AARHUS UNIVERSITY JONAS ASKJÆR-JØRGENSEN DEADLINE: 17/10
AU-id: AU713502 Studienr: 202207230

Programmering for Computerteknologi

Hand-in Assignment Exercises

Week 7: Programming larger software projects

1. (Code answer) In the lecture we discussed how we could implement an approximate sine function using the Taylor series equation:
   a) Write the following in a C program:

Write this function in a C program so that it calculates the sine function with precision up to n Taylor series terms, e.g. the above example shows 4 terms. Your sine function should accept both the x value and n precision as input. The signature of your function should be:

```
double taylor_sine(double x, int n)
{
  /* implement your function here */
}
```

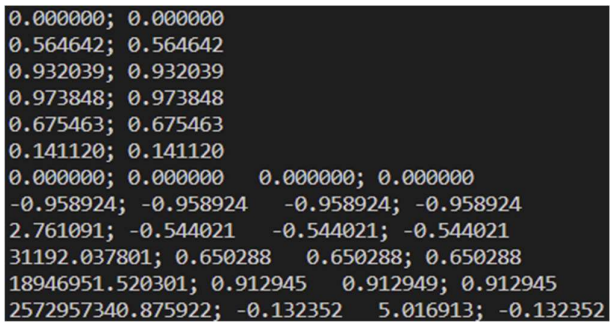
Write your Taylor series implementation of sine as a **library** consisting of: a header file and a source file (with no **main()** function).

To see the function, I have implemented to calculate see $taylor\_sine.c$

   b) Write some tests for different values of x.

(Try both small and large input values) and compare your function output with the ANSI C sin function. Write your test program (which will have a $main()$ function) separately from your library, i.e., you should only include the library header file. Compile your test program by linking with your Taylor Sine library.

   c) Answer the following questions using your test program, and please provide your answers as comments in your test program: Which intervals of input x did your function give a similar result to the ANSI C sin function? What impact did increasing the precision have (i.e., increasing the number of Taylor series terms)?



Increasing the precision ensured that my input in the Taylor series looked like the sin function implemented using $< math.h >$, when I input accuracy 10, the values were not similar while increasing to accuracy 30 makes the numbers accurate, except a few decimals. Larger accuracy makes the Taylor expressions closer to the sin function.
I made a loop that runs the values of pi in increments of 0.6 to accuracy 10, seen in the two rows to the left they are accurate. Smaller inputs worked with lower accuracy while larger inputs need higher accuracy. See my code for the calculations.

2. (Code answer) Stacks are containers where items are retrieved according to the order of insertion, independent of content. Stacks maintain last-in, first-out order (LIFO).
   a. Implement a stack based on singly linked lists as discussed in the lecture.
   b. Test your implementation. You should expect the following "laws" to hold for any implementation of a stack. Hint: you should enforce these conditions using assert statements:

   (A) After executing `Initialize(s);` the stack `s` must be empty.
   (B) After executing `Push(x,s); y = Pop(s);` the stack `s` must be the same as before execution of the two commands, and `x` must equal `y`.
   (C) After executing `Push(x0,s); Push(x1,s); y0 = Pop(s); y1 = Pop(s);` the stack `s` must be the same as before execution of the two commands, `x0` must equal `y1`, and `x1` must equal `y0`.

For the results of assignment 2 see my code. Everything is there including comments.