# Answers to Even-Numbered Exercises 16

*from page 891*

1. What function does every C program have? Why should you split large programs into several functions?

2. What command could you use to compile **prog.c** and **func.c** into an executable named **cprog**?

```
$ cc -o cprog prog.c func.c
```

3. Show two ways to instruct the C preprocessor to include the header file **/usr/include/math.h** in your C program. Assuming that the **declar.h** header file is located in the subdirectory named **headers** of your home directory, describe two ways to instruct the C preprocessor to include this header file in your C program.

4. Both C functions, **getchar** and **putchar**, appear in the standard C library **libc.so**. Show that **getchar** and **putchar** are also macros on your system. Can you think of more than one way to show this?

```
$ egrep 'getchar|putchar' /usr/include/*.h | grep define
```

5. How are the names of system libraries abbreviated on the gcc command line? Where does gcc search for libraries named in this manner? Describe how to specify your own library on the gcc command line.

6. What command can you use to create an RCS file for a file named **answers**? What command can you use to retrieve an editable version of **answers**?

```
$ ci answers
$ co -l answers
```

7. Write a **makefile** that reflects the following relationships:

   a. The C source files **transactions.c** and **reports.c** are compiled to produce an executable **accts**.

   b. Both **transactions.c** and **reports.c** include a header file **accts.h**.

   c. The header file **accts.h** is composed of two other header files: **trans.h** and **reps.h**.

8. How can you retrieve the RCS working file **answers** so that

   a. Only Barbara and hls can make changes?

```
$ rcs -e -abarbara,hls answers
```

   b. No one except the owner and Superuser can make changes to Release 2?

```
$ rcs -e -b2 answers
```

9. If you retrieve Version 4.1 of the file **answers** for editing and then attempt to retrieve the same version again, what will RCS do? Why is RCS set up this way?

10. Answer the questions from exercise 9 for CVS.

    When you retrieve Version 4.1 of the file **answers** for editing and then attempt to retrieve the same version again, CVS does nothing. If you do not change the file between the two retrievals, the file is still checked out and ready for editing. If you do make changes to the file after you check it out and attempt to check it out again, CVS gives you a message that starts with `M` (modified) which indicates that the file has been modified but not checked in; the file is still checked out with the changes you made.

    If you want to start over again with the original version of the file, you can remove the file (**rm answers**) and then run **cvs update** to check out a new version.

## Advanced Exercises

11. What RCS commands modify the access list? Explain how to ensure that only the owner of a file or Superuser can check out a locked revision of an RCS file.

12. Modify the **badtabs.c** program so that it exits cleanly (with a specific return value). Compile the program, and run it using the dbx or another debugger. What values does the debugger report when the program finishes executing?

    Put the following code at line 33:

    ```
    return 0;
    ```

    The **badtabs** program will, when its other problems are fixed and it is compiled, exit with a return value of 0; this value should be reflected by any debugger.

13. For the following makefile

    ```
    $ cat Makefile
    leads: menu.o users.o resellers.o prospects.o
                gcc -o leads menu.o users.o resellers.o prospects.o

    menu.o: menu.h dialog.h inquiry.h

    users.o: menu.h dialog.h

    prospects.o: dialog.h
    ```

    Identify:

    a. Targets

    b. Construction commands

    c. Prerequisites

14. Refer to **Makefile** in exercise 13 to answer the following questions:

    a. If the target **leads** is up-to-date and you then change **users.c**, what happens when you run make again? Be specific.

    The **users.o** file would be rebuilt using an implicit dependency from **users.c**, and then gcc would relink **leads**.

b. rewrite the makefile to include the following macros:

```
OBJECTS = menu.o users.o resellers.o prospects.o
HFILES = menu.h dialog.h
```

```
$ cat Makefile
OBJECTS = menu.o users.o resellers.o prospects.o
HFILES = menu.h dialog.h
leads: $(OBJECTS)
            gcc -o leads $(OBJECTS)

menu.o: $(HFILES) inquiry.h
users.o: $(HFILES)
prospects.o: dialog.h.
```

15. Read about make on page 1247 in Part III and the make man page to answer the following questions:

    a. What does the **–t** option do?

    b. If you have files named **makefile** and **Makefile** in the working directory, how can you instruct make to use **Makefile**?

    c. Give two ways to define a variable so that you can use it inside a makefile.

16. Suppose that the file named **synchr.c** has four revisions numbered 1.1 through 1.4. Show how to

    a. Check out the latest revision for editing.

    ```
    $ co -l synchr.c
    ```

    b. Check out the latest revision for compiling only.

    ```
    $ co synchr.c
    ```

    c. Check in a new revision after editing the latest revision, but allow editing of the working file to continue.

    ```
    $ ci -l synchr.c
    ```

    d. Check out revision 1.2 for editing.

    ```
    $ co -l1.2 synchr.c
    ```

    e. Delete revision 1.2.

    ```
    $ rcs -o1.2 synchr.c
    ```

17. Read about the RCS system, or experiment with it, to answer the following questions:

    a. How do you assign a symbolic name to a revision?

b. How is a branch revision created?

c. What happens when you attempt to check in an editable revision that has not been modified?

d. How do you delete a revision? Can this cause other revisions to be renumbered?

18. Refer to the makefile for **compute** on page 853. Suppose that a file in the working directory is named **clean**. What is the effect of giving the following command. Explain.

```
$ make clean
```

If none of the targets to be cleaned (***.o**, **core**, and so on) exists, this command terminates with an error and removes the file named **clean**.

If some of the targets exist, this command displays `Make: `clean' is up to date.` and exits.

The discussion of the makefile on page 851 states that the following command is not normally seen in makefiles:

```
cat num.h table.h > form.h
```

a. Discuss the effect of removing this construction command from the makefile while retaining the dependency line.

Removing the construction while keeping the dependency causes the **form.h** dependency to have no effect.

b. The preceding construction command works only because the file **form.h** is made up of **num.h** and **table.h**. More often **#include** directives in the target define the dependencies. Suggest a more general technique that updates **form.h** whenever **num.h** or **table.h** has a more recent modification date.

There are several techniques.

You can use proper includes: You can include either **num.h** or **table.h** on the **length.o** line.

You can define a macro:

```
INCLUDES = num.h form.h table.h
length.o: length.c $(INCLUDES)
```

You can split **length.o** into two dependencies:

```
length.o: length.c form.h
length.o: table.h num.h
```

Finally, you can cause **form.h** to be updated when **num.h** or **table.h** is newer than **form.h**.

```
form.h: num.h table.h
     @touch $@
```