2. First Programs

2.1 Some Settings For vi Editor

If you decide you will use the vi editor for your programs, there are a couple of settings that you can implement that can make your life easier.

Setting Automatic Tabs To set automatic tabs, follow the instructions below. This will automate the indentation of your files. At your root directory, open up the file `.vimrc` for editing. Add the following lines:

```
set autoindent
set smartindent
set tabstop=3
set shiftwidth=3
syntax on
```

then save the file and type `source .vimrc` at the Unix command for it to take effect (or log out and log back in for it to take effect).

You can Google ".vimrc settings" and probably find other useful settings – let us know if you find a setting that you think will be really helpful!!

Color-Coding the Text If you are using SSH (for Windows machines) and the text in your files is not color coded in Vim, try the following. Choose "Edit" on the menu bar, and then "Settings". In the Settings Window, under "Profile Settings" on the left hand side, click on "Connection". Then on the right hand side, in the box next to "Terminal Answerback", choose `vt100` from the drop down list.

2.2 *Programs From Chapter 2*

Program 2.1 revisited:

The following shows Program 2.1 from page 11 in the book with a few added lines of code, along with comments that explain the code.

C Program:

```
/*  prog2_1.c
    this is the very first program in the book
    Program 2.1, on page 11
    which is explained in detail in Chapter 2
*/

#include <stdio.h>

int main (void)
{
    printf("\nProgramming is fun. \n");

    return 0;
}
```

Note from above things like:

- header comment at the top (comments are ignored by the compiler)
- indentation – notice that the lines inside the main function are indented
- semi-colons at the end of statements (if missing a semi-colon where one is required – that's what type of error??)
- curly braces surrounding block of code inside of main function
- blank lines in certain places to break up groups of lines of code, for easier readability

Developing Program 2.1

Following the steps outlined on page 4/slide 14 of the Chapter 1 notes/slides:

1. Using an editor, such as vim or pico or gedit, type up the code in a file; save it with a `.c` extension, such as `prog2_1.c` keeping in mind the things noted above (header comment, indentation, etc.)
2. Compile the program using `gcc`. If you named it as suggested in step 1, type: `gcc prog2_1.c` at the command prompt.
3. If there were errors, you will see them listed to the screen. They must be fixed and the program re-compiled to produce an executable.
4. When there are no errors, an executable file called `a.out` will have been created. Type the Unix/Linux command `ls` at the command prompt to verify that `a.out` exists.
5. To run the program, just simply type at the command prompt: `./a.out`

6. **Reminder #1:** when you type `gcc prog2_1.c` at the prompt, the preprocessor runs, then the assembler, and then the linker. These intermediate files created through those steps are not saved to a file, unless you use a flag with your `gcc` command to tell it to capture and save those files. If you want to capture the preprocessor results, use the `-E` flag with a redirect to a filename for the results to be saved to; to capture the assembly language file, use the `-S` flag; to capture the object code, use the `-c` flag.

`gcc -E prog2_1.c > ppf`	will create file called `ppf` containing preprocessor results (will be a `.i` file)
`gcc -S prog2_1.c`	will create `.s` file (assembly code)
`gcc -c prog2_1.c`	will create `.o` file (object code)

   Or you can use this one flag to save all three of those files: `gcc -save-temps prog2_1.c`

7. **Reminder # 2:** to rename the executable to something other than a.out, use the `-o` flag and provide a new name for it, such as: `gcc prog2_1.c -o prog1` Then, in this case, to run the program, you would just type `./prog1` at the command prompt.

8. **Reminder #3:** to make it so that you don't have to type the `./` in front of the executable file name, add the following line to your `.bashrc` file (which is located at your home directory):

   ```
   alias a.out='./a.out'
   ```

   then save the file and type `source .bashrc` at the Unix command for it to take effect (or log out and log back in for it to take effect).

C Program:

```
/*  prog2_1.c modified (similar to program 2.2 in the book, page 14)
     this is the very first program in the book, program 2.1, on page 11,
     with an added line of output + comments added
*/

// we need stdio.h library file because this is where the code for the
//    printf function is located
// the preprocessor goes to this file  "stdio.h"  on the system and copies
//    everything from that file into this file at this point (Chapter 12)
#include <stdio.h>


// main() is the entry point where programs start execution
// "main" is the name of the function
// the main function returns an integer
// this main function is "void" of arguments – it does not take in any //
      arguments
int main (void)
{
     /* printf is a function in the C library that is used to print
        "Programming is fun." to the screen by sending the argument (the
        string in quotes) to the function.  The  \n  denotes a newline
        character.  */
      printf("\nProgramming is fun. \n");

     /* another print statement with extra "new line characters";
        note the difference when you run the program */
     printf("I'm going to love this class! \n\n");

     // return 0 to the system to indicate successful execution;
     // any number can be used here, but 0 is used by convention
     //      to indicate a successful run of the program
     return 0;
}
```

Note from above things like:

- two different ways of commenting
- header comment – should ALWAYS have a header comment at the top of all your files that contains information about the program, sometimes may include information about how to run the program or anything that you think would be useful to remind yourself or someone else of. With assignments in this class, your header comment should also include information like name, user id, assignment #, date, etc. More specifics about that will be provided to you.
- indentation – notice that the comments are indented at the same level as the line to which they belong
    - **NOTE:** The above commenting is only meant to show you *style* of commenting (i.e. two different ways to comment, and they are indented at same level as the lines to which they belong) – and NOT meant to indicate that you should comment every single line explaining what it is . I did this in this example only to explain these C lines of code with this early example.
- semi-colons at the end of statements (if missing a semi-colon where one is required – that's what type of error??)
- curly braces surrounding block of code inside of main function
- blank lines in certain places to break up groups of lines of code, for easier readability
- more on the `return` statement: Different numbers can be used to indicate different error conditions that occurred (such as a file not being found). This exit status can be tested by other programs (such as the Unix shell) to see whether the program ran successfully (something that someone on the systems staff can do, if needed for some reason, if you don't know how to do this yourself).

C Program:

```
/*  prog2_3.c, page 15, but using the print statements from my last example
    above

    example showing how to print multiple lines of output with one print
    statement
*/


#include <stdio.h>


int main (void)
{
    printf("\nProgramming is fun. \n I'm going to love this class! \n\n ");

    return 0;
}
```

The output would look like the following:

```

Programming is fun.
 I'm going to love this class!
```

Notice the space before the word "`I'm`" on the second line because of the space after the newline character…

C Program:

```
/* -----------------------------------------------------------------------
    prog2_4.c       Displaying Variables

    Program 2.4, on page 15
    with an added line or two of code

---------------------------------------------------------------------- */

#include <stdio.h>


int main (void) {

   /* declaring a variable called "sum", which will hold the value
      of two numbers added together
          * a variable needs to be declared somewhere before it is used;
            otherwise, it would cause a semantic error
          * this variable is of type "int", which means it is an integer
            and not a decimal (float) number
   */
    int sum;

    sum = 50 + 25;


    // printf is used to print 2 known values and 1 unknown value, "sum"
    /* this call to the printf function sends 2 arguments:
            1. the string in quotes, and
            2. a variable called "sum"
       the format string  "%i"  in the quote portion will be replaced by the
       value of "sum"
    */
    printf ("\nThe sum of 50 and 25 is %i. \n\n", sum);


    // the variable "sum" is reused here, being assigned a new value
    sum = 483 + 379;
    printf ("The sum of 483 and 379 is %i. \n\n", sum);


    return 0;
}
```

Note from above things like:

- header comment separated with dashes (looks nice, but not necessary)
- the types of comments: I took out the unnecessary comments from the previous example but added other comments to explain the new code; in your programs, you should include comments whenever something is not obvious or whenever the code is complicated and you want/need to explain what the following block of code does so that when you/fellow programmers look at the code later, you/they will be able to see what's going on (or if you want to explain something to me, such as on an assignment)
- curly braces surrounding block of code inside of main function – in this example, the opening curly brace is after the parentheses instead of on the next line; either way is fine, as long as it is consistent (don't do it one way sometimes and another way other times)
- the variable named `sum` of type `int`
- added components inside the `printf` statement
- blank lines in certain places to break up groups of lines of code, for easier readability

C Program:

```
/* ------------------------------------------------------------------
    prog2_5.c       Displaying Multiple Variables

    Program 2.5, on page 17

-------------------------------------------------------------------- */

#include <stdio.h>


int main (void) {

    int value1, value2, sum;

    value1 = 50;
    value2 = 25;
    sum = value1 + value2;
    printf ("\nThe sum of %i and %i is %i. \n\n", value1, value2, sum);

    return 0;
}
```

Note from above things like:

- the printf statement above now has 4 arguments:
    - the string in quotes, containing format strings `%i` for each of the other 3 arguments
    - value1
    - value2
    - sum

- The declaration of the three variables could have been on separate lines, such as:

```
#include <stdio.h>

int main (void) {

      int value1;
      int value2;
      int sum;

      value1 = 50;
      value2 = 25;
      sum = value1 + value2;

      printf ("\nThe sum of %i and %i is %i. \n\n", value1, value2, sum);

      return 0;
}
```

- The initial values could have been on the lines where they were declared, either this way:

```
#include <stdio.h>

int main (void) {

	int value1 = 50, value2 = 25, sum;

	sum = value1 + value2;
	printf ("\nThe sum of %i and %i is %i. \n\n", value1, value2, sum);

	return 0;
}
```

- Or this way:

```
#include <stdio.h>

int main (void) {

	int value1 = 50;
	int value2 = 25;
	int sum;

	sum = value1 + value2;
	printf ("\nThe sum of %i and %i is %i. \n\n", value1, value2, sum);

	return 0;
}
```

2.3 ***FYI***

NOTE: Refer to the documents “**Programming Assignment Requirements**” and “**Formatting Examples**” posted on Blackboard for details about the general programming requirements and programming style expected in this course.