Programming Guidance

John Dempsey
COMP-232: Programming Languages
California State University, Channel Islands

Before you turn in your work, take a step back and ask yourself is my work correct.

Here are general comments with regard to student work previously turned in.

1. Programs and functions should follow this format:

```
#include <stdio.h>                  ← List system include files first.

                                    ← Blank line

#include "myinclude.h"              ← List user include files.

                                    ← Blank line

#define MAX_VALUE    25             ← List all defines

                                    ← Blank line

int functionA(int i);               ← List function prototypes, if any, in alphabetical order.

                                    ← Blank line

FILE     *fd_input;                 ← File descriptors defined.

                                    ← Blank line

struct node_struct { }              ← Structure definitions

                                    ← Blank line

int       i;                        ← Variable can be initialized, if needed.
int       month=2;                  ← List variables in alphabetical order.
int      *ptr;                      ← Star should stand out instead of being lined up.

function max_value()                ← Function calls in alphabetical order.
{ … }

main(int argc, char *argv[]) {}     ← Main program at bottom.
```

2. All of the above should be defined in alphabetical order and on their own line, with the one exception that #include <stdio.h> should be the first line.

3. Don't define i in the for loop itself: for (int i=0; i<10; i++) { …}. Define the i variable with the other variables.

4. If you have an array such as keywords and you read in a file to initialize the array keywords, one approach is to have the number of keywords loaded into an integer named keywords_count. Then you can print all keywords using:

```
for (i=0;i<keywords_count;i++) {
    printf("%d. %s\n", i, keywords[i]);
}
```

5. Indenting. I create a ~/.vimrc file defined with the following lines:

```
set noswapfile
set tabstop=4
set shiftwidth=4
```

Then you can vi a file and type:

```
gg=G
```

This will indent the source code using 4 blanks per indentation.

6. If you ask a question, like "What year is it?" You need to put a space between the question mark and the user's answer. You don't want the user to see: What year is it?2023.

7. Do not do this when defining FILEs:

```
FILE    *fd_in = fopen("input.txt", "r");
FILE    *fd_out = fopen("output.txt", "w");
```

When opening a file, you need to check to see if NULL was returned, indicating an error has occurred. And if an error does occur, you need to print out a short error message indicating the file name and telling the user the program will now exit. Here's an example:

```
if ((fd_in = fopen("input.txt", "r")) == NULL) {
    perror("ERROR: Cannot open file input.txt for writing.");
    printf("Program will now exit.\n");
}
```

If fopen does return a NULL, you can check the errno value to print the actual error message as to why the file couldn't be opened or use perror() instead.

8. If the program errors out for any reason, instead of using "return 1" only, you should print out a helpful message indicating the problem before exiting, like:

```
printf("Cannot fork process.\nProgram will now exit.\n");
exit(1);
```

9. When using case statements, instead of having two case statement repeated, a cleaner way is to use:

```
switch (toupper(value_entered)) {
   case 'F':
        printf("Full\n");
        break;
   …
}
```

10. Make sure you initialize your variables and pointers before use.

11. Use // instead of /* comment */. That way you can comment out an entire group of code at once, if desired.

12. If you are given examples of what your program should do, you should run your programs using these examples and check your answers before turning in your work.

13. I like to put my main() program at the bottom of the file such that when editing with vi, I can type 'G' to go to the bottom, then '%" to match brackets {}, which then display main. Being that functions are above main, you may not need define function prototypes for functions inside the file.

14. If a program requires parameters on startup and no parameters are provided by the user, you should print out a usage message, such as:

```
Usage:
        retire age amount
where:
        age      – How old you are when you start saving.
        amount – Dollar amount saved per month.
Please try again.
```

15. I prefer function call parameters defined with their data types like:

```
int max_value(int num1, int num2)
{ …
```

versus:

```
int max_value(num1, num2)
int     num1;
int     num2;
{ …
```

Less lines of code plus, if needed, you can copy the function call and easily use it as a prototype definition by just adding the semicolon to the end.

16. If the printf statement is very long and wraps, make sure you make it more readable.

Instead of writing this all on one line:

printf(" Name=%s, Address=%s, City=%s, State=%s, Zip=%d\n", person[index].name, person[index].address, person[index].city, person[index].state, person[index].zip);

Make the line more readable, more maintainable, and easier to spot an error by writing the same printf using this approach instead:

```
printf(" Name=%s, Address=%s, City=%s, State=%s, Zip=%d\n",
        person[index].name,
        person[index].address,
        person[index].city,
        person[index].state,
        person[index].zip);
```

17. Save backups daily on small programs. For RepoLeaf, I'm running for October 11th:

% **mkdir 11** ← where 99 is today's day number, like 11 for October 11.
% **cp * 11**

% **mkdir -p BACKUP/OCTOBER** ← Once a month, move files to BACKUP/MONTH, like OCTOBER
% **mv [0-3]* BACKUP/OCTOBER**