Monograph on Stack

Pradeep chauhan M.I.E.T, Meerut, U.P

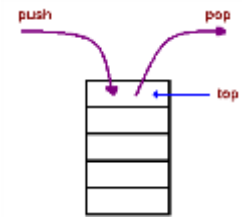
History of Stack

The stack was first proposed in 1946, in the computer design of Alan M. Turing (who used the terms "bury" and "unbury") as a means of calling and returning from subroutines. The Germans Klaus Samelson and Friedrich L. Bauer of Technical University Munich proposed the idea in 1955 and filed a patent in1957. The same concept was developed, independently, by the Australian Charles Leonard Hamblin in the first half of 1957.

Stack Data Structure

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last in First Out) or FILO (First in Last Out).

In the stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top. A pile of books, a stack of dinner plates, a box of Pringles potato chips can all be thought of examples of stacks. You can remove only the top book also you can add a new book on the top. The basic operating principle is that last item you put in is first item you can take out



Stack As an Abstract Data Type

This Abstract Data Type definition of a stack **does not define how a stack is implemented** in a computer program, it **only defines the operations on a stack**.

The ADT is made of with primitive data types, but operation logics are hidden.

Here we will see the stack ADT. These are few operations or functions of the Stack ADT.

- isFull(), This is used to check whether stack is full or not
- isEmpry(), This is used to check whether stack is empty or not
- push(x), This is used to push x into the stack
- pop(), This is used to delete one element from top of the stack
- peek(), This is used to get the top most element of the stack
- size(), this function is used to get number of elements present into the stack

Software stacks

Implementation

In most high level languages, a stack can be easily implemented either through an array or a linked list. What Identifies the data structure as a stack in either case is not the implementation but the interface: the user is only Allowed to pop or push items onto the array or linked list, with few other helper operations. The following will Demonstrate both implementations, using C.

Array implementation of stack

```
void push(int item)
{
        if (top==n-1)
        printf("Overflow");
        else
        {
                top= top+1.
                stack[top]=item.;
        }
}
void pop( )
{
        if (top = =-1)
        printf("underflow");
        else
        {
                printf("%d is deleted",stack[top]);
                top= top-1.
        }
}
void display( )
{
    int i;
     if (top = =-1)
     printf("Stack  Empty");
     else
     {
     for(i=top;i>=0;i--)
     printf("%d\t",stack[i]);
     }
}
```

Linked list implementation of stack

```
     struct node
     {
     int info;
     struct node *next;
     };
     struct node *top=NULL;
     void push()
     {
```

```
        struct node *n;
        n=(struct node *)malloc(sizeof(struct node));
        if(n==NULL)
                printf("\n over flow\n");
        else
                {
                        printf("\n enter value of node");
                        scanf("%d",&n->info);
                        n->next=top;
                        top=n;
                    }
}
void pop()
{
        struct node *ptr;
        if(top==NULL)
         printf("under flow");
        else
        {
             ptr=top;
             printf("%d is deleted\n",ptr->info);
             top=top->next;
             free(ptr);
        }
}
void display()
{
        struct node *ptr;
        ptr=top;
        if(top==NULL)
        printf("stack is empty");
        else
        {
                while(ptr!=NULL)
                {
                        printf("%d\t",ptr->info);
                        ptr=ptr->next;
                }
        }
}
```

Hardware stacks

A common use of stacks at the architecture level is as a means of allocating and accessing memory.

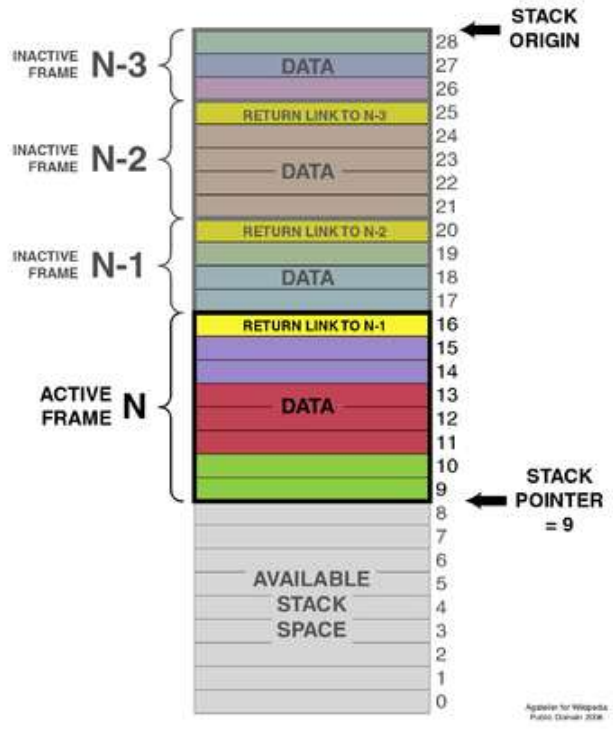
Basic architecture of a stack

A typical stack is an area of computer memory with a fixed origin and a variable size. Initially the size of the stack is zero. A stack pointer, usually in the form of a hardware register, points to the most recently referenced location on the stack; when the stack has a size of zero, the stack pointer points to the origin of the stack.

The two operations applicable to all stacks are:

• a push operation, in which a data item is placed at the location pointed to by the stack pointer, and the address in the stack pointer is adjusted by the size of the data item.

• a pop or pull operation: a data item at the current location pointed to by the stack pointer is removed, and the stack pointer is adjusted by the size of the data item.



A typical stack, storing local data and call information for nested procedure calls (not necessarily nested procedures!). This stack grows downward from its origin. The stack pointer points to the current topmost datum on the stack. A push operation decrements the pointer and copies the data to the stack; a pop operation copies data from the stack and then increments the pointer. Each procedure called in the program stores procedure return information (in yellow) and local data (in other colors) by pushing them onto the stack. This type of stack implementation is extremely common, but it is vulnerable to buffer overflow attacks (see the text).

Stacks and programming languages

Some languages, like LISP and Python, do not call for stack implementations, since push and pop functions are available for any list. All Forth-like languages (such as Adobe PostScript) are also designed around language defined stacks that are directly visible to and manipulated by the programmer.

C++'s Standard Template Library provides a "stack" template class which is restricted to only push/pop operations. Java's library contains a Stack class that is a specialization of Vector this could be considered a design flaw, since the inherited get() method from Vector ignores the LIFO constraint of the Stack. PHP has an SplStack class.

Application of Stacks

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.

Backtracking Procedure

Backtracking is one of the algorithms 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 and return process

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

The assignment of memory takes place in contiguous memory blocks. We call this stack memory allocation because the assignment takes place in the function call stack. The size of the memory to be allocated is known to the compiler. When a function is called, its variables get memory allocated on the stack. When the function call is completed, the memory for the variables is released. All this happens with the help of some predefined routines in the compiler. The user does not have to worry about memory allocation and release of stack variables.

Other applications

Stack is also used in web browser history, undo operation in text editors and Tower of Hanoi problem.

References

[1]. https://www.researchgate.net/publication/259397239_Lecture_Notes_-_Algorithms_and_Data_Structures_-_Part_1_Introduction
[2]. http:// www.faceprep.in/data-structures/stack-applications-in-data-structure/
[3]. https://www.tutorialspoint.com/applications-of-stack-in-data-structure
[4]. https://www.cse.unr.edu/~sushil/class/cs202