Compute Sine using Taylor Series

Overview

This project stresses the use of floating point instructions to create a program that computes the sine of an angle given to you in degrees on the command line.

Taylor Series

The sine of an angle given in radians can be found using the Taylor Series:

```
sin x = x - x^3/3! + x^5/5! - x^7/7! ...
```

Notice each term flips from addition to subtraction.

Notice each term is based on the odd integers starting at 1. While the "1" case might look different, it is the same as all the others since 1 is just 1 to the first power divided by 1 factorial.

Command line

You are to accept two arguments on the command line. `getopt` is not being used here to concentrate on the floating point math. Both arguments are therefore required.

- The angle in degrees whose sine you wish to calculate. Take this to be a double.
- The number of terms to evaluate. The number of terms must lie between 1 and 10 inclusive. Note the value of 10 as an upper bound is new. It was 8 in the original specification.

C version

To assist your efforts, here is a version of this project written in C. This has been updated to print nice debugging output which is not part of the project.

This C version also demonstrates a different way of calculating the toggle. This version flips the sign of the toggle by multiplying by -1. The previous version used odd and even values of the term.

Sample executions

```
pk_taylor_series > gcc main.S -o a
pk_taylor_series > ./a 0 10
The sine of 0.00 degrees is 0.00000000.
pk_taylor_series > ./a 30 10
The sine of 30.00 degrees is 0.50000000.
pk_taylor_series > ./a 45 10
The sine of 45.00 degrees is 0.70710678.
```

```
pk_taylor_series > ./a 90 10
The sine of 90.00 degrees is 1.00000000.
pk_taylor_series > ./a 180 10
The sine of 180.00 degrees is -0.00000000.
pk_taylor_series > ./a 360 10
The sine of 360.00 degrees is -0.00104818.
pk_taylor_series > ./a 360 100
Number of terms is out of range.
pk_taylor_series > ./a 360 -1
Number of terms is out of range.
pk_taylor_series >
```

Floating point instructions I used

These are the floating point instructions I used in my implementation.

- `fmov`
- `scvtf`
- `fmul`
- `fdiv`
- `fadd`

How I broke up the program

I have functions named:

- main
- HandleOptions - gets, parses and checks the command line
- Factorial
- IntegerPower - x to the nth power
- ComputeSine - The main calculation
- PrintAnswer
- ConvertTheta - Wrap D2R
- D2R - Degrees to radians

Sad story

In writing the assembly language version of this project, I decided to check my intermediate answers with ChatGPT. It was easy enough to frame the request and I did so in many ways **getting a different wrong answer every time**. I wasted **hours** looking for a bug in my code ultimately to discover ChatGPT's lies and deceit.

For example, here are some gems:

Three minus four is about zero...

```
~ 3.14159 - (3.14159^3 / 3!)
~ 3.14159 - 4.9348
~ -0.0005
```

or this (ChatGPT even generated this code):

```c
# include <stdio.h>
# include <math.h>
int main() {
    const double pi = 3.14159265358979323846;
    double result = pi *(1 - pi*pi / 6.0);
    printf("%f\n", result);
    return 0;
}
```

This is a hard coded version of a 2 term Taylor Series evaluation of 180 degrees (PI radians).

And Chat's answer: 2.958040

The correct answer is: -2.026120.

Bottom line: ChatGPT lies as much as a politician.

ChatGPT for President!

CSC3510

The following applies to Carthage College CSC3510 students.

Work rules

Work is to be done solo.

What to hand in

Just the .S file. **Your name must be at the top of the file.**

Setting expectations

With extensive commenting, my solution is about 250 lines. This is not a challenge, rather a figure by which to set expectations. Should you find you're writing a thousand lines, for example, you're doing something wrong.