

C++ LAB ASSIGNMENT

Compute pi

$\pi = 3.14159265358979323\dots$



The constant π is represented in this [mosaic](#) outside the mathematics building at the [Technische Universität Berlin](#).

Dan McElroy, September 2017

 This presentation is offered under a **Creative Commons Attribution Non-Commercial Share** license. Content in this video can be considered under this license unless otherwise noted.

What is pi π

Definition

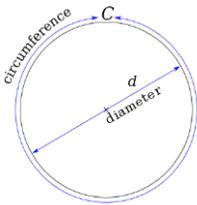
The circumference of a circle is slightly more than three times as long as its diameter. The exact ratio is called π .

π is commonly defined as the ratio of a circle's circumference to its diameter d : The ratio C/d is constant, regardless of the circle's size. For example, if a circle has twice the diameter of another circle it will also have twice the circumference, preserving the ratio C/d .

Properties

π is an irrational number, meaning that it cannot be written as the ratio of two integers (fractions such as $22/7$ are commonly used to approximate π ; no common fraction (ratio of whole numbers) can be its exact value). Since π is irrational, it has an infinite number of digits in its decimal representation, and it does not end with an infinitely repeating pattern of digits.

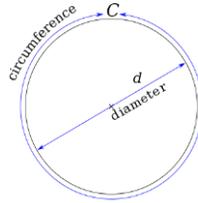
$\pi = \frac{C}{d}$



<http://en.wikipedia.org/wiki/Pi>

What is pi π

$$\pi = \frac{C}{d}$$



$$\frac{22}{7} = 3.1428571428571\dots$$

$$\pi = 3.1415926535898\dots$$

$$\text{Error} = 0.0012644892673\dots$$

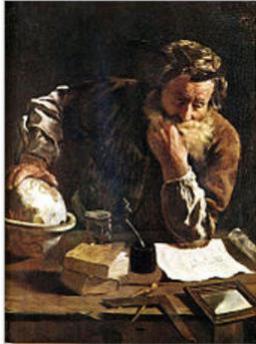
Computers and pi

Several different algorithms have been used to compute pi. Below are two of the more common algorithms. When doing your lab assignment, you can choose either (or both) of the algorithms. Only the first algorithm is discussed here.

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

$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \dots$$

pi Pioneers



Archimedes developed the polygonal approach to approximating π .



Isaac Newton used infinite series to compute π to 15 digits, later writing "I am ashamed to tell you to how many figures I carried these computations".



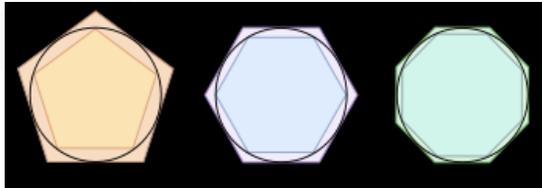
William Jones was most noted for his proposal for the use of the symbol π (the Greek letter pi) to represent the ratio of the circumference of a circle



Leonhard Euler popularized the use of the Greek letter π in works he published in 1736 and 1748.



John von Neumann was part of the team that first used a digital computer, ENIAC, to compute π .

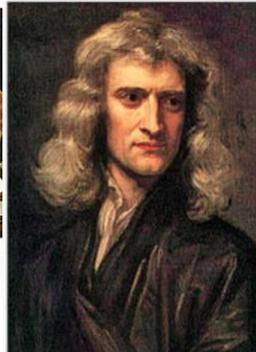


<http://en.wikipedia.org/wiki/Pi>

pi Pioneers



Archimedes developed the polygonal approach to approximating π .



Isaac Newton used infinite series to compute π to 15 digits, later writing "I am ashamed to tell you to how many figures I carried these computations".



William Jones was most noted for his proposal for use of the symbol π (the Greek letter pi) to represent the ratio of the circumference of a circle to its diameter.



Leonhard Euler popularized the use of the Greek letter π in works he published in 1736 and 1748.



John von Neumann was part of the team that first used a digital computer, ENIAC, to compute π .

<http://en.wikipedia.org/wiki/Pi>

pi Pioneers



Archimedes developed the polygonal approach to approximating π .



Isaac Newton used infinite series to compute π to 15 digits, later writing "I am ashamed to tell you how many figures I carried these computations".



William Jones was most noted for his proposal for the use of the symbol π (the Greek letter *pi*) to represent the ratio of the circumference of a circle to its diameter.

π



Leonhard Euler popularized the use of the Greek letter π in works he published in 1736 and 1748.



John von Neumann was part of the team that first used a digital computer, ENIAC, to compute π .

<http://en.wikipedia.org/wiki/Pi>

pi Pioneers



Archimedes developed the polygonal approach to approximating π .



Isaac Newton used infinite series to compute π to 15 digits, later writing "I am ashamed to tell you how many figures I carried these computations".



William Jones was most noted for his proposal for the use of the symbol π (the Greek letter *pi*) to represent the ratio of the circumference of a circle to its diameter.

π



Leonhard Euler popularized the use of the Greek letter π in works he published in 1736 and 1748.



John von Neumann was part of the team that first used a digital computer, ENIAC, to compute π .

<http://en.wikipedia.org/wiki/Pi>

pi Pioneers



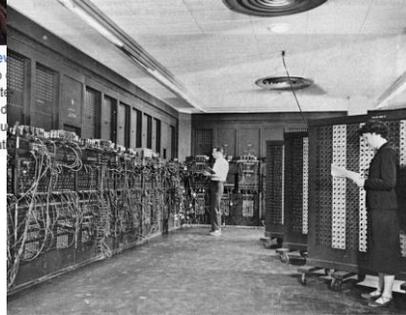
Archimedes developed the polygonal approach to approximating π .



Isaac Newton developed a series to approximate π using infinite series to digits, later he was ashamed of many figures in his computational work.



John von Neumann was part of the team that first used a digital computer, ENIAC, to compute π .



<http://en.wikipedia.org/wiki/Pi>

Progress of Computing pi

3.14159265358979323846264338327...

Pi decimal places	Date	Who / Where / How
3.1416	2589-2566 BC	Egyptian Pyramid
3.1250	1900-1600 BC	Babylon
3.1622	600 BC	India
3.1408 to 3.1429	250 BC	Greece - Archimedes
3.1416	150 AD	Greek-Roman - Ptolemy
3.141592920	480 AD	China – Zu Chongzhi
16 digits	1424	Persia – Jamshīd al-Kāshī
39 digits	1630	Polygon approach
71 digits	1699	Infinite series
620 digits	1964	Daniel Ferguson – without a calculator
2037 digits	1949	ENIAC - Reitwiesner and John von Neumann
1 million digits	1973	Computer

Progress of Computing pi

3.14159265358979323846264338327...

Pi decimal places	Date	Who / Where / How
3.1416	2589-2566 BC	Egyptian Pyramid
3.1250	1900-1600 BC	Babylon
3.1622	600 BC	India
3.1408 to 3.1429	250 BC	Greece - Archimedes
3.1416	150	British mathematician William Shanks famously took 15 years to calculate π to 707 digits, but made a mistake in the 528th digit, rendering all subsequent digits incorrect. (1853 AD)
3.141592920	480	
16 digits	142	
39 digits	163	
71 digits	169	
620 digits	1964	Daniel Ferguson – without a calculator
2037 digits	1949	ENIAC - Reitwiesner and John von Neumann
1 million digits	1973	Computer

Two π Computation Algorithms

Gregory-Leibniz series: [\(covered here\)](#)

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

Number of terms ----->
that pi is computed (times through the loop)

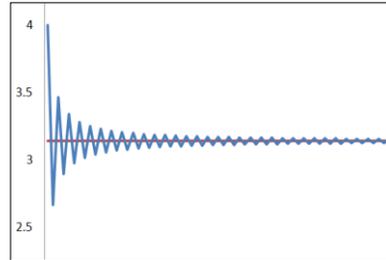
Nilakantha series: [\(discussed very briefly\)](#)

$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \dots$$

Two π Computation Algorithms

Gregory-Leibniz series: See how the computed value of pi converges on the actual value of pi.

Loop	Numerator	Denominator	Fraction	Computed	Diff
1	4	1	4.00000	4.00000	0.85841
2	4	3	1.33333	2.66667	0.47493
3	4	5	0.80000	3.46667	0.32507
4	4	7	0.57143	2.89524	0.24635
5	4	9	0.44444	3.33968	0.19809
6	4	11	0.36364	2.97605	0.16555
7	4	13	0.30769	3.28374	0.14215
8	4	15	0.26667	3.01707	0.12452
9	4	17	0.23529	3.25237	0.11077
10	4	19	0.21053	3.04184	0.09975



The difference between the **computed** value and the **true** value of pi gets smaller each time through the loop.

Program Organization

The title block at the top of the program is made up of comments that briefly describe the program, date, version, programmer, etc.

The **include** files come next.

The name of the program section, in this case it is **main**, and its "argument" list.

Curly braces { and } surround the block of code that belongs to main.

The body of the program is organized into the list of variables, input, processing and output.

A return statement ends the program.

Title Block

Program Name, Date, Version
Programmer Name

#include <filenames>

```
int main (int argc, char* argv[ ])
```

```
{
```

```
// list of variables
```

```
// input
```

```
// process
```

```
// output
```

```
return 0;
```

```
}
```

Include Statements

```
#include <iostream> // for cin and cout
#include <iomanip> // digits past decimal
#include <cmath> // for M_PI constant
using namespace std;
```

NOTE: It can be difficult to access M_PI when using some versions of Microsoft Visual C++. If your program is not able to get M_PI using **#include <cmath>** then define it at the top of the program with:

```
const double M_PI = 3.14159265358979;
```

Title Block

Program Name, Date, Version
Programmer Name

#include <filenames>

```
int main (int argc, char* argv[ ])
{
    // list of variables
    // input
    // process
    // output
    return 0;
}
```

int main (...) and open curly-brace

```
int main (int argc, char* argv[ ])
{
```

Title Block

Program Name, Date, Version
Programmer Name

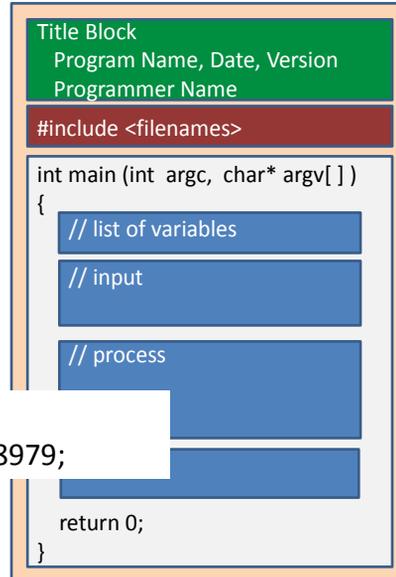
#include <filenames>

```
int main (int argc, char* argv[ ])
{
    // list of variables
    // input
    // process
    // output
    return 0;
}
```

List of Variables and Constants

```
// list of variables and constants
double numerator = 4.0;
double denominator = 1.0;
double fraction = 1.0;
double pi = 0.0;
int terms;    // read from user

// for Microsoft Visual C++
const double M_PI = 3.14159265358979;
```



Input the Number of Terms

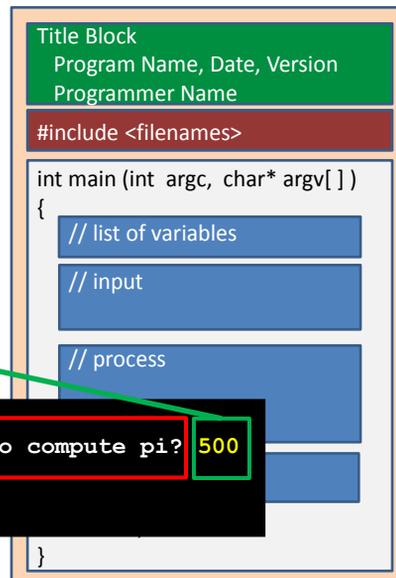
INPUT:
Use a cout and a cin to input the number of **terms**

cout

cin
User input

How many terms do you want to compute pi?

500



Program Organization

PROCESS:

Use a loop to compute
the value of pi.
More information later

```

Title Block
Program Name, Date, Version
Programmer Name
#include <filenames>

int main (int argc, char* argv[ ])
{
    // list of variables
    // input
    // process
    // output
    return 0;
}

```

Program Organization

OUTPUT:

- 1) computed pi
- 2) M_PI constant (expected)
- 3) difference between the
computed value and the
M_PI constant.

```

Title Block
Program Name, Date, Version
Programmer Name
#include <filenames>

int main (int argc, char* argv[ ])
{
    // list of variables
    // input
    // process
    // output
    return 0;
}

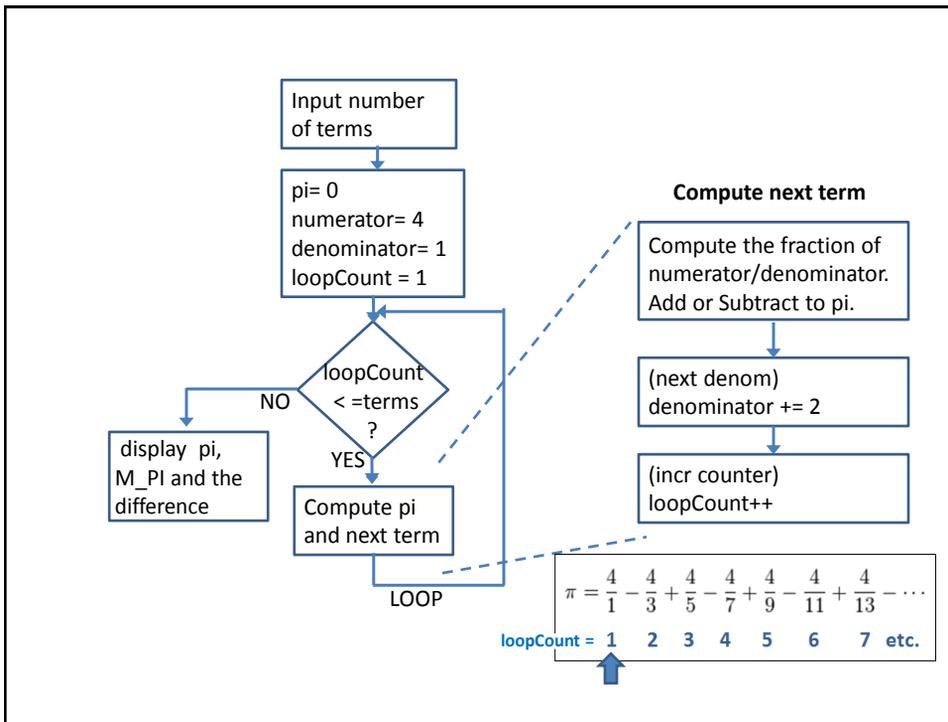
```

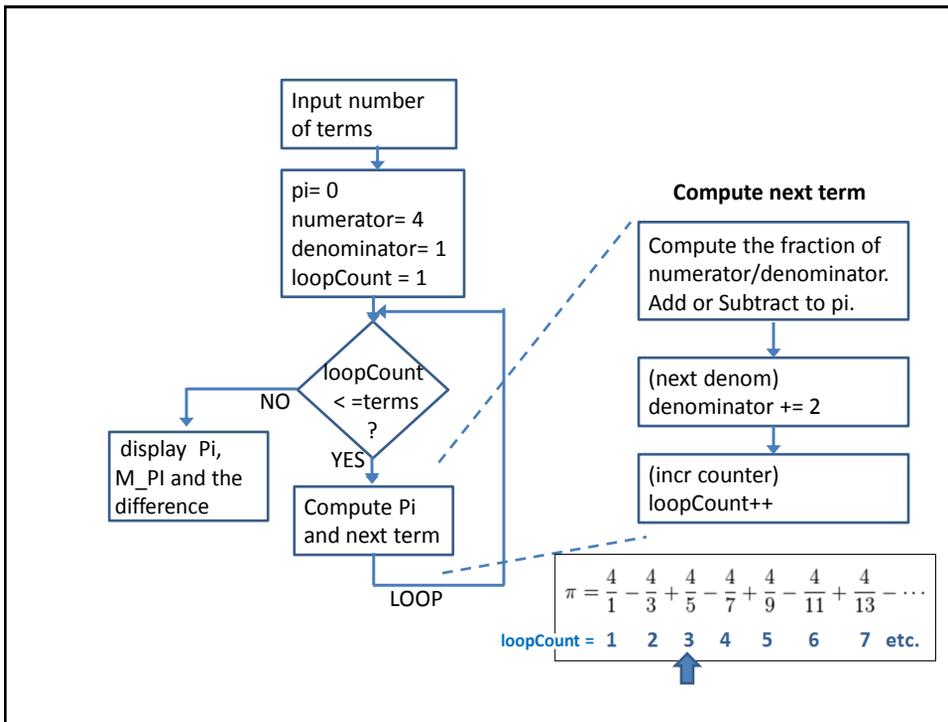
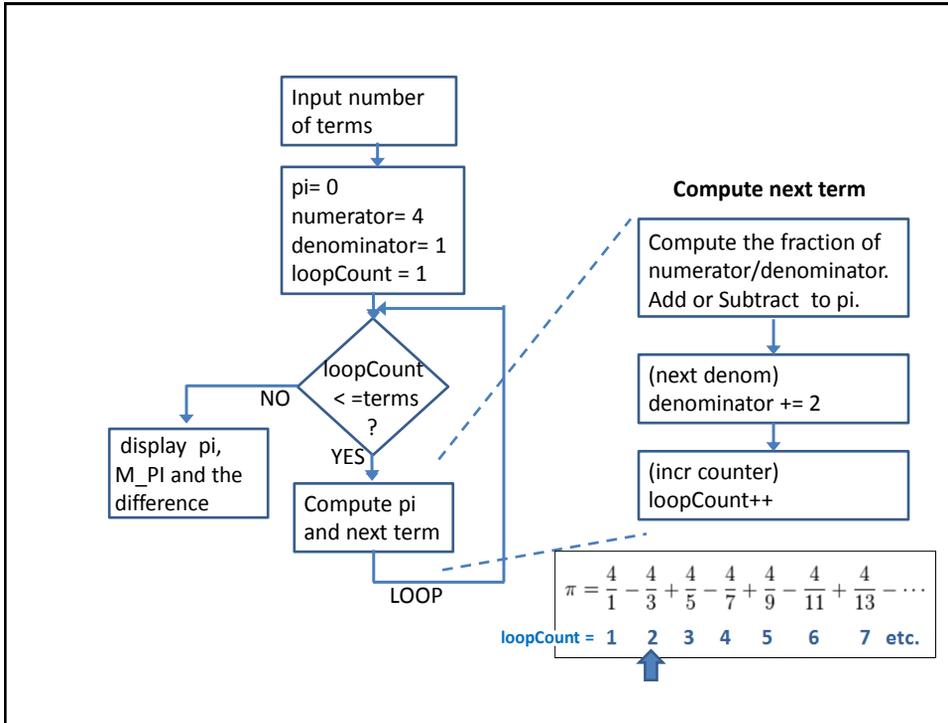
Use a Loop

Gregory-Leibniz series:

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

loopCount **1** **2** **3** **4** **5** **6** **7** **etc.**





Loop Control (**for** loop)

Use either a **while** loop or a **for** loop.

```
for (int i=1; i<=terms; i++)
{
    // body of loop
    // add sum of fractions
}
// Use cout to display the results
// after the loop ends
```

Loop Control (**for** loop)

```
cout << "How many terms do you want to compute pi? ";
cin  >> terms;
```

```
for (int i=1; i<=terms; i++)
{
    // body of loop
    // add sum of fractions
}
// Use cout to display the results
// after the loop ends
```



terms was input from the keyboard using cin before the loop starts

numerator = 4

Gregory-Leibniz series:

numerator always
is equal to 4

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

loopCount **1** **2** **3** **4** **5** **6** **7** **etc.**

Compute denominator

Gregory-Leibniz series:

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

Add 2 to denominator
each pass through the
loop

loopCount **1** **2** **3** **4** **5** **6** **7** **etc.**

Don't use the `int` Data Type

to compute the fraction

This is what happens if you use `int` data types for the numerator, denominator and fraction computation:

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

integer data types:

$$\begin{array}{r} Q=0, R=4 \\ 4 \overline{) 5} \end{array}$$

$$\begin{array}{cccccccc} 4 & - & 1 & + & 0 & - & 0 & + & 0 & - & 0 & + & 0 & = & 3 \\ Q=4, R=0 & & Q=1, R=1 & & Q=0, R=4 & & \end{array}$$

Use the `double` Data Type

Use the `double` data type for numerator, denominator and the fraction computation

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

integer data types:

$$\begin{array}{r} 0.800 \\ 4.0 \overline{) 5.0} \end{array}$$

$$\begin{array}{cccccccc} Q=4.0000 & Q=1.3333 & Q=0.8000 & Q=0.5714 & Q=0.4444 & Q=0.3636 & Q=0.3077 \\ \end{array}$$

Add or Subtract?

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

Add or subtract each
pass through the loop
+ - + - etc.

Add or Subtract the fraction

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

loopCount	1	2	3	4	5	6	7	etc.
	Odd	Even	Odd	Even	Odd	Even	Odd	

```

if (i % 2) // determine Odd or Even
    pi += fraction; // Odd, then add fraction
else
    pi -= fraction; // Even, subtract fraction
  
```

Why Does %2 Work?

If all we are concerned about is determining if loopCount is ODD or EVEN, use loopCount%2

With integer division, / gives the quotient

1 / 2 Q=0 R=1 % gives the remainder

2 / 2 Q=1 R=0

3 / 2 Q=1 R=1

4 / 2 Q=2 R=0

5 / 2 Q=2 R=1

Why Does %2 Work?

If all we are concerned about is determining if an integer is ODD or EVEN, then use %2

Examples:

1 % 2 = 1

2 % 2 = 0

3 % 2 = 1

4 % 2 = 0

5 % 2 = 1

Since TRUE is defined as non-zero and FALSE is defined as zero in C and C++, loopCount%2 can be used as a TRUE/FALSE evaluation.

Compute pi with loopCount == 1

```

pi = 0.0;
numerator = 4.0;
denominator = 1.0;
for (int i=1; i<=terms; i++)
{
    fraction = numerator/denominator; // compute 4.0/1.0
    if (i % 2) // Determine ODD or EVEN counter
        pi += fraction; // Add fraction if loopCount is ODD
    else
        pi -= fraction; // Subtract if loopCount is EVEN

    // prepare for next iteration through the loop
    denominator += 2.0; // 1.0, 3.0, 5.0, 7.0, 9.0, etc.
}

```

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

Compute pi with loopCount == 1

```

pi = 0.0;
numerator = 4.0;
denominator = 1.0;
for (int i=1; i<=terms; i++)
{
    fraction = numerator/denominator;
    if (i % 2) // Determine ODD or EVEN counter
    pi += fraction; // Add fraction if loopCount is ODD
    else
    pi -= fraction; // Subtract if loopCount is EVEN

    // prepare for next iteration through the loop
    denominator += 2.0; // 1.0, 3.0, 5.0, 7.0, 9.0, etc.
}

```

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

Prepare denominator for Next Loop

```

pi = 0.0;
numerator = 4.0;
denominator = 1.0;
for (int i=1; i<=terms; i++)
{
    1
    fraction = numerator/denominator;
    if (i % 2) // Determine ODD or EVEN counter
        pi += fraction; // Add fraction if loopCount is ODD
    else
        pi -= fraction; // Subtract if loopCount is EVEN

    // prepare for next iteration through the loop
    denominator += 2.0; // 1.0, 3.0, 5.0, 7.0, 9.0, etc.
}

```

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

A blue arrow points to the denominator '1' in the first term of the series. A red arrow points from the '2.0' in the code to the '3.0' in the series, indicating the next denominator value.

loopCount is Incremented

```

pi = 0.0;
numerator = 4.0;
denominator = 1.0;
for (int i=1; i<=terms; i++)
{
    2
    fraction = numerator/denominator;
    if (i % 2) // Determine ODD or EVEN counter
        pi += fraction; // Add fraction if loopCount is ODD
    else
        pi -= fraction; // Subtract if loopCount is EVEN
    // prepare for next iteration through the loop
    denominator += 2.0; // 1.0, 3.0, 5.0, 7.0, 9.0, etc.
}

```

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

A blue arrow points to the denominator '1' in the first term of the series. A red arrow points from the '2' in the code to the '3.0' in the series, indicating the next denominator value.

Compute pi with loopCount == 2

```

pi = 0.0;
numerator = 4.0;
denominator = 1.0;
for (int i=1; i<=terms; i++)
{
    fraction = numerator/denominator; // Compute 4.0/3.0
    if ( loopCount % 2) // Determine ODD or EVEN counter
        pi += fraction; // Add fraction if loopCount is ODD
    else
        pi -= fraction; // Subtract if loopCount is EVEN

    // prepare for next iteration through the loop
    denominator += 2.0; // 1.0, 3.0, 5.0, 7.0, 9.0, etc.
}

```

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

Compute pi with loopCount == 2

```

pi = 0.0;
numerator = 4.0;
denominator = 1.0;
for (int i=1; i<=terms; i++)
{
    fraction = numerator/denominator;
    if ( i % 2) // Determine ODD or EVEN counter
        pi += fraction; // Add fraction if loopCount is ODD
    else
        pi -= fraction; // Subtract if loopCount is EVEN

    // prepare for next iteration through the loop
    denominator += 2.0; // 1.0, 3.0, 5.0, 7.0, 9.0, etc.
}

```

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

Compute pi with loopCount == 2

```

pi = 0.0;
numerator = 4.0;
denominator = 1.0;
for (int i=1; i<=terms; i++)
{
    fraction = numerator/denominator;
    if ( i % 2) // Determine ODD or EVEN counter
        pi += fraction; // Add fraction if loopCount is ODD
    else
        pi -= fraction; // Subtract if loopCount is EVEN

    // prepare for next iteration through the loop
    denominator += 2.0; // 1.0, 3.0, 5.0, 7.0, 9.0, etc.
}

```

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



Ready to Display the Results

Format 10 Digits Past Decimal

Use `#include <iomanip>` to define how many digits are to be displayed past the decimal. Then you have access to the `setiosflags` and `setprecision`. Type this once before the other `cout` statements that display the output:

```

cout << setiosflags(ios::fixed)
      << setiosflags(ios::showpoint);

```

Display the **M_PI** Predefined Constant

NOTE: It can be difficult to access `M_PI` when using some versions of Microsoft Visual C++. If your program is not able to get `M_PI` using `#include <cmath>` then define it in at the top of the program with:

```
const double M_PI = 3.14159265358979;
```

or in the C-language (without `=` or `;`)

```
#define M_PI 3.14159265358979
```

Display the **M_PI** Predefined Constant

The math header files contain a predefined constant for pi named `M_PI` of type `double`. The C++ program can display ten digits:

```
cout << "pi constant = "  
      << setprecision(10) << M_PI << endl;
```

Display the Computed pi Value

```
cout << "Computed pi = "  
      << setprecision(10) << pi << endl;
```

Display the Difference Between Values

```
cout << "Difference = " << setprecision(10)  
      << fabs(pi - M_PI) << endl;
```

The difference between the computed pi (π) and the constant from the math header file (M_PI) is

$$\pi - M_PI$$

The `fabs()` function returns the absolute value of a floating point number so that the difference will always be shown as a positive number

$$\text{fabs}(\pi - M_PI)$$

The Compute_pi Program (in C++)

```
// C++ language version: Compute_pi.cpp
//
// Compute pi = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 ... + (4*(-1)^n/(2n+1) )
//

#include <iostream>    // used by cin and cout
#include <iomanip>      // used by setios, precision, etc.
#include <cmath>       // math functions and constants
using namespace std;

int main(int argc, char* argv[])
{
    // list of variables and constants
    int loopCount;
    int terms;        // read from user
    double pi;
    double numerator;
    double denominator;
    double fraction;
    const double M_PI = 3.14159265358979;    // for Microsoft Visual C++

    // get the number of times through the loop from the user
    cout << "How many terms do you want to compute Pi? ";
    cin >> terms;
}
```

The Compute_pi Program (in C++)

```
// Initialize the variables and go through the loop
pi = 0;
numerator = 4;
denominator = 1;
for (loopCount=1; loopCount<=terms; loopCount++)
{
    fraction = numerator / denominator;
    if ( loopCount % 2)    // Determine ODD or EVEN counter
        pi += fraction;    // Add fraction if loopCount is ODD
    else
        pi -= fraction;    // Subtract if loopCount is EVEN
    // prepare for next iteration through the loop
    denominator += 2;    // 1, 3, 5, 7, 9, etc
}

// Set flags to display a set number of digits past the decimal
cout << setiosflags(ios::fixed) << setiosflags(ios::showpoint);
cout << "PI constant = " << setprecision(10) << M_PI << endl;
cout << "Computed PI = " << setprecision(10) << pi << endl;
cout << "Difference = " << setprecision(10) << fabs(pi - M_PI) << endl;

return 0;
}
```

Lab Result

When your program is done computing pi, have the program display the computed value of pi, the M_PI constant from the math header file and the difference between your computed value and the constant from the header file.

```
How many terms do you want to compute pi? 500
Computed pi    = 3.1395926556
M_PI constant  = 3.1415926536
Difference     = 0.0019999980
```

Conclusion

This presentation shows all of the pieces to construct a C++ program to compute pi.

You need to run the program several times with different number of terms (terms), **10, 100, 1000, 10000 and 100000** to see how close the computed value of pi becomes with more terms.

Option #2

Here is some additional information if you are interested in computing pi using the Nilakantha series. You only need to use one method to compute pi for the lab assignment, but you can choose either the Gregory-Leibniz or the Nilakantha series.

Nilakantha Series

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

Nilakantha Series

The Nilakantha series converges on the correct value of pi much faster than the Gregory-Leibniz series, but the denominator has three values:

$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \dots$$

If you choose to use the Nilakantha series, then set d1=2, d2=3, d4=5, compute the denominator before computing the fraction. Add 2 to these values at the end of the loop to prepare for the next calculation of the denominator.

The End

ACKNOWLEDGEMENTS:

IMAGES were taken from Wikimedia Commons