C / C++

Computing Pi

π = 3.14159265358979323…

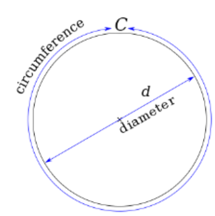
The constant π is represented in this mosaic outside the mathematics building at the *Technische Universität Berlin*.

What is Pi π

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

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*.



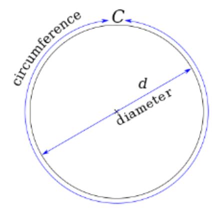
Properties

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.

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

What is Pi π

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



$\frac{22}{7}$ = 3.1428571428571…

π = 3.1415926535898…

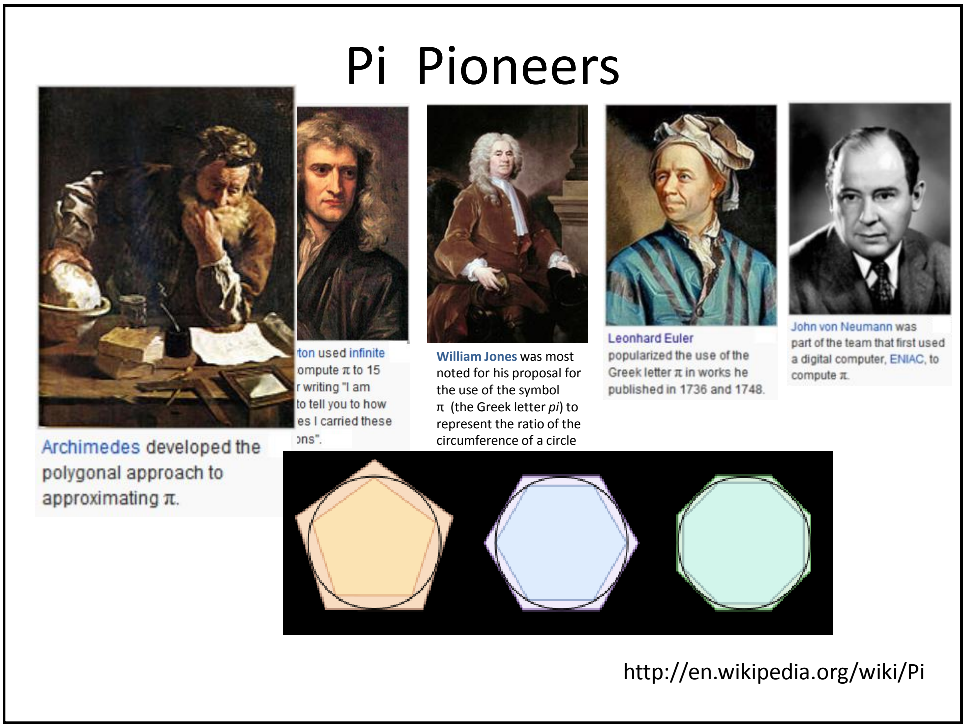
Error = 0.0012644892673…

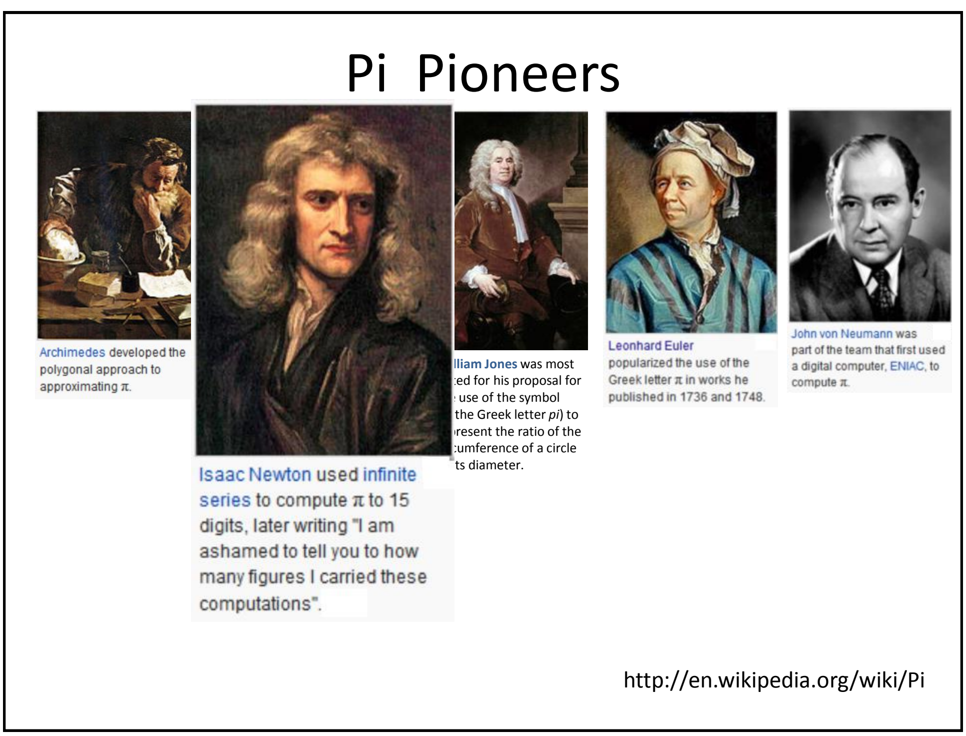
Computers and pi

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.

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

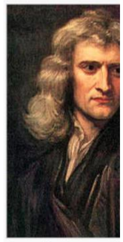
$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \cdots$$


Pi Pioneers
Archimedes developed the polygonal approach to approximating π.
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".
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 i series to compute π digits, later writing "I ashamed to tell you t many figures I carrie 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.

eonhard Euler opularized the use of the reek letter π in works he ublished in 1736 and 1748.

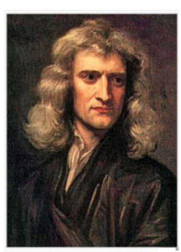
John von Neumann was part of the team that first used a digital computer, ENIAC, to compute π.

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 w noted for his pr the use of the s π (the Greek le represent the r circumference to its diameter.

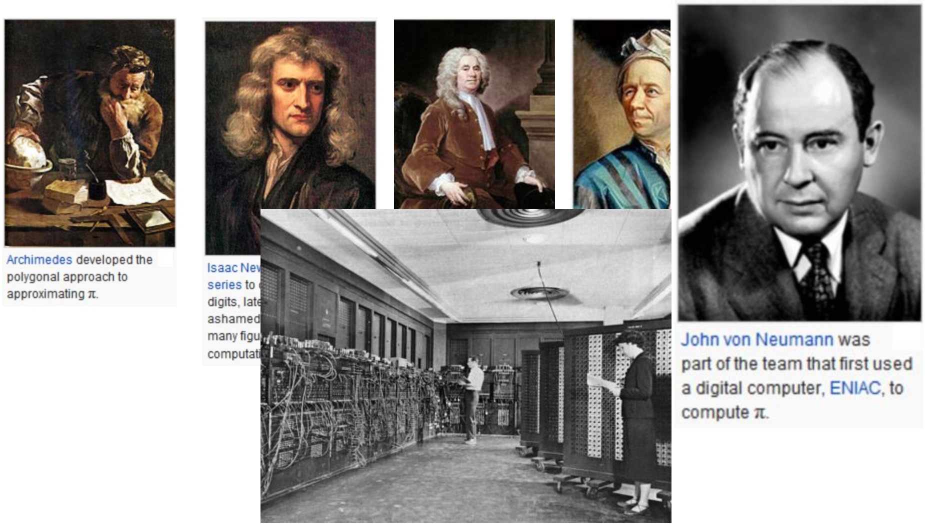
Leonhard Euler popularized the use of the Greek letter π in works he published in 1736 and 1748.

ohn von Neumann was art of the team that first used digital computer, ENIAC, to ompute π.

π

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

Pi Pioneers



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	
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

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)

Two π Computation Algorithms

Gregory-Leibniz series: (covered here)

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

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

Nilakantha series: (discussed very briefly)

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

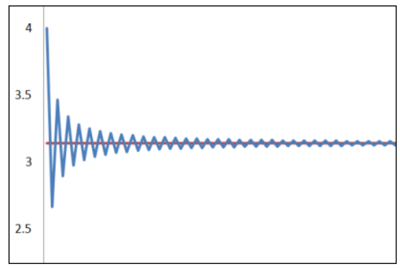
Two π Computation Algorithms

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

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

PI
3.141593

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.

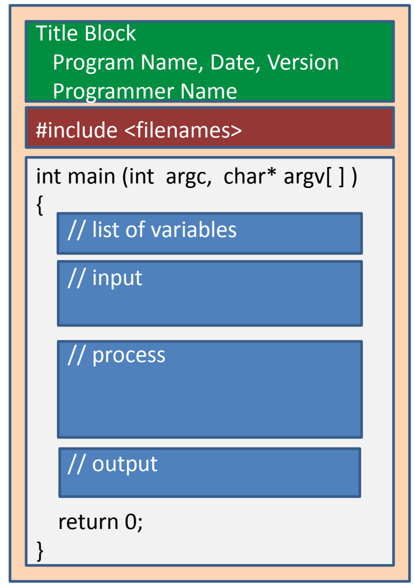
```
Title Block
   Program Name, Date, Version
   Programmer Name

#include <filenames>

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

Include Statements

```
#include "stdafx.h"    // if using Microsoft C++
#include <iostream>  // for cin and cout
#include <iomanip>   // digits past decimal
#include <cmath>      // for M_PI constant
using namespace std;
```

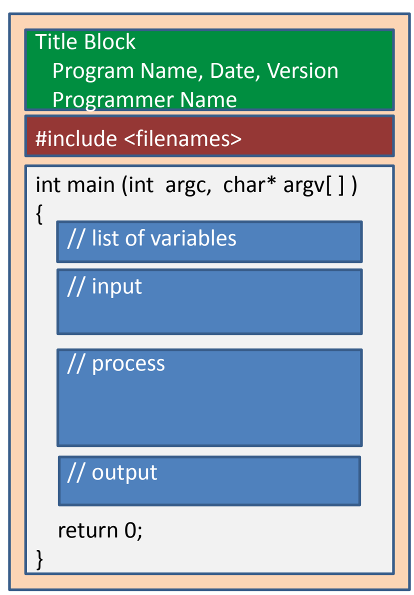
NOTE: It can be difficult to access M_PI when using some versions of Microsoft Visual C++. If your program is not able to get M_PI using **#include <cmath>** then define it at the top of the program with:

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



int main (…) and open curly-brace

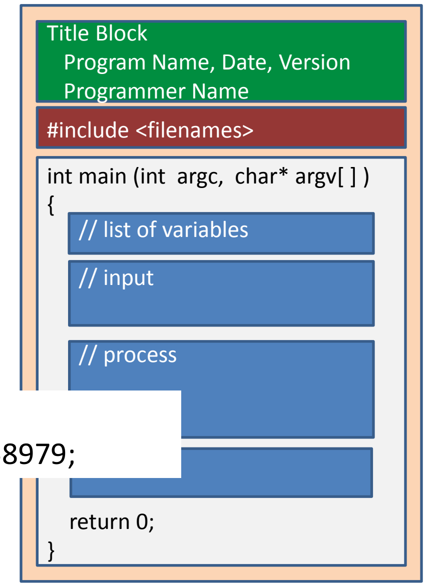
```
int main (int  argc,  char* argv[ ] )
{
```



List of Variables and Constants

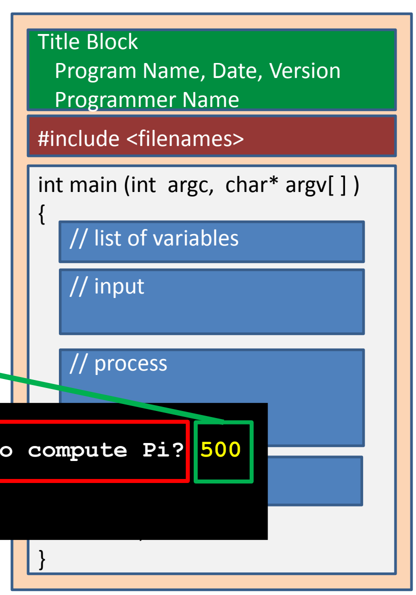
// list of variables and constants
double numerator = 4.0;
double denominator = 1.0;
double fraction = 1.0;
double pi = 0.0;
int terms; // read from user

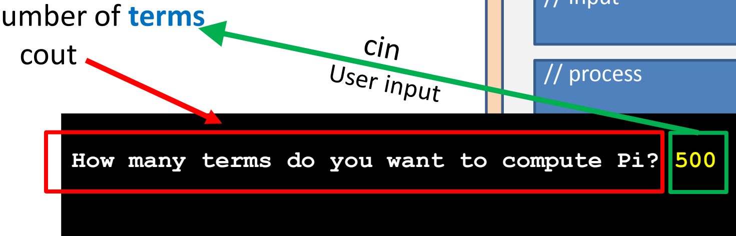
// for Microsoft Visual C++
const double M_PI = 3.14159265358979;



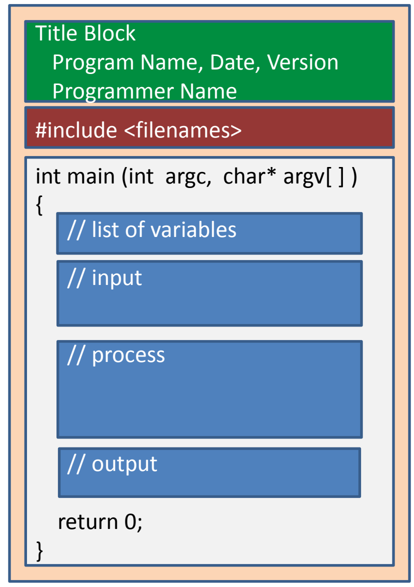
Input the Number of Terms

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



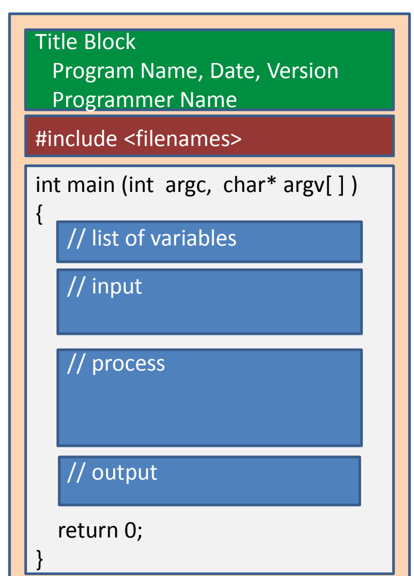


Program Organization



PROCESS:
Use a loop to compute the value of Pi.
More information later

Program Organization



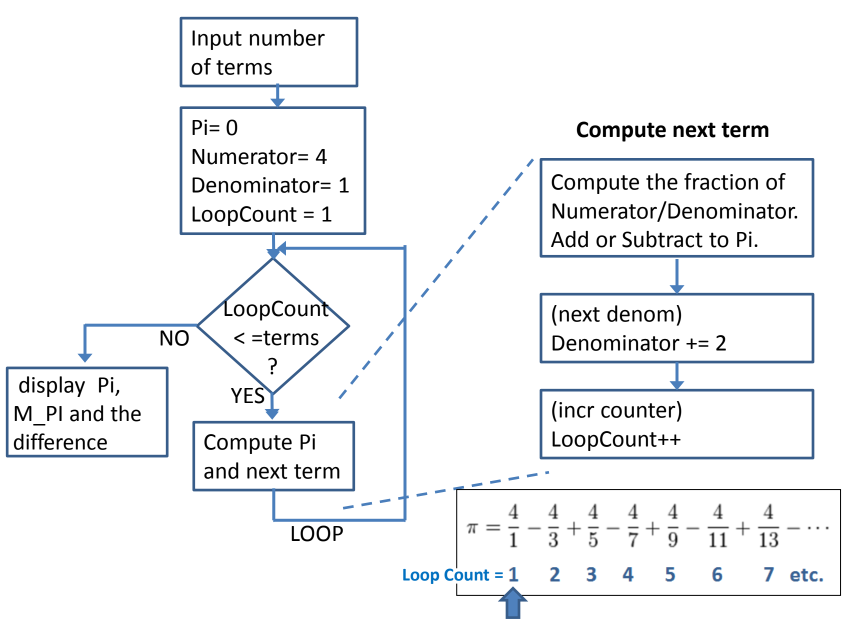
OUTPUT:
1) computed Pi
2) M_PI constant (expected)
3) difference between the computed value and the M_PI constant.

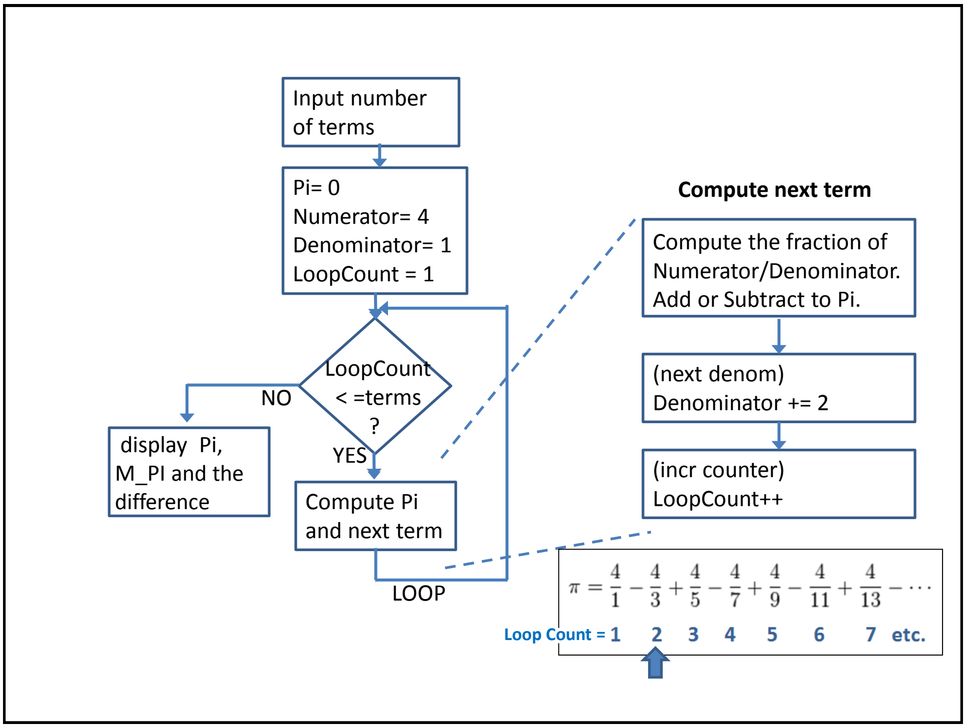
Use a Loop

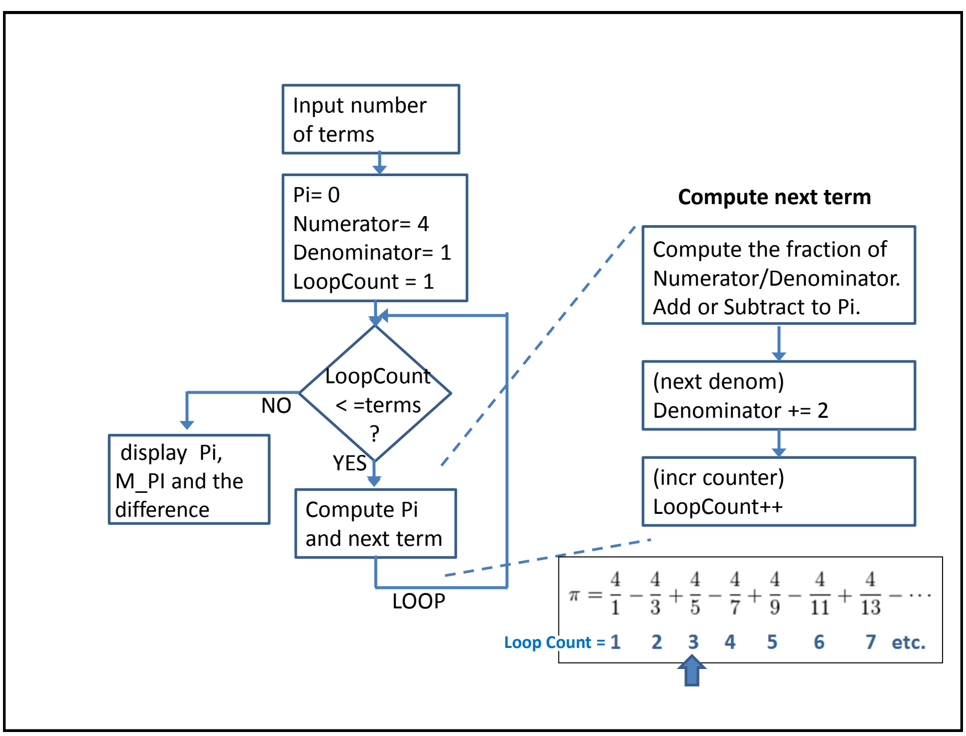
Gregory-Leibniz series:

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

LoopCounter 1 2 3 4 5 6 7 etc.




Input number of terms
Pi= 0
Numerator= 4
Denominator= 1
LoopCount = 1
LoopCount < =terms ?
NO
YES
display Pi, M_PI and the difference
Compute Pi and next term
LOOP
Compute next term
Compute the fraction of Numerator/Denominator. Add or Subtract to Pi.
(next denom) Denominator += 2
(incr counter) LoopCount++
$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \frac{4}{13} - \cdots$
Loop Count = 1 2 3 4 5 6 7 etc.


Input number of terms
Pi= 0
Numerator= 4
Denominator= 1
LoopCount = 1
LoopCount < =terms ?
NO
YES
display Pi, M_PI and the difference
Compute Pi and next term
LOOP
Compute next term
Compute the fraction of Numerator/Denominator. Add or Subtract to Pi.
(next denom) Denominator += 2
(incr counter) LoopCount++
$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \frac{4}{13} - \cdots$
Loop Count = 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

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

LoopCount 1 2 3 4 5 6 7 etc.

Compute Denominator

Gregory-Leibniz series:

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

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 <u>int</u> data types for the numerator, denominator and fraction computation:

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

integer data types:

Q=0, R=4

4) 5

4 - 1 + 0 - 0 + 0 - 0 + 0 = 3

Q=4, R=0 | Q=1, R=1 | Q=0, R=4 | Q=0, R=4 | Q=0, R=4 | Q=0, R=4 | Q=0, R=4

Use the **double** Data Type

Use the <u>double</u> data type for Numerator, Denominator and the Fraction computation

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

integer data types:

0.800

4.0) 5.0

Q=4.0000 | Q=1.3333 | Q=0.8000 | Q=0.5714 | Q=0.4444 | Q=0.3636 | Q=03077

Add or Subtract?

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

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

Add or Subtract the Fraction

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

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++, LoopCounter%2 can be used as a TRUE/FALSE evaluation.

Compute Pi with LoopCount == 1

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

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

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

Compute Pi with LoopCount == 1

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

```
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 LoopCounter is ODD
    else
        pi -= Fraction;     // Subtract if LoopCounter is EVEN

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

Prepare Denominator for Next Loop

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

```
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 LoopCounter is ODD
    else
        pi -= fraction;      // Subtract if LoopCounter is EVEN

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

LoopCount is Incremented

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

```
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 LoopCounter is ODD
    else
        pi -= fraction;      // Subtract if LoopCounter is EVEN
    // prepare for next iteration through the loop
    denominator += 2.0;    // 1.0,  3.0,  5.0,  7.0,  9.0,  etc.
}
```

Compute Pi with LoopCount == 2

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

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

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

Compute Pi with LoopCount == 2

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

```
pi = 0.0;
numerator = 4.0;
denominator = 1.0;
for (int i=1; i<=terms; i++)
               3
{
    fraction = numerator/denominator;
    if ( i % 2)    // Determine ODD or EVEN counter
        pi += Fraction;      // Add fraction if LoopCounter is ODD
    else
        pi -= Fraction;      // Subtract if LoopCounter is EVEN

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

Compute Pi with LoopCount == 2

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

```
pi = 0.0;
numerator = 4.0;
denominator = 1.0;
for (int i=1; i<=terms; i++)
                4
{
    fraction = numerator/denominator;
    if ( i % 2)    // Determine ODD or EVEN counter
        pi += Fraction;      // Add fraction if LoopCounter is ODD
    else
        pi -= Fraction;      // Subtract if LoopCounter is EVEN

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

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;
```

The C program can use printf to display ten digits:

```
printf ("%.10lf\n", M_PI);
```

Number-one

Small letter-L (use long float to display a double)

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 (pi) 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

```
fabs(Pi - M_PI)
```

The **Compute_PI** Program (in C++)

```
//
//  Compute PI = 4/1 - 4/3 + 4/5 - 4/7 + 4/9  ... + (4*(-1)^n/(2n + 1) )
//
#include "stdafx.h"         // only for Microsoft Visual C++
#include <iostream>         // used by cin and cout
#include <iomanip>          // used to set number of digits past decimal
#define _USE_MATH_DEFINES   // may help to get M_PI
#include <math.h>           // math functions and constants
using namespace std;

// If you are not able to use M_PI from math.h then,
// const double M_PI = 3.14159265358979323846264338832795;

int main(int argc, char* argv[])
{
    double numerator = 4.0;
    double denominator = 1.0;
    double fraction = 1.0;
    double pi = 0.0;
    int terms;          // read from the user

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

The **Compute_PI** Program (in C++)

```
    for (int i=1; i<terms; i++)
    {
        fraction = numerator/denominator;
        if (i % 2)          // determine if loop count is odd or even
            pi += fraction; // add fraction to pi if loop count is ODD
        else
            pi -= fraction; // subtract from pi if loop count is EVEN
        denominator += 2;   // denominator will be 1, 3, 4, 7, 9, etc.
    }
    // set flags to display a set number of digits past the decimal
    cout << setiosflags(ios::fixed) << setiosflags(ios::showpoint);
    cout << " PI  = " << pi << endl;
    cout << "M_PI = " << M_PI << endl;
    cout << "difference = " << pi-M_PI << endl << endl;
    system("PAUSE");        // stop the screen from disappearing
    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.

C header file #include <math.h>

C++ header file #include <cmath>

```
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

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

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:

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

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