Assignment 1: Communicating with Programs

EC602 Design by Software

Fall 2017

Contents

1 Introduction

1.1 Assignment Goals

The first assignment goals are to ensure

- you have the development environments available to you
- you are able to use the assignment submission system
- you can write and run simple C++ and Python programs

- you can use the unix terminal

1.2 Group Size

For this assignment, the maximum group size is 1. Everyone must complete the assignment.

However, you may help each other as much as you wish for this assignment.

1.3 Due Date

This assignment is due 2017-09-12 at midnight

1.4 Late policy

Late assignments will be accepted through a grace period of H hours.

Usually, H will be the number of hours until the beginning of the lecture immediately following the due date (the normal H is 14.5).

The grade will be scaled on a linear scale from 100% at the deadline to 50% at the end of the grade period.

If the *natural grade* on the assignment is g, the number of hours late is h, and the grace period is H, then the grade is

```
grade = (h > H) ? 0 : g * (1- h/(2*H));
```

in C++ or

```
grade = 0 if h>H else g * (1- h/(2*H))
```

in python.

If the same assignment is submitted ontime *and* late, the grade for that component will be the maximum of the ontime submission grade and the scaled late submission grade.

1.5 Submission Link

You can submit here: week 1 submit link

The submission link will activate on Monday morning.

1.6 Points

This assignment is worth 3 points.

2 Tutorials

If you are a total beginner at python, unix, or C++, here are three good places to start:

- C++: tutorial as cplusplus.com
- Python: Official Python Tutorial
- Unix: Unix tutorial for beginners

3 Background

In unix, the shell or terminal provides an environment in which the program can interact with the operating system and user. One of the ways it does this is to provide input and output facilities to the terminal. The other way is by passing programs configuration options on the command line. These are called "command-line arguments."

3.1 Input and Output

The input and output "streams" are

- standard input (`stdin`): the program can read whatever is "typed" at the terminal
- standard output (`stdout`): normal output from the program goes here.
- standard error (`stderr`): when errors occur during execution of a program, the program normally sends the error messages here.

By default, both `stdout` and `stderr` will appear together on the terminal, but they can be *redirected* to illustrate the difference

Here are some examples:

Put the list of files into `my_file_list.txt`, a new text file to be created:

```
ls > my_file_list.txt
```

Compile `my_prog.cpp` and put errors, if any, in `my_errors.txt`

```
g++ my_prog.cpp 2> my_errors.txt
```

3.2 Command-line arguments

Most unix programs reserve `stdin` and `stdout` for data. Configuration options are typically specified as arguments specified on the "command line."

Normally, `ls` lists all files in the current directory. With command line arguments, this behavior can be modified:

Examples:

List the files in the directory `/media`

```
ls /media
```

Compile `fourargs.cpp` and make an executable called `fourargs`

```
g++ fourargs.cpp -o fourargs
```

In this example, there are three arguments:

- argument 1 is the name of the program to compile
- argument 2 is a symbol that `g++` recognizes which means "the next argument is the name of the output"
- argument 3 is the name of the executable file to create

4 The Assignment

The programs use command-line arguments and `stdout` and `stderr`. Your python and C++ programs should accept up to four arguments. Any more arguments are considered errors.

4.1 Python `fourargs.py` (1 point)

Write a python program `fourargs.py` that outputs the first four arguments to *stdout* and any additional arguments, if any, to *stderr*.

Each argument should appear on a line by itself.

So, for example, if the program is run like this:

```
python fourargs.py one two 3 four five six
```

then the output to `stdout` will be

```
one
two
3
four
```

and the output to `stderr` will be

```
five
six
```

4.2 C++ `fourargs.cpp` (1 point)

Write a C++ program `fourargs.cpp` that outputs the first four arguments to *stdout* and any additional arguments, if any, to *stderr*. Each argument should appear on a line by itself.

So, for example, if the program is run like this:

```
fourargs one two 3 four five six
```

then the output to `stdout` will be

```
one
two
3
four
```

and the output to `stderr` will be

```
five
six
```

4.3 Shell `fourargs.sh` (1 point)

Write a script `fourargs.sh` that

- compiles the C++ program to an executable called "fourargs"
- runs the python program with 6 arguments
- runs the python program with 3 arguments
- runs the C++ program with 6 arguments
- runs the C++ program with 3 arguments

4.4 Hints

For python, research the `sys` module.

For C++, research the `iostream` module.

Caution: both python and C++ include an extra argument as "argument 0". This extra argument is the name of the program. These programs should not print out argument 0.

5 Assignment checker

You can check your work prior to submission using the assignment checker, which is a python program.

- fourargs_checker.py

To use the checker, download it into the same directory that has your three programs `fourargs.cpp`, `fourargs.py` and `fourargs.sh` and then run

```
python fourargs_checker.py
```