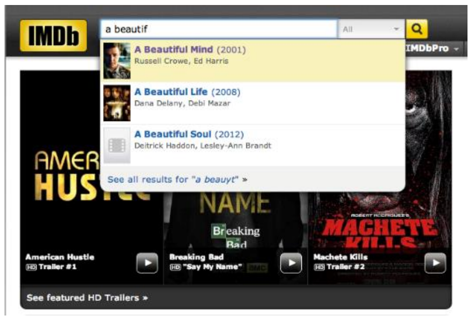
CS 251
Data Structures
Spring 2017

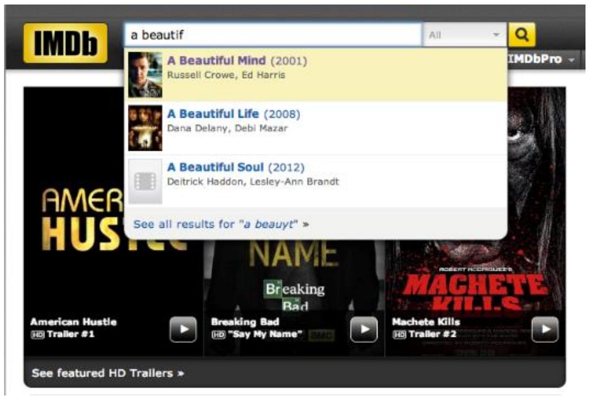


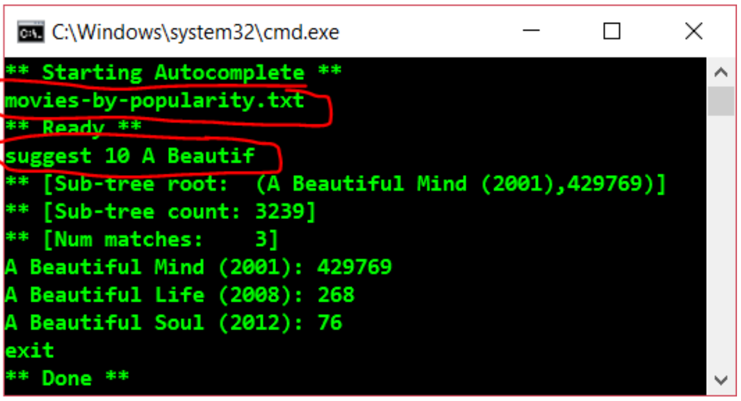
Project #03: Auto-complete using Binary Search Trees

Complete By: Saturday, February 11th @ 11:59pm
Assignment: C program to yield auto-complete suggestions
Policy: Individual work only, late work *is* accepted (see "Policy" section on last page for more details)
Submission: online via zyLabs ("Project03 Auto-Complete", section 4.12)

Assignment

The assignment is to build a system that implements "auto-complete". This is a feature we use every day, where we enter a partially complete piece of text and the system suggests various ways to complete. The "internet movie database" offers this feature to help find movies:



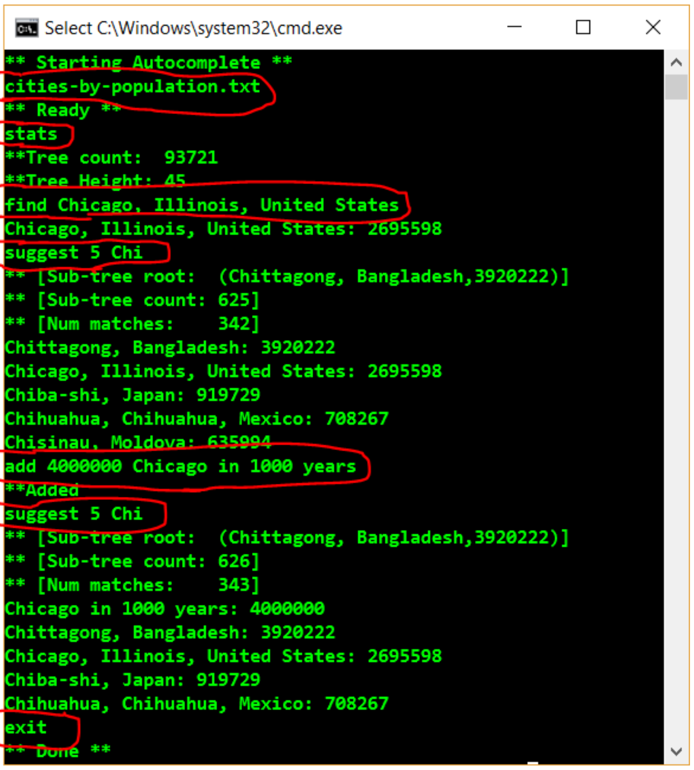

```
C:\Windows\system32\cmd.exe
** Starting Autocomplete **
movies-by-popularity.txt
** Ready **
suggest 10 A Beautif
** [Sub-tree root:  (A Beautiful Mind (2001),429769)]
** [Sub-tree count: 3239]
** [Num matches:    3]
A Beautiful Mind (2001): 429769
A Beautiful Life (2008): 268
A Beautiful Soul (2012): 76
exit
** Done **
```

Our visually less-appealing but equally-powerful system is shown on the right. Given an input file of textual phrases and weights, the user can ask the system to suggest **k** ways to complete the given piece of text, in **decreasing order by weight**. In the example shown on the right, we are telling the program to read the phrases and weights from the input file "movies-by-popularity.txt", and then to suggest the top **10** ways to complete the textual phrase "**A Beautif** ". Only 3 ways are shown because there are only 3 matches in the given input file; the integers shown to the right of the movie name are the weights (in this case the # of votes cast for this movie on the IMDB site).

Assignment Details

Your program is going to support 4 different commands, listed here in order of difficulty. This way you can earn some partial credit even if you don't complete the full assignment:

1. **stats**
2. **find** phrase
3. **add** weight phrase
4. **suggest** k phrase
5. **exit**



Select C:\Windows\system32\cmd.exe

```
** Starting Autocomplete **
cities-by-population.txt
** Ready **
stats
**Tree count:  93721
**Tree Height: 45
find Chicago, Illinois, United States
Chicago, Illinois, United States: 2695598
suggest 5 Chi
** [Sub-tree root:  (Chittagong, Bangladesh,3920222)]
** [Sub-tree count: 625]
** [Num matches:    342]
Chittagong, Bangladesh: 3920222
Chicago, Illinois, United States: 2695598
Chiba-shi, Japan: 919729
Chihuahua, Chihuahua, Mexico: 708267
Chisinau, Moldova: 635994
add 4000000 Chicago in 1000 years
**Added
suggest 5 Chi
** [Sub-tree root:  (Chittagong, Bangladesh,3920222)]
** [Sub-tree count: 626]
** [Num matches:    343]
Chicago in 1000 years: 4000000
Chittagong, Bangladesh: 3920222
Chicago, Illinois, United States: 2695598
Chiba-shi, Japan: 919729
Chihuahua, Chihuahua, Mexico: 708267
exit
** Done **
```

The user enters the name of an input file, and then enters N > 0 commands until he/she enters **exit**. A screenshot of one possible session is shown to the right. Your program will need to produce this output exactly as we are using the zyante system for the first level of grading.

Your program is required to solve this problem using a binary search tree (do not attempt to balance the tree). The "stats" command outputs the # of nodes in the tree, followed by the height. The "find" command searches the tree for an exact match to the phrase input by the user; if found, the phrase and weight are output, otherwise "**Not found…" is output. The "add" command inserts a new node into the tree with the given phrase and weight; if the insert is successful then "**Added" is output, otherwise "**Not added…" is output. Why would the insert fail? For example, if the user inserts a duplicate phrase. *[Note: the "add" command does not change the underlying input file, just the tree created in memory.*]

The "suggest" command is the interesting one, and implements the *auto-complete* functionality. While there are many ways to solve this problem, the idea is to solve this problem using the binary search tree built from the input file. While you have some freedom in designing your solution, you are required to search and extract the data from the tree in a reasonably efficient manner based on the following approach:

1. The user enters an integer **k**>0 and the start of a textual phrase, what we'll call the **prefix**
2. Search the tree for the first node whose key starts with this prefix; call this node R.
3. Output R's (key, value) pair; this is shown above as the "Sub-tree root".
4. All possible suggestions are contained in the sub-tree rooted by R; output the # of nodes in this sub-tree, shown above as the "Sub-tree count".
5. Now search this sub-tree for all nodes that start with prefix; this is the # of matches M. Output M, shown above as the "Num matches".
6. Sort these matches in decreasing order by weight; if 2 matches have the same weight, order by text. Since M is not very large, use any sort you want as long as you write the sort yourself.
7. Output the first min(k, M) suggestions; if M = 0, output "**No suggestions…".

The reason for the tree is to quickly locate the first possible suggestion. This narrows down the search space in O(lgN) time. There are other ways to solve this problem, e.g. you can simply load the phrases and weights into an array and perform a linear search. However, this would yield an O(N) solution, which is unacceptable for large N. In this assignment, you are required to build, search and extract the data from a binary search tree --- in particular, from the appropriate subset of the tree that matches the entered prefix. Using other approaches, even if they yield the correct answer, will not be graded.

Input Files

A number of input files are supplied on the course web site under "Projects", "project03-files". Each file has the same format: one or more lines where each line contains a weight followed by a textual phrase. The smallest file, and thus a good one for testing, is "fortune1000-by-revenue.txt":

```
219812     Wal-Mart Stores
191581     Exxon Mobil
177260     General Motors
162412     Ford Motor
.
.
.
```

This file contains the top 1000 companies in terms of revenue. The **weight** is an integer in the range 0 <= weight <= 2^63 - 1. This implies you need to store the weight as a 64-bit integer, declared as follows:

```
long long  weight;
```

The type "long long" guarantees an integer that is at least 64-bits long. When using scanf and printf, use the formatting code "%lld":

```
scanf("%lld", &weight);
printf("%lld\n", weight);
```

Note that you cannot use scanf to input the textual phrase, since it may contain more than one word. Use fgets( ), but remember to strip off the EOL characters:

```
char text[512];
int  textsize = sizeof(text) / sizeof(text[0]);

fgets(text, textsize, file);
text[strcspn(text, "\r\n")] = '\0';  // strip EOL char(s):
```

The other input issue to worry about is that there may be one or more blanks and tabs between the weight and the text phrase. This means fgets( ) will input this "whitespace" and make it part of the text phrase. You need to advance past the whitespace and obtain a pointer **cp** to the first non-whitespace character:

```
char *cp = text;

while (*cp == ' ' || *cp == '\t')  // advance past whitespace:
  cp++;
```

The pointer **cp** now points to the start of the string that you should duplicate and use as the key to order your tree; it would be wise to also store this pointer, along with the weight, as part of BSTValue so you can easily obtain the text and the weight.

Getting Started

The course web page contains sample input files, along with some initial code in "main.c" to help you with the keyboard I/O. Add the "bst.h" and "bst.c" files that you have been developing. Other than that, the same suggestions apply: start slowly, build step by step, and apply what we've been working on in class, homework, and projects.

Submission

The program is to be submitted via zyLabs: see "**Project03 Auto-Complete**", section 4.12. Submit your three files in the tabs provided: main.c, bst.h, and bst.c. When you're ready to submit, copy-paste your program into the zyLabs editor pane, switch to "Submit" mode, and click "SUBMIT FOR GRADING".

Keep in mind this is only a first level grading mechanism to ensure you have (a) submitted the correct files, and (b) are following the basic requirements and passing some simple tests. The grade you receive from your zyLabs submission is only a preliminary grade. After the deadline, the TAs will then manually review and test each program for the following:

- adherence to requirements
- overall design
- correctness against additional tests (involving other input files)
- commenting, indentation, whitespace and readability

Policy

Late work *is* accepted. You may submit as late as 24 hours after the deadline for a penalty of 25%. After 24 hours, no submissions will be accepted.

All work is to be done individually — group work is not allowed. While I encourage you to talk to your peers and learn from them (e.g. via Piazza), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is described here:

http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.pdf .

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at http://www.uic.edu/depts/dos/studentconductprocess.shtml .