Abstract:-

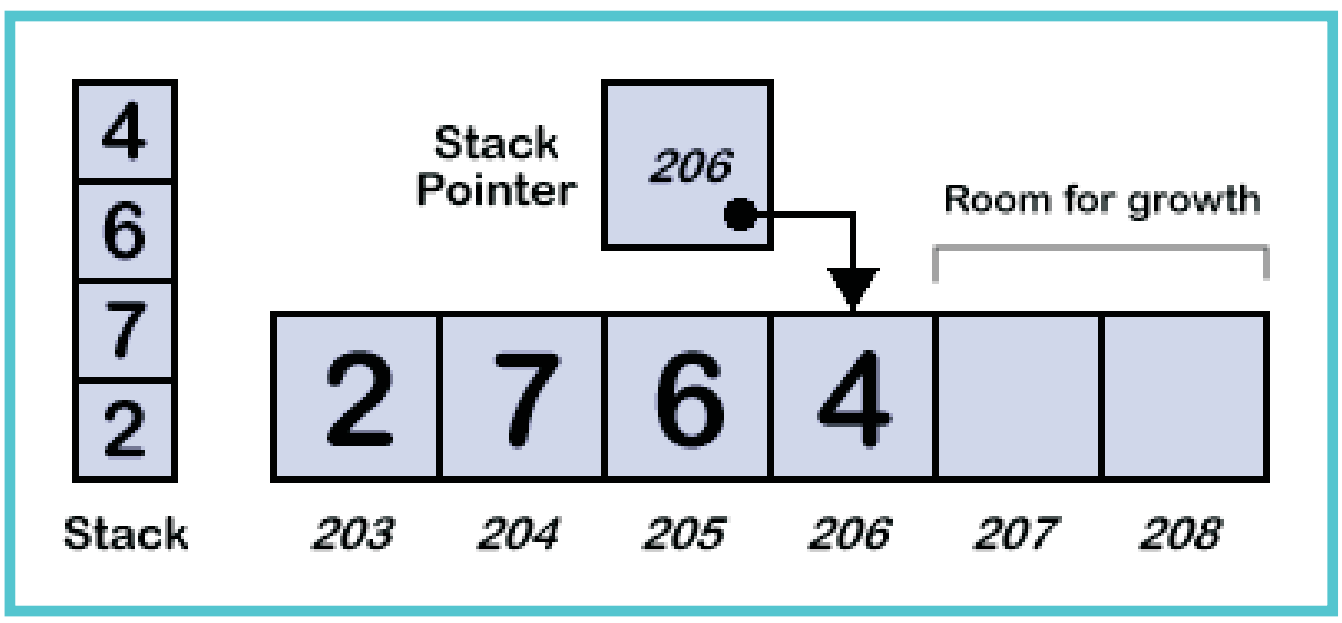
This application note shows how to implement stack in data structure. This note shows using stack to create a hexadecimal number and how to convert decimal number to binary using stack.

Sagar S. Shillewar and Gajanan S. Jadhav

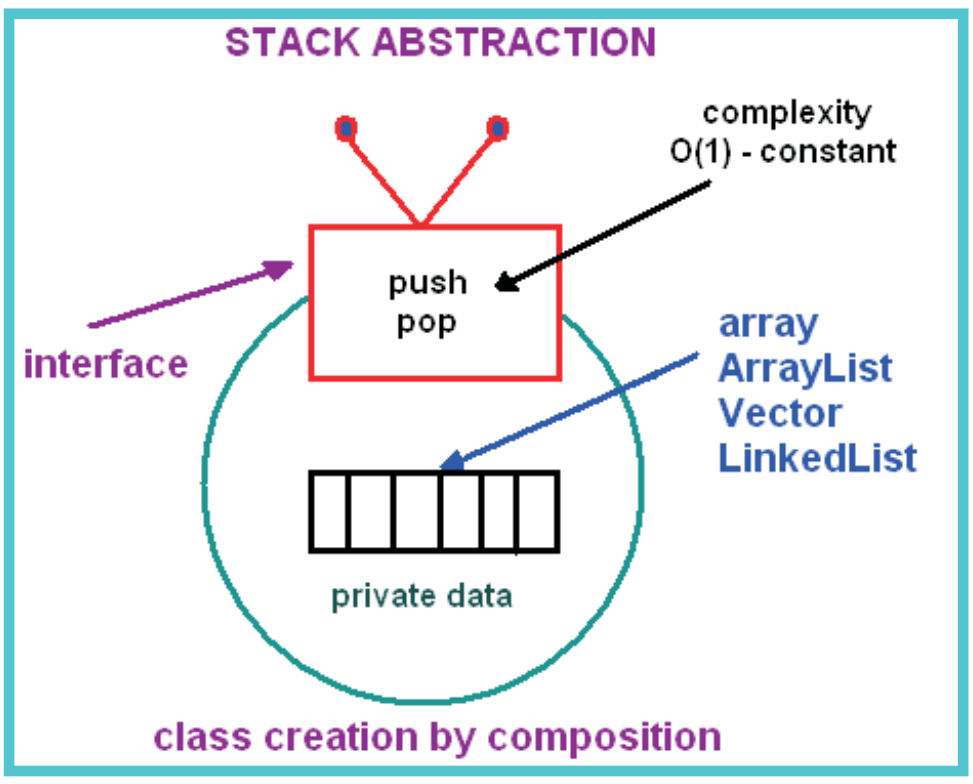
Chhatrapati Shivaji Raje Mahavidayalaya, Kinwat Dist Nanded.

Keywords:

Adapter, Hexa, Auxiliary.



IMPLEMENTATION OF STACK



INTRODUCTION-

A stack is an ordered list of items. Items are added to the list at the top and items are removed from the top. Therefore the last item are added to the list is the first removed from the list, stack are also known as "Last in First Out"(LIFO) LISTS. A stack is easily implemented in an array, requiring only a point to the position of the top element of the stack.

The operation needed to use a stack is: Create the stack, delete the stack, add an item to the stack (push), delete an item from the stack (pop), and check the length of the stack. A stack may be implemented to have a bounded capacity. If the stack is full & does not contain enough space to accept an entity to be pushed, the stack is then considered to be in overflow state.

Stack description:

The pop operation removes an item from the top of the stack. A pop either reveals previously considered items or results in an empty stack, but, if the stack is empty, it goes into underflow state, which means no items are present in stack to be removed.

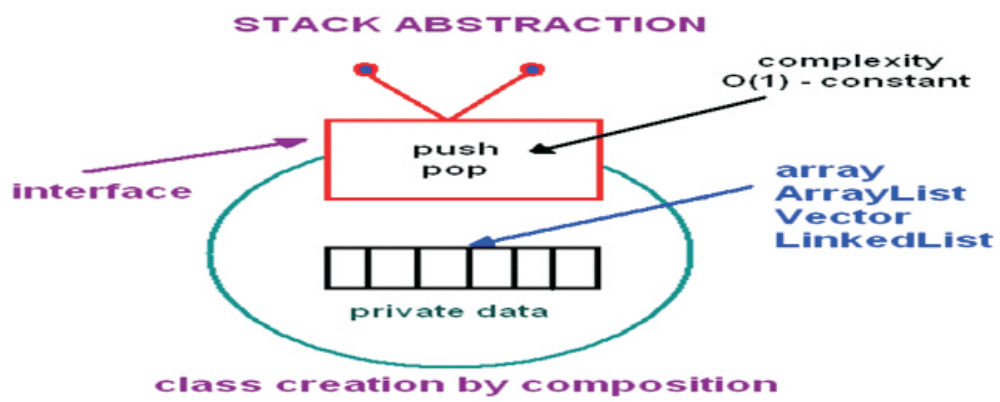
A stack is restricted data structure, because only a small number of operations are performed on it. The nature of the pop & push operation also means that stack element have a natural order. Element are removed from the stack in the reverse order to the order of their addition. Therefore the lower element is those that have been on the stack the longest.

IMPLEMENTATION

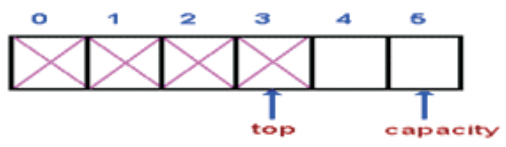
The data type stack is an adapter class in the standard library of classes, i.e. the stack is built on the top of other data structure. The underlying structure for a stack could an array, vector, an ArrayList, a linked list, or any other collection. Regardless of the type of the underlying data structure, a stack must implement the same functionality. This is achieved by providing a unique interface.

```
Public interface stackInterface<AnyType>
{ Public void push(AnyType e)
Public AnyTypepop();
Public AnyTypepeek();
Public Boolean isEmpty();
   }
```

The following picture demonstrates the idea of implementation by composition:



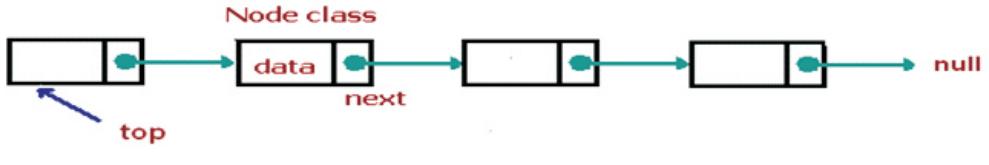
Array based implementation



In an array-based implementation maintain the following field: Default size of array A(≥ 1), the variable top that refers to the top element in the stack and the capacity that refers to the array size. The variable top changes from -1 to capacity -1. We say that a stack is empty when top = -1, and the stack is full when top = capacity - 1.

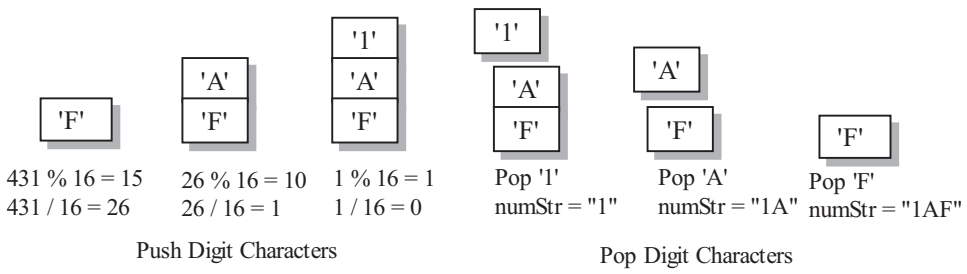
In a fixed-size stack abstraction, the capacity stays unchanged; therefore when top reaches capacity, the stack object throws an exception. In a dynamic stack abstraction when top reaches capacity, we double up the stack size.

Linked list-based implementation



Linked list-based implementation provides the best (from the efficiency point of view) dynamic stack implementation.

Using a stack to create Hexadecimal number



Example:c++ run time stack

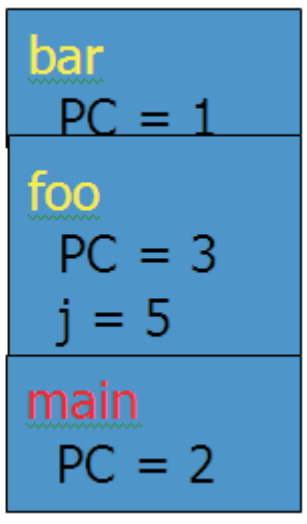
a.The c++ run time system keeps track of the chain of active function with stacks.
b.When a function is called, the run-time system pushes on the stack a frame containing

a. Local variable and return value.
b. Program counter, keeping track of the statement being executed.
c.When a function returns, its frame is popped from the stack and control is passed to the methods on top of the stack.

```
main() {
        inti = 5; foo(i);}

foo(int j) {
        int k ;k = j+1;
        bar(k);}

bar(int m) {
        …
        }
```



Decimal to Binary Conversion:

```
// Precondition: n >= 0.
 // Postcondition: The binary equivalent of n has been
 //  printed. The worstTime(n) is O(log n).
voidwriteBinary (int n)
                {   if (n == 0 || n == 1)
```

```
cout<< n;
else
                { writeBinary (n / 2);
cout<< n % 2;
                }// else
             }// writeBinary
```

HERE IS A STACK-BASED VERSION:

```
voidwriteBinary (int n)
 {          stack<int>myStack;
            myStack.push (n);

while (n > 1)
            {           n = n / 2;
                        myStack.push (n);
             }// pushing
            while (!myStack.empty())
            {           n = myStack.top();
                        myStack.pop();
                        cout<< (n % 2);
            }// popping
            cout<<endl<<endl;
}// method writeBinary
```

Stack application:

Direct application:

Page-visited history in a web browser
Undo sequence in a text editor
Savinglocal variables when one function calls another and this one calls another, and so on.

Indirect application:

Auxiliary data structure for algorithms
Component of other data structure.

REFERENCE:

1.http://www.cprogramming.com/tutorial/computerscience/stack.htmlcprogramming.com
2.Edition. MIT Press and McGraw-Hill, 2001
3.Data Structures Thro. C++ Tutor by YashavantKanetkar
4.S.B. Kishor Data Structures, Edition 3. Das GanuPrakashan, Nagpur, 2008.