Prependix B  What You Need To Know Now

B.1  General

The student beginning this course must have a good working knowledge of elementary algebra, pre-calculus or analytic geometry, and the elements of calculus. A knowledge of matrix notation and simple linear algebra is required.

The student must know how to write a computer program, with one or more subprograms, in some high-level computer language such as C, C++, Java, Pascal, Basic, or Fortran. Recursion, pointers, structures, and dynamic storage allocation will not be necessary.

Radians

All angles will be measured in radians unless otherwise specified! The student who works a problem with a calculator set on DEG or GRAD instead of on RAD will likely get wrong answers. Don’t do that! Use radians only.

Logarithms

\[ y = \log_b(x) \] means exactly the same thing as \[ x = b^y \]

Formulas involving logarithms:

\[ \log(xy) = \log(x) + \log(y) \]
\[ \log(x/y) = \log(x) - \log(y) \]
\[ \log(x^p) = p \log(x) \]

We can evaluate a logarithm \( \log_b(x) \) in a different base \( B \) using the formula

\[ \log_b(x) = \frac{\log_a(x)}{\log_a(b)} \]

**Example:** Compute \( \log_2(170) \).

**Solution:** Use base \( B=e=2.718\ldots \)

\[ \log_2(170) = \frac{\log_e(170)}{\log_e(2)} \]
\[ = \frac{\ln(170)}{\ln(2)} \]
\[ \approx 5.13579844 / 0.693147181 \approx 7.40939094 \]
Or, we could use base \( B=10 \) and get the same answer:

\[
\log_2(170) = \frac{\log_{10}(170)}{\log_{10}(2)}
\approx \frac{2.3044892}{0.30103000} \approx 7.40939094
\]

**Inverse functions**

\( f^{-1}(y) \) means “the inverse of the function \( f \)”: 

\[
x = f^{-1}(y),
\]

if it exists, means exactly the same thing as 

\[
y = f(x)
\]

The problem “Find the inverse of the function \( f(x) \)” means the same thing as “Solve the equation \( y=f(x) \) for \( x \) as a function of \( y \).” It may or may not be possible to do this analytically; there are many functions \( f(x) \) that can be expressed in terms of simple functions of \( x \) but for which the inverse function \( f^{-1}(y) \) cannot be expressed in terms of simple functions of \( y \).

The inverse function \( x = f^{-1}(y) \) exists for \( x \) on an interval \([a,b]\) only if the function \( y = f(x) \) is continuous and monotonic on that interval.

In plotting a graph of \( x=f^{-1}(y) \) from a graph of \( y=f(x) \), we interchange the \( x \) and \( y \) axes or, equivalently, “flop the paper over, keeping the 45-degree line \( y=x \) fixed”.

**Examples:**

If \( y = f_1(x) = x^2 \) then \( x = f_1^{-1}(y) = \sqrt{y} \) for \( y \geq 0 \) or \( x = -\sqrt{y} \) for \( y \geq 0 \)

If \( y = f_2(x) = e^x \) then \( x = f_2^{-1}(y) = \ln(y) \) for \( y > 0 \)

If \( y = f_3(x) = x^2 - 3x + 2 \) then by the quadratic formula,

\[
x^2 - 3x + (2 - y) = 0
\]

and either \( x = f_3^{-1}(y) = (1/2)(3 + \sqrt{9 - 4(2 - y)}) \) for \( x \geq 3/2 \)

or \( x = f_3^{-1}(y) = (1/2)(3 - \sqrt{9 - 4(2 - y)}) \) for \( x \leq 3/2 \)

[2 graphs]
Sums of powers of \( i \)

For counting the number of operations in some algorithms, the student will need to know the following formulae.

**Formula 1:** \[ \sum_{i=1}^{n} 1 = n \quad , \quad n \geq 0 \]

**Example:** \[ \sum_{i=1}^{4} 1 = 1 + 1 + 1 + 1 = 4 \]

**Formula 2:** \[ \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \quad , \quad n \geq -1 \]

**Example:** \[ \sum_{i=1}^{4} i = 1 + 2 + 3 + 4 = 10 = \frac{4(4 + 1)}{2} \]

**Formula 3:** \[ \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \quad , \quad n \geq -1 \]

**Example:** \[ \sum_{i=1}^{4} i^2 = 1 + 4 + 9 + 16 = 30 = \frac{4(4 + 1)(2(4) + 1)}{6} \]

These can also appear with lower limits other than \( i=1 \), for example

\[ \sum_{i=m}^{n} 1 = \sum_{i=1}^{n} 1 - \sum_{i=1}^{m-1} 1 = n - (m - 1) = n - m + 1 \quad , \quad n \geq m - 1 \]

This is known as “the fencepost theorem”: if you walk along a fence and count fenceposts from number \( m \) through number \( n \) inclusive, you have counted \( n-m+1 \) fenceposts.
Factorials

n! means “n factorial”, which is equal to n(n - 1)(n - 2)(n - 3)...(3)(2)(1) if n is an integer greater than zero. 0!, “zero factorial”, is defined to be equal to 1. This may seem to be a strange definition for 0!, but with this definition the point (0,0!) lies on a simple smooth curve passing through the infinite set of points (x,y) = (x,x!), namely the complete gamma function of Euler, \( \Gamma(x+1) \), and with any other definition it would not.

Infinite geometric series

If \(|x|<1\), then \[
1 + x + x^2 + x^3 + ... = \frac{1}{1 – x}
\]

Example: \[
1 + \frac{1}{3} + \frac{1}{9} + \frac{1}{27} + ... = \frac{1}{1 – \frac{1}{3}} = \frac{1}{\frac{2}{3}} = \frac{3}{2}
\]

The triangle inequality

If x, y, and z are positive, negative, or zero real numbers (or complex numbers) and if

\[
z = x + y , \quad \text{then} \quad |z| \leq |x| + |y| .
\]

The same inequality is also true if \( z = x - y \).

Continuity

If a function \( f(x) \) is continuous on a closed interval \([a,b]\), then the function is also bounded on this interval.

Example:

The function \( f(x) = \sqrt{x} \) is continuous on the interval \([0,1]\).

The first derivative of \( f(x) \), \( f'(x) = 1/(2 \sqrt{x}) \), is unbounded as \( x \) approaches zero, so \( f'(x) \) is not continuous on \([0,1]\).
**Linear algebra**

Students must be familiar with vector and matrix notation, be able to multiply matrices, know what the inverse of a matrix is, and on homework be able to compute the inverse of a 2-by-2 or 3-by-3 matrix.

**Common Errors**

1) \(1/(x + y)\) is usually not equal to \((1/x) + (1/y)\), although generations of students have assumed that it is. A generalization of this is to assume that \((x + y)^n = x^n + y^n\) which is not usually true. Similarly, \(\ln(x + y)\) is usually not equal to \(\ln(x) + \ln(y)\), \(\sin(x + y)\) is usually not equal to \(\sin(x) + \sin(y)\), etc.

   Another common algebraic error is to assume that \(\log(5x^3) = 3 \log(5x)\) which is also not true in general.

2) \(\tan^{-1}(x)\) means arctan(x), the inverse tangent function. \(\tan^{-1}(x)\) is not equal to \(1/\tan(x)\); the latter is equal to \(\cot(x)\), the cotangent function. This terminology is standard although admittedly confusing, since \(x^{-1} = 1/x\), etc.

Suggestions of other common errors will be welcomed.

**Exercises**

1. Compute the first and second derivatives of the functions below. Sketch \(f(x)\), \(f'(x)\), and \(f''(x)\) over some representative domain of \(x\). Indicate on your sketches the values of \(f(x)\), \(f'(x)\), and \(f''(x)\) at important points.

   \[
   \begin{align*}
   (a) & \quad \sqrt{x} & (b) & \quad \sin(x) & (c) & \quad \cos(x) & (d) & \quad \tan(x) \\
   (e) & \quad \cot(x) & (f) & \quad \sin^{-1}(x) & (g) & \quad \cos^{-1}(x) & (h) & \quad \tan^{-1}(x) \\
   (i) & \quad e^x & (j) & \quad \ln(x) & (k) & \quad \exp(-x^2) & (l) & \\
   (m) & \quad 1/(1 + x^2) & (n) & \quad \sin(3x^2) & (o) & \quad \sin(x)/x & (p) & \quad \sin(x^2) \\
   (q) & \quad \exp(-1/x^2)
   \end{align*}
   \]
2. (a) Give the values of

   (i) $\ln(2)$  
   (ii) $\ln(10)$  
   (iii) $\log_{10}(2)$  
   (iv) $\log_{2}(10)$

Compute all of these quantities using the ln key on a calculator (along with any appropriate multiplications and/or divisions), not using the Log ($\log_{10}$) key at all.

(b) Repeat part (a), but using only the Log key, not the ln key.

3. Simplify (write in terms of $\ln(x)$):

   (a) $\ln(3x)$  
   (b) $\ln(x^3)$  
   (c) $\ln(5x^3)$

4. (a) Sketch $y = f(x) = x^2 + x - 2$ for $x$ on $[-3, +3]$.

   Draw the part of the curve for $x$ on $[0, 2]$ as a solid curve, and the other parts of the curve as dashed curves.

   (b) Sketch $x = f^{-1}(y)$ for the $f(x)$ in part (a), for $x$ on $[0, 2]$, that is, for the solid part of the curve in part (a).

   Note: Plot $x = f^{-1}(y)$ with $y$ as the horizontal axis and $x$ as the vertical axis, not $y = f^{-1}(x)$.

   (c) Write the $f^{-1}(y)$ in part (b) analytically as a formula.

   Hint: Solve the equation $y = x^2 + x - 2$ for $x$ in terms of $y$.

5. Work Exercise 2 above, but for $x$ on $[-2,-1]$ instead of $[0,2]$.

6. $\arctan(x)$ on a calculator computes the inverse of the piece of the $\tan(x)$ function that is defined on $-\pi/2 < x < \pi/2$. In terms of the calculator $\arctan(x)$ function, show how to compute the inverse of the piece of $\tan(x)$ that is defined on $\pi/2 < x < 3\pi/2$.

7. (a) $\arccos(x)$ on most calculators computes the inverse of the piece of the $\cos(x)$ function that is defined on $0 \leq x \leq \pi$. In terms of this calculator $\arccos(x)$ function, show how to compute the inverse of the piece of $\cos(x)$ that is defined on $-\pi \leq x \leq 0$.

   (b) Repeat part (a) for the piece of $\cos(x)$ that is defined on $\pi \leq x \leq 2\pi$.

8. For what values of $x$ and $y$, if any, is $1/(x + y) = 1/x + 1/y$ ?
9. Evaluate the sums below using the formulas given above for sums, if this is possible. If it is impossible, say why.

(a) \[ \sum_{i=1}^{3} i \]

(b) \[ \sum_{i=3}^{2} i \]

(c) \[ \sum_{i=1}^{3} i^2 \]

(d) \[ \sum_{i=1}^{3} 1 \]

10. Write a segment of computer code in a language of your choice (or pseudocode) to evaluate

\[ \sum_{i=1}^{n} \frac{1}{i} \]

for any given value of \( n \geq 0 \). Be sure to handle the possibility of an empty sum correctly!

11. Write a segment of computer code in a language of your choice (or pseudocode) to evaluate

\[ \prod_{i=1}^{n} \frac{i}{1 + i^2} \]

for any given value of \( n \geq 0 \). Be sure to handle the possibility of an empty product properly!

12. In each case where it can be expressed in terms of elementary functions, give the indefinite integral of each function in Exercise #1 above.

13. Give the inverse of the matrix \( A = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \)

14. Solve numerically for \( x \):

(a) \[ 5x - x^3 = x \]

(b) \[ x^4 + 4x^2 - 2 = 0 \]
B.2 Number Systems

The fraction 1/7 can be expressed as the infinite repeating decimal expansion 0.14285714285714... where the digits “go on forever”, repeating the pattern 142857 which has a period of six digits.

The square root of two, \( \sqrt{2} \), can also be expressed as an infinite decimal expansion 1.41421356... However there is no fraction that is exactly equal to \( \sqrt{2} \). We express this fact by saying that \( \sqrt{2} \) “is irrational”, that is, it is not the ratio of any two integers. As a result, the decimal expansion of \( \sqrt{2} \) never gets to a point where it repeats a finite pattern forever from that point onward. This is true not only in base ten but in any base, such as base=2: the expansion of \( \sqrt{2} \) never repeats forever from any point on. \( \pi=3.14159... \) and \( e=2.718... \) are also irrational, as are all non-integer fractional roots of prime numbers, and many other numbers.

Rational numbers

If m and n are integers and n is nonzero, then the fraction m/n, expressed as an expansion in any base, is a repeating expansion. That is, from some point on, there is a pattern of repeating digits with some period. The repeating digits may all be zero in some cases, in which case we say the expansion terminates. The period of the repetition is the shortest period or cycle in the expansion.

Examples:

In base ten, \( 11/4 = 2.75000... \) The expansion repeats the digit 0 forever, with period=1; therefore the expansion terminates.

\( 1/3 = 0.333... \) The expansion repeats the digit 3 forever, with period=1.

\( 22/70 = 0.3142857142857... \) The expansion repeats the pattern of the six digits 142857 forever, so the period is six.

Similarly, every expansion that repeats forever beyond some point must be a fraction. Many numbers such as \( \sqrt{2} \), \( \pi \), e, etc. never repeat, so they are not fractions. They are called “irrational” numbers, meaning “not a ratio”.

Converting a periodic expansion to a fraction

If any decimal expansion, or an expansion in any other base, does repeat a finite pattern forever from some point on, then that number is exactly equal to a fraction. We can find the fraction by a process of “multiply and subtract”:

Example: Let \( x = 41.538943294329432... \) (repeating the pattern 9432 forever)

What is the fraction that is equal to \( x \)?

Because the period of the pattern is four decimal digits, multiply \( x \) by \( 10^4 \) and subtract \( x \) from it:
\[ 10^4 x = 415389.432943294329... \]
minus \[ x = 41.538943294329... \]
The result is \((10^4 - 1) x = 415347.894000000000000000000...
\]
Multiplying both sides by \(10^3\) to get rid of the fractional part, we have
\[ 10^3(10^4 - 1) x = 415347894 \]
\[ 9999000 x = 415347894 \]
\[ x = 415347894/9999000 \]
which is the fraction that is the answer to our question.

The value of a decimal expansion such as 743.062 is given by a sum of products of the digits and the appropriate powers of the base. (Wherever the base is not indicated explicitly, it is equal to ten.) That is,
\[ 743.062 = 7(10^2) + 4(10^1) + 3(10^0) + 0(10^{-1}) + 6(10^{-2}) + 2(10^{-3}) \]
\[ = 700 + 40 + 3 + 0/10 + 6/100 + 2/1000 \]
Other integers greater than one can also be used as the bases for number systems. We will represent a number in a base other than base ten by an unbalanced right parenthesis at the right end of the number, followed by the base as a subscript. Numbers in those systems can be converted into values by using powers of the base. For example, a number in base five might be
\[ 413.201)_5 = 4(5^2) + 1(5^1) + 3(5^0) + 2(5^{-1}) + 0(5^{-2}) + 1(5^{-3}) \] (Eq. 1)
\[ = 4(25) + 1(5) + 3(1) + 2/5 + 0/25 + 1/125 \]
\[ = 100 + 5 + 3 + 0.4 + 0 + 0.008 \]
\[ = 108.408 \]

We have represented this value as a decimal expansion, but it is important to realize that we could have represented it in any base. If we do the arithmetic in Eq. 1 above in any computer or on any calculator, we will get the correct value, expressed in whatever base is being used in that machine. If we use decimal digits and decimal arithmetic as we have done above, we get a decimal representation of the value, so it is often said that we “have converted the number from base b (here b=5) to base ten”, but what we have really done is compute the value of the number from its representation in base b.
Changing the base

To convert an integer from base ten to any other base b, we can do repeated division by the new base b, keeping the integer remainders and then using them in reverse order, bottom-to-top. (It is convenient to do the arithmetic here in base ten, but it could be done in any base.) For example, convert 1729 to base 8:

\[
\begin{array}{c}
\text{216} \\
8 ) 1729 \\
\text{27} \\
8 ) 216 \\
\text{3} \\
8 ) 27 \\
0 \\
8 ) 3 \\
\end{array}
\]

remainder = 1
remainder = 0
remainder = 3
remainder = 3

Now we read off the remainders in reverse order, upwards, \(3301\)\(_8\) = 1729. We can check that this is correct by expanding in powers of the base, b=8:

\[
3301\)\(_8\) = 3(8^3) + 3(8^2) + 0(8^1) + 1(8^0)
\]

\[
= 3(512) + 3(64) + 0 + 1(1)
\]

\[
= 1536 + 192 + 1
\]

\[
= 1729 \text{ which is correct.}
\]

Fractional expansions can be converted from base ten to any other base b by repeatedly multiplying by the new base b and removing the integer parts of the products.
For example, convert $0.1$ to base $b=2$. Write the integer parts of the products to the left:

\[
\begin{array}{c}
0.1 \\
\times 2 \\
\hline
0.2 \\
0.2 \\
\times 2 \\
0.4 \\
0.4 \\
\times 2 \\
0.8 \\
0.8 \\
\times 2 \\
1.6 \\
\end{array}
\]

To proceed to the next step, remove the 1, leaving 0.6.

\[
\begin{array}{c}
0.6 \\
\times 2 \\
\hline
1.2 \\
1.2 \\
\times 2 \\
0.4 \\
0.4 \\
\end{array}
\]

and now we have repeated a previous fraction, 0.2, so we are finished. The results are read off top-to-bottom,

\[
0.1 = 0.000110011001100\ldots_2
\]

So 0.1 (one-tenth) has a non-terminating expansion in binary (base two)! This has important implications for floating point arithmetic in computers. The numbers 0.2, 0.3, 0.4, 0.6, 0.7, 0.8, and 0.9 also have non-terminating binary expansions. On the other hand, 0.5, being an integral power of two ($0.5 = 2^{-1}$), has a terminating expansion, $0.5 = 0.1_2$. 
Similarly, let us convert 0.2 to octal (base 8).

\[
\begin{array}{c|c}
0.2 & \times 8 \\
1 & .6 \\
\hline
0.6 & \times 8 \\
4 & .8 \\
\hline
0.8 & \times 8 \\
6 & .4 \\
\hline
0.4 & \times 8 \\
3 & .2 \\
\end{array}
\]

The fractional part has repeated: 0.2 → 0.6 → 0.8 → 0.4 → 0.2 so the answer is 
0.2 = 0.14631463\ldots_8.

To convert a number such as 193.2, which has both an integer part and a fractional part, to a new base, convert the integer part (193) and the fractional part (0.2) separately, then add the two converted parts together. For example, to convert 193.2 to octal (base 8), we get 193 = 301_8 and 0.2 = 0.14631463\ldots_8, so 193.2 = 301.14631463\ldots_8.

Digits in base two are called “bits”. The word “bit”, to describe a binary digit, as well as the word “software”, were invented by statistician John Tukey.
Binary expansions can be converted to octal (base 8) easily by grouping the bits in groups of three (because $8 = 2^3$):

\[
\begin{align*}
000)_2 &= 0, & 001)_2 &= 1, & 010)_2 &= 2, & 011)_2 &= 3, \\
100)_2 &= 4, & 101)_2 &= 5, & 110)_2 &= 6, & 111)_2 &= 7, & \text{so}
\end{align*}
\]

\[
\begin{align*}
0.1 &= 0.000 110 011 001 100 110 \ldots)_2 \\
 &= 0.063146\ldots_8
\end{align*}
\]

Exercises

1. Express as a fraction each of the following.
   
   (a) $0.111\ldots$ (period = 1)
   
   (b) $0.0909\ldots$ (period = 2)
   
   (c) $0.9999\ldots$ (period = 1)
   
   (d) $0.487932932932\ldots$ (period = 3)
   
   (e) $2.7182818281828\ldots$ (period = 4)

   This last number is not exactly equal to $e = 2.718281828459\ldots$, but it is close to it.

2. Write the numbers below in decimal form.

   (a) $1234.03)_5$
   
   (b) $1011011.1011)_2$

3. (a) Express $193.2$ in binary (base two). Then express the result in octal (base 8).

   (b) Express $0.27$ in octal.

4. Express in base seven: $193.2$

5. If a number with three decimal places, such as $42.137$ or $0.001$, is converted to binary, what is the maximum possible period of the repeating binary fraction that might be produced?
The Fundamental Theorem of Algebra (FTA) (one form of it):

Any polynomial \( p_n(x) \) that is not identically zero for all \( x \), with coefficients \( c_i \) that are real or complex numbers,

\[
p_n(x) = c_n x^n + c_{n-1} x^{n-1} + c_{n-2} x^{n-2} + \ldots + c_2 x^2 + c_1 x + c_0 , \quad c_n \neq 0
\]

can be written in the form

\[
p_n(x) = C (x - x_1)(x - x_2)(x - x_3)\ldots(x - x_n), \quad C \neq 0
\]

in one and only one way, apart from possible interchanges of the factors \( (x - x_i) \).

The \( x_i \) are real or complex numbers. The \( x_i \) are called the zeros of the polynomial, or the roots of the equation \( p_n(x)=0 \). We may also refer to the \( x_i \) as roots of the polynomial, to avoid having to use such terminology as “the nonzero zeros of the polynomial \( p_n(x) \)”; it makes a lot more sense to say “the nonzero roots of the polynomial \( p_n(x) \)”.

The theorem says that any polynomial that is not identically zero has exactly \( n \) real or complex zeros. Some or all of these zeros could be equal. If exactly \( m \) zeros \( x_i \) are equal, we say that \( x_i \) is a zero of multiplicity \( m \). A zero of multiplicity one is called a simple root; a root of multiplicity two is called a double root, etc.

Example: \( p_2(x) = 3x^3 - 9x + 6 \)

\[
= 3(x + 2)(x - 1)^2
\]

\( x=-2 \) is a simple root of the equation \( p_2(x)=0 \), and \( x=+1 \) is a double root.

Exercises

1. Sketch the polynomial \( p_3(x) = 2x^3 - 8x^2 + 2x + 12 \).
   Where are the zeros of this polynomial?
   (Hint: The zeros of this polynomial are all small integers.)
   Write this polynomial in the form shown in the FTA.
   What is the multiplicity of each zero?

2. Repeat Exercise 1 above for \( p_3(x) = x^3 - 3x + 2 \).
Programming guidelines

A good computer program is

1) Correct.
   Speed and all other considerations are secondary to correctness.

2) Robust.
   The program tests for invalid data and unexpected results, and reacts appropriately. For example, a good program never divides by a variable without making sure the value of the variable is not zero. Whenever it is possible to detect grossly incorrect results and print a warning message, this should be done.

3) Clear.
   The code is clear and well documented. Every subprogram is documented independently.

4) Accurate.
   A good program uses stable algorithms (to be discussed in Chapter 6 below). That is, it avoids unnecessary subtractive cancellation of leading significant digits (to be discussed in Chapters 2 and 6 below). The documentation should discuss the accuracy that can be expected and how the accuracy is related to convergence tolerances and other user input quantities.

5) General.
   A good program usually solves a general problem in a general way. It rarely contains a succession of special cases or repeated sections of code that are highly similar. Sometimes generality conflicts with clarity (#3 above), with speed (#7 below), or with reasonably low storage requirements (#8 below), but more often it does not.

6) Well modularized.
   A good program is organized into subprograms in an appropriate way. This is part of making a program clear and general. Often the main program will do nothing except read in data, call subprograms, and print results, and sometimes even the reading and printing will be done in subprograms.

7) Fast.
   A good program does not take a lot more time to solve a problem than necessary. However, it is not important to squeeze out every little bit of speed if this would compromise the robustness, generality, or accuracy of the program.

8) Compact.
   A good program does not have unnecessarily great requirements for computer storage ("memory").
Discussions for all programming assignments

Document the program thoroughly with comment lines. Document each subprogram independently. Indent neatly and consistently. Use good programming practices throughout. (There should be no possibility that the program could ever divide by zero, for example.)

Discuss the results at length. Every discussion of program results should include “What I Did”, “What Happened”, and “What I Learned”. Assume that the grader will read your discussion but will not read your source code, so your discussion should describe in English the organization and content of your source code.

Do the results appear to be reasonable? Do not turn in output that is obviously absurdly wrong!

Submit the source code, output, and discussion, on paper. Do not submit any material on diskettes or in the form of e-mailed computer files.

A perfect program with no discussion will receive no credit. (Program #1 may be an exception.)

Compiling a C program

To compile a program using the gcc compiler for the C language and cause it include one or more library functions, use the Unix command

```
gcc -lm prog1.cc
```

where prog1.cc is the name of the file containing the C language source code. -lm causes the source code for the math library functions to be loaded. #include by itself does not accomplish this.
Programming Assignment #1(a): Printing large and small numbers

Show how to print out floating point numbers in a high-level programming language of your choice so that, even if they are very small in magnitude, or very large, a given number of significant digits will always be printed. If the number is not very small or very large in magnitude, it should preferably be printed in fixed format rather than in scientific notation with an exponent.

In Java, this is possible just using System.out.println(). Do not use the DecimalFormat or NumberFormat classes with a fixed number of decimal places.

In Fortran, use an E15.7 or G15.7 format specification, or to get more digits, E24.16 or G24.16. Even better is to use 1P,G15.7 to get one nonzero digit before the decimal point.

In Standard C or C++, you can use `printf("Value of x = %24.16#g\n", x);` (This is really just the G24.16 format carried over from Fortran. For fewer digits on variables of type float, use 15.7 instead of 24.16.) The # prevents the stripping of trailing zeroes after the decimal point, and the stripping of the decimal point itself if there are no digits after the decimal point. For numerical work, the # is preferable, so please use it. In some non-Standard compilers you might have to omit the #; do not use non-Standard compilers in this course. The above format works fine with the gcc compiler.

Java language now allows the formatting of numbers as done above in C or C++.

In the programming language you plan to use in this course, set a variable of type double equal to a number of moderate size such as 1.23456781414214693057 and print it out. Repeat with a very large value such as 9.87654321936759276472e34, using the same format or method of printing as for the number of moderate size. Repeat with a very small value such as 5.67891234825491038563e-34, again using the same format or method of printing. Make sure that at least fourteen significant digits are visible in each part of your output. Try to use a method that automatically prints the number of moderate size (the first value above) in fixed format (without any E or e), but this is not as important as getting at least fourteen significant digits printed for each value.

How many correct significant digits were printed, for each of the three values above? What does this tell you about the precision of type double on this computer? Compute the approximate number of bits in the mantissa of a value of type double on this computer.

Hand in paper hardcopy of your source code, test data if any, and output.

Follow good programming practices throughout. See Prependix B.
Programming Assignment 1(b): Printing a vector of arbitrary length

Write a general subprogram (a method, function, procedure, or subroutine) that will accept as input the following four parameters:

- a vector (a one-dimensional array) of type “double” named vec[], of any size,
- an integer base index named vBase (vBase can be any integer value),
- an integer n that is the number of elements to be printed (n \(\geq\) 0), and
- an integer nPerLine, the maximum number of elements to be printed on each line (nPerLine \(\geq\) 1).

This general subprogram should then print the vector elements vec[vBase], vec[vBase+1], vec[vBase+2], ..., vec[vBase+n-1], with nPerLine elements on each line on the paper, separating the values by at least one blank and automatically wrapping the values around to as many lines as necessary. For example, if

\[
\begin{align*}
\text{vec}[0] &= -0.719165446 & \quad \text{vBase} &= 0 \\
\text{vec}[1] &= -0.738798065 & \quad \text{n} &= 4 \\
\text{vec}[2] &= 0.739019885 & \quad \text{nPerLine} &= 3 \\
\text{vec}[3] &= 0.739110549
\end{align*}
\]

then the subprogram should print

\[-0.719165446 \quad -0.738798065 \quad 0.739019885 \quad 0.739110549\]

The subprogram has printed n=4 values with nPerLine=3 values on a line, separating the values by at least one blank character and automatically wrapping the fourth value around to the second line. The subprogram should check n and nPerLine to make sure they satisfy the constraints given above on their possible legal values. In case of illegal values, the subprogram should print an error message and then return. The subprogram should return a boolean value that is true if there was no error and false if there was an error. (In C language, return 1 or 0.)

It is essential and absolutely required that your subprogram must be able to handle any positive value of n, however large, such as n = 1000 or even larger! It must also handle any reasonable value of nPerLine, such as nPerLine = 1, 2, 3, 4, etc. up to whatever might fit on the paper. Write code that is general. Do not write a section to handle one particular case, such as nPerLine = 1, followed by another section to handle nPerLine = 2, etc. This is no way to program.

Test your subprogram by calling it at least three times in one run with various values of n and nPerLine, both valid and invalid. Include one case with n = 120 and nPerLine = 3.

A subprogram is required. Handing in only a main program would not earn any credit. Also, as shown above, the subprogram must be general.

Hand in paper hardcopy of your source code, test data if any, and output.

Follow good programming practices throughout. See Prepending B.