

## Homework 5: Extracting parts from strings

(Points and due date are in Ctools)

The goal of this assignment is to use functions and string processing to read movie information from a string and then write the movie information in HTML format. The movie information is represented in a string in the following format:

```
<ID> | <movie title> | <movie date> | <ignore> | <movie URL> | <arbitrary stuff>
```

Notice the vertical `|` that separate the various fields in the movie. This format is actually used in a research project on recommender systems for netflix-like movie database consisting of 100,000 movies. We provide you that database in `ml-100k`, though you will not be actually using that database in this assignment. See the file `u.item` in the `ml-100k` folder for example movies in the above format.

In this assignment, we are just going to assume that the information is available in string form, rather than read from the movie database files. The goal is to extract the following 4 fields from a string in the above format:

- <ID>

- <movie title>

- <movie date>

- <movie URL>

An example movie information (referred to as `movieitem` in the program) in the above format is:

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
```

(The above is the first movie in the file ml-100k/u.item).

For the above movieitem, here are the fields of interest:

- <ID> is "1". This is a string.
- <movie title> is "Toy Story (1995)".
- <movie date> is "01-Jan-1995".
- <movie URL> is "http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)"

## C++ Part of the Assignment

You will be writing functions in the file `extract.cpp` to do the extraction of the above 4 fields. We have given you the code for some of the functions. For others, you have to fill it in, following the instructions in that file. The functions are:

- `string extract_id(string movieitem)`: this takes a `movieitem` in the above format with vertical bars and extracts the movie ID from it. For example, if `movieitem` is:

[illegible]

the above function should return the string "1".

- `string extract_movie_title(string movieitem)`: This extracts and returns the movie title. For the above `movieitem`, it should return the string "Toy Story (1995)".
- `string extract_moviedate(string movieitem)`: This extracts and returns the movie date. For the above `movieitem`, it should return the string "01-Jan-1995".
- `string extract_movieurl(string movieitem)`: This extracts and returns the movie url. For the above `movieitem`, it should return the URL string "http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)".

You will also be writing functions that help do testing. Every time one writes a function, it is good to write another function that checks the correctness of the implementation with some sample input by verifying the output. Those functions are to be written in the file `test_extract.cpp`. We have given you one of the functions there: `test_extract_id.cpp`. Follow the instructions to write the other functions.

Finally, you will be writing a function `helper_gen_html()` in the file `print_movieitem.cpp` to convert the above 4 fields in HTML format. The HTML format must be looks like the following:

```
<html>

<body>

<p>ID</p>

<p><a href="movie url">movie title</a></p>

<p>movie date</p>

</body>

</html>
```

Another function in `printmovieitem.cpp`, `get_html_from_movie(string movieitem)`, makes use of your extract functions in `extract.cpp` and the helper `_gen_html()` function above to prints the HTML string to a file `<ID>.html`, where `<ID>` is extracted from the `movieitem`. You can view that file by opening it in a browser. For the Toy Story movie, the correct HTML is the following:

```
<html>

<body>

<p>1</p>

<p><a href = "http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)">Toy Story
(1995)</a></p>

<p>01-Jan-1995</p>

</body>

</html>
```

Make sure you include the quotes around the URL.

We have given you a `main.cpp`. you are free to make changes to `main.cpp`, though we are not going to be grading `main.cpp`. We are only grading `extract.cpp`, `test_extract.cpp`, and `printmovieitem.cpp`.

Go in the following sequence:

### Step 0.

Get the code to compile. To check if the code compiles, do the following:

```
% g++ -Wall -g main.cpp extract.cpp printmovieitem.cpp test_extract.cpp
```

(Your program consists of functions in multiple files. You need to provide all the `.cpp` files to `g++` to build the executable file.)

A successful compilation should generate an `a.out` file.

It is OK if `a.out` doesn't run correctly. Write empty functions that return some dummy values of the right type in all the files to match the functions in the corresponding header files. For example, `test_extract.h` declares 4 functions. Make sure you have at least empty versions of those functions in `extract.cpp`.

We have given you one of the tests, `test_extract_id()`, as well as its corresponding function, `extract_id()`, to get you started. These functions are correct. You do have to write the remaining 3 tests and the remaining 3 extractions functions.

**Step 1:** Write an additional test in `test_extract.cpp`, e.g., `test_extract_movie_title()`, followed by the corresponding function, `extract_movie_title()`, in `extract.cpp`.

Always write the tests in `test_extract.cpp` before you write the corresponding functions. Usually, writing a test is easier than writing a function. To figure out what goes in the test, you may need to look at the description of the function in `extract.cpp`.

As you write a test, make sure your code continues to compile. You may have to write an empty (stub) version of the function being tested so that the code compiles.

When you run the program with a stub function, the test should initially fail - MAKE SURE you can make a test fail. Easiest thing is to just write an empty version of the function being tested that compiles, but doesn't do anything. If you can't make a test fail, you really would not know if the test is correct.

### **Step 2 and 3:**

Once you pass the test for `extract_movie_title` in Step 1, repeat step 1 for the remaining two extraction functions. Write the test first and then the extraction function. Make sure you pass all the tests.

### **Step 4:**

Write the function in `print_movieitem.cpp` so that it generates the correct output.

### **Step 5:**

For comparison purposes, we also provide you an executable file, `prof.run`, which contains the professor's solution. If you run that file, it will output `1.html` and `64.html`, two files corresponding to two sample movies in your `main.cpp`. You can compare the HTML files generated from your program with the HTML files generated by `prof.run`. Any output from `prof.run` on the Terminal does not have to be matched by your code. You are free to add any couts you need for debugging to your program. That will not affect the auto-grader.

To run `prof.run`, simply do in Terminal:

... make sure you are in the HW5 folder before doing the following ...

```
% chmod +x prof.run      # Makes prof.run executable. One-time.
% prof.run                # Executes prof.run
```

If by chance `prof.run` doesn't work (if you are on a CAEN machine, it may not), then try:

```
% ./prof.run
```

## Auto-grading your C++ Program

We provide you a local autograder so that you can determine your likely score on the C++ part of the assignment. You will be able to run that on Linux within Mint and on CAEN machines.

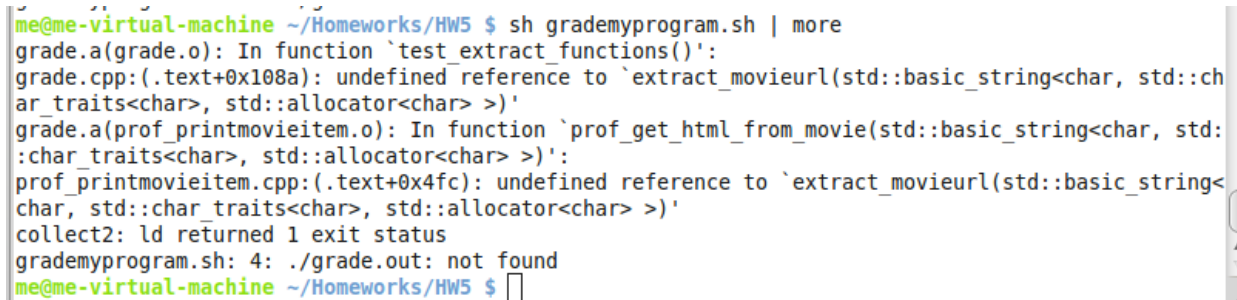
To run the autograder, do the following from within your HW5 folder:

```
% sh grademyprogram.sh
```

If the output is too much and flies by, you can "pipe" the output from the above command through "more" so that you get one screen at a time output:

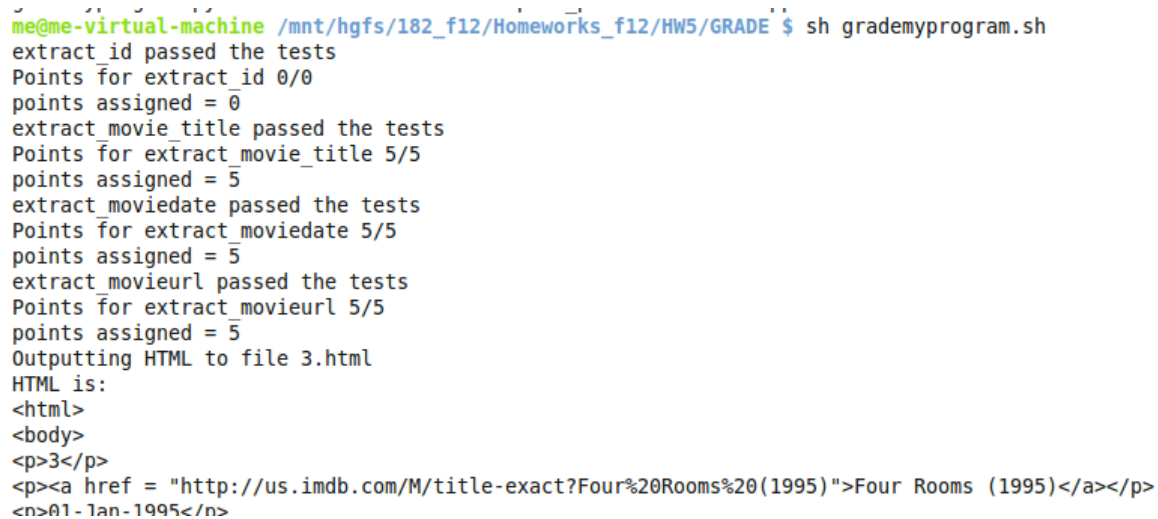
```
% sh grademyprogram.sh | more
```

Read the output carefully. Below is one example thing you may see if you run the autograder immediately with the assignment as originally distributed:

A terminal window screenshot showing the output of the autograder. The prompt is 'me@me-virtual-machine ~/Homeworks/HW5 \$'. The command 'sh grademyprogram.sh | more' has been executed. The output shows several error messages: 'grade.a(grade.o): In function `test\_extract\_functions()': grade.cpp:(.text+0x108a): undefined reference to `extract\_movieurl(std::basic\_string<char, std::char\_traits<char>, std::allocator<char> >)'', 'grade.a(prof\_printmovieitem.o): In function `prof\_get\_html\_from\_movie(std::basic\_string<char, std::char\_traits<char>, std::allocator<char> >)'': prof\_printmovieitem.cpp:(.text+0x4fc): undefined reference to `extract\_movieurl(std::basic\_string<char, std::char\_traits<char>, std::allocator<char> >)'', and 'collect2: ld returned 1 exit status'. The final line shows 'grademyprogram.sh: 4: ./grade.out: not found' and the prompt returns to 'me@me-virtual-machine ~/Homeworks/HW5 \$'.

The autograder failed to work because you have not yet written all the functions that are required. So, your score is essentially zero.

Once you complete all the functions, you should see something like the following:

A terminal window screenshot showing the successful output of the autograder. The prompt is 'me@me-virtual-machine /mnt/hgfs/182\_f12/Homeworks\_f12/HW5/GRADE \$'. The command 'sh grademyprogram.sh' has been executed. The output shows that all tests passed: 'extract\_id passed the tests', 'extract\_movie\_title passed the tests', 'extract\_moviedate passed the tests', and 'extract\_movieurl passed the tests'. It also shows the points assigned for each test (0/0, 5/5, 5/5, 5/5) and the total points assigned (0, 5, 5, 5). Finally, it shows the HTML output: 'Outputting HTML to file 3.html', 'HTML is:', and the HTML code for 'Four Rooms (1995)'. The prompt returns to 'me@me-virtual-machine /mnt/hgfs/182\_f12/Homeworks\_f12/HW5/GRADE \$'.

The total points will be reported at the end of the output. The above screenshot only shows the partial output.

## Python Part of the assignment:

This is going to be relatively small part, just to get experience with Python functions. Modify `printmovieitem.py` so that it prints out a movie in HTML, given the fields.

You only need to modify the `print_movie_in_html` function. Nothing else is required. Make sure that it prints out exactly the same value your C++ code for the same movie.

This function should be easy to write once you have completed the corresponding C++ functions in `print_movieitem.cpp`.

## Additional notes on the example data

The example data from which we are drawing the inspiration of this assignment is in the folder `ml-100k`. That has information on about 100,000 movies.

We are not actually reading the files from that folder in this assignment, but it may help motivate you. This data is only permitted for use for academic purposes.

Note that this data is several years old. Therefore, the URLs for the movies are unlikely to work today. That is not crucial for this assignment. The URLs can be fixed with more current data. They should all take you somewhere in IMDB's web site.

## What to Submit

- All your `.cpp` files
- Your python file: `printmovieitem.py`
- Screenshots showing that your Python and C++ programs run successfully.

If you have trouble with uploading multiple files to CTools, zip the HW5 folder as follows in Mint:

- `cd` to the folder above HW5:

```
% cd
```

```
% cd Homeworks
```

- zip the HW5 folder and generate the archive `HW5.zip` using the "zip" Terminal command. The "-r" option instructs zip to recursively include any files or folders within HW5 into the zip archive.

```
% zip -r HW5.zip HW5
```

- Upload the HW5.zip to Ctools and Submit this single zip file as your solution.
- Verify the submission by downloading the zip file from Ctools, extracting files from it, and making sure that all your intended files are there. A common mistake is to submit an incorrect zip file or a zip file without all the files. So, it is a good idea to verify the submission.