CS242: Object-Oriented Design and Programming

Program Assignment 2
Part 1: Due Midnight, Wednesday March 22^{nd}, 2006
Part 2: Due Midnight, Wednesday March 29^{th}, 2006

Key Concepts

ADT Stack and Push-Down Automaton.

Program Description

In this program, you will create a *deterministic push-down automaton* (DPDA) using a *stack* abstract data type (ADT). This DPDA will determine whether or not a string of symbols contains balanced delimiters from the set $(,), <, >, \{, \}, [,]$.

For example, the following strings do **not** have balanced delimiters (can you spot the errors?):

```
(){{<[]>}}([(([[{}{()}]))])
{{({[()})}}]
[]{}<><[>]
```

However, the next string **does** contain balanced delimiters:

```
<()<>{{<[]>}}([([]([[{}{()}]))])])>
```

Program Tasks

Your tasks are as follows:

- **Part 1: Stack Adapter implementation** – You must implement a `Stack_Adapter` class that uses the Adapter pattern to inherit the virtual methods of a `Stack_Base` class and then delegate all of its operations to a `STACK` passed as a template parameter. This design enables the dynamic selection of either `AStack` or `LStack` that do not inherit from a common base class with virtual methods.
- **Part 2: Main driver program** – You must implement a driver program that dynamically selects the desired type of stack and uses it to perform the algorithm for determining whether the input strings are balanced.

It's very important to modularize your code and use patterns, such as Abstract Factory, Singleton, Adapter, Bridge, and Strategy from your "Gang of Four" textbook. Even though the algorithm for checking balanced delimiters is straightforward I want you to design and implement your program in an elegant and concise manner. In particular, make it general so that you don't "hard code" assumptions into your solution. For example, your program should be flexible with respect to the following:

- *The length of an input line – e.g.*, there should be no arbitrary limits on the size of an input line.
- *The type of stack – i.e.*, it should be possible to select either an `AStack` or an `LStack` strategy based on command-line options to the program.

Getting Started

You can get the program shells from `www.cs.wustl.edu/~schmidt/cs215/assignment4`. The `Makefile` and `delim-test.cpp` files are written for you. You need to fill in the methods that implement the balanced delimiter checking program, as well as all the methods in the other classes whose skeletons are given to you in the "shell" files. Please note that these shells are provided "as is" as hint/suggestions to help structure your solution effectively. If you have a better way to design/implement your program please go ahead and use it.

Note that if your compiler lacks the `getopt()` function I've provided the `getopt.c` file for your use.

After you've written and compiled everything the `delim-test` file will illustrate how your program should work. For example, to try it out on UNIX run the following

```
% ./delim-test < delimiters
```

This input file contains a number of example strings of delimiters for you to test your program:

```
()
(]
(({<[]>})
[]{}<><[>]
[]<{}]<>
({<[]>})
{{({(})})}
{{({[(})})}]
(){{<[]>}}([((([{}{()}]))])])
<()<>{{<[]>}}([([]([[{}{()}]))])])>
{}}
```

Each line of input should be considered a separate string. The output from your program should look like this:

```
1: succeeded
2: unmatched left delimiter '(' at column 2
3: stack not empty, popping contents: '('
4: unmatched left delimiter '[' at column 9
5: unmatched left delimiter '<' at column 6
6: succeeded
7: unmatched left delimiter '(' at column 6
8: unmatched left delimiter '(' at column 7
9: unmatched left delimiter '(' at column 26
10: succeeded
11: unmatched right delimiter '}' at column 3
```

Additional Requirements for CS 291

If you are taking the course as CS 291 please enhance your program to support the following additional capabilities.

- *Effiency of delimiter-pair matching – e.g.,* your solution should make it possible to trade-off time and space to produce an efficient solution that is appropriate to your needs.
- *Type of delimiters – i.e.,* it should be easy to extend your program to handle arbitrary "pairs" of single-character delimiters by selecting the appropriate strategy.

Concluding Remarks

I recommend that you begin this assignment immediately so that you don’t have any “surprises” right before it is due. Keep in mind that I reserve the right to modify the assignment at any point...