal of data only in one way.



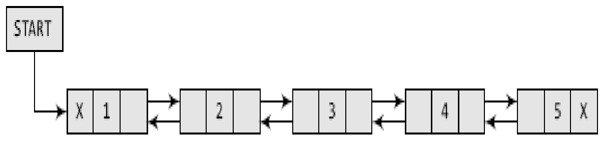
2.2 CICULAR LINKED LISTS

In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular singly linked list as well as a circular doubly linked list. While traversing a circular linked list, we can begin at any node and traverse the list in any direction, forward or backward, until we reach the same node where we started. Thus, a circular linked list has no beginning and no ending.



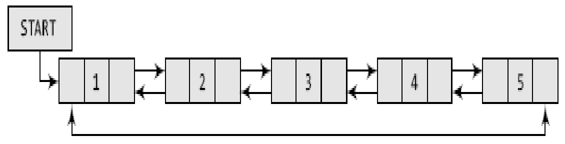
2.2 DOUBLE LINKED LISTS

A doubly linked list or a two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence. Therefore, it consists of three parts data, a pointer to the next node, and a pointer to the previous node as shown in Fig.



2.3 CICULAR DOUBLE LINKED LISTS

A circular doubly linked list or a circular two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence. The difference between a doubly linked and a circular doubly linked list is same as that exists between a singly linked list and a circular linked list. The circular doubly linked list does not contain NULL in the previous field of the first node and the next field of the last node. Rather, the next field of the last node stores the address of the first node of the list, i.e., START. Similarly, the previous field of the first field stores the address of the last node. A circular doubly linked list is shown in Fig.

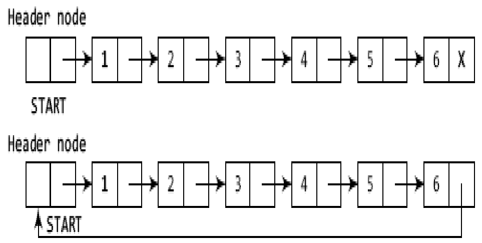


2.4 HEADER LINKED LISTS

A header linked list is a special type of linked list which contains a header node at the beginning of the list. So, in a header linked list, START will not point to the first node of the list but START will contain the address of the header node. The following are the two variants of a header linked list.

Grounded header linked list which stores NULL in the next field of the last node.

Circular header linked list which stores the address of the header node in the next field of the last node. Here, the header node will denote the end of the list. Look at Fig which shows both the types of header linked lists.



3. STACKS

Stack is an important data structure which stores its elements in an ordered manner. A stack is a linear data structure which uses the same principle, i.e., the elements in a stack are added and removed only from one end, which is called the TOP. Hence, a stack is called a LIFO (Last-In-First-Out) data structure, as the element that was inserted last is the first one to be taken out.

3.1 OPERATIONS ON A STACK

A stack supports three basic operations: push, pop, and peek. The push operation adds an element to the top of the stack and the pop operation removes the element from the top of the stack. The peek operation returns the value of the topmost element of the stack.

3.1.1 PUSH OPERATION

The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack. However, before inserting the value, we must first check if TOP=MAX–1, because if that is the case, then the stack is full and no more insertions can be done. If an attempt is made to insert a value in a stack that is already full, an OVERFLOW message is printed.

3.1.2 POP OPERATION

The pop operation is used to delete the topmost element from the stack. However, before deleting the value, we must first check if TOP=NULL because if that is the case, then it means the stack is empty and no more deletions can be done. If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed.

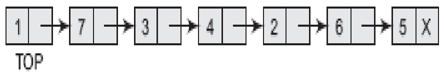
3.1.3 PEEK OPERATION

Peek is an operation that returns the value of the topmost element of the stack without deleting it from the stack. However, the Peek operation first checks if the stack is empty, i.e., if TOP = NULL, then an appropriate message is printed, else the value is returned

4. IMPLEMENTION OF STACK USING SINGLE LINKED LIST

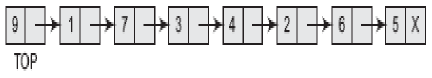
PUSH OPERATION

The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack. Consider the linked stack shown in Fig



Linked stack

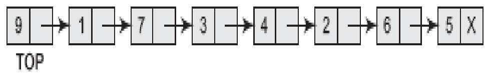
To insert an element with value 9, we first check if TOP=NULL. If this is the case, then we allocate memory for a new node, store the value in its DATA part and NULL in its NEXT part. The new node will then be called TOP. However, if TOP!=NULL, then we insert the new node at the beginning of the linked stack and name this new node as TOP. Thus, the updated stack becomes as shown in Fig.



Linked stack after inserting a new node

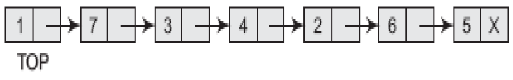
POP OPERATION

The pop operation is used to delete the topmost element from a stack. However, before deleting the value, we must first check if TOP=NULL, because if this is the case, then it means that the stack is empty and no more deletions can be done. If an attempt is made to delete a value from a stack that is already empty, an UNDERFLOW message is printed. Consider the stack shown in Fig



Linked stack

In case TOP!=NULL, then we will delete the node pointed by TOP, and make TOP point to the second element of the linked stack. Thus, the updated stack becomes as shown in Fig



Linked stack after deletion of the topmost element

5. CODE FOR IMPLEMENTATION OF STACK USING SINGLE LINKED LIST

5.1 PUSH OPERATION

```
void push(int st[], int val)
{
if(top == MAX-1)
{
printf("\n STACK OVERFLOW");
}
else
{
top++;
st[top] = val;
}
}
```

5.2 POP OPERATION

```
int pop(int st[])
{
int val;
if(top == -1)
{
printf("\n STACK UNDERFLOW");
return -1;
}
else
{
val = st[top];
top--;
return val;
}
}
```

5.3 PEEK OPERATION

```
int peek(int st[])
{
if(top == -1)
{
printf("\n STACK IS EMPTY");
return -1;
}
else
return (st[top]);
}
```

6. CONCLUSION

The technique of creating a stack using array is easy but the drawback is that the array must be declared to have some fixed size. In case the stack is a very small one or its maximum size is known in advance, then the array implementation of the stack gives an efficient implementation. But if the array size cannot be determined in advance, then the other alternative, i.e., linked representation, is used. The storage requirement of linked representation of the stack with n elements is O(n), and the typical time requirement for the operations is O(1).

7. REFERENCES

1. Allen Newell, Cliff Shaw and Herbert A. Simon "Linked List".

2. Search engines like google and yahoo.

3. T. Lev-Ami and S. Sagiv. TVLA: A system for implementing static analyses. InSAS 00: Static Analysis Symposium, volume 1824 ofLNCS, pages 280–301.Springer-Verlag, 2000.

4. A. Møller and M. I. Schwartzbach. The pointer assertion logic engine.In PLDI 01: Programming Language Design and Implementation,pages 221–231, 2001.

5. G. Nelson. Verifying reachability invariants of linked structures.InPOPL 83: Principles of Programming Languages, pages 38–47.ACM Press, 1983.

6. Collins, William J. (2005) [2002]. Data Structures and the Java Collections Framework. New York: McGraw Hill. pp. 239–303. ISBN 0-07-282379-8.

7. "Definition of a linked list". National Institute of Standards and Technology. 2004-08-16. Retrieved 2004-12-14.

8. Antonakos, James L.; Mansfield, Kenneth C., Jr. (1999). Practical Data Structures Using C/C++. Prentice-Hall. pp. 165–190. ISBN 0-13-280843-9.

9. Cormen, Thomas H.; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein (2003). Introduction to Algorithms. MIT Press. pp. 205–213 & 501–505. ISBN 0-262-03293-7

10. Green, Bert F. Jr. (1961). "Computer Languages for Symbol Manipulation". IRE Transactions on Human Factors in Electronics (2): 3–8.doi:10.1109/THFE2.1961.4503292

11. McCarthy, John (1960). "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I". Communications of the ACM **3** (4): 184.doi:10.1145/367177.367199

12. Juan, Angel (2006). "Ch20 –Data Structures; ID06 - PROGRAMMING with JAVA (slide part of the book "Big Java", by CayS. Horstmann)" (PDF). p. 3