the Stack

1. The Stack Abstract Data Type
   - stack ADT
   - using the STL stack

2. An Application: Test Expressions
   - are parentheses balanced?
   - algorithm uses a stack

3. Stack Implementations
   - adapting the STL vector class
   - adapting the STL list class

MCS 360 Lecture 13
Introduction to Data Structures
Jan Verschelde, 27 September 2017
the Stack

1. The Stack Abstract Data Type
   - stack ADT
     - using the STL stack

2. An Application: Test Expressions
   - are parentheses balanced?
   - algorithm uses a stack

3. Stack Implementations
   - adapting the STL vector class
   - adapting the STL list class
A stack is a LIFO (Last In First Out) sequence: we can only get (or pop) the element on top.

Despite this restriction, many applications:

- parsing expressions,
- store information about function calls.

In the STL, a stack is an *adapter*: either a vector or a list implements a stack.
stack ADT

abstract <typename T> stack;
/* a stack is a sequence of elements, 
   stored as Last In First Out (LIFO) */

abstract bool empty ( stack s );
postcondition: empty(s)
   == true if s is empty,
   == false if s is not empty;

abstract T pop ( stack s );
precondition: not empty(s);
postcondition: push(s,pop(s)) == s;

abstract void push ( stack s, T e );
postcondition: push(s,e); pop(s) == e;
1. The Stack Abstract Data Type
   - stack ADT
   - using the STL stack

2. An Application: Test Expressions
   - are parentheses balanced?
   - algorithm uses a stack

3. Stack Implementations
   - adapting the STL vector class
   - adapting the STL list class
using the STL stack

```cpp
#include <iostream>
#include <stack>
using namespace std;

int main()
{
    stack<int> s;
    int e;

    do
    {
        cout << "give element (0 to stop) : ";
        cin >> e;
        if(e <= 0) break;
        s.push(e);
    }
    while(true);
}
```
while(!s.empty())
{
    int e = s.top();
    s.pop();
    cout << "popped " << e << endl;
}

Members of stack<T> class:

- T top() const : returns the top of the stack,
- void pop() : removes the top element.
the Stack

1. The Stack Abstract Data Type
   - stack ADT
   - using the STL stack

2. An Application: Test Expressions
   - are parentheses balanced?
   - algorithm uses a stack

3. Stack Implementations
   - adapting the STL vector class
   - adapting the STL list class
Given an expression \((w \times (x+y) / z - (p / (9 - 8)))\), test if every closing bracket \) matches an opener \(.

Simple counting algorithm:

- +1 if encounter opener \), and
- −1 if encounter closer \).

Expression is balanced if final count equals zero.

Harder if different type of brackets, braces, and parentheses can be used, e.g.:
\((w \times [x+y] / z - [p/\{9-8\}])\).
running the program

$ /tmp/match_brackets
give an expression: \((w*[x+y] / z - [p/{9 -8}])\)
checking "(w*[x+y] / z - [p/{9 -8}])" ...
pushed (  
pushed [  
] matches [  
popped [  
pushed [  
pushed {  
pushed {  
} matches {  
popped {  
] matches [  
popped [  
) matches (  
popped (  
parenthesis in "(w*[x+y] / z - [p/{9 -8}])" \  
are balanced
$
1 The Stack Abstract Data Type
   - stack ADT
   - using the STL stack

2 An Application: Test Expressions
   - are parentheses balanced?
   - algorithm uses a stack

3 Stack Implementations
   - adapting the STL vector class
   - adapting the STL list class
algorithm uses a stack

A stack stores all opening brackets:
- we push every opening bracket,
- for every matching closing bracket, we pop.

For every character $c$ in a given expression:
- if $c$ is '(', '{', or '[' then
  - push $c$ to a stack
- else if $c$ is ')', '}', or ']' then
  - if top of stack matches $c$ then
    - pop the stack
  - else
    - break out of loop: parenthesis unbalanced.
```cpp
data and variables

#include <iostream>
#include <string>
#include <stack>

using namespace std;

int main()
{
    string expression;

    cout << "give an expression : ";
    getline(cin,expression,'
');

    stack<char> brackets;
    string opening_brackets = "({["
    string closing_brackets = "]")";
```
recall the `find`

For any string `s` and character `c`:

```
s.find(c) either
  • returns `string::npos` if `c` does not occur in `s`,
  or
  • returns the first index `k` for which `s[k] == c`.
```

We use `find with c == expression[i]` on `s == opening_brackets`
or `s == closing_brackets`. 
bool balanced = true;

for(int i=0; i<expression.size(); i++)
    if(opening_brackets.find(expression[i])
        != string::npos)
        brackets.push(expression[i]);
    else if(closing_brackets.find(expression[i])
        != string::npos)
    {
        int k = closing_brackets.find(expression[i]);
        char c = brackets.top();
        if(c != opening_brackets[k])
            {balanced = false; break;}
    }
else
    brackets.pop();
1. The Stack Abstract Data Type
   - stack ADT
   - using the STL stack

2. An Application: Test Expressions
   - are parentheses balanced?
   - algorithm uses a stack

3. Stack Implementations
   - adapting the STL vector class
   - adapting the STL list class
A stack is said to be an *adapter* class: we adapt a sequential container, get a stack implementation providing another interface to vector or list.

A dictionary between STL stack and vector, for any item \( t \) of type \( T \):

<table>
<thead>
<tr>
<th>stack(&lt;T&gt;\ s)</th>
<th>vector(&lt;T&gt;\ v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s.\text{push}(t) )</td>
<td>( v.\text{push_back}(t) )</td>
</tr>
<tr>
<td>( \text{if(!s.\text{empty}())} )</td>
<td>( \text{if(!v.\text{empty}())} )</td>
</tr>
<tr>
<td>( t = s.\text{top()} )</td>
<td>( t = v[v.\text{size()-1}] )</td>
</tr>
<tr>
<td>( t = s.\text{top()} )</td>
<td>( t = v.\text{back()} )</td>
</tr>
<tr>
<td>( s.\text{pop()} )</td>
<td>( v.\text{pop_back()} )</td>
</tr>
</tbody>
</table>
STL vector as stack

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector<int> s;
    int e;

    do
    {
        cout << "give element (0 to stop) : ";
        cin >> e;
        if(e <= 0) break;
        s.push_back(e);
    }
    while(true);
}
```
while(!s.empty()) {
    int e = s[s.size()-1];
    cout << "popped " << e
         << " = " << s.back() << endl;
    s.pop_back();
}

A stack is a natural data structure to reverse the order in any sequence.

Application: check if word is a palindrome. A palindrome can be read forward and backward, some examples: dad, testset, racecar.
The Stack

1. The Stack Abstract Data Type
   - stack ADT
   - using the STL stack

2. An Application: Test Expressions
   - are parentheses balanced?
   - algorithm uses a stack

3. Stack Implementations
   - adapting the STL vector class
   - adapting the STL list class
as an alternative to adapting the STL vector class, we can implement a stack adapting the STL list class.

A dictionary between STL stack and list, for any item $t$ of type $T$:

<table>
<thead>
<tr>
<th>stack$&lt;T&gt;$ $s$</th>
<th>List$&lt;T&gt;$ $L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$.push($t$)</td>
<td>$L$.push_back($t$)</td>
</tr>
<tr>
<td>if(!$s$.empty())</td>
<td>if(!$L$.empty())</td>
</tr>
<tr>
<td>$t$ = $s$.top()</td>
<td>$t$ = $L$.back()</td>
</tr>
<tr>
<td>$s$.pop()</td>
<td>$v$.pop_back()</td>
</tr>
</tbody>
</table>
STL list as stack

```cpp
#include <iostream>
#include <list>
using namespace std;

int main()
{
    list<int> s;
    int e;

    do
    {
        cout << "give element (0 to stop) : ";
        cin >> e;
        if(e <= 0) break;
        s.push_back(e);
    }
    while(true);
```
while(!s.empty())
{
    int e = s.back();
    s.pop_back();
    cout << "popped " << e << endl;
}

Observe that the uniform naming of methods in STL leads to another type of generic programming: in the description of the algorithm, we may declare s as `vector<T>` s, or as `list<T>` s.
Started Chapter 5 on *Stacks*. Although simpler than vectors or lists, some algorithms reduce to mere loops with a stack.

**Exercises:**

1. Write code using a stack to test if a string, given by the user, is a palindrome.

2. The phrase "*murder for a jar of red rum*" is a palindrome. Adjust the code of the first exercise to ignore spaces.

3. Adjust the test for balanced parentheses to match single (left ‘ and right ’) and double quotes ".

4. Use a stack to compute the value of a number given as string in some given basis (< 10). For example: "537" in octal (base 8) evaluates to $7 + 3 \times 8 + 5 \times 8^2$. 