المكدس

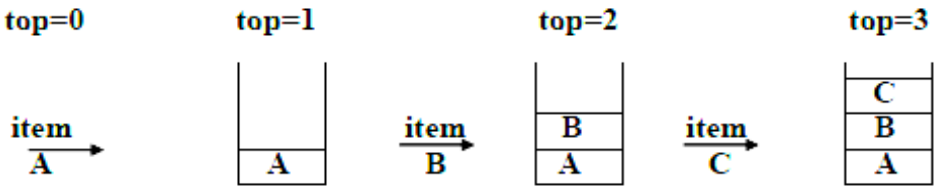
Stack

هو هيكل بياني من نوع هياكل البيانات الخطية Linear structures يحتوي نهاية واحدة يتم من خلالها اجراء عمليات الاضافة و الحذف وتسمى عملية اضافة عنصر الى المكدس stack بـ (push) وتسمى عملية حذف عنصر من المكدس بـ (pop) ، و للمكدس مؤشر يسمى (top) حيث ان هذا المؤشر يشير الى اخر عنصر دخل الـ stack او يشير الى اول عنصر سوف يخرج او يحذف من الـ stack ،وعندما يكون الـ stack فارغ فان قيمة المؤشر تساوي الى الصفر ، ويتمتع الـstack بخاصية او مبدأ (Last in first out) (اخر من يدخل اول من يخرج) (LIFO) ويمكن تمثيل الـ Stack في مصفوفة احادية.

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

- عملية الدفع (**Push**) :

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

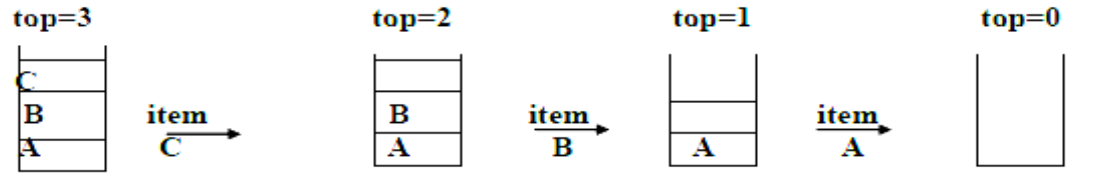


size= 3

stack is over flow المكدس ممتلئ

– عملية السحب (Pop) :

يتناقص المؤشر بمقدار واحد على شرط كون المكدس غير فارغ.



stack is under flow المكدس فارغ

Program of stack

```
# include <iostream .h>
# defined   max 3
        int stack [ max];
        int top = 0;

Void push (int a)
{
        if (top == 3) Cout <<" stack is full";
        else
        {
                top ++ ;
                Stack [top] =a;
        }
}
```

```
Void pop ( )
{
int x ;
        if (top == 0) cout <<" stack is empty";
        else
        {
                x = stack [top];
                top -- ;
                Cout <<" deleted" << x;
        }
}


Main ( )
        Push (1);
        Push (2);
        Push (3);
        Push (4);
        Pop () ;
        Pop () ;
        Pop () ;
        Pop () ;
}
```

اهم تطبيقات الـ stack :

1. معالجة البرامج التي تحتوي على برامج فرعية (Functions) يستخدم الـ stack من قبل المترجمات (Compilers) في معالجة البرامج التي تحتوي على برامج فرعية (Functions) ، فعند استدعاء برنامج فرعي داخل البرنامج الرئيسي ، فان ذلك يتطلب خزن عنوان الايعاز التالي بعد ايعاز الاستدعاء لكي يستطيع البرنامج الرئيسي تنفيذ البرنامج الفرعي والعودة بشكل صحيح الى موقع الايعاز التالي .
2. يستخدم الـ stack كهيكل لخزن البيانات التي تحتاج استرجاعها بصورة معكوسة (بترتيب معكوس).
3. استخدام الـ stack في معالجة التعابير الحسابية .

Arithmetic expressions

التعابير الحسابية

من المعروف ان التعابير الحسابية تكتب بثلاث صيغ:

1- infix (Operand Operator Operand)
صيغة infix notation: حيث تكون اشارة العملية الحسابية تتوسط العوامل وهذا هي الصيغة الاعتيادية 3+4، A-B ، x/20

1- prefix (Operator Operand Operand)
صيغة prefix notation :حيث تكون اشارة العملية الحسابية تسبق العوامل اي تكون الاشارة من جهة اليسار مثل : + 34 , -AB , /x20

2- postfix (Operand Operand Operator)
صيغة postfix notation:اذ تكون اشارة العملية الحسابية تلحق العوامل اي تكون الاشارة من جهة اليمين مثل : 34 + , AB- , x20/

Infix = A+B
Prefix = +AB
Postfix = AB+

* ملاحظة : لتنفيذ تعبير حسابي مكتوب بصيغة (infix) فان العمليات تنفذ من جهة اليسار الى جهة اليمين وحسب اعلى اسبقية للعملية الحسابية وهي :

Operator	precedence
++ or -- قبل المتغير	1
–	2
* or /	3
+ or –	4
=	5
++ or -- بعد المتغير	6

Ex: convert the following expression to prefix & postfix?

Ans.

a + b*c /d + e 1= b*c
  1
a +1/d +e 2=1/d
  2
a +2 + e 3= a +2
 3
3+e
 4

Prefix	**Postfix**
3+e	3+e
+3e	3e+
+a+2e	a +2e+
++a2e	a2+e+
++a1/de	a 1/d +e+
++a/b*c de	a 1d /+e+
++a/* b c d e	a b*c d /+e+
	abc*d /+e+

Ex: if postfix expression is ABS /D *- she the phases of execute this expression and find finish value when A=5, B=4 , C=2 and D=2 ?

Sol:

Postfix	**current operator**	**current operant**
ABC/D*+	/	T1=B/C
AT1D*+	*	T2= T1*D
AT2 +	+	T3=A+T2
T3		

T3=A+T2
=A+T1*D
=A+B/C*D
=5+4/2*2
=9

- ملاحظة :- يجب ان يكون المعاملات (operates) في كل تعبير حسابي اكثر من عدد العمليات الحسابية (operaters) بمقدار واحد ،فاذا اعطينا لكل معامل قيمة (1+) ولكل عملية حسابية قيمة (1-) ، فيجب ان تكون الدرجة النهائية (rank) لاي تعبير حسابي مساوي الى (1).

Ex: find the Rank of the following expression and Shaw is expression valid or invalid?

Infix	**postfix**	**rank**	**case**
1- a+*b	a b*+	0	invalid
2- a+ b*c	a b c*-	1	valid
3- a +b/d-e	a b d/+e-	1	valid
4- a b +c	a b c+	2	invalid

- لتحويل التعبير الحسابي من infix الى postfix باستخدام الـ stack تحتاج الى جدول الاسبقيات والذي يكون كما يلي :

Symbol	**precedence**	**rank**
+, -	1	-1
*, /	2	-1
A, B, C,…	3	+1
$	0	

Algorithm of convert infix to postfix

Let char s[n] is stack.
Top is pointer of stack.
Next char (infix) is a function to input infix.
Pop () is a function to delete element from a stack.
Push (next) is a function to insert element to a stack.
Next is a string
Step 1: [initial stack]
Set top 1, s [top] $
Step 2: [initial output string and rank]
Set rank 0, postfix " "
Step 3: [get first input symbol]
Next next char (infix)
Step 4: [scan the infix expression]
Do step 5, 6
While next = $
Step 5: [remove symbol with greater or equal precedence from the stack]
While prec (next) < prec (s [Top]) Temp pop ()
Postfix postfix +temp
Rank rank + r (Temp)
If rank < 1 then cout <<" invalid "and exit.
Step 6: [push next on stack and get next input symbol]
Push (next)
Set next next char (infix)
Step 7: [remove remaining elements from stack]
While s [top]! = '$'
Set temp pop ( )
Postfix postfix + Temp
Rank rank + r (Temp)
If rank < 1 then cout <<" invalid" and exit.
Step 8: [is expression valid]
If rank =1 then cout << postfix else cout <<"invalid" and exit.

Ex: convert the following expression from infix to postfix using stack. $ A+B/ C* D – E$

Next

Infix	**stack**	**postfix**	**rank**
-	$	-	0
A	$A	-	0
+	$+	A	1
B	$+B	A	1
/	$+/	AB	2
C	$+/C	AB	2
*	$+*	ABC	2
D	$+*D	ABC/	2
-	$-	ABC/D*+	1
E	$-E	ABC/D*+	1
$	$	ABC/D*+E-	1 valid

خوارزمية تحويل التعابير الحسابية التي تحتوي على اقواس من صيغة الـ infix الى صيغة الـ postfix .

لغرض تحويل التعابير الحسابية التي تحتوي على اقواس فان بعض التغيرات سوف تجرى على الخوارزمية السابقة وهذاه التغيرارت هي :

1- عندما يكون next = ' (' فان القوس سوف يدخل الى الـ stack بدون اي مقارنة .
2- اذا جاء متغيير اسبقيته مساوية لاسبقية العنصر الموجود في الـ stack فان ه سوف يدخل الى الـ stack .
3- اذا جاء القوس المغلق next = ') ' فانه يودي الي خروج جميع العناصر الموجودة في الـ stack والتي بعد القوس المفتوح .ويلغي القوس المفتوح.

Ex: convert the expression ((A+B/ C –D)/E from infix hotation to postfix notation. using stack?

Infix	**stack**	**postfix**
-	$	-
(	$(	-
(	$((	-
A	$((A	-
+	$((+	A
B	$((+ B	A
)	$(	AB+
/	$(/	AB+
C	$(/C	AB+
-	$(-	AB+C/
D	$(-D	AB+C/
)	$	AB+C/D-
/	$/	AB+C/D-
E	$/E	AB+C/D-E/
$		