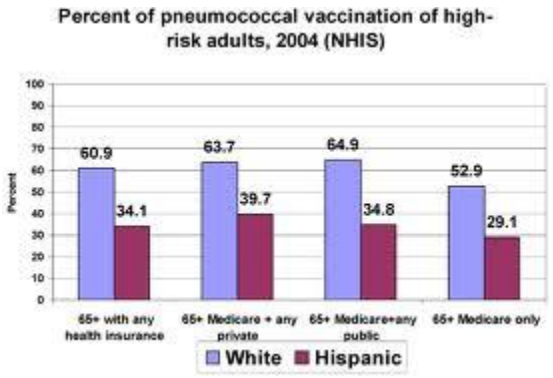
CS246 Asgn03. Cluster Bar Chart V1.0

In this assignment, you are to write a C++ program to produce *clustered bar chart.*

Clustered bar charts. Here is an example of a clustered bar chart from
http://www.academyhealth.org/ahrq/elders



This chart has a title: *Percent of … (NHIS).* It has 5 clusters each consisting of 2 bars. The length (such as 60.9) is given for each bar. In this example, the bars are drawn vertically. Each kind of bar has a label, e.g., blue bars have the label *White.* Each cluster has a label, e.g., *65+ with any health insurance.*

This chart is drawn using pixel graphics. In this assignment you program is to produce equivalent clustered bar charts drawn using ASCII characters instead of pixels with the bars drawn **horizontally**. You are provided with example input data that your program is to read and the example horizontal ASCII charts that you program is to produce. See "Provided files" at the end of this write-up for instructions to find these examples.

Practice with C++ features. This assignment provides practice in various features of C++ such as the abstract types *vector* and *string*. These two are abstract in that they not part of the actual C++ language, but instead are implemented using templates, which are prepended to your program using “includes”. As a result of these *includes*, it *appears* to the programmer that vectors and strings are part of a superset of C++.

Assertions. In your program, use at least one *assert* in a useful, rational way. Be sure to write
#include <assert.h>
Exits. In your program, use at least one *exit* in a useful, rational way. Be sure to write
#include <cstdlib>
IO manipulators. In your program, use at least one *IO manipulator* in a useful, rational way. Be sure to write
#include <iomanip>
C++ strings. In your program, use C++ strings and not (except when necessary) C strings. Be sure to write

#include <string>

C++ vectors. In your program, use C++ vectors and not C arrays. Be sure to write
#include <vectors>
In your program, when subscripting of any vector A, use the safe notation A*.at(subscript)* and not the unsafe notation A*[subscript].* You should use vectors of strings to hold data such as cluster labels and bar labels. You should use vectors of chars to hold the characters to represent bars. You should use a two dimensional vector V, to store the values of the bars within clusters. You will need to create (declare) V with a syntax similar to the following:

vector < vector < double > > V; // Leave a space between the two '>' characters.

Use a meaningful name (not V). You can set (*resize* or *pushback*) the number of rows and columns in V so that V is large enough to handle reasonable charts.

Software architecture. Software architecture refers to the overall organization of a program. It is suggested that for this program you use a simple organization that matches the following description. Use a single C++ file to contain all of your variables and functions, i.e., to contain your whole program. Declare all (or almost all) of your variables outside of (between) functions. As a result, your variables become *globals.* While this sort of organization is not viable for large programs, it is convenient for very small programs.

Programming style. In this assignment concentrate on giving meaningful (or mnemonic) names to your functions, variables and constants . There are exceptions such as: In *for* loops, counters may be given simple names such as *i* and *j.*- Example meaningful name:

String changeUnderscoresToBlanks (string s)

This function changes underscore characters in a string to blanks. Example meaningful name:

int maxBarLabelWidth;

This variable contains the length (in terms of characters) of the longest bar. Use many simple, easily understood, well named function, such as:

void readClusterLabels ()

This function inputs all of the labels of clusters. Avoid using *magic numbers* such as 12 or 45 except when initializing named constants.

Tags. The input files that your program reads contain *tags* such as <numBarsPerCluster>. The purpose of these tags is to make the input easier for a person to read it. It also allows your program to do a small amount of checking of the input data (by seeing if a tag appears in the input where it is expected). Your program should read these tags and discard them, checking that the discarded tag was expected, as per the example input files provided to you. However, if your program is expecting to read a non-tag (such as a cluster label) you can assume that the input is as expected.

Underscore characters. In order to simplify reading, the strings being read contain no blanks. Instead, they contain underscores. You should change any such underscores to blanks before outputting them.

Only positive bars. The length of each bar will be non-negative. You should draw your graphs such that zero is the leftmost end of the bar.

Number of characters in bars. The length of a bar (the number of characters in the bar) should be based on its given length, then scaled (and finally rounded) according the longest bar. You will need to search for the bar with the greatest length.

Find chart on web. As a warm-up to this assignment, you are to find a clustered bar chart of your choice on the web. Translate it to the input format illustrated by the examples given to you. Name your input: **webexamp.chin**. You are to then create the correct output for your input. Name this output file **webexamp.chout**. Zip both files and name this zip file a3p1.zip and submit the file to marmoset.

Naming. Name your C++ program *chart.cpp* with its executable version named *chart*. Data to be read by your program is to be named *XXX.chin* and your corresponding ASCII chart is to be named *XXX.chout*.

File I/O. Your program should read from the standard input and write to the standard output (and to the standard error when appropriate). Use redirection so your program actually reads from XXX.chin and writes to XXX.chout for various XXX.

Provided files. These files contain ASCII charts (suffix chout) and their corresponding data (suffix chin). Study the chin files to understand the format of input data for your program. Make copies of them using the following commands.

```
cp ~holt/cp-for-students/cs/246/2013/asgn03/pneu.chin .
cp ~holt/cp-for-students/cs/246/2013/asgn03/pneu.chout .
cp ~holt/cp-for-students/cs/246/2013/asgn03/perform.chin .
cp ~holt/cp-for-students/cs/246/2013/asgn03/perform.chout .
cp ~holt/cp-for-students/cs/246/2013/asgn03/trivial.chin .
cp ~holt/cp-for-students/cs/246/2013/asgn03/trivial.chout .
```

Marks.
10 marks for a3p1.zip
40 marks for chart.cpp