customized C++ tools for CIS 130

- **NOTE #1:** These tools are all currently available on nrs-labs.
- **NOTE #2:** For most of these tools, note that you should be *in* the directory you want to work in *before* you call these tools. That is, use the `cd` (change directory) command to navigate to the desired directory/folder BEFORE you use these. (Remember that `cd ..` lets you move "up" a directory, and `cd` by itself takes you back to your home directory.)
- **NOTE #3:** This handout assumes that you have copied all of the tools below *except* for `~st10/130submit` into your `bin` directory on nrs-labs. See **Appendix 1** at the end of this handout for how to do this, if you haven't yet.
- **NOTE #4:** The GNU C++ compiler, gcc/`g++`, includes the **line number** where it got confuzzled near the beginning of its error messages (usually right after the file name). In nano, you can go right to a particular line number within a file by typing **^W** (where-is/search), then **^T** (enter line number), then typing the desired line number and then typing the ENTER key.

expr_play

USE WHEN:

...you want to experiment with C++ expressions (including function calls of already-written-and-compiled C++ functions)

KNOWN QUIRKS:

- To test a function, you need to type in the names of all functions that that function calls (or that functions it calls call...!)
- You may get odd results when using this with **`void`** functions (functions that do no return any value) and with functions that print to the screen.
- The current version of this tool leaves the little C++ programs that it builds to do its work in your current working directory -- I left them in deliberately (rather than deleting them) in case some of you might find them to be interesting examples of tiny C++ programs. It is certainly safe to remove these little files `try_expr1.cpp`, `try_expr1`, `try_expr2.cpp`, `try_expr2`, etc. that it builds and leaves in your directory.

funct_play2

USE WHEN:

...you want to type in a C++ function from "scratch" using the design recipe.

KNOWN QUIRKS:

- It is a Perl script -- you **cannot** back up to a previous step. If you are a good ways in, I'd advise simply finishing and then editing the resulting .cpp and .h files as desired. Then, you can use **funct_compile** (see below) to compile and test the result.
- If you make a typo in a **constant declaration** and/or **function header**, you may need to edit **both** the function's `.h` and `.cpp` files. Otherwise, you should usually just need to edit the function's `.cpp` file.
- If you aren't a good ways in and you want to start completely over, type ^C (ctrl key and c key

together) to just exit. Then you can simply run `funct_play2` again.

- if a function will call other functions, be sure to type the *names* of those functions when prompted.
- this also builds an examples-tester program from the examples you type in when prompted. This program's file is the name of the function followed by **_ck_expect.cpp** -- that's what you should edit if you make a typo in your examples. **funct_compile** will try to compile it if it isn't already there.

funct_compile

USE WHEN:

...you simply want to recompile and test an already-written-and-compiled C++ function in the current working directory. It will also try to compile the tester program for the function (function name + **_ck_expect.cpp**) if it can.

KNOWN QUIRKS:

- if a function calls other functions, you need to type in the names of all such functions when prompted for them.

~st10/130submit

USE WHEN:

...you want to submit homework files. You can simply submit all of the **.cpp** and **.h** files in the current working directory by answering that question with a `y` (for yes) when prompted.

A "receipt" of your submission will be placed in a directory named **submitted** in the current working directory. Remember to keep this "receipt" until the grades for a homework have been posted to the course Moodle site.

KNOWN QUIRKS:

- IF you exit this before the end using ^C, YOUR FILES MIGHT NOT BE SUBMITTED - beware!
- See the tool below, **unziptar_all**, if you'd like to see copies of the files you've submitted.
- Remember that I don't mind if you submit MULTIPLE VERSIONS of your homework files - unless I hear from you otherwise, I simply grade the **latest** version turned in **before the deadline**. This is to encourage you to play with your code!

unziptar_all

USE WHEN:

...you want to look at the files in your "receipt" created by **130submit**. You want to **cd** into the **submitted** directory, and THEN run this.

The zipped and tarred receipt file will be unzipped and un-tarred into a directory containing copies of your submitted files. **ls *** is a quick-and-sleazy way to see the names of these files --- **more */*** is a quick-and-sleazy way to page through the contents of the submitted files.

KNOWN QUIRKS:

- Be sure to run this while **in** the **submitted** directory containing your zipped-and-tarred "receipt" file.

APPENDIX 1 - installing these tools in your `bin` directory on nrs-labs

Most of these tools (except for `~st10/130submit`) are easier to use if you have made copies of them in your nrs-labs account, in a special directory named `bin`.

(Note that, if you don't want to do this, all should work just fine if you precede their names with `~st10/` -- that is,

```
~st10/expr_play

~st10/funct_play2

~st10/funct_compile

~st10/unziptar_all
```

...should all work fine from any directory on nrs-labs.)

Step 1:

`ssh` to, and log into your account on, `nrs-labs.humboldt.edu`

Step 2:

See if you have a `bin` directory in your home directory on nrs-labs, and create it if not.

```
[you@nrs-labs ~]$ ls bin
```

If the above command results in the message:

```
ls: bin: No such file or directory
```

...then create a `bin` directory (and protect it) with the commands:

```
[you@nrs-labs ~]$ mkdir bin

[you@nrs-labs ~]$ chmod 700 bin
```

Step 3:

Copy over these tools into your new `bin` directory using the following commands:

```
[you@nrs-labs ~]$ cp ~st10/expr_play bin

[you@nrs-labs ~]$ cp ~st10/funct_play2 bin

[you@nrs-labs ~]$ cp ~st10/funct_compile bin

[you@nrs-labs ~]$ cp ~st10/unziptar_all bin
```

Step 4:

You need to add some lines to a certain file, named `.bashrc`, in your home directory, so that nrs-labs knows to look for these programs in your `bin` directory. First, open up this special file using the text editor of your choice -- I'm using `nano` below:

```
[you@nrs-labs ~]$ nano .bashrc
```

Using the down-arrow key, move down to the bottom of this file `.bashrc` -- do not remove what is there, just type in or paste the following two lines:

```
# User specific aliases and functions
PATH=$PATH:$HOME/bin:.
```

The first line is just a comment -- as long as you put the #, anything will work there. But the second line needs to be typed in *exactly* as shown.

Save the modified `.bashrc` file (using ^O in `nano`), and then it is safe to exit the editor (using ^X in `nano`).

Either log out and `ssh` to nrs-labs again, OR type the following command, to execute this file `.bashrc` for the first time (`.bashrc` will executed for you every time you `ssh` to nrs-labs from now on):

```
[you@nrs-labs ~]$ source .bashrc
```

Step 5:

Now you should be able to run these commands in any of your directories on nrs-labs by simply typing:

```
expr_play

funct_play2

funct_compile

unziptar_all
```