# CSC 236 Programming Assignment

# FLOAT

## Summary

The FLOAT assignment is worth 2% of your grade. For this assignment, you will be submitting a file named **float.ans**. This file is created by the grading system when you grade your program. The grade is based on correct behavior from your completed program (no documentation or efficiency score this time).

If you have questions about the assignment, feel free to use our EdStem forum, but be sure not to post any code from your solution in a public edstem post (even code that's not working). You can also stop by during office hours to get help from anyone on the teaching staff. Remember, if you need help, you need to make sure your code is documented, including block comments above sections of your code and line comments saying what you are trying to do with each instruction. We'll also expect you to have a design for the program that you're trying to follow.

## Specifications

With some help from the starter code, you will be writing a program to calculate the constant pi using the Salamin-Brent algorithm (Microcomputers and Mathematics by Bruce, Giblin and Rippon. Cambridge University Press 1990. ISBN 0-521-31238). This algorithm was announced independently in 1976 by both mathematicians.

The pseudocode below gives a complete implementation of the algorithm. There's also a full C program that mirrors the code needed for the assignment, and assembly code for all the supporting functions. You will write the floating point code for the first three statements in the algorithm. This is should require about 26 instructions.

The following summarizes the Salamin-Brent technique. The estimate of pi improves with increasing values of n.

$$\pi \approx \frac{4a_n^2}{1 - \sum_{k=0}^{n} 2^k (a_k - b_k)^2}$$

The values of $a_n$ and $b_n$ are a series, with the in initial elements defined as:

$$a_0 = 1$$
$$b_0 = 1/\sqrt{2}$$

Given elements ai and bi, the next elements of the series are defined as:

$$a_{i+1} = (a_i + b_i)/2$$
$$b_{i+1} = \sqrt{a_i b_i}$$

The following pseudo-code for the algorithm loops until it reaches the limit of the precision of the variables and value of pi being calculated stops improving. You'll be implementing the part of the algorithm in blue.

```
float a, b, c, d, s, t, pi, old;

a = 1.0                                  // a_0 = 1
b = 1.0 / sqrt( 2.0 )                    // b_0 = 1/sqrt(2)
s = 1.0                                  // the sum in the denominator
t = 1.0                                  // 2^0 is the first power of 2
old = 0.0                                // previous pi approximation

while ( true ) {                         // Loop until enough precision
    s = s - t * (a - b ) * (a - b)       // Update denominator value
    pi = 4 * a * a / s                   // Latest value of pi

    c = (a + b) / 2.0                    // Compute next value of a_i
    d = sqrt( a * b )                    // compute next value of b_i

    a = c                                // Copy back to a and b.
    b = d

    t = 2 * t                            // Compute next power of 2
    output( pi )                         // Report on progress
    if ( pi == old ) break               // Check for convergence

    old = pi                             // Remember latest estimate.
}
```

## Convergence Illustration

The following table shows the convergence of this algorithm across a few iterations. This was originally calculated using excel.

| | n=0 | | n=1 | | n=2 | | n=3 | |
|---|---|---|---|---|---|---|---|---|
| | a0 | b0 | a1 | b1 | a2 | b2 | a2 | b2 |
| Initial values | 1.00000 | 0.7071 | | | | | | |
| $a_{n+1} = ( a_n + b_n ) / 2$ | | | 0.85355 | | 0.84722 | | 0.84721 | |
| $b_{n+1} = \text{sqrt} ( a_n * b_n )$ | | | | 0.84090 | | 0.84720 | | 0.84721 |
| $4 * ( a_n)^2$ | 4.00000 | | 2.91421 | | 2.87116 | | 2.87108 | |
| $k0 = 2^0 * (a_0 - b_0)^2$ | 0.08579 | | | | | | | |
| 1-k0 | 0.91421 | | | | | | | |
| $PI = 4 * (a_0)^2 / ( 1 - k0)$ | 4.37535 | | | | | | | |
| $k1 = 2^1 * (a_1 - b_1)^2$ | | | 0.00032 | | | | | |
| 1-k0-k1 | | | 0.91389 | | | | | |
| $PI = 4 * (a_1)^2 / ( 1 - k0-k1)$ | | | 3.18879 | | | | | |
| $k2 = 2^2 * (a_2 - b_2)^2$ | | | | | 0.00000 | | | |
| 1-k0-k1-k2 | | | | | 0.91389 | | | |
| $PI = 4 * (a_2)^2 / ( 1 - k0-k1-k2)$ | | | | | 3.14168 | | | |
| $k3 = 2^3 * (a_3 - b_3)^3$ | | | | | | | 0.00000 | |
| 1-k0-k1-k2-k3 | | | | | | | 0.91389 | |
| $PI = 4 * (a_3)^2 / ( 1 - k0-k1-k2-k3)$ | | | | | | | 3.14159 | |

## Creating your Program

## Step 1 : Create a design

Make sure you're familiar with the machine instruction set for floating-point arithmetic. Then, come up with a plan for calculating the three statements in the algorithm that you're expected to implement.

## Step 2 : Code your solutions

Retrieve the testing and grading files. All files are packed together in one self-extracting file named **unpack.exe**. You should be able to download this file from the course Moodle page.

Put this file in the FLOAT directory in your shared, P23X dosbox file space. Run DOSBox and execute **unpack.exe** to create the testing and grading files. The starter archive should create the following files

| | |
|---|---|
| FLOATPI.C | A C solution for the whole assignment. |
| FLOATPI.EXE | The executable for the compiled C code. |
| SQROOT.ASM | Assembler source code for the square root routine. |
| SQROOT.OBJ | The object code you link with your float main program. |
| OUTPUT.ASM | The assembler source code to display the current value of pi. |
| OUTPUT.OBJ | The object code you link with your float main program. |
| FLOAT.m | Starter file for your part of the program. |

The **float.m** is a starter for your implementation. Rename it to **float.asm** and add your code to **float.asm**.

Two of the other files contain subroutines needed to complete the assignment.

- The sqroot subroutine calculates the square root of a number.
- The output subroutine displays your current value of pi.

Use the commands below to assemble and link your main float assignment with the two subroutines used to calculate square roots and output the current value of pi.

- To compile your source file to an object:
  ```
  ml /c /Zi /Fl float.asm
  ```
- To link everything into an executable:
  ```
  link /CO float.obj sqroot.obj output.obj
  ```

## Step 3 : Test and debug

We are not logging your testing for this assignment. When you're ready, you can run your program by entering: **float**

Your output should look like the following. Each line represents one iteration of the algorithm. Using 32-bit (single-precision) floating-point values we can correctly calculate six digits.

```
4.37534
3.18879
```

```
3.14168
3.14159
3.14159
```

## Step 4 : Grading

Run the grading system to determine how many points your program has earned. Go to the FLOAT subdirectory. If you successfully unpacked the archive for the assignment, you should be able to run the grading program by entering:

```
gradfl
```

It should generate a grading results file, **float.ans**, where your grade is based on your program getting correct results.

## Step 5 : Submit your Solution

Electronically submit the file **float.ans** created by the grading system. This file should be submitted to the FLOAT assignment in Moodle.

Incorrect electronic submissions will result in your program not being graded and considered late.

As usual, do not edit or modify any of the grading system files.