Note: the exercises in this set are designed to practice simple control constructs. Since we have not yet covered the C library, they are a little silly. Sorry.

To get a feel for what the programs do, run the versions in the directory `control` on hills.

**Part One**

The programs in this part manipulate a string entered on the command line when the program is started. In a very simple program, this is easy to implement. To run the program with a string as its single argument, do this:

```
progname 'string goes here'
```

e.g.,

```
a.out 'C is Wonderful'
```

To implement accessing the string, code the start of `main()` as follows:

```c
int main (int argc, char **argv) {
    char *arg1;
    if (argc != 2) {
        /* output an error message and exit (return) with an error */
    }
    argv++;
    arg1 = *argv;
    /* now arg1 is a pointer to the string on the command-line */
```

Don't worry - we'll go over command line arguments in gory detail later. For now, this will allow us to have more interesting programs! Take the opportunity, however, to code these simple programs carefully. A user should not be able to cause your program to fault, even if it is given illegal command-line arguments.

**PS** - For this exercise set, don't use C library routines to manipulate the string (like `strlen`, `strcpy`, `strchr`), even if you know about them. We will assume the ascii char set and will manipulate the characters directly. It will be more general to use the C library to do this after we learn how.

*revarg1.c*

This program should use the technique described above and output the string supplied as the argument reversed.

Use the function `putchar()` to output each character to standard output. `putchar()` takes a single character as its argument.

*anal.c*

This program does a simple analysis on the argument string, outputting the number of characters whose numeric values were even and the number of characters whose numeric values were odd. Do not include the nul byte at the end of the string.

*pathsegs.c*

Write a program that reads in a Unix path from the command line and outputs the segments of the path one-per-line. Make sure your program works correctly no matter whether the path you give it is absolute or relative.

*revname.c*

This program takes a single argument on the command-line in the form of a proper name (like `Sue Ann Doe`). It outputs the name reversed like this: `Doe, Sue Ann`.

For practice, see if you can write your program so that the result is output using a single `printf` call like this:

```
printf("%s, %s\n", ptr1, ptr2);
```

Test how your program acts when you give it unexpected input like an empty string, or a string with only one word. Practice coding your program so that it doesn't fault when the user gives it illegal input.
Part Two

In this part we will write a couple of simple programs to investigate characteristics about the standard types available in different compilation environments on both hills and linux. (i.e., depending on the options you give to the compiler to change the language subset used, these programs will execute differently). We will write two programs. Each will investigate a particular type and answer a few questions about it.

One way to accomplish this task is to rely on C preprocessor definitions in the standard system header files `<limits.h>` and `<float.h>` Most types available have definitions in these header files: `INT_MAX`, `INT_MIN`, `SHRT_MAX`, ... `CHAR_MAX`, `CHAR_MIN`, `LONG_LONG_MAX`, `LONG_LONG_MIN` (all in `<limits.h>`) and corresponding characteristics of floating point types `DBL_MAX`, `LDBL_MAX`, etc) in `<float.h>`. Most of the types described by these definitions are standard, but it is supposed to be the case that the optional types will only have definitions if the types exist. We will use this assumption.

(Note: this type of assumption is insufficient for production work, as it relies on the implementation being very consistent. A better method is to actually try to compile a small program that uses the type and see if the compilation fails. This piece of information is then used to configure the main program to either use the type or to circumvent it.)

We will investigate one standard type: `int` and one optional type: `long long`. In assignment one, you will investigate other types, but the basic techniques are the same.

`ints.c`

Write a program to output the maximum and minimum value that an `int` can hold. In addition, answer this question: if you set an `int` to its maximum value and then add one to it, what is its value?

`llongs.c`

Write a program to determine whether the type `long long` is available. If it is not, your program should simply output a message. If it is, it should answer the following questions:

- what is the maximum value a variable of type `long long` can hold? (If the header file indicates the type exists, you may assume that `printf` also supports its conversion specifier "%lld")
- how many bytes does the type `long long` use?

Note that you have not really tested the type until you have actually used it. (i.e., dont just use the preprocessor definition: declare a variable of that type and make sure it works!)

You should test your program on both hills and linux for each of the following language subsets:

- the default subset (without any special compilation options)
- ANSI C99 mode. (you must use an option for this: look up how to set the language subset in the `cc` (or `gcc`) man page.

Answers and Hits

Part One

`revarg1.c`

Use a counter and a loop to find the nul byte at the end of the string. Keep a copy of the original pointer. Then generate a pointer to the last character in the string. Finally, output each character in the string from the last to the first, using the starting pointer to know when you are finished.

`anal.c`

Each time you look at a character, place it in a variable `c` and check the value of `(c%2)` to see if the numeric value is even or odd.

`pathsegs.c`

See if you can alter the string passed on the command line so that you can output it one path segment per line with a single `printf` statement like this: `printf("%s\n", arg1);`
revname.c
This is a little more ambitious than the other programs, but the solution is there if you need it.

Part Two
These are simple programs. You should be able to do them easily. If you get stuck, look at the code.