CSI 402 Systems Programming SUNY at Albany Spring 1999
Professor Chaiken Proj 1-Part 1 Friday, Jan. 28 Due Wednesday, Feb. 3

Readings: To complete this part, you will have to these terms and how they relate to assembly language: symbol, undefined and defined symbols, symbol table, location counter, operation codes, labels, directives, etc. Read material from your CSI333 text or Beck Chapter 2 as necessary. Also, read about the SPARC assembler (2.5.2).

Over the next 2 weeks, read Beck Chapter 1: 1.1-1.2 for orientation, 1.3 only as needed to understand illustrations with SIC, 1.4.2 Pentium Pro, 1.5 on RISC especially 1.5.1 on SPARC. Chapter 2: items above, the SIC object program format (Fig. 2.2), the assembly algorithm (Fig. 2.4a-b), and relocation (section 2.2.2; relocation will be covered in detail during the course.

Part 0 DO NOW!!

1. Get an account on the University at Albany’s central UNIX cluster (“CSC”) if you do not have one already.
   
   Your account for using samson, moses and joshua is NO GOOD for csi402. You must be able to remotely log into eve.albany.edu and to use the “veggie” SPARC Ultra machines in LC-4.

   If you are registered in the course you can use CAAPS to get an account. If you are not registered, register right away and then use CAAPS to get an account the next day. See me if you cannot do this.

2. Make sure you know how to use the standard UNIX commands: man, ls (or dir), mkdir, cd, pwd, cp, mv (and you know the difference), logout, cat, less (this is the GNU improved version of more), rm (Never, never count on getting deleted files back!), rmdir, lpr, etc. You will learn others. The first source way to get information on a command is to run man <command name>; for example, you can always begin with man man. One can also try man -k <some keyword>. For example, to locate commands relevant to files, try man -k files. Also, make sure that you can use emacs. See the “Quick introduction to emacs” linked from the class home page.

3. Make sure that you can browse the World Wide Web. To get started, go to LC-3 or LC-4 get any assistance you need. Web browsers are also available in the Library and other locations around campus. When you work on a “veggie” machine, you can use netscape. When you log in remotely to eve with a terminal emulator, you can use Lynx. The course home page is http://www.albany.edu/~csi402

   If you use your own computer, make sure that you can view Postscript (.ps) and Adobe Acrobat (.pdf) documents.

4. Make sure that you check the course conversation and announcement newsgroups sunya.class.csi402 and suny.class.csi402.announce at least twice a week. You can read news using M-x gnus in emacs. Also, make sure that you can send and receive email. The email address for your account at the CSC is username@csc.albany.edu, where username is the letter and number combination user name assigned by CAAPS.
Part 1-Turnin due 7:30 PM Wednesday, February 3.

The goals of Part 1 are to practice building multiple module C++ programs, to learn about revision control and the required procedures for submitting CSI402 project work, and to see how symbols representing functions and global variables are referred to by some modules (C/C++ code, assembly code and object file symbol tables) and defined in others.

**FOLLOWING INSTRUCTIONS:** Project instructions are given to teach specific skills and to provide laboratory procedures that work and don’t waste your time. If you do not follow them accurately, you risk losing significant amounts of points for the project (and eventually your final grade).

If you run into difficulties, resolve them right away by seeking the instructor or the TA, the CSC helpdesk (LC-27) if the problem is system-related, or posting a question on the newsgroup.

1. Make a new subdirectory for your Project1 work and copy all the .cc and .h files, plus build.sh, from ~csi402/pub/Project1/demo into it. Write a list of all those files’ names in your notebook.

2. Make a subdirectory of your Project1 work directory called RCS. Do this by using the cd command to go to your Project1 work directory; then do the command “mkdir RCS”.

3. For each of the files you copied, execute the command

   ```
   ci -l <filename>
   ```

   where <filename> is the name of the file. Each time it will prompt you for a description. Type in a one line description of the file.

   **EXPLANATION:** You are to maintain a revision control database for every source, header, Makefile and other files that go into your project work for this course. Revision control systems are part of the software engineering process requirements of many professional software teams\(^1\). A primary benefit of revision control is that it maintains backup copies of previous versions so that if you mess up a file when you try to add or debug a new feature, you can back it up to earlier, stable version and begin again.

   Another benefit is that it helps maintain a log in which you can record design decisions, bugs, new functionalities, etc., tied to the code itself. This provides an “audit trail” of the development process.

   For CSI402, you are required to use the standard, freeware “RCS” system available on the CSC systems and most Linux distributions.

   The minimum, basic command you need to know is “ci -l <filename>” where <filename> is the name of a file in the current directory. What this command does “check-in” the current version of the file into the RCS database file named RCS/<filename>,v and

---

\(^1\) Teams find it useful because locking the file by one person in order to edit it prevents another person from editing the file at the same time.
keep the file available for you to edit further. (Specifically, `ci` -1 means do an ordinary “check-in,” which registers a new version (except if it is identical to the current version), followed by a locking “check-out” so that you can modify the file to create the next version if you want to.)

Once an RCS database is set up, you can read the log of a file with the command “`rlog <filename>`”.

To replace the file you are working on with an earlier version, do “`co -rn.m <filename>`” where n.m is the version number that you want to go back to. (get that number from rlog.) Tell co to remove a writable file. Then make the checked out file writable by using `chmod`. Checking it in locked will now make it the latest version.

When you are ready to turn in your project work, (1) make sure the final versions of all the files that comprise your project are “checked-in,” (3) compose a README file and put it in your project work directory if there are issues you want the graders to be aware of, and (3) run the turnin command to submit the RCS directory and all of its contents, and the README file if you made one:

cd <your project directory, containing the RCS subdirectory>
ls RCS/* README
# here, make sure there is a .v file for EVERY file you want to submit
turnin-csi402 -p proj1-part1 RCS/* README
turnin-csi402 -p proj1-part1 -v

The last command very important to verify that we received all the files that you intend to submit. Unfortunately, the turnin command does not report if some of the files you specify are not found.

Projects MUST be submitted as RCS database files plus a README, and MUST compile and link successfully in order to get any credit at all!! Make absolutely sure ALL source files and ALL your header files they “include” are submitted.

Now to continue your project1 work:

4. Run the command “`g++ -E`” on each of the .cc files. This command makes the preprocessor run on the specified file and output the result to standard output (your terminal unless you redirect it). If too much goes by to see, pipe it to a pager; for example,
g++ -E demo.cc | less

Observe how the preprocessor inserts a “copy” of each included file into the given file. Is it really a copy? What about comments? Why should comments be removed?

Can you think of uses for the C/C++ preprocessor besides building software in C/C++?

5. Edit `demo.cc` so that the first thing the program prints

Project 1 Skeleton
is replaced by
CSI402 Project 1 work by <your name>, February 1999.
where of course <your name> is replaced by your real name.

6. Compile each module separately using the g++ -g -c command; edit or copy in missing header files if necessary to get successful compilations. Finally, link the object files with the command g++ -o part1 <the object files>. (Here of course <the object files> is the sequence of object (".o") file names of the object files to be linked to build the executable file to be named project1.
Do what is necessary to get a successful link step.

7. Check in all source files you modified. To make sure, you can check them in with the ci command with no options, eg ci main.cc. This will make the checked in file disappear from the working directory. (You can use co -l <filename> to make the file re-appear.)

8. (10 points) Add code into demo.cc to recognize a command of the form delete <key>.

9. (15 points) Add a member function the StringDict class that will delete the record with a given key from a StringDict. Add member functions to Listof2CharPtrs to support this. Make the delete command\(^2\) call invoke the deletion.

The way member functions refer to components of struct or classes is explained in the lecture. If you do not understand the code, you should review your data structures notes and text about operating on singley linked lists.

You need one fact about referring to struct data members by a member function: Within the scope of the function body, the name of the data member refers to the data member of the object (i.e., data type instance) for which the member function is called.

The demoC version shows how the access to data members works in terms of plain C.

I suggest you implement deletion by writing one member function for Listof2CharPtrs similar to lookup but will return the pointer to the preceding list node when the record with the given key is found. You will need to know that the reserved C++ word "this" is a pointer to the object for which the member function was called. For example, if the node containing the given key is the first after the “dummy” node in the StringDict object, the value of “this” should be returned.

Write another to actually delete the Listof2CharPtrs object. Make StringDict responsible for reclaiming the storage for the key and value strings.

Check in each file after each successful compilation of a new version.

10. (15 points) Make the StringDict class behave reasonably and do nothing if the key to be deleted is not there.

\(^2\)New commands, like the commands already in the skeleton, should ignore excess tokens on the command line.

4
11. (20 points) Add the “dump” command that will print all the key/value pairs in the
dictionary, in the order they appear in the linked list, using the format given in the
skeleton program.

12. (20 points) Before the project is due, whether or not your code is debugged, prepare a
version with member functions you added which compiles and links successfully, and
prints your name as specified above when it is run. Check it in.

13. (20 points) Run “g++ -S ” on your final demo.cc, StringDict.cc and
Listof2CharPtrs.cc files; observe the “.s” assembly language files that are produced.
Also, run nm and3 “/usr/local/binutils/objdump --sym” on each of the three
object (“.o”) files from your last step.
Write a report into a file named README based on the assemblyA and symbol table
output: For each member function (including constructors) of the two classes, report
(a) the C++ “mangled” symbol name,
(b) the ordinary C++ name and signature,
(c) which object file defines the symbol and what value (relative to the .text section
beginning) it has,
(d) in which object file(s) is that symbol undefined.

14. Now, submit the RCS directory and the README file by following the instructions
above.

**LATE POLICY for proj1-part1:** Due time is 7:30 PM Wednesday, February 3.
Late submissions will count according to the following formula: $G = \max(0, OG \times (2 - ND)/2)$ where $OG$ is the grade you would have gotten had the submission been on
time, $ND$ is the number of days past 7:30 PM Wednesday, and $G$ is the grade that
your project will get, including the late penalty. That means if your submission is 1
hour late, you get $(2 - 1/24)/2 = 1 - 1/48$ or about 2% off. But if it is 24 hours late,
you get $(2 - 1)/2 = 1/2$ or 50% off.

15. Observe the symbol table of the executable file. What is the nature of the symbols
in it that are undefined? Can you imagine how an executable file can have undefined
symbols?
Run “ldd” to help find out..

---

3Tip: Run setenv PATH /usr/local/binutils:$PATH to make your shell locate objdump without you
typing the full path name each time
4You can do a search for “assembly language” on Sun’s documentation Web server
http://doc.sun.com to find the SPARC assembly language manual which some of you might want for
deciphering the assembly language you will see in this assignment. This url is referenced from the “Textbooks
and other Reading Resources” link on the class home page. However, for the purposes of Project 1, it suffices
to make a good guess about what the .s listing means based on your general knowledge of assemblers, machine
instructions, symbols, etc.