

Parsing Tab-separated Input

Parsing Examples 1

Consider the problem of parsing a script file which contains lines of the following form:

```
<command> <tab> <tab-separated parameters>
```

For example:

```
; Parser test input 01
;
reverse parse this line
;
sort    gamma    alpha    delta
;
add     17       43       29
exit
```

The lines beginning with semicolons are comment lines which should be ignored, but we'll ignore that issue for now and focus on the actual command lines.

The Issues

Parsing Examples 2

Given the line

```
reverse parse this line
```

the program should identify the command "reverse" and then take the appropriate action with the remainder of the line, which should result in something like:

```
"parse this line" reversed is:  esrap siht enil
```

There are two parsing issues here:

- How do we deal with identifying the command?
- How do we break the line up into logical tokens?

The first issue may be handled flexibly by making use of `strings`, `stringstreams`, and an enumerated type.

Top-Level Organization

Parsing Examples 3

Here's one approach:

```
void main() {
    string inputFileName = "script.txt";
    ifstream iFile(inputFileName.c_str());

    if (iFile.fail()) {
        cout << "File not found: " << inputFileName << endl;
        return;
    }

    string Next = Parser(iFile);           // get first line of input
    while ( iFile ) {                     // quit on stream failure
        if ( ProcessCmd(Next) == EXIT) return; // process this command line
        Next = Parser(iFile);             // try for another line
    }
    iFile.close();
}
```

Getting the Next Input Line

Parsing Examples 4

This will return the next non-empty line, if any, of the input file as a string:

```
string Parser(istream& In) {
    string nextLine;
    getline(In, nextLine, '\n');           // eat a line

    while ( In && ( nextLine.length() == 0 ) ){ // don't accept an empty one
        getline(In, nextLine, '\n');
    }

    return nextLine;
}
```

Note that this does not take the comment lines into account.

Since `main()` makes no provision for dealing with comments, this must be extended to also reject comment lines.

Identifying the Command

Parsing Examples 5

The current command line can be parsed with a stringstream:

```
Command ProcessCmd(string cmdLine) {  
  
    string cmdString;  
    stringstream In(cmdLine);           // attach a stream to the string  
  
    getline(In, cmdString, '\t');       // read the command string  
  
    Command thisCmd = Classify(cmdString); // map it to an enumerated value  
  
    switch ( thisCmd ) {                // so it can be sorted out  
    case ADD:    handleAdd(In);         // with a switch statement  
                break;                // and the stream can then  
    case REVERSE: handleReverse(In);   // be passed on to the  
                break;                // appropriate handler  
    case SORT:  handleSort(In);  
                break;  
    };  
    return thisCmd;  
}
```

```
enum Command {ADD, REVERSE, SORT, EXIT, UNKNOWN};
```

Mapping a String to a Command

Parsing Examples 6

The mapping can be done with a simple sequence of if statements:

```
Command Classify(string cmdString) {  
  
    if (cmdString == "add")    return ADD;  
    if (cmdString == "reverse") return REVERSE;  
    if (cmdString == "sort")  return SORT;  
    if (cmdString == "exit")  return EXIT;  
    return UNKNOWN;  
}
```

A few points:

- The comparisons are case-sensitive (that can be changed).
- This is easily extended to handle different or additional commands.
- A default value is needed in case no matching command can be found.

Handling a Command

Parsing Examples 7

The reverse command is handled easily with stream and string members:

```
void handleReverse(istream& In) {  
  
    string Next;  
    getline(In, Next, '\t'); // the istream is read just the  
                            // same as any other stream  
  
    while ( In ) {  
  
        for (int Idx = Next.length() - 1; Idx >= 0; Idx--) {  
            cout << Next.at(Idx);  
        }  
        cout << '\t';  
  
        getline(In, Next, '\t');  
    }  
    cout << endl;  
}
```