SYNOPSIS
This case study looks at the re-engineering of a program to calculate the value of $\pi$ written in Fortran from 1966.

<table>
<thead>
<tr>
<th>Type:</th>
<th>legacy re-engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language:</td>
<td>Fortran</td>
</tr>
<tr>
<td>Compiler:</td>
<td>gfortran</td>
</tr>
<tr>
<td>Skills:</td>
<td>program re-engineering</td>
</tr>
<tr>
<td>Experience Level:</td>
<td>novice-intermediate</td>
</tr>
</tbody>
</table>

"Black holes result from God dividing the universe by zero.”
unknown.
INTRODUCTION

In 1966 Donald G. Fink published a book entitled “Computers and the Human Mind”, which introduced the idea of artificial intelligence as it related to computers. One of the first problems he introduced to illustrated the utility of computers was the calculation of $\pi$. He posed the question as to why compute $\pi$ to 100,000 decimal places, when 30 decimal places suffices to calculate the circumference of the known universe? This estimate of 30 decimal places was made by astronomer Simon Newcomb (1835-1909). In 2006, Arndt & Haenel estimated that “39 digits of $\pi$ are sufficient to calculate the volume of the universe to the nearest atom”.

So why calculate $\pi$? Because it’s easy and computers are good for exhaustive calculations.

Fink used a series of the following form:

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \frac{4}{13} \ldots$$

Then goes on the say that using this formula to derive 100,000 decimal places of $\pi$ would require $10^{100,000}$ terms. So many in fact that “shortcuts must be used even with a fast computer”. It was 1966, and computers were slow, and used punch cards!

He postulates that a speed-up comes by way of simple averaging. Here’s a summary:

- sum of the first 98 terms = 3.1313888 (too low)
- sum of the first 99 terms = 3.1516934 (too high)
- sum of the first 100 terms = 3.1315929 (too low)
- average of 98 and 99 term sums = 3.1415411 (still too low)
- average of the 99 and 100 terms = 3.1416432 (still too high)
- average of the two averages = 3.1415921
THE CODE

The program itself is (likely) coded in Fortran IV (or earlier).

```
DIMENSION TERM(100)
N=1
3   TERM(N)=(-1**(N+1))*((4.)/(2.*N-1.))
   N=N+1
   IF (N-101) 3,6,6
6   N=1
7   SUM98 = SUM98+TERM(N)
   PRINT 28, N, TERM(N)
   N=N+1
   IF (N-99) 7, 11, 11
11  SUM99=SUM98+TERM(N)
   SUM100=SUM99+TERM(N+1)
   IF (SUM98-3.141592) 14,23,23
14  IF (SUM99-3.141592) 23,23,15
15  IF (SUM100-3.141592) 16,23,23
16  AV89=(SUM98+SUM99)/2.
   AV90=(SUM99+SUM100)/2.
   COMANS=(AV89+AV90)/2.
   IF (COMANS-3.1415920) 21,19,19
19  IF (COMANS-3.1415930) 20,21,21
20  PRINT 26
   GO TO 22
21  PRINT 27 COMANS
22  STOP
23  PRINT 25
   GO TO 22
25  FORMAT(25H ERROR IN MAGNITUDE OF SUM)
26  FORMAT(14H PROBLEM SOLVED)
27  FORMAT(16H PROBLEM UNSOLVED, E14.6)
28  FORMAT(I3, E13.6)
END
```
DISASSEMBLE AND ANALYZE THE PROGRAM

The first task is to assess the program, by disassembling it. Assume no prior knowledge, the task here is to re-engineer the program, not create it from scratch. Don’t be fooled by the programs mere 28 lines of code. Re-engineering a program can be a challenging experience, no matter the size of the program.

Some basic observations can be made. Column 1 contains a C to denote comments, and statement numbers in columns 1-5 to be used in jump instructions for transferring control. there are NO variable declarations - variables are implicitly declared, meaning that variables beginning with i, j, k, l, m, n are always integers, and the rest are reals. Basically only N is an integer in this program, the rest are all reals.

Try to compile this and it won’t compile. The first major problem is the PRINT statement, which archaically tries to print directly to a printer. First stop is to modify these to print statements. Therefore:

```
PRINT 28, N, TERM(N)
```

becomes

```
WRITE(*,28) N, TERM(N)
```

In addition, the FORMAT statements make reference to HOLLERITH constants. Therefore:

```
26 FORMAT(14H PROBLEM SOLVED)
```

becomes

```
26 FORMAT(‘PROBLEM SOLVED’)
```

Oh, and the program had no program header (first line), so one was added. The result is 32 lines of legacy yuck. It should now compile - but won’t give the right answers. It is riddled with seven archaic arithmetic if statements, some of which control loops. Remember this is prior to the real introduction of structured programming as we know it today. Jumps in programs were a normal way of doing things prior to the widespread use of loops. There are also two go to statements.
THE CODE THAT COMPILES

The program that compiles and runs - although it doesn’t work properly.

```
PROGRAM PI
DIMENSION TERM(100)
N=1
3   TERM(N)=((-1)**(N+1))*(4./((2.*N-1.))
N=N+1
IF (N-101) 3,6,6
6   N=1
7   SUM98 = SUM98+TERM(N)
WRITE(*,28) N, TERM(N)
N=N+1
IF (N-99) 7, 11, 11
11  SUM99=SUM98+TERM(N)
14  IF (SUM98-3.141592) 14,23,23
15  IF (SUM99-3.141592) 23,23,15
16  AV89=(SUM98+SUM99)/2.
17  AV90=(SUM99+SUM100)/2.
18  COMANS=(AV89+AV90)/2.
19  IF (COMANS-3.1415920) 21,19,19
20  WRITE(*,26)
21  WRITE(*,27) COMANS
22  STOP
23  WRITE(*,25)
24  GO TO 22
25  FORMAT('ERROR IN MAGNITUDE OF SUM')
26  FORMAT('PROBLEM SOLVED')
27  FORMAT('PROBLEM UNSOLVED', F14.6)
28  FORMAT(I3, F14.6)
END
```

See listing `pi.f90`

For a program listing with a comment for every line of code, see `pi2.f90`.

Before proceeding any further, convert the program to lowercase letters - you DON’T want to work in uppercase.
REENGINEERING - STEP 0

So the program runs. Great. Problem is the output is wrong and the program doesn’t look very nice. You would think that aesthetics have very little to do with re-engineering a program. Ever looked at 1000 lines of uppercase where the only “structure” is an unstructured mess of jump instructions?

Probably not.

The easiest place to start may be to deal with improving readability. Translate the code entirely to lowercase.

So why does the code not work? The idea behind the algorithm is to alternate between different terms of the sequence, adding one, subtracting the next, adding the next one etc. Look at the output from the program - it prints all the terms of the sequence - and they are all negative. A huge red flag. The problem lies in the code on this line:

```
3    term(n) = (-1**(n+1)) * (4./(2.*n-1.))
```

Turns out that in Fortran ** (power) has the highest precedence, so it will be done before the unary minus. This is not like in languages such as C, where the unary operators have higher precedence than the arithmetic operators. So to fix this problem, enclose the unary minus sign and 1 in parentheses:

```
3    term(n) = ((-1)**(n+1)) * (4./(2.*n-1.))
```

Now explicitly declare the variables:

```
real, dimension(100) :: term
real :: sum98, sum99, sum100, av89, av90, comans
```

NOW IT WORKS - RIGHT?

Compile and run it, and it seems to work, although it still doesn’t - it should solve the problem because it works on paper. When the problem is run though, it now prints “problem unsolved”. How can this be? Why is it even necessary to get it running properly, we are re-engineering the whole program - right? Firstly, the program needs to work properly before it is re-engineered, otherwise there won’t be a benchmark to calibrate and test the program throughout the re-engineering process. Imagine if it was a larger program, didn’t work, and had little documentation? Throw it away and start from scratch - there’s no painless way of getting much from such a program.

So how to fix this program? Check the numbers it’s crunching and see what happens with the algorithm. If the program is run, and the appropriate write statements added to print out the relevant data, this is what is output:
So now work through the algorithm to see where is goes wonky. Consider this line of code:

```c
if (comans - 3.1415920) 21,19,19
```

It calculates the difference between the variable `comans` and 3.1415920. If the difference is greater than or equal to zero, control is passed to the statement at label 19, otherwise (if it is negative) control is passed to label 21, where the message “problem unsolved” is output and the program terminates. Clearly

```
3.14159179 - 3.1415920 = -0.00000021
```

Which means the program terminates. Why does this happen? Probably because precision was less in 1966, so the calculations would have produced a more inaccurate number. The program works when the code is modified to:

```c
if (comans - 3.1415917) 21,19,19
```

The last fix involves the output. When the problem is unsolved, the output includes the value calculated, whereas when the problem is solved, the value calculated is not output. This is an easy fix, the following output code for the two statements is swapped. Also, the output includes all 100 terms of the series calculation - which isn’t really needed except for debugging, so the statements invoking this can be commented out.
THE CODE THAT WORKS

The program that compiles and runs - and works. Now the legacy features just need to be removed.

```fortran
program pi

  real, dimension(100) :: term
  real :: sum98, sum99, sum100, av89, av90, comans

  n=1
  3  term(n)=((-1)**(n+1))*(4./(2.*n-1.))
     n=n+1
     if (n-101) 3,6,6
  6  n=1
  7  sum98 = sum98+term(n)
     n=n+1
     if (n-99) 7, 11, 11
  11 sum99=sum98+term(n)
     sum100=sum99+term(n+1)
     if (sum98-3.141592) 14,23,23
  14 if (sum99-3.141592) 23,23,15
  15 if (sum100-3.141592) 16,23,23
  16 av89=(sum98+sum99)/2.
     av90=(sum99+sum100)/2.
     comans=(av89+av90)/2.
     if (comans-3.1415917) 21,19,19
  19 if (comans-3.1415930) 20,21,21
  20 write(*,26) comans
     go to 22
  21 write(*,27)
     stop
  22 write(*,25)
     go to 22
  25 format('error in magnitude of sum')
  26 format('problem solved', f14.8)
  27 format('problem unsolved')
end
```

See listing `pi3.f95`
REENGINEERING - STEP 1

The first step is to rationalize the loops in the code. The first “loop” in the code is of the form:

```fortran
n=1
3 term(n)=((-1)**(n+1))*(4./(2.*n-1.))
n=n+1
if (n-101) 3,6,6
6 n=1
```

Basically this calculates the 100 terms of the sequence. The arithmetic **if** statement controls iteration, while `n` has a value of 100 or less. As soon as `n` has a value of 101, the **if** statement jumps to label 6. This can be replaced with a do statement of the form:

```fortran
do n = 1,100
   term(n)=((-1)**(n+1))*(4./(2.*n-1.))
end do
```

This effectively does away with an arithmetic **if**, and two labels (3,6) - the code at label 6 is redundant because the code that follows is also a loop, so there is no need to reset `n` to 1. So the second loop, whose code is of the form:

```fortran
7 sum98 = sum98+term(n)
n=n+1
if (n-99) 7, 11, 11
```

Which sums the first 98 terms of the series, now becomes:

```fortran
sum98 = 0.0
do n = 1,98
   sum98 = sum98 + term(n)
end do
```

Another arithmetic **if**, and label (7) disappear. Both loops are now real loops, not quasi-loops composed of jump instructions. Next there are two statements deriving values for `sum99` and `sum100` - these can be cleaned up, removing another label (11). So

```fortran
11 sum99=sum98+term(n)
sum100=sum99+term(n+1)
```

becomes:

```fortran
sum99 = sum98 + term(n)
sum100 = sum99 + term(n+1)
```

See listing *pi4.f95*
REENGINEERING - STEP 2

Next the remaining five arithmetic if's are dealt with. This involves the remaining code:

```
if (sum98-3.141592) 14,23,23
14 if (sum99-3.141592) 23,23,15
15 if (sum100-3.141592) 16,23,23
16 av89=(sum98+sum99)/2.
   av90=(sum99+sum100)/2.
   comans=(av89+av90)/2.
   if (comans-3.1415917) 21,19,19
   19 if (comans-3.1415930) 20,21,21
   20 write(*,26) comans
      go to 22
   21 write(*,27)
   22 stop
   23 write(*,25)
      go to 22
   25 format('error in magnitude of sum')
   26 format('problem solved', f14.8)
   27 format('problem unsolved')
```

It may be challenging to re-engineer this one line at a time - because the code is all interlaced due to the jump instructions from the arithmetic if's. It is clear from the logic of the first block of code above that the three averages are calculated only if:

```
sum98 < 3.141592 AND sum99 > 3.141592 AND sum100 < 3.141592
```

otherwise, the “error in magnitude of sum” message is generated. So the three arithmetic if's could be turned into a single if statement of the form:

```
if ((sum98 < 3.141592) .and. (sum99 > 3.141592)
   .and. (sum100 < 3.141592)) then
   av89 = (sum98 + sum99) / 2.0
   av90 = (sum99 + sum100) / 2.0
   comans = (av89 + av90) / 2.0
else
   write (*,*), "Error in magnitude of sum"
   stop
end if
```

Three arithmetic if's and three labels (14, 15, 16) are removed.
The remaining code now looks like:

```fortran
if (comans-3.1415917) 21,19,19
19  if (comans-3.1415930) 20,21,21
20  write(*,26) comans
    go to 22
21  write(*,27)
22  stop
26  format('problem solved', f14.8)
27  format('problem unsolved')
```

The remaining two arithmetic if's can be transformed in a similar fashion. Basically if

```
comans >= 3.1415917 AND comans < 3.1415930
```

the problem is solved, and \( \pi \) is printed, otherwise the problem is unsolved. So the code above becomes:

```fortran
if ((comans >= 3.1415917) .and. (comans < 3.1415930)) then
    write (*,'(A,f14.8)') "problem solved", comans
else
    write (*,*) 'problem unsolved'
end if
```

Now all the labels are gone, and the format specification has been compressed inside the write statement.

See listing `pi5.f95`
THE RE-ENGINEERED CODE

The program that has been re-engineered, with no archaic structures.

```fortran
program pi

  real, dimension(100) :: term
  real :: sum98, sum99, sum100, av89, av90, comans

  do n = 1,100
    term(n)=((-1)**(n+1))*(4./((2.*n-1.)))
  end do

  sum98 = 0.0
  do n = 1,98
    sum98 = sum98+term(n)
  end do

  sum99 = sum98 + term(n)
  sum100 = sum99 + term(n+1)

  if ( (sum98 < 3.141592) .and. (sum98 > 3.141592) .and.
      (sum100 < 3.141592)) then
    av89 = (sum98 + sum99) / 2.0
    av90 = (sum99 + sum100) / 2.0
    comans = (av89 + av90) / 2.0
  else
    write (*,*) "Error in magnitude of sum"
    stop
  end if

  if ( (comans >= 3.1415917) .and. (comans < 3.1415930)) then
    write (*,'(A,f14.8)') "problem solved", comans
  else
    write (*,*) 'problem unsolved'
  end if

end
```

What is left to do? The two if statements created in Step 2 could be combined in the following manner:

```fortran
if ((sum98 < 3.141592) .and. (sum99 > 3.141592) .and. (sum100 < 3.141592)) then
    av89 = (sum98 + sum99) / 2.0
    av90 = (sum99 + sum100) / 2.0
    comans = (av89 + av90) / 2.0
    if ((comans >= 3.1415917) .and. (comans < 3.1415930)) then
        write (*,'(A,f14.8)') "problem solved", comans
    else
        write (*,*') "problem unsolved"
    end if
else
    write (*,*') "Error in magnitude of sum"
end if
```

Beyond that, adding comments back into the program, and modifying the indenting.

See listing `pi6.f95`