Assignment 8

Experiment Control Applied to Matrix Multiply

EC602 Fall 2016

Contents

1 Introduction

Many tests and research experiments require the use of automated programs for their operations.

This week, we show how to use python to manage C++ programs, under the assumption that the *fast and efficient code* is written in C++.

1.1 Assignment Goals

The assignment goals are to

- learn about command line arguments
- learn about the subprocess library in python
- learn file I/O in C++ and python

1.2 Due Date

This assignment is due 2016-11-07 at midnight.

1.3 Submission Link

You can submit here: week 8 submit link

2 Command Line arguments for C++

The `main()` function of a C++ program is passed values from the command line. These values are typically called *arguments*, which is confusing because they are not precisely the same as function *arguments*. Since the operating system (the shell) separates these values, we can also think of the values as *tokens*, which means the shell has parsed the line and split it into *tokens*.

These values (*tokens* or *arguments*) are typically used to select options for the program and adjust the behavior of the program.

The first argument to `main` is called `argc` and is an integer representing the number of tokens or *arguments to the program* that were on the command line.

The second argument to `main` is called `argv` and is an array of C-strings containing the tokens or *arguments to the program* that were on the command line.

A C-string is a pointer to a consecutive block of member containing `char` values. The last `char` is indicated by a null character.

C-strings are hard to use properly and are dangerous, and this is the only situation in C++ that we are really forced to use C-strings instead of C++ `<string>` objects.

2.1 Example: printing out the tokens

The following example show_arguments.cpp demonstrates how to read the arguments to `main` which are *tokens* or *arguments* on the command line.

```
// Show the command line argument values.
#include <iostream>
#include <string> // provides stoX where X can be i (int), d (double), etc.

using namespace std;
int main(int argc, char const *argv[])
{
    // the following example shows what argc and argv are
    //
    // c-strings are hard to use properly and are dangerous.
    int i,j;

    cout << "argc is " << argc << endl;
    for (i=0; i<argc; i++){
        //print out argv[i]

        cout << "argv of " << i << " is " << argv[i] << endl;

        //print out argv[i], discover its length
        j=0;
        while ( argv[i][j])  // equivalent to (argv[i][j] != '\0')
             cout << argv[i][j++];

        cout << " len " << j;

        // alternative
        string s(argv[i]);
        cout << " len " << s.size();
```

```
        cout << endl;
    }
}
```

2.2 Processing arguments

To extract numerical values, you will need to convert C-strings to `int` or `double`.

The following program convert_arguments.cpp shows how to do this.

Notice the use of `try {} catch (...) {}` for *exception handling*.

```
// Convert command line values to double
#include <iostream>
#include <string> // provides stoX where X can be i (int), d (double), etc.

using namespace std;
int main(int argc, char const *argv[])
{

    for (int i=1; i<argc ; i++)
    {
        cout << i << " ";
        try
        {
            double v = stod(argv[i]);
            cout << v << endl;
            continue;
        }
        catch (...)
        {
            cout << argv[i] << " is not a double.";
        }
        cout << endl;
    }
}
```

3 Making Python "executables"

We can make a python program look exactly like a compiled C++ program as long as python is installed on the host computer.

Here is the program:

```
#!/usr/bin/env python

import sys

print("argc is",len(sys.argv))
for i,token in enumerate(sys.argv):
    print('argv of',i,'is',token)
```

This program is called fake_show_arguments and must be modified on your computer using a command like

```
chmod 755 fake_show_arguments
```

This makes it an official "executable" program, which means you can type

```
./fake_show_arguments 6 7 8
```

from terminal and the program will be run. Note that this program can still be run as

```
python fake_show_arguments
```

4 File I/O

4.1 C++ file I/O

C++ file I/O is very similar to using `cin` and `cout`, but the files also need to be opened and closed.

Here are some examples:

- make_table_of_squares.cpp : create a file
- read_table_of_squares.cpp : read the file
- read_table_of_squares_bad.cpp : this program contains a common error.

4.2 Python file I/O

Python file I/O is well explained here: input/output tutorial

5 Scripting: Using Python to run other programs

Some people consider python to be a *scripting language*, which means roughly that it is useful to accomplish system administration and other operating system related tasks.

One function of a scripting language is to control, execute, and monitor other programs.

We will make use of python's `subprocess` library to do this.

Here are the docs for subprocess

5.1 Example one: getting a file list

Here is an example of running the shell command `ls` to get a directory listing.

This is such a useful command that python includes it directly in the `os` module, so we check that the two commands give the same result.

```
import subprocess
import os

T = subprocess.run(['ls','-t'],stdout=subprocess.PIPE)
Files = T.stdout.decode().splitlines()

OSfiles = os.listdir('.')

print(Files)
print(OSfiles)
assert sorted(Files) == OSfiles
```

6 Matrix Multiply

Matrix multiply is a common and important linear algebra operation. The following is an example of multiplying two integer matrices. Notice that the program assumes that the vector of vectors are of the correct shape.

```
typedef vector< vector<int> > int_matrix;

int_matrix multply(const & int_matrix A,const & int_matrix B){

 int M = A.size()
```

```
 int K = A[0].size()
 int L = B[0].size()

 int_matrix c(M,L);

 for (int i=0;i<M;i++)
    for (int j=0;j<L;j++)
        for (int n=0;n<N;n++)
            c[i][j] = A[i][n]*B[n][j]


}
```

The program is multiply.cpp.

7 The Assignment

7.1 Part A: Matrix multiply with C++

Write a C++ program that reads in two matrices from two text files, and outputs the result in a third text file.

The data in the files will be in the natural, readable format, like this:

```
5 6 7 8
3 4 5 6
1 2 -2 0
```

This is a 3x4 matrix, with 3 lines of data each containing 4 numbers.

The sizes and types of the data are controlled via command line arguments, as follows.

7.1.1 Square matrix mode

If the command line looks like this:

```
w8c_multiply dtype N file1 file2 file3
```

then all matrices are `NxN` and the other information means:

- `dtype` is either `int` or `double`, indicating what type of data is in the files
- `file1` is the name of the file containing the first matrix
- `file2` is the name of the file containing the second matrix
- `file3` is the name of the file containing the result matrix

7.1.2 General matrix mode

If the command line looks like this:

```
w8c_multiply dtype M N L file1 file2 file3
```

then

- `dtype` is either `int` or `double`, indicating what type of data is in the files
- `file1` is the name of the file containing the first matrix, size `MxN`
- `file2` is the name of the file containing the second matrix, size `NxL`
- `file3` is the name of the file containing the result matrix

The program should be called `w8c_multiply.cpp`

7.1.3 Details

The program should handle all possible error conditions, including (but not limited to?):

- the command line arguments are invalid (return code 1)
- one or more of the input files do not exist (return code 2)
- the data in the files does not conform to the expectations (return code 3)
- the result matrix cannot be created (return code 4)

If there are cases I have not thought of yet, use return codes 5 and up.

The return code is the integer value returned by `main`.

7.2 Part B: Matrix multiply with Python

Write a python "executable" that does exactly the same thing, with the same interface as described in part A. The program should be called `w8p_multiply` (notice, there is no `.py` at the end)

You may (and probably should) use the python `@` operator for numpy.ndarray objects.

Python can generate a return code using `exit(rc)` where `rc` is the return code.

Your goal with this program is to make it both fast and as short as possible.

7.3 Part C: Testing your code with python

Write a program `w8_tester` (a python script) which compares the execution time of the programs from part A (under different optimization levels) and part B. The compiler optimization levels (1,2, and 3) control various tricks that the compiler tries to speed up your compiled code.

The program from part A should be available as `w8c_multiply_x` where `x` is the optimization level provided to the C++ compiler, like this:

```
g++ -std=c++14 -O2 w8c_multiply.cpp -o w8c_multiply_2
```

Your program should use `subprocess` to compile the three versions of part A.

The program should perform tests according to the following command line switches:

```
w8_tester dtype Nlist Ntrials
```

- `dtype` is `int` or `double`
- `Nlist` is a list of matrix sizes to test the programs on. We will test square matrices for now.
- `Ntrials` is the number of different files to average the results over.

For each trial, you should generate a random matrix of appropriate values and store those values into a file. The same file should be used for each of the four programs.

Here is an example:

```
w8_tester int "[5,10,100]" 3
```

The output should be in sorted order of average time, like this:

```
py 100 5.61
O1 100 4.52
O2 100 4.33
O3 100 2.33
py 10 1.03
O1 10 0.61
O3 10 0.54
O2 10 0.31
O1 5 0.10
py 5 0.08
O3 5 0.04
O2 5 0.01
```