Data Structure using C++

Lecture 04

Reading Material

Data Structures and algorithm analysis in C++ Chapter. 3 3.1, 3.2, 3.2.1

Summary

- Stack
- Operations on a stack
- Representing stacks
- Converting an expression from index to postfix
- Infix, Postfix , and prefix Expressions

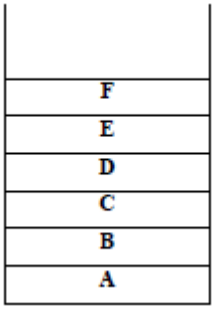
م.م اثير هادي عيسى الرماحي

Stack

A stack is an ordered collection of items into which new items may be inserted and from which items may be deleted at one end, called the **top of the stack**.

The definition of the stack provides for the insertion and deletion of items, of items so that a stack is a dynamic constantly changing object. The definition specifies that a single and of the stack is designated as the stack top. New items may be put on top of the stack, or items which are at the top of the stack may be removed.

In the figure below, F is physically higher on the page then all the other items in the stack, so F is the current top element of the stack, If any new items are added to the stack they are placed on top of F, and if any items are deleted, F is the first to be deleted. Therefore, a stack, is collected a last – in – first – out. **(LIFO)** list.



Operations on a stack

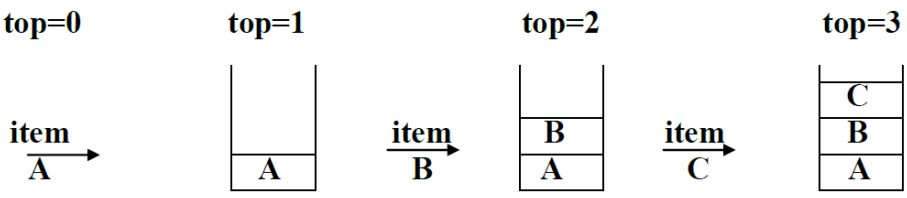
The two main operations which can be applied to a stack are given spatial names, when an item is added to a stack, it is pushed onto the stack, and when an item is removed, it is poped from the stack.

Given a stack s, and an item I, performing the operation push (s,i) adds the item I to the top of stack s. similarity, the operation pop(s) removes the top element and returns ir as a function value. Thus the assignment operation I = pop(s); removes the element at the top below is a motion picture of a stack s as it expands (items pushed) and shrinks (item poped) with the passage of item.

هناك عمليتين اساسيتين لادارة المكدس ((الداخل اولا يخرج آخرا)) **LIFO** :-

- عملية الدفع (**Push**) :

يزداد المؤشر بمقدار واحد على شرط كونه اقل من حجم المكدس.



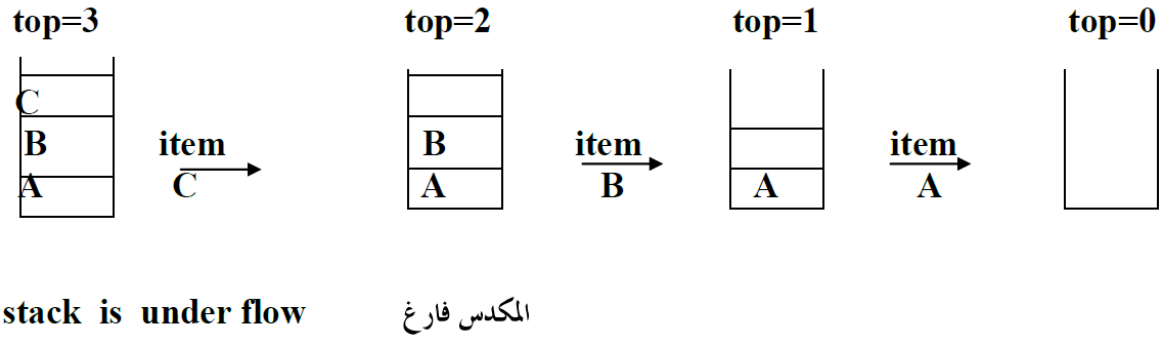
size= 3

stack is over flow المكدس ممتلئ

م.م اثير هادي عيسى الرماحي

– عملية السحب (Pop) :

يتناقص المؤشر بمقدار واحد على شرط كون المكدس غير فارغ.



stack is under flow المكدس فارغ

The original idea of the stack is that there is no upper limit on the number of items that may be kept in a stack. Pushing another item onto a stack produces a larger collection of items. When the stack been implemented in a program, it can be represented as an array, therefore we need to give the maximum size of this stack to be shrieked. So, we need as operation called Full (S) (which determines whether or not a stack is full (overflow) to be applied before push operation. On the other hand, if a stack contains a single item and the stack is poped, the resulting stack contains no item and is called an empty stack.

Therefore, before applying the pop operator to a stack, we must ensure that a stack is not empty. The operation empty (s) determine whether or not a stack is empty.

Another operation that can be performed on a stack is to determine what the top item on a stack is without removing it. This operation is written stack top (s) and return the top element of stack s.

I = stacktop (s)

Representing stacks:

Before programming a problem solution uses a stack, we must decide how to represent a stack using the data structures that exit in our programming language (C or C++) . the simplest method to represent a stack is to use an array to be home of the stack. During the program execution, the stack can grow and shrink within the space reserved for it. One end of the array is the fixed bottom of the stack, while the top of the stack, Thus another field is needed that, at each point during program execution, keeps tracks of the current position of the top of the stack.

The stack may therefore be declared as a structure containing two objects: an array to hold the elements of the stack, and an integer to indicate the position of the current stack top within the array.

```
# define stacksize 100
Struct stack {
    int top ;

    int item [stacksize] ;
};
```


م.م اثير هادي عيسى الرماحي

The empty stack contain s no elements and can therefore, be indicated by top equaling -1 .
To initialize a stack s to the empty stack, we may execute.

```
S top = -1;
```

The function empty(s) that is used to test whether a stack is empty, may be written as follow:

```
int empty (struct stack *st)
   {
    If (st →top = = -1)
    Return (TRUE);
    Else
    Return (FALSE)
   }
```

The function full (s) that is used to test whether a stack is full, may be written as fellow:

```
int full (struct stack *ST)
   {
    If (st → top = = maximize -1)
    Return (TRUE);
    Else
    return (FALSE);
  }
```

The function pop that is used to remove the element in the top of the stack, must perform the following three actions:

1- If the stack is empty, print a warning message and halt execution.
2- Remove the top element from the stack.
3- Return this element to the calling program.

```
int pop (srtuct stack *ST)
   {
    If (empty (ST)) {
    Cout << "stack underflow" ;
    Exit (q) ;
    }
     Return (ST → item [st → top P-] ;
   }
```

The function push is used to add a new element to the top of the stack. This function must perform the following operations:

1- If the stack if full, print a warning message and halt execution.
2- Add a new element into the top of the stack.

```
Void push (struct stack *ST, int x)
      {
       If (full (S+)) {
       Cout << "stack is overflow";
        Exit (1);
```


م.م اثير هادي عيسى الرماحي

```
}
Else
ST → item [+ + (ST →top)] = x;
Return ;
}
```

The function stacktop (), which returns the top element of a stack without removing it from the stack, may be written as follows:

```
int stacktop (struct stack *ST)
{
If (empty (ST)) {
Cout << "stack is underflow";
Exit (1);
}
else
return (ST → item {ST → top});
}
```

Convert the following infix expression to postfix using stack:

A + (B /C) #

ch	opstack	postfix	commentary
	#		Push # to opstack read ch
A	# #	A A	Add ch to postfix read ch
+	# #+	A A	Push ch to opstack read ch
(	#+ #+(	A A	Push # to opstack read ch
B	#+(#+(	A A B	Push # to opstack read ch
/	#+(#+(	A B A B	Push # to opstack read ch
C	#+(/ #+(/	A B A B C	Add ch to postfix read ch
)	#+(/ #+	A B C A B C /	Pop and add to postfix until (is reached. Read ch
#	#+	A B C / A B C / + #	Pop and add to postfix until the opstack is empty.

Example

Convert the following infix expression to postfix using stack:

((A – (B + C)) * D) / (E +F) #

ch	opstack	postfix	commentary
	#		Push # to opstack read ch
(	# # (		Push # to opstack read ch
(	#((#((		Push ch to opstack read ch
A	#((#((	 A	Add ch to postfix read ch
–	#((#((-	A A	Push # to opstack read ch
(	#((- #((- (	A A	Push # to opstack read ch
B	#((- (#((- (#((- (	A A A B	Add ch to postfix read ch
+	#((- (#((- (+	A B A B	Push # to opstack read ch
C	#((- (+ #((- (+	A B A B C	Add ch to postfix read ch
)	#((- (+ #((-	A B C A B C +	Pop and add to postfix until (is reached
)	#((- #(	A B C + A B C + -	Pop and add to postfix until (is reached
*	#(	A B C + -	Push # to opstack
	#(*	A B C + - D	read ch
)	#(* #	A B C + - D A B C + - D *	Pop and add to postfix until (is reached
/	# # /	A B C + - D * A B C + - D *	read ch Push # to opstack
(	# / # / (	A B C + - D * A B C + - D *	Push # to opstack read ch
E	# / (# / (	A B C + - D * A B C + - D * E	Add ch to postfix read ch
E	# / (# / (	A B C + - D * E A B C + - D * E	Add ch to postfix read ch
+	# / (# / (+	A B C + - D * E A B C + - D * E	read ch Push # to opstack
F	# / (+ # / (+	A B C + - D * E A B C + - D * E F	read ch Add ch to postfix
)	# / (+ # /	A B C + - D * E A B C + - D * E F +	read ch Pop and add to postfix until (is reached
#	# /	A B C + - D * E F + A B C + - D * E F + #	read ch Pop and add to postfix until (is reached

Infix, Postfix , and prefix Expressions:

Consider the sum of A and B. we think of applying the operator t" to the operands A and B and write the Sum as A+B . This particular representation is called infix . There are two alternate notations for expressing the Sam of A and Busing the symbols A, B and t . These are

T A B prefix

A B + postfix

In prefix notation the operator precedes the two operands , in postfix notation the operator follows the two operands , and in infix notation the operand. Is between the two operands. The evaluation of the expression A+B*c, as written in standard infix notation , require Knowledge of which of the two operations + or * is to be performed first. In the case of +

and * we Know that mutilation is to be done before addition . Thus A+B*c is interpreter As a (B*C) .

Foe the Same example , if we want need to write the expression as (A+B)*C .Therefore , Infix notation may require parentheses to specify a decimal or of operations.

Using postfix and prefix notation , the notation, the need for parentheses is eliminated because the operator is placed directly after (before) the two operands to which it applied.

To convent infix expression to postfix forms ,we use the following algorithm :

1- Completely parenthesize the infix expression .
2- Move each operator to the space hold by its corresponding right parenthesis.
3- Remove all parentheses.

We can apply this algorithm to the expression :

A / B$C + D*E – A*C
(((A / (B $ C)) + (D*E)) – (A * C))
ABC$ / DE*+AC*_

The conversion algorithm for infix to prefix specify that , after compactly parenthesizing the infix expression with order of priority , we move each operator to its corresponding left parent . And after that , delimiting all parentheses

For example :-

(((A / (B $ C)) + (D * E)) – (A*C))
(((A / (B $ C)) + (D * E)) – (A*C))
- + / A$BC *DE*AC

In the above example , we have consider five binary operations : addition , subtraction , , multiplication division, and exponentiation . The first four are available in C and C+ + and are dented by the usual operators + ,- , * and / > The fifth , expunction is demotion by $. The value of the expression A $ B is a raised to the B power , So that 3$ 2 is of For these binary operators the following is the order of precedence (highest to lowest) :

Exponentiation
Multiplication / division
Addition / Subtraction

When un parenthesize operators of the Same precedence are scanned , the order is assume to be left to right except in the case of the exponentiation , where the order is assumed to be from right to left . Thus A+B+C means (A+B) +C, whereas A $ B $C means A $ (B$C) .

Evaluating a postfix Expression

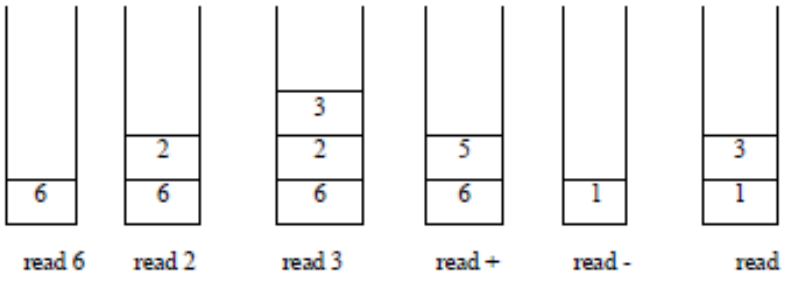
Stacks con be used to evaluate the different expression notation in sedition to converting from one expression notation to another as we will see later. Here, we consider evaluating postfix expression using stack .

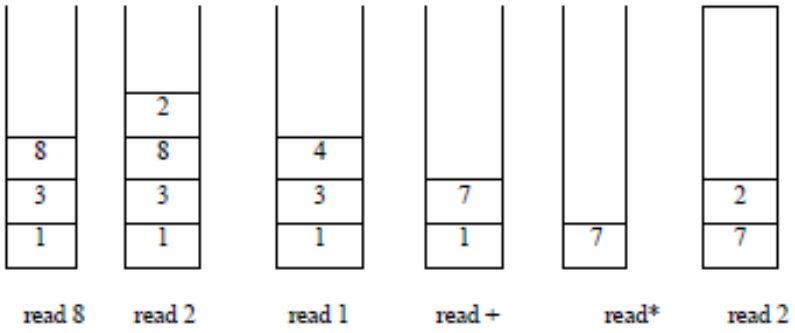
As an example , consider the postfix expansion

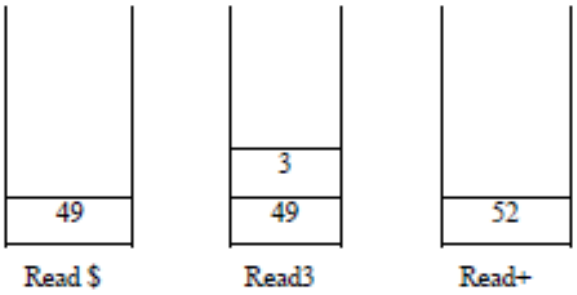
623 + - 382 / + * 2 $ 3 +

To evaluate such expression, we reputedly read characters from the postfix expression. If the character read is can operand , push the value associational with it onto the stack . If it is an operator , pop two values from the stack, copy the operator to them, and push the result back onto the stack.

م.م اثير هادي عيسى الرماحي







م.م اثير هادي عيسى الرماحي