E02: Supplement “Code reading” and “cmdline”

This is the supplemental sheet for applying technique “code reading” to the program “cmdline”

Overview

The program **cmdline** performs a simple parsing of the command line to determine if it is valid. It then prints out a description of the command line or an indication of where it is incorrect. Note that it does not carry out any of the commands, but merely validates the input.

Below is a simple example of its usage:

```
C:\ cmdline -measure GKOM ifile
Command summary is:
Measure: GKOM
Number of Input Files: 1
Input Files are:
      ifile

C:\ cmdline -measure JUNK ifile
Invalid Measure option given
Option Arguments are Incorrect
```

The presence of input files is not checked so they do not have to be created. The measures (GKOM in the above example) are acronyms for operations to be performed on the data in the files — for the purposes of this exercise you do not need to know exactly what these are.

Brief description of the library functions used

- int strlen(char *s)

  Returns the length of the string s (the null character on the end is not counted).

- int strcmp(char *s1, char *s2)

  Compares two strings. Returns a value greater than, equal to, or less than 0 depending on whether st1 is lexicographically (i.e., ASCII value) greater than, equal, or less than s2.

- int strncmp(char *s1, char *s2, int n)

  Like strcmp, but compares maximum n characters.

- int fprintf(stderr, "Text"...)

  Prints the text on the standard-error output stream.

- int isdigit(char c)

  Returns true if the character is a digit, otherwise false.

- int atoi(char *str)

  Converts the string str to an integer. Leading white space is ignored, and scanning of the string ends with the first non-numeric character. If no numeric characters could be read, 0 is returned.

Reminder

Don’t produce any abstractions for the test-harness functions.

E03: Supplement “Functional testing” and “cmdline”

This is the supplemental sheet for applying technique “functional testing” to the program “cmdline”

Overview

The program **cmdline** performs a simple parsing of the command line to determine if it is valid. It then prints out a description of the command line or an indication of where it is incorrect. Note that it does not carry out any of the commands, but merely validates the input.

Below is a simple example of its usage:

```
C:\ cmdline -measure GKOM ifile
Command summary is:
Measure: GKOM
Number of Input Files: 1
Input Files are:
      ifile

C:\ cmdline -measure JUNK ifile
Invalid Measure option given
Option Arguments are Incorrect
```

The presence of input files is not checked so they do not have to be created. The measures (GKOM in the above example) are acronyms for operations to be performed on the data in the files — for the purposes of this exercise you do not need to know exactly what these are.

Documents and Code

For this exercise you require the specification of the component, and the executable code of the component, which you will receive after you have written test cases.

Fetching the necessary files

The only program you need is the executable cmdline. Create a new directory to avoid conflict with any other testing exercises and copy the file cmdline.exe into this directory.

Creating equivalence classes and test cases

The component “cmdline” can be tested directly. No input files are required.

E04: Supplement “Structural testing” and “cmdline”

This is the supplemental sheet for applying technique “structural testing” to the program “cmdline”

Overview

The program **cmdline** performs a simple parsing of the command line to determine if it is valid. It then prints out a description of the command line or an indication of where it is incorrect. Note that it does not carry out any of the commands, but merely validates the input.

Below is a simple example of its usage:

```
C:\ cmdline -measure GKOM ifile
Command summary is:
Measure: GKOM
Number of Input Files: 1
Input Files are:
      ifile

C:\ cmdline -measure JUNK ifile
Invalid Measure option given
Option Arguments are Incorrect
```

The presence of input files is not checked so they do not have to be created. The measures (GKOM in the above example) are acronyms for operations to be performed on the data in the files — for the purposes of this exercise you do not need to know exactly what these are.

Documents and code

For this exercise you require the source code of the component, and the specification of the component, which you will receive after you have created test cases and attempted to reach 100% coverage.

Fetching the necessary files

The only program you need is the executable cmdline. Create a new directory to avoid conflict with any other testing exercises and copy the file cmdline.exe into this directory.

Writing test cases

All of the program’s functions are fundamentally tested via the invocation

```
cmdline [ -argument ... ]
```

“cmdline” requires no input files to be created.

Which arguments are allowed on the command line and which functions are invoked by those commands can be seen in the function `process_switches` in the file `cmdline.c`. There is also a small test harness which does not need to be tested.

Brief description of the library functions used

- int strlen(char *s)

  Returns the length of the string s (the null character on the end is not counted).

- int strcmp(char *s1, char *s2)

  Compares two strings. Returns a value greater than, equal to, or less than 0 depending on whether st1 is lexicographically (i.e., ASCII value) greater than, equal, or less than s2.

- int strncmp(char *s1, char *s2, int n)

  Like strcmp, but compares maximum n characters.

- int fprintf(stderr, "Text"...)

  Prints the text on the standard-error output stream.

- int isdigit(char c)

  Returns true if the character is a digit, otherwise false.

- int atoi(char *str)

  Converts the string str to an integer. Leading white space is ignored, and scanning of the string ends with the first non-numeric character. If no numeric characters could be read, 0 is returned.