

ARRAY <i>pila_array.ccp</i>	container STACK <i>pila_stack.ccp</i>
<pre>#include <iostream> #include <fstream> using namespace std;</pre>	<pre>#include <iostream> #include <fstream> using namespace std;</pre>
<pre>#define MAX 10 int Testa=MAX; char ESPR [MAX]; <i>//-- dichiarazione array ESPR</i></pre>	<pre>#include <stack> stack <char> ESPR; <i>//-- dichiarazione stack ESPR</i></pre>
<pre>ifstream finp ("parentin.txt");</pre>	<pre>ifstream finp ("parentin.txt");</pre>
<pre>void scriviPila (char x) { cout<<"ESPR="; <i>//-- stampa tutti gli elementi nella Pila</i> for(int k = MAX-1; k >= Testa; k--) cout << ESPR [k]; <i>//-- elemento nella Pila</i> if (x == 'p') cout<<"\n\n errore: Pila piena"; }</pre>	<pre>void scriviPila () { <i>//-- stampa:</i> cout << " " << ESPR.size(); <i>//-- numero elementi pila</i> cout << '\t' << ESPR.top (); <i>//-- ultimo elemento</i> <i>//-- si può accedere solo all'ultimo elemento inserito nella Pila</i> }</pre>
<pre>int Push (char let) { if (Testa == 0) { <i>//-- errore: array ormai pieno</i> scriviPila ('p'); return -1; }</pre>	<pre>int Push (char let) { ESPR.push (let); <i>//-- inserimento nella Pila</i> return 0; }</pre>
<pre>Testa--; ESPR [Testa] = let; <i>//-- inserimento nella Pila</i> return 0; }</pre>	<pre>int Pop (char& LTesta) { if (ESPR.empty()) { <i>//-- errore: Pila vuota</i> cout<<"\n\n errore: Pila vuota"; return -1; }</pre>
<pre>int Pop (char& LTesta) { if (Testa == MAX) { <i>//-- errore: Pila vuota</i> cout<<"\n\n errore: Pila vuota"; return -1; }</pre>	<pre> cout<<"\n\n errore: Pila vuota"; return -1; }</pre>
<pre>scriviPila ('n'); <i>//-- OK, Pila non vuota</i> LTesta = ESPR[Testa]; <i>//-- salvato in LTesta ultimo elem.</i> Testa++; <i>//-- aggiornam. indice Testa a penult.elem.</i> return 0; }</pre>	<pre> scriviPila (); <i>//-- OK, Pila non vuota</i> LTesta = ESPR.top(); <i>//-- salvato in LTesta ultimo elem.</i> ESPR.pop(); <i>//-- rimosso ultimo elem. dalla Pila</i> return 0; }</pre>
<pre>return 0; }</pre>	<pre>return 0; }</pre>

```

int main(){
    char LET, letPila;    //-- LET parentesi letta da file
    int N, esito=0;      //-- letPila prelevata da array ESPR

    finp >> N;
    for (int i=0; i<N && esito==0; i++){
        finp >> LET;
        cout<<"\nelab. LET " << LET <<'\t';
        if (LET == '{' || LET == '['
            || LET == '(' || LET == '<') //-- parentesi aperta
            esito = Push(LET);        //-- inserimento in Pila
        else {                        //-- parentesi chiusa
            esito = Pop (letPila);    //-- prelievo da Pila
            if (esito == 0)          //-- e controllo
                if (LET == '}' && letPila == '{'
                    || LET == ']' && letPila == '['
                    || LET == ')' && letPila == '('
                    || LET == '>' && letPila == '<')
                    esito = 0;
                else
                    esito = -1;
        }
    }
    if ( esito < 0)
        cout << "\n\n sequenza errata";
    else
        if (Testa < MAX ) {
            cout << "\n\n sequenza errata, Pila piena\n\n\t\t";
            scriviPila ('e');
        }
        else
            cout << "\n\n sequenza corretta";
    cout << endl;
    finp.close();    } //-- FINE main

```

```

int main(){
    char LET, letPila;
    int N, esito=0;
    cout << "\nparentesi in Pila: num \t top";
    finp >> N;
    for (int i=0; i<N && esito==0; i++){
        finp >> LET;
        cout<<"\nelab. LET " << LET <<'\t';
        if (LET == '{' || LET == '['
            || LET == '(' || LET == '<')
            esito = Push(LET);
        else {
            esito = Pop (letPila);
            if (esito == 0)
                if (LET == '}' && letPila == '{'
                    || LET == ']' && letPila == '['
                    || LET == ')' && letPila == '('
                    || LET == '>' && letPila == '<')
                    esito = 0;
                else
                    esito = -1;
        }
    }
    if ( esito < 0)
        cout << "\n\n sequenza errata";
    else
        if ( ! ESPR.empty() ) { //-- errore: Pila NON vuota
            cout << "\n\n sequenza errata, Pila piena\n\n\t\t";
            scriviPila ();
        }
        else
            cout << "\n\n sequenza corretta";
    cout << endl;
    finp.close();    } //-- FINE main

```

Esempio di gestione di una PILA mediante **Array** oppure **container Stack**

p.3/4

casi di Test

prof.ssa P.Grandillo

OUTPUT con <i>pila_array.exe</i>	OUTPUT con <i>pila_stack.exe</i>
----------------------------------	----------------------------------

caso 1 sequenza errata	4)[(
---	----------

```

C:\> Prompt dei comandi
>>pila_array.exe

elab. LET )

errore: Pila vuota

sequenza errata

>>_
    
```

```

C:\> Prompt dei comandi
>>pila_stack.exe

parentesi in Pila: num top
elab. LET )

errore: Pila vuota

sequenza errata

>>_
    
```

caso 2 - sequenza corretta ma pila piena (non gestibile con array di soli 10 elementi)	26 {[(<{[(<{[(<()>)]}>)]}>)]}
--	----------------------------------

```

C:\> Prompt dei comandi
>>pila_array.exe

elab. LET {
elab. LET [
elab. LET (
elab. LET <
elab. LET {
elab. LET [
elab. LET (
elab. LET <
elab. LET {
elab. LET [
elab. LET (      ESPR={[((<{[((<{[

errore: Pila piena

sequenza errata

>>_
    
```

```

C:\> Prompt dei comandi
>>pila_stack.exe

parentesi in Pila: num top
elab. LET {
elab. LET [
elab. LET (
elab. LET <
elab. LET {
elab. LET [
elab. LET (
elab. LET <
elab. LET {
elab. LET [
elab. LET (
elab. LET <
elab. LET (
elab. LET )      13 (
elab. LET >      12 <
elab. LET )      11 (
elab. LET ]      10 [
elab. LET }      9 {
elab. LET >      8 <
elab. LET )      7 (
elab. LET ]      6 [
elab. LET }      5 {
elab. LET >      4 <
elab. LET )      3 (
elab. LET ]      2 [
elab. LET }      1 {

sequenza corretta

>>_
    
```

NOTA:
 con il container **stack** non è possibile accedere alle informazioni contenute nella Pila, ma solo all'ultimo elemento inserito;
 l'accesso si effettua con il **metodo top()**
 esempio: **ESPR.top()**

per approfondimenti consultare :
<http://www.cplusplus.com/reference/stack/stack/>

OUTPUT con <i>pila_array.exe</i>	OUTPUT con <i>pila_stack.exe</i>
caso 3 10 sequenza corretta (){}[()]} 	
<pre> Prompt dei coma... - □ >>pila_array.exe elab. LET (elab. LET) ESPR=(elab. LET { elab. LET (elab. LET) ESPR={({ elab. LET [elab. LET (elab. LET) ESPR={[(elab. LET] ESPR={[elab. LET } ESPR={ sequenza corretta >>_ </pre>	<pre> Prompt dei coma... - □ >>pila_stack.exe parentesi in Pila: num top elab. LET (elab. LET) 1 (elab. LET { elab. LET (elab. LET) 2 (elab. LET [elab. LET (elab. LET) 3 (elab. LET] 2 [elab. LET } 1 { sequenza corretta >>_ </pre>
caso 4 4 sequenza errata ([])	
<pre> Prompt dei coma... - □ >>pila_array.exe elab. LET (elab. LET [elab. LET) ESPR=(sequenza errata >>_ </pre>	<pre> Prompt dei coma... - □ >>pila_stack.exe parentesi in Pila: num top elab. LET (elab. LET [elab. LET) 2 [sequenza errata >>_ </pre>
caso 5 12 sequenza errata ([{()})](){}	
<pre> Prompt dei coma... - □ >>pila_array.exe elab. LET (elab. LET [elab. LET { elab. LET (elab. LET) ESPR=([{{(elab. LET (elab. LET) ESPR=([{{(elab. LET } ESPR=([{{ elab. LET] ESPR=([elab. LET (elab. LET) ESPR=((elab. LET { sequenza errata, Pila piena ESPR=({ </pre>	<pre> Prompt dei coma... - □ >>pila_stack.exe parentesi in Pila: num top elab. LET (elab. LET [elab. LET { elab. LET (elab. LET) 4 (elab. LET (elab. LET) 4 (elab. LET } 3 { elab. LET] 2 [elab. LET (elab. LET) 2 (elab. LET { sequenza errata, Pila piena 2 { </pre>