Assignment 1 - Software Project, Fall 2008/2009

Due: December 1, 2008

Before starting to answer the questions, please read very carefully the "Submission Guidelines" at the end of this document. You can find information on developing `C` under `UNIX` (e.g. working with makefiles) in the presentation `unix_c_development.ppt` at the course website.

Ex 1.1 exp

This program reads a real number x from the standard input (using type `double` and `args = scanf("%lf", &num ))`, computes, and outputs $exp(x) = e^x$. The value $exp(x)$ should be computed according to the formula:

$$exp(x) = 1 + \sum_{i=1}^{i=M} \frac{x^i}{i!}$$

where $i! = 1 \cdot 2 \cdot 3 \cdots i$. Note that the i-th element in this summation can be easily computed from the (i-1)-st element. In this exercise define M to be 20. Assume that $-10 <= x < 30$.

Ex 1.2 exp_limit

In exercise 1.1 the calculation of $exp(x)$ was approximated by a series of 20 elements. In this exercise for each value of x your program should calculate the optimal value of M. As in the previous exercise, use the `double` data type to carry out all calculations. Due to the limited precision of this data type, you will soon see that enlarging the value of M does not have any effect on the resulting sum. The optimal value of M is the index of the element, such that the next elements in the series (stored as `double`) are equal to 0. Your program should output the value of $exp(x)$ followed by the optimal value of M. Assume that $-10 <= x < 79$. Examples:

```
./exp_limit
1
2.718282 177

./exp_limit
10
22026.465795 306

./exp_limit
```

```
0.1
1.105171 121
```

Ex 1.3 get_digit

Write a program that reads a decimal number x (positive or negative), prints the number of digits of x, reads an index i of a digit (index 1 means least significant digit of x) and prints the digit of x at index i. If i is not a valid index of digit, the program prints an error message and terminates. Use `long` type to represent x; use an appropriate format in `scanf` and `printf`. To perform the computations, you are not allowed to use loops. Instead, use `log10`, `pow` and `labs` functions from the standard math library. To use these functions, add the line

```
#include <math.h>
```

at the top of the source file, and link with the math library (add `-lm` to the command line) to use these functions.

Submission Guidelines

Files framework

- Create the directory `assign1` under `~soft-proj09`. All the files in this assignment should be placed under this directory.
- Create the directories `exp`, `exp_limit`, and `get_digit` under `~/soft-proj09/assign1`. Each of these directories should contain the corresponding source files: a makefile, which is found in the course website, and a `C` file.
- The names of the `C` files should be: `exp.c`, `exp_limit.c`, and `get_digit.c`. Each of these should be placed under the appropriate directory. For example, `exp.c`, the `C` file for program `exp`, should be under the directory `~/soft-proj09/assign1/exp/`.

Note that file names in `UNIX` are case sensitive (e.g. `foo.C` is different than `foo.c`).

Setting Access Permissions to Files

Make sure that you have correct permissions for all the directories and files. Set permissions by executing the following commands:

```
chmod 755 ~
chmod -R 755 ~/soft-proj09
```

Compilation

Makefiles for each of the programs, `exp`, `exp_limit` and `get_digit`, are provided at the website. All of your exercises should pass the compilation test, which will be performed by running the "make all" command in a `UNIX` terminal window.

Testing

Make sure your programs detect invalid input data, and print out appropriate error messages. Do **not** add "friendly" messages to your programs, as they are tested automatically.

In order to test your programs, we provide samples of input and output files - like the ones that will be used by the automatic check. The input files have a `.in` extension, and the output files have a `.out` extension. In order to read the input from a file and write the output to a file, you should use redirection. For example:

```
exp < infile > outfile
```

The output of your program should be exactly as in the provided samples! Use the command `diff` or `diff -b` in order to compare your output to the appropriate sample output file.

Submission

Printouts submission: You should submit in pairs. Printouts of the code should be submitted to the checker's mailbox (#375, Reuven Aronashvili). The printouts should include the id-numbers and user names of both partners (one manual submission per pair).

Code submission: Although the submission is in pairs, **every** student must have all the exercise files under his home directory as described above. The exercise files of both partners must be identical. Each exercise directory (e.g. assign1) must contain a file named "partners.txt" that contains the following information:

```
Full Name: your-full-name
Id No 1: your-id
User Name 1: your-user-name
Id No 2: partner-id
User Name 2: partner-user-name
Assignment No: the-assignment-number
```

Important note: Exercises that can not be automatically checked due to problems in the above described definitions will not be tested and the appeals will not be considered.

Good Luck!