Polynomials

As with Matlab, polynomials are written in the order of increasing powers. Instead of \( x^2 - 2x + 3 \), write \( 3 - 2x + x^2 \). Represent \( (3)x^0 + (-2)x^1 + (1)x^2 \) with the vector \( p = [3, -2, 1] \) of its coefficients. Declare it as \( \text{real} : : p(0 : 2) \), since the degrees range from 0 to 2. Thus \( p(0) = 3 \), \( p(1) = -2 \), \( p(2) = 1 \). Write 0 for any missing coefficient: write \( x^2 - 1 \) as \( -1 + 0x + 1x^2 \) with vector \( p = [-1, 0, 1] \) of degree \( n = 3 \). Evaluating a polynomial \( p \) at a real \( r \) means the result of substituting \( r \) for \( x \). \( x^2 - 2x + 3 \) evaluated at \( x = 10 \) is \( 10^2 - 2(10) + 3 = 100 - 20 + 3 = 83 \)
Classwork 16.1(2) polyeval.f95 Write a real function polyeval(p,n,r) which evaluates a polynomial p of degree n at x = r. Print with 2-place decimals.

c16_1_2polyeval.f95

!p = a polynomial of degree n, polyeval = p(r)
real function polyeval(p,n,r) result(y)
real::p(0:n),r
y=0
do i=___,___ ;y=y+p(___)*r**___;enddo
endfunction

program test_decimal2base
real::p(0:2),polyeval,r,y
p=(/-1,0,1/) !x^2-1
do i=1,4
  y=polyeval(p,2,real(i))
  print*,'x=',real(i),'x^2-1=',y
enddo; print*
endprogram !change print* so decimals have two places
Write a subroutine `polyproduct(p,n,q,m,r)` which gives the coefficient vector for the polynomial `r` which is the product of the two polynomials `p` of degree `n` and `q` of degree `m`.

```fortran
subroutine polyproduct(p,n,q,m,r)
    real::p(0:n),q(0:m),r(0:n+m)
    r(0:n+m)=0
    do i=0,n; do j=0,m
        r(i+j)=r(i+j)+ ______
    enddo; enddo
endsubroutine
```

```fortran
program test_polyproduct
    real::p(0:1),q(0:2),r(0:4)
p=[1,-1] ! p=(-1+x). q=(-1+x)^2, r=(-1+x)^4
print*,"(-1+x), (-1+x)^2, (-1+x)^4"
```
CLASSWORK 16.2(4) estate.f95  An estate has $M$ dollars which is to divided among $n$ heirs $1, 2, 3, \ldots, n$. Let \( a = [a(1), a(2), \ldots, a(n)] \) where \( a(i) \) is the amount (in integer dollars, no change) the \( i \)th heir gets. Write a subroutine \textit{estate}(M,n,a) which calculates \( a \) by dividing the \( M \) dollars among the \( n \) heirs (at most 12 heirs) so that all get the same amount plus/minus one dollar, \( a(i) \) equals \( a(j) \) plus or minus 1. \([3, 4, 4], [4, 3, 4]\) are acceptable divisions of $11 among 3 heirs; but not \([3, 3, 5], [5, 4, 2]\). Find the quotient \( q \) and remainder \( r \) of $M/n$. Give each heir \( q \) dollars. Then give \( r \) of the \( n \) heirs one more dollar.
subroutine estate (M, n, a)
!M = inheritance dollars, n= # heirs, a(i)= ith heir's inheritance.
integer:: M, n, a(n), r, q

... finish the subroutine
endsubroutine

program test_estate
integer:: M, n, a(12)
do M=11,13; do n=3,4
call estate (M, n, a)
print'(a,i2,i2,a,20i2)','$',M,n,' heirs, each gets ',a(1:n)
endo; enddo
endprogram
Write a subroutine \texttt{estate2}(M,n,a) as in the classwork problem but without quotients or remainders (\texttt{no mod}) and using only one do-loop \texttt{do } j=1,M \texttt{ where } j \texttt{ ranges over the } M \texttt{ dollars in the estate. Initially set the vector } a \texttt{ to all 0's. Give each heir in turn one dollar until the money runs out. Let } i \texttt{ be the heir whose turn it is to get a dollar. Start with heir } i=1. \texttt{ After heir } i \texttt{ has been given a dollar, the next heir to get a dollar is heir } i+1, \texttt{ unless } i=n \texttt{ in which case the next heir is } 1. \texttt{ Include the } \texttt{test_estate2} \texttt{ program of Classwork 15.3(2). Writing a program that works isn't enough, your code must use the steps just described: give each heir one dollar in turn until the money runs out.}

\begin{verbatim}
!estate2.f95
subroutine estate2(M,n,a)
!M = inheritance dollars, n= # heirs, a(i)= ith heir's inheritance.
integer::M,n,a(n)
i=1; a=0
   ... finish the subroutine, no mod, no quotients
endsubroutine
\end{verbatim}
program test_estate2
integer:: M,n,a(12)
do M=11,13; do n=3,4
    call estate2(M,n,a)
print'(a,i2,i2,a,20i2)','$',M,n,' heirs, each gets ',a(1:n)
enddo; enddo
endprogram

DO LOOP CONTROLS

return exits a subroutine.
exit leaves a do-loop.
cycle jumps to the next iteration of the loop (skipping the remaining do-loop lines).
if(...) cycle; abbreviates
if(...) then; cycle; endif;
if(...) return; abbreviates
if(...) then; return; endif;
if(...) exit; abbreviates
if(...) then; exit; endif;
The hailstorm sequence. Start with a positive integer $n$:
If $n=1$, stop.
If $n$ is even, replace it with $n/2$ and cycle.
If $n$ is odd, replace it with $3n + 1$ and cycle.

Open problem (worth $2000):$ Do you always eventually get to 1 and stop? Or is there an $n$ on which the sequence cycles forever? Fill the five blanks.
program test_hailstorm

integer::n=97

!Print*, 'Enter a positive integer.'
!read*, n
do
    print *, n
    if(n==1) then  !put your cursor in a blank, double click.
        _________
        ! return? exit? cycle?
    endif
    if(mod(n, 2)==0) then
        n= _________
        _________
        ! return? exit? cycle?
    endif
    if(mod(n, 2)/=0) then
        n= _________
        _________
        ! return? exit? cycle?
    endif
enddo;  endprogram  ! □ Answer: 97, 292, 146, 73, ...
**Magic squares**

A *magic square* is an \( n \times n \) array of containing the integers 1, 2, 3, ..., \( n^2 \). Each of these integers occurs exactly once in the matrix and all rows, columns and the two diagonals all have the same sum. In the \( 3 \times 3 \) magic square below, all rows, columns and diagonals total to 15, 2+7+6=15, 2+9+4=14, 2+5+8=15.

<table>
<thead>
<tr>
<th>2</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Here’s an algorithm for constructing a magic square for odd \( n \). We will place the integers one at a time in increasing order \( k = 1, 2, 3, ..., n^2 \). Put 1 in the center of the top row. After filling a square with an integer \( k \), the next integer, \( k+1 \) will go in the square which is diagonally up and to the right of \( k \)’s square. If there is no higher row, wrap to the bottom row. If there is no column on the right, wrap to the first column. If the next square is already filled, then move down from \( k \)’s square. Here's the \( 3 \times 3 \) square.
We represent a magic square with an $n \times n$ matrix of integers where 0 indicates an empty square.

The center of the top row has position $(1, (1 + n)/2)$. 
Write a program `magic_square` which fills a 3×3 magic square using the algorithm above with a subroutine `next_square` which calculates the next position.
program magic_square
    integer :: board(3,3) = _____
    integer :: n=3, k=1, i, j, next_i, next_j
    ! start at top middle k=1,2,3,... the cell numbers
    ! (i,j), (next_i,next_j) are the board positions.
    i= ___ ; j= ___ ; board(i,j)= ___
    call print_board
    do k=2,n**2
        call next_square(i,j,next_i,next_j)
        ... delete this line, finish the do-loop
        call print_board
    enddo
contains
    subroutine next_square(i,j,next_i,next_j)
        if(i>1)then; next_i=i-1; else; next_i=n; endif
        ... replace this line with the condition for columns
    endsubroutine
    subroutine print_board
        integer :: i
        print*, (board(i,:), char(10), i=1,3)
    endsubroutine
endprogram
FORMATING

To make columns line up in a word processor, you use tabs. In Fortran you use format statements. Instead of

\texttt{print *, ..., where * means default format, write}

\texttt{print '( *** )',..., where *** are format instructions.}

The following are equivalent:

\texttt{print *, (a(i,j), j=1,3)}
\texttt{print *, a(i,1), a(i,2), a(i,3)}
\texttt{print *, a(i,1:3)}
is for character strings of unspecified length.
a12 reserves 12 spaces for characters (letters, numerals).
i5 reserves 5 spaces for an integer including any “-”.
i0 uses whatever space is needed for the integer.
4i5 reserves 4 integer fields of 5 spaces each.
f7.2 reserves 7 spaces with two decimal places.
4f7.2 reserves consecutive fields of type f7.2.
f7.2 is wrong, you can’t have variables in formats
4x prints 4 spaces as does ‘   ’.
/ prints a carriage return, i.e., begins a new line (not \n).
Quoted character strings print as is. ‘   ’ equals 1x.

Items are right-justified in their fields (blanks added on the left). In the f7.2 real format, 1. is printed “  1.00”.
print *,x,y uses Fortran’s default format,
print ' (20f7.2)',x,y
prints each number in a field of 7 spaces (including “.” and “−”) and 2 decimal places. The 20 specifies that the format may be used up to 20 times. The variables x, y use only 2 of the 20 fields. The remaining 18 unused fields are ignored.
Here are three ways to print \( x \) in the \( f7.2 \) real format. All except the last advance to the next line after printing.

\[
\begin{align*}
\text{print} & \quad '(f7.2)', x \\
\text{11 format}(f7.2) & \quad !use with FTN95 \\
\text{print 11}, x \\
\text{write(*,11), x} \\
\text{write(*,11,advance='no'), x}
\end{align*}
\]

The last doesn’t advance to the next line after printing.
Change things in pink to print a nice title for \( n \) homework scores. Change \texttt{print 11} to make the title print on one line. Change \texttt{a1,i1} to get (items should be one space apart). Result should be like this:

\begin{verbatim}
Hw1 Hw2 Hw3 Hw4 Hw5 Hw6 Hw7 Hw8 Grade
Hw1 Hw2 Hw3 Hw4 Hw5 Hw6 Hw7 Hw8 Hw9 Grade
\end{verbatim}

\begin{verbatim}
!c16_4_2print_title.f95
subroutine print_title(n)
integer::n;
11 format(a1,i1)  !change this
do i=1,n
    print 11,'Hw',i
    !replace \texttt{print 11,} with \texttt{write(*,11,advance='no')}
enddo
print '(a6)','Grade'  !change if using \texttt{FTN95}
end subroutine

program test_print_title
do i=1,9
    call print_title(i)
enddo
end program
\end{verbatim}
Write a program `print_title2` which asks the user for the number \( n \) of exams and then prints, on one line, the heading

Exam 1, Exam 2, ..., Exam \( n \), Final

If the user enters 4, the printed heading should be

Exam 1, Exam 2, Exam 3, Exam 4, Final

The heading must include the commas. Use a write statement with `advance='no'`. See previous program `print_title`. With FTN95, format statements are written the old way, e.g.

```
10 format(a9)  
print 10,'Grade'
```
Characters, strings, words

Printing or writing \texttt{abcd} to a 10-space field gives "      abcd" -- it is \textit{right-justified} when trimmed.

\begin{verbatim}
  character(10)::s10='abcd'
  character(2)::s2='abcd'
\end{verbatim}

Assign the 4-letter string \texttt{abcd} to 10-spaces and to 2-spaces.

Assigning or reading \texttt{abcd} to a 10-space field gives "abcd      " -- it is \textit{left-justified}.

Assigning or reading \texttt{abcd} to a 2-space field gives "ab" -- it is \textit{truncated}, the tail is lost.
EXAMPLE  conversions.f95  New window, save, run.
!conversions.f95

program character_conversions
character(10)::s10
character(2) ::s2

s10 = 'abcd' ! 6 spaces added on the right.
s2 = 'abcd' ! 'abcd' truncated tp 'ab'

print *, '/' , s10 , '" / s10='abcd';print*,s10"
print *, '/' , trim(s10) , '/' print*,trim(s10)'
print ' (a,a10,a,a)', ' / ', trim(s10), &
    '" / print ' (a10)',trim(s10)"
print *, '/' , s2 , '" / s2='abcd';print*,s2" 
print*  !prints a blank line
endprogram
What went wrong? **Correct the 4 pinks. Add two spaces.**

*Want*    Hello John Doe!

*not*   HelloJohn     Doe        !

*not*   HelloJohnDoe!

**Common error**  If you don’t see say the last !, it might be off screen due to untrimmed blank spaces.

Recall that read *,s reads one word, read '(a)',s reads one line.
program name2
character(20) :: fullname

print *, 'Enter first name, space, last name'
read *, fullname

print *, 'Aloha ', trim(fullname), '!
endprogram

Correct the pink line to read in all of the fullname.

Want   Aloha John Doe!
not    Aloha John