Introductory Fortran Programming  Part 6

Gunnar Wollan$^1$

Dept. of Geosciences, University of Oslo$^1$

January 2007
Outline

1. From small examples to real programs
List of Topics

1. From small examples to real programs
Solving a real problem

The case of missing data

Very often we have an inconsistent time series where there are missing data.

To be able so use the time series we have to "fill in the blanks" so to speak.

There are several methods for doing this, but here we will look at the simple linear interpolation.

The equation for the linear interpolation is this:

$$y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0} \tag{1}$$
Solving a real problem

- The case of missing data
- Very often we have an inconsistent time series where there are missing data
- To be able so use the time series we have to "fill in the blanks" so to speak
- There are several methods for doing this, but here we will look at the simple linear interpolation
- The equation for the linear interpolation is this:

$$
y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0}
$$

(1)
Solving a real problem

- The case of missing data
- Very often we have an inconsistent time series where there are missing data
- To be able to use the time series we have to "fill in the blanks" so to speak
- There are several methods for doing this, but here we will look at the simple linear interpolation
- The equation for the linear interpolation is this:

\[ y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0} \] (1)
The case of missing data

Very often we have an inconsistent time series where there are missing data

To be able so use the time series we have to "fill in the blanks" so to speak

There are several methods for doing this, but here we will look at the simple linear interpolation

The equation for the linear interpolation is this:

\[ y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0} \]  

(1)
Solving a real problem

- The case of missing data
- Very often we have an inconsistent time series where there are missing data
- To be able so use the time series we have to "fill in the blanks" so to speak
- There are several methods for doing this, but here we will look at the simple linear interpolation
- The equation for the linear interpolation is this:

\[ y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0} \] (1)
How do we start??

First of all we have to read the time series containing missing data

Let us assume that the file contains an integer with the length of the time series as the first entry

A way to begin our program is to declare the necessary variables

USE linear_interpolation
REAL(KIND=4), POINTER :: array(:)
REAL, PARAMETER :: mval = -9999.99
INTEGER, PARAMETER :: lun = 10
INTEGER :: i, j, k
CHARACTER(LEN=80) :: ifname, ofname
INTEGER :: rstat
INTEGER :: argc
INTEGER, EXTERNAL :: iargc
Real Programs, Part 2

Solving the linear interpolation

How do we start??

First of all we have to read the time series containing missing data

Let us assume that the file contains an integer with the length of the time series as the first entry

A way to begin our program is to declare the necessary variables

```fortran
USE linear_interpolation
REAL(KIND=4), POINTER :: array(:)
REAL, PARAMETER :: mval = -9999.99
INTEGER, PARAMETER :: lun = 10
INTEGER :: i, j, k
CHARACTER(LEN=80) :: ifname, ofname
INTEGER :: rstat
INTEGER :: argc
INTEGER, EXTERNAL :: iargc
```
How do we start?

First of all we have to read the time series containing missing data.

Let us assume that the file contains an integer with the length of the time series as the first entry.

A way to begin our program is to declare the necessary variables:

```fortran
USE linear_interpolation
REAL(KIND=4), POINTER :: array(:)
REAL, PARAMETER :: mval = -9999.99
INTEGER, PARAMETER :: lun = 10
INTEGER :: i, j, k
CHARACTER(LEN=80) :: ifname, ofname
INTEGER :: rstat
INTEGER :: argc
```

```fortran
```

How do we start??

First of all we have to read the time series containing missing data

Let us assume that the file contains an integer with the length of the time series as the first entry

A way to begin our program is to declare the necessary variables

```fortran
USE linear_interpolation
REAL(KIND=4), POINTER :: array(:) :: array(:)
REAL, PARAMETER :: mval = -9999.99 :: mval = -9999.99
INTEGER, PARAMETER :: lun = 10 :: lun = 10
INTEGER :: i, j, k :: i, j, k
INTEGER :: argc :: argc
CHARACTER(LEN=80) :: ifname, ofname :: ifname, ofname
INTEGER :: rstat :: rstat
INTEGER :: argc :: argc
INTEGER, EXTERNAL :: iargc :: iargc
```

Wollan
Introductory Fortran Programming Part 6
Some explanation

- The module `linear_interpolation` contains all the necessary methods for the interpolation.
- The array to contain the time series is declared as a single precision array using the `KIND=4` to specify the precision.
- We use a constant to define the missing value (the number -9999.99 is often used for this).
- We also use a constant for the file unit number and two character strings to contain the input and output filenames.
- To be able to retrieve the number of input arguments we declare an external integer function `iargc` and an integer to hold the number of input arguments (`argc`).
Solving the linear interpolation

- Some explanation
- The module `linear_interpolation` contains all the necessary methods for the interpolation
  - The array to contain the time series is declared as a single precision array using the `KIND=4` to specify the precision
  - We use a constant to define the missing value (the number -9999.99 is often used for this)
  - We also use a constant for the file unit number and two character strings to contain the input and output filenames
  - To be able to retrieve the number of input arguments we declare an external integer function `iargc` and an integer to hold the number of input arguments (`argc`)
Solving the linear interpolation

- Some explanation
- The module *linear_interpolation* contains all the necessary methods for the interpolation
- The array to contain the time series is declared as a single precision array using the *KIND=4* to specify the precision
- We use a constant to define the missing value (the number -9999.99 is often used for this)
- We also use a constant for the file unit number and two character strings to contain the input and output filenames
- To be able to retrieve the number of input arguments we declare an external integer function *iargc* and an integer to hold the number of input arguments (*argc*)
Solving the linear interpolation

- Some explanation
- The module *linear_interpolation* contains all the necessary methods for the interpolation
- The array to contain the time series is declared as a single precision array using the *KIND=4* to specify the precision
- We use a constant to define the missing value (the number -9999.99 is often used for this)
- We also use a constant for the file unit number and two character strings to contain the input and output filenames
- To be able to retrieve the number of input arguments we declare an external integer function *iargc* and an integer to hold the number of input arguments (*argc*)
Solving the linear interpolation

- Some explanation
- The module `linear_interpolation` contains all the necessary methods for the interpolation
- The array to contain the time series is declared as a single precision array using the `KIND=4` to specify the precision
- We use a constant to define the missing value (the number `-9999.99` is often used for this)
- We also use a constant for the file unit number and two character strings to contain the input and output filenames
- To be able to retrieve the number of input arguments we declare an external integer function `iargc` and an integer to hold the number of input arguments (`argc`)
Solving the linear interpolation

- Some explanation
- The module *linear_interpolation* contains all the necessary methods for the interpolation
- The array to contain the time series is declared as a single precision array using the $KIND=4$ to specify the precision
- We use a constant to define the missing value (the number -9999.99 is often used for this)
- We also use a constant for the file unit number and two character strings to contain the input and output filenames
- To be able to retrieve the number of input arguments we declare an external integer function *iargc* and an integer to hold the number of input arguments (argc)
Retrieving the number of input arguments and get the input filename

```fortran
argc = iargc()
IF(argc == 1) THEN
  CALL getarg(1, ifname)
ELSE
  ifname = "indata.txt"
END IF
```
Solving the linear interpolation

- Some explanation

- We test to see if we have one command line argument stored in the argc variable.

- If we have, we retrieve the input filename calling the `getarg` subroutine with a one to specify we want to retrieve the first command line argument and then the ifname as the second argument.

- Note that all command line arguments are text strings even if we type in a number.
Solving the linear interpolation

- Some explanation
- We test to see if we have one command line argument stored in the argc variable
  - If we have, we retrieve the input filename calling the getarg subroutine with a one to specify we want to retrieve the first command line argument and then the ifname as the second argument
  - Note that all command line arguments are text strings even if we type in a number
Solving the linear interpolation

- Some explanation
- We test to see if we have one command line argument stored in the argc variable
- If we have, we retrieve the input filename calling the getarg subroutine with a one to specify we want to retrieve the first command line argument and then the ifname as the second argument
- Note that all command line arguments are text strings even if we type in a number
Solving the linear interpolation

- Some explanation
- We test to see if we have one command line argument stored in the argc variable
- If we have, we retrieve the input filename calling the getarg subroutine with a one to specify we want to retrieve the first command line argument and then the ifname as the second argument
- Note that all command line arguments are text strings even if we type in a number
The next step is to open the input file, read the length of the time series, allocate space for the data and read the values from the file.

```
OPEN(UNIT=lun,FILE=ifname,FORM="FORMATTED",IOSTAT=rstat)
READ(UNIT=lun,FMT='(I6)',IOSTAT=rstat) array_length
ALLOCATE(array(array_length),STAT=rstat)
DO i = 1, array_length
   READ(UNIT=lun,FMT='(F10.4)',IOSTAT=rstat) array(i)
END DO
CLOSE(UNIT=lun)
```
Solving the linear interpolation

The next step is to open the input file, read the length of the time series, allocate space for the data and read the values from the file.

```fortran
OPEN(UNIT=lun,FILE=ifname,FORM="FORMATTED",IOSTAT=rstat)
READ(UNIT=lun,FMT='(I6)',IOSTAT=rstat) array_length
ALLOCATE(array(array_length),STAT=rstat)
DO i = 1, array_length
   READ(UNIT=lun,FMT='(F10.4)',IOSTAT=rstat) array(i)
END DO
CLOSE(UNIT=lun)
```
Real Programs, Part 2

Solving the linear interpolation

Some explanation

- Using the \texttt{FORM="FORMATTED"} tells the system that the input file is an ascii file.
- Reading the first entry we specify that it is an integer in the \texttt{FMT='(I6)'} argument.
- Then we allocate space for the array. Note that we should always test if our calls to the operating-system were successful.
- The integer \texttt{rstat} is used here to receive the return value. If the return value is other than zero an error has occurred.
- A similar format statement is used in reading the values from the file, but here we use a floating point number with a total of 10 positions including the decimal separator and with 4 decimals.
- Always remember to close the file when we are finished using it.
Solving the linear interpolation

- Some explanation
- Using the \texttt{FORM="FORMATTED"} tells the system that the input file is an ascii file
  - Reading the first entry we specify that it is an integer in the \texttt{FMT='(I6)'} argument
  - Then we allocate space for the array. Note that we should always test if our calls to the operating-system were successful
  - The integer \texttt{rstat} is used here to receive the return value. If the return value is other than zero an error has occurred
  - A similar format statement is used in reading the values from the file, but here we use a floating point number with a total of 10 positions including the decimal separator and with 4 decimals
  - Always remember to close the file when we are finished using it
Solving the linear interpolation

- Some explanation
- Using the `FORM="FORMATTED"` tells the system that the input file is an ascii file
- Reading the first entry we specify that it is an integer in the `FMT='(I6)'` argument
- Then we allocate space for the array. Note that we should always test if our calls to the operating-system were successful
- The integer rstat is used here to receive the return value. If the return value is other than zero an error has occurred
- A similar format statement is used in reading the values from the file, but here we use a floating point number with a total of 10 positions including the decimal separator and with 4 decimals
- Always remember to close the file when we are finished using it
Solving the linear interpolation

- Some explanation
- Using the `FORM="FORMATTED"` tells the system that the input file is an ascii file
- Reading the first entry we specify that it is an integer in the `FMT=’(I6)’` argument
- Then we allocate space for the array. Note that we should always test if our calls to the operating-system were successful
- The integer rstat is used here to receive the return value. If the return value is other than zero an error has occurred
- A similar format statement is used in reading the values from the file, but here we use a floating point number with a total of 10 positions including the decimal separator and with 4 decimals
- Always remember to close the file when we are finished using it
Solving the linear interpolation

- Some explanation
- Using the `FORM="FORMATTED"` tells the system that the input file is an ascii file
- Reading the first entry we specify that it is an integer in the `FMT='(I6)'` argument
- Then we allocate space for the array. Note that we should always test if our calls to the operating-system were successful
- The integer rstat is used here to recieve the return value. If the return value is other than zero an error has occurred
- A similiar format statement is used in reading the values from the file, but here we use a floating point number with a total of 10 positions including the decimal separator and with 4 decimals
- Always remember to close the file when we are finished using it
Solving the linear interpolation

- Some explanation
- Using the `FORM="FORMATTED"` tells the system that the input file is an ascii file.
- Reading the first entry we specify that it is an integer in the `FMT=’(I6)’` argument.
- Then we allocate space for the array. Note that we should always test if our calls to the operating-system were successful.
- The integer rstat is used here to receive the return value. If the return value is other than zero an error has occurred.
- A similar format statement is used in reading the values from the file, but here we use a floating point number with a total of 10 positions including the decimal separator and with 4 decimals.
- Always remember to close the file when we are finished using it.
Solving the linear interpolation

- Some explanation
- Using the `FORM="FORMATTED"` tells the system that the input file is an ascii file
- Reading the first entry we specify that it is an integer in the `FMT='(I6)'` argument
- Then we allocate space for the array. Note that we should always test if our calls to the operating-system were successful
- The integer rstat is used here to receive the return value. If the return value is other than zero an error has occurred
- A similar format statement is used in reading the values from the file, but here we use a floating point number with a total of 10 positions including the decimal separator and with 4 decimals
- Always remember to close the file when we are finished using it
Now we shall check the data for missing values

We loop through the array and test each element for missing value

\[
i = 1 \\
\text{DO} \\
\quad \text{IF}(\text{array}(i) == \text{mval}) \text{ THEN} \\
\quad \quad \text{CALL find_index(array,i,j,mval)} \\
\quad \quad \text{CALL interpolate(array,i-1,j)} \\
\quad \quad i = j \\
\quad \text{ELSE} \\
\quad \quad i = i + 1 \\
\quad \text{END IF} \\
\quad \text{IF}(i == \text{array_length}) \text{ THEN} \\
\quad \quad \text{EXIT} \\
\quad \text{END IF} \\
\text{END DO}
\]
Solving the linear interpolation

- Now we shall check the data for missing values
- We loop through the array and test each element for missing value

```fortran
i = 1
DO
  IF(array(i) == mval) THEN
    CALL find_index(array,i,j,mval)
    CALL interpolate(array,i-1,j)
    i = j
  ELSE
    i = i + 1
  END IF
  IF(i == array_length) THEN
    EXIT
  END IF
END DO
```
Solving the linear interpolation

**Some explanation**

- We use a potential endless loop by using the standalone `DO` so we use a manually updated index.
- If this array element contains a missing value, we call the `find_index` subroutine to find the index of the next nonmissing value element.
- Then we call the `interpolate` subroutine with the array and the index of the elements on both sides of the sequence of elements with the missing value.
- We then set the index to the element after the last missing value in the sequence.
- Then we test to see if we have reached the end of the array and exit the loop if we have.
Solving the linear interpolation

- Some explanation
- We use a potential endless loop by using the standalone *DO* so we use a manually updated index
  - If this array element contains a missing value we call the *find_index* subroutine to find the index of the next nonmissing value element
  - Then we call the *interpolate* subroutine with the array and the index of the elements on both sides of the sequence of elements with the missing value
  - We then set the index to the element after the last missing value in the sequence
  - Then we test to see if we have reached the end of the array and exit the loop if we have

Introductory Fortran Programming Part 6
Solving the linear interpolation

- Some explanation
- We use a potential endless loop by using the standalone DO so we use a manually updated index
- If this array element contains a missing value we call the \texttt{find_index} subroutine to find the index of the next nonmissing value element
- Then we call the \texttt{interpolate} subroutine with the array and the index of the elements on both sides of the sequence of elements with the missing value
- We then set the index to the element after the last missing value in the sequence
- Then we test to see if we have reached the end of the array and exit the loop if we have
Solving the linear interpolation

- Some explanation
- We use a potential endless loop by using the standalone DO so we use a manually updated index
- If this array element contains a missing value we call the find_index subroutine to find the index of the next nonmissing value element
- Then we call the interpolate subroutine with the array and the index of the elements on both sides of the sequence of elements with the missing value
- We then set the index to the element after the last missing value in the sequence
- Then we test to see if we have reached the end of the array and exit the loop if we have
Some explanation

We use a potential endless loop by using the standalone *DO* so we use a manually updated index

If this array element contains a missing value we call the *find_index* subroutine to find the index of the next nonmissing value element

Then we call the *interpolate* subroutine with the array and the index of the elements on both sides of the sequence of elements with the missing value

We then set the index to the element after the last missing value in the sequence

Then we test to see if we have reached the end of the array and exit the loop if we have
Solving the linear interpolation

- Some explanation
- We use a potential endless loop by using the standalone *DO* so we use a manually updated index
- If this array element contains a missing value we call the *find_index* subroutine to find the index of the next nonmissing value element
- Then we call the *interpolate* subroutine with the array and the index of the elements on both sides of the sequence of elements with the missing value
- We then set the index to the element after the last missing value in the sequence
- Then we test to see if we have reached the end of the array and exit the loop if we have
The last thing we have to do is to write the array to a new output file

This is how we can do it

```fortran
ofname = TRIM(ifname)//".data"
OPEN(UNIT=lun,FILE=ofname,FORM='FORMATTED',IOSTAT=rstat)
WRITE(UNIT=lun,FMT='(I6)',IOSTAT=rstat) array_length
DO i = 1, array_length
    WRITE(UNIT=lun,FMT='(F10.4)',IOSTAT=rstat) array(i)
END DO
CLOSE(UNIT=lun)
```
The last thing we have to do is to write the array to a new output file.

This is how we can do it:

```fortran
ofname = TRIM(ifname) // "data"
OPEN(UNIT=lun, FILE=ofname, FORM='FORMATTED', IOSTAT=rstat)
WRITE(UNIT=lun, FMT='(I6)', IOSTAT=rstat) array_length
DO i = 1, array_length
   WRITE(UNIT=lun, FMT='(F10.4)', IOSTAT=rstat) array(i)
END DO
CLOSE(UNIT=lun)
```
Some explanation

- We create the output filename from the input filename by concatenating the input filename with an extra extension ".data"

- Note the concatenation is performed using a double slash. The TRIM function returns the character string without trailing spaces

- The code for writing is just the same as the one for reading, but we use of course WRITE instead of READ
Some explanation

We create the output filename from the input filename by concatenating the input filename with an extra extension ".data"

Note the concatenation is performed using a double slash. The \texttt{TRIM} function returns the character string without trailing spaces.

The code for writing is just the same as the one for reading, but we use of course \texttt{WRITE} instead of \texttt{READ}. 
Solving the linear interpolation

- Some explanation
- We create the output filename from the input filename by concatenating the input filename with an extra extension ".data"
- Note the concatenation is performed using a double slash. The \textit{TRIM} function returns the character string without trailing spaces
- The code for writing is just the same as the one for reading, but we use of course \texttt{WRITE} instead of \texttt{READ}
Some explanation

We create the output filename from the input filename by concatenating the input filename with an extra extension ".data"

Note the concatenation is performed using a double slash. The TRIM function returns the character string without trailing spaces.

The code for writing is just the same as the one for reading, but we use of course WRITE instead of READ.
Now we have finished the main program, but we still have to write the two subroutines \textit{find\_index} and \textit{interpolate}.

So how do we go about doing this??

We have to take into consideration that we can have data either as single or double precision.

To solve this we will make a module containing all the necessary routines.

Since we do not want to have a lot of different subroutine names to call we will use an interface which allows us to have one subroutine name for both the single and double precision data.

Also it is convenient to have a global variable containing the array length.
Solving the linear interpolation

Now we have finished the main program, but we still have to write the two subroutines find_index and interpolate.

So how do we go about doing this??

- We have to take into consideration that we can have data either as single or double precision.
- To solve this we will make a module containing all the necessary routines.
- Since we do not want to have a lot of different subroutine names to call we will use an interface which allows us to have one subroutine name for both the single and double precision data.
- Also it is convenient to have a global variable containing the array length.
Now we have finished the main program, but we still have to write the two subroutines `find_index` and `interpolate`.

So how do we go about doing this??

We have to take into consideration that we can have data either as single or double precision.

To solve this we will make a module containing all the necessary routines.

Since we do not want to have a lot of different subroutine names to call we will use an interface which allows us to have one subroutine name for both the single and double precision data.

Also it is convenient to have a global variable containing the array length.
Now we have finished the main program, but we still have to write the two subroutines `find_index` and `interpolate`.

So how do we go about doing this??

We have to take into consideration that we can have data either as single or double precision.

To solve this we will make a module containing all the necessary routines.

Since we do not want to have a lot of different subroutine names to call we will use an interface which allows us to have one subroutine name for both the single and double precision data.

Also it is convenient to have a global variable containing the array length.
Now we have finished the main program, but we still have to write the two subroutines *find_index* and *interpolate*

So how do we go about doing this??

We have to take into consideration that we can have data either as single or double precision

To solve this we will make a module containing all the neccessary routines

Since we do not want to have a lot of different subroutine names to call we will use an interface which allows us to have one subroutine name for both the single and double precision data

Also it is convenient to have a global variable containing the array length
Now we have finished the main program, but we still have to write the two subroutines `find_index` and `interpolate`.

So how do we go about doing this??

We have to take into consideration that we can have data either as single or double precision.

To solve this we will make a module containing all the necessary routines.

Since we do not want to have a lot of different subroutine names to call we will use an interface which allows us to have one subroutine name for both the single and double precision data.

Also it is convenient to have a global variable containing the array length.
Real Programs, Part 2

Solving the linear interpolation

Let us code some more

 INTEGER :: array_length
 INTERFACE interpolate
   MODULE PROCEDURE sinterp
   MODULE PROCEDURE dinterp
 END INTERFACE interpolate
 INTERFACE find_index
   MODULE PROCEDURE find_indexs
   MODULE PROCEDURE find_indexd
 END INTERFACE find_index
Real Programs, Part 2

Solving the linear interpolation

Some explanation

- A global variable array_length will contain the length of the time series we read from the input file.
- The construct `INTERFACE interpolate` gives us the opportunity to use the `interpolate` for both the single and double precision data.
- Note that the subroutines `sinterp` and `dinterp` will be selected and linked in at compile time depending on the input arguments when we call the `interpolate` subroutine.
- The same goes for the `find_index`.
Solving the linear interpolation

- Some explanation
- A global variable array_length will contain the length of the time series we read form the input file
- The construct INTERFACE interpolate gives us the opportunity to use the interpolate for both the single and double precision data
- Note that the subroutines sinterp and dinterp will be selected and linked in at compile time depending on the input arguments when we call the interpolate subroutine
- The same goes for the find_index
Solving the linear interpolation

- Some explanation
- A global variable array_length will contain the length of the time series we read from the input file
- The construct `INTERFACE interpolate` gives us the opportunity to use the `interpolate` for both the single and double precision data
  - Note that the subroutines `sinterp` and `dinterp` will be selected and linked in at compile time depending on the input arguments when we call the `interpolate` subroutine
- The same goes for the `find_index`
Solving the linear interpolation

- Some explanation
- A global variable array_length will contain the length of the time series we read from the input file
- The construct `INTERFACE interpolate` gives us the opportunity to use the `interpolate` for both the single and double precision data
- Note that the subroutines `sinterp` and `dinterp` will be selected and linked in at compile time depending on the input arguments when we call the `interpolate` subroutine
- The same goes for the `find_index`
Solving the linear interpolation

- Some explanation
- A global variable array_length will contain the length of the time series we read from the input file
- The construct `INTERFACE interpolate` gives us the opportunity to use the `interpolate` for both the single and double precision data
- Note that the subroutines `sinterp` and `dinterp` will be selected and linked in at compile time depending on the input arguments when we call the `interpolate` subroutine
- The same goes for the `find_index`
Now let us code the *find_index* subroutine

```fortran
SUBROUTINE find_indexs(array, s, e, mv)
  IMPLICIT NONE
  REAL(KIND=4), POINTER :: array(:)
  INTEGER, INTENT(IN) :: s
  INTEGER, INTENT(OUT) :: e
  REAL(KIND=4), INTENT(IN) :: mv
  INTEGER :: i
  DO i = s, array_length
    IF(array(i) /= mv) THEN
      EXIT
    END IF
  END DO
  e = i
END SUBROUTINE find_indexs
```
Solving the linear interpolation

- Some explanation
- This is a fairly straightforward subroutine which takes 4 arguments, the array, the starting index which both are input only.
- Then the return value containing the index of the next nonmissing value element and the missing value itself.
- If we are at the end of the array the returning index will be that of the last element.
- Note that the variable `array_length` is the global variable.
Solving the linear interpolation

Some explanation

This is a fairly straightforward subroutine which takes 4 arguments, the array, the starting index which both are input only

Then the return value containing the index of the next nonmissing value element and the missing value itself

If we are at the end of the array the returning index will be that of the last element

Note that the variable array_length is the global variable
Solving the linear interpolation

Some explanation
- This is a fairly straightforward subroutine which takes 4 arguments, the array, the starting index which both are input only
- Then the return value containing the index of the next nonmissing value element and the missing value itself
- If we are at the end of the array the returning index will be that of the last element
- Note that the variable array_length is the global variable
Some explanation

This is a fairly straightforward subroutine which takes 4 arguments, the array, the starting index which both are input only

Then the return value containing the index of the next nonmissing value element and the missing value itself

If we are at the end of the array the returning index will be that of the last element

Note that the variable array_length is the global variable
Solving the linear interpolation

- Some explanation
- This is a fairly straighe forward subroutine which takes 4 arguments, the array, the starting index which both are input only
- Then the return value containing the index of the next nonmissing value element and the missing value itself
- If we are at the end of the array the returning index will be that of the last element
- Note that the variable array_length is the global variable
Now for the really tricky part

Let us program the interpolation subroutine for the single precision data and begin with some declarations

```fortran
RECURSIVE SUBROUTINE sinterp(array,s,e)
  IMPLICIT NONE
  REAL(KIND=4), POINTER :: array(:)
  INTEGER, INTENT(IN) :: s, e
  INTEGER, INTENT(IN) :: i, j, k
  REAL(KIND=4) :: tmp_val
```
Now for the really tricky part

Let us program the interpolation subroutine for the single precision data and begin with some declarations

```fortran
RECURSIVE SUBROUTINE sinterp(array,s,e)
  IMPLICIT NONE
  REAL(KIND=4), POINTER :: array(:)
  INTEGER, INTENT(IN) :: s, e
  INTEGER :: i, j, k
  REAL(KIND=4) :: tmp_val
```
Some explanation

- The first thing to note is that we use a recursive subroutine.
- This means that the subroutine can call itself with new arguments.
- It is a nice way to be able to split a problem into smaller more manageable parts.
- The rest of the declaratons are well known from earlier code.
Solving the linear interpolation

- Some explanation
- The first thing to note is that we use a recursive subroutine
  - This means that the subroutine can call itself with new arguments
  - It is a nice way to be able to split a problem into smaller more manageable parts
- The rest of the declarations are well known from earlier code
Solving the linear interpolation

- Some explanation
- The first thing to note is that we use a recursive subroutine
- This means that the subroutine can call itself with new arguments
- It is a nice way to be able to split a problem into smaller more manageable parts
- The rest of the declaratons are well known from earlier code
Some explanation

The first thing to note is that we use a recursive subroutine.

This means that the subroutine can call itself with new arguments.

It is a nice way to be able to split a problem into smaller more manageable parts.

The rest of the declarations are well known from earlier code.
Some explanation

The first thing to note is that we use a recursive subroutine. This means that the subroutine can call itself with new arguments.

It is a nice way to be able to split a problem into smaller more manageable parts.

The rest of the declarations are well known from earlier code.
One element with missing value

\[ k = e - s - 1 \]

```fortran
SELECTCASE k
  CASE (1)
    j = s + 1
    tmp_val = array(s) + FLOAT(j-s) * ((array(e)-array(s))/FLOAT(e-s))
    array(j) = tmp_val
```

Some explanation

This code needs to be explained a little thoroughly.

The variable k receives the number of elements containing missing values.

Then we use a case construct to check if we have 1, 2, or more than 2 elements containing missing values.

If we have only one we calculate the interpolated value according to the formula and replaces the missing value with the interpolated one.
Solving the linear interpolation

- Some explanation
- This code needs to be explained a little thorough
  - The variable k receives the number of elements containing missing values
  - Then we use a case construct to check if we have 1, 2, or more than 2 elements containing missing values
  - If we have only one we calculate the interpolated value according to the formula and replaces the missing value with the interpolated one
Some explanation

This code needs to be explained a little thorough

The variable k receives the number of elements containing missing values

Then we use a case construct to check if we have 1, 2, or more than 2 elements containing missing values

If we have only one we calculate the interpolated value according to the formula and replaces the missing value with the interpolated one
Solving the linear interpolation

- Some explanation
- This code needs to be explained a little thorough
- The variable k receives the number of elements containing missing values
- Then we use a case construct to check if we have 1, 2, or more than 2 elements containing missing values
- If we have only one we calculate the interpolated value according to the formula and replaces the missing value with the interpolated one
Solving the linear interpolation

- Some explanation
- This code needs to be explained a little thorough
- The variable \( k \) receives the number of elements containing missing values
- Then we use a case construct to check if we have 1, 2, or more than 2 elements containing missing values
- If we have only one we calculate the interpolated value according to the formula and replaces the missing value with the interpolated one
Two consecutive elements with missing values

With two consecutive elements containing a missing value we will need a little modified code:

```
CASE(2)
  j = s + 1
  tmp_val = array(s) + FLOAT((j-s)) * 
             ((array(e)-array(s))/FLOAT((e-s)))
  array(j) = tmp_val
  CALL sinterp(array,j,e)
```
Solving the linear interpolation

- Two consecutive elements with missing values
- With two consecutive elements containing a missing value we will need a little modified code

```fortran
CASE(2)
j = s + 1
tmp_val = array(s) + FLOAT((j-s))* &
         ((array(e)-array(s))/FLOAT((e-s)))
array(j) = tmp_val
CALL sinterp(array,j,e)
```
Some explanation

Here we just calculate the first missing value and then use a recursive call to the subroutine to calculate the next value.

The value calculated in the recursive call is not really linear, but it is as close we can come with this algorithm.
Solving the linear interpolation

- Some explanation
- Here we just calculate the first missing value and then use a recursive call to the subroutine to calculate the next value
- The value calculated in the recursive call is not really linear, but it is as close we can come with this algorithm
Solving the linear interpolation

- Some explanation
- Here we just calculate the first missing value and then use a recursive call to the subroutine to calculate the next value
- The value calculated in the recursive call is not really linear, but it is as close we can come with this algorithm
If we have more than two consecutive elements with missing values we will have to extend our code.

Here is what can be done:

```
CASE DEFAULT
    j = s + (e - s)/2
    tmp_val = array(s) + FLOAT((j-s))* &
               ((array(e)-array(s))/FLOAT((e-s)))
    array(j) = tmp_val
    CALL sinterp(array,s,j)
    CALL sinterp(array,j,e)
END SELECT
```

END SUBROUTINE sinterp
If we have more than two consecutive elements with missing values we will have to extend our code

Here is what can be done

```fortran
CASE DEFAULT
    j = s + (e - s)/2
    tmp_val = array(s) + FLOAT((j-s))* &
              ((array(e)-array(s))/FLOAT((e-s)))
    array(j) = tmp_val
    CALL sinterp(array,s,j)
    CALL sinterp(array,j,e)
END SELECT
END SUBROUTINE sinterp
```
Solving the linear interpolation

Some explanation

- Note the use of the statement `CASE DEFAULT` which will be reached in the case we have more than two consecutive elements with missing values.
- First of all we here calculates the midpoint in the sequence.
- Then we calculate the interpolated value for the midpoint and use this for further calculations.
- Then we use the recursive call to the subroutine first for the left part of the sequence with the midpoint as the upper element.
- Then we do the same for the right part again using the midpoint as the lower element.
Solving the linear interpolation

- Some explanation
- Note the use of the statement *CASE DEFAULT* which will be reached in the case we have more than two consecutive elements with missing values
  - First of all we here calculates the midpoint in the sequence
  - Then we calculate the interpolated value for the midpoint and use this for further calculations
  - Then we use the recursive call to the subroutine first for the left part of the sequence with the midpoint as the upper element
  - Then we do the same for the right part again using the midpoint as the lower element
Real Programs, Part 2

Solving the linear interpolation

- Some explanation
- Note the use of the statement *CASE DEFAULT* which will be reached in the case we have more than two consecutive elements with missing values
- First of all we here calculates the midpoint in the sequence
  - Then we calculate the interpolated value for the midpoint and use this for further calculations
  - Then we use the recursive call to the subroutine first for the left part of the sequence with the midpoint as the upper element
  - Then we do the same for the right part again using the midpoint as the lower element
Solving the linear interpolation

- Some explanation
- Note the use of the statement `CASE DEFAULT` which will be reached in the case we have more than two consecutive elements with missing values
- First of all we here calculates the midpoint in the sequence
- Then we calculate the interpolated value for the midpoint and use this for further calculations
- Then we use the recursive call to the subroutine first for the left part of the sequence with the midpoint as the upper element
- Then we do the same for the right part again using the midpoint as the lower element
Solving the linear interpolation

- Some explanation
- Note the use of the statement `CASE DEFAULT` which will be reached in the case we have more than two consecutive elements with missing values
- First of all we here calculates the midpoint in the sequence
- Then we calculate the interpolated value for the midpoint and use this for further calculations
- Then we use the recursive call to the subroutine first for the left part of the sequence with the midpoint as the upper element
- Then we do the same for the right part again using the midpoint as the lower element
Some explanation

Note the use of the statement `CASE DEFAULT` which will be reached in the case we have more than two consecutive elements with missing values.

First of all we here calculates the midpoint in the sequence.

Then we calculate the interpolated value for the midpoint and use this for further calculations.

Then we use the recursive call to the subroutine first for the left part of the sequence with the midpoint as the upper element.

Then we do the same for the right part again using the midpoint as the lower element.
Solving the linear interpolation

- We now have a tool for performing a linear interpolation for floating point numbers.
- Note that the subroutines for the double precision numbers are exactly the same with the exception that we use a double precision array and missing value variable.
- The code for the main program is found here.
- The code for the module program is found here.
Solving the linear interpolation

- We now have a tool for performing a linear interpolation for floating point numbers.
- Note that the subroutines for the double precision numbers are exactly the same with the exception that we use a double precision array and missing value variable.
- The code for the main program is found [here](#).
- The code for the module program is found [here](#).
We now have a tool for performing a linear interpolation for floating point numbers.

Note that the subroutines for the double precision numbers are exactly the same with the exception that we use a double precision array and missing value variable.

The code for the main program is found here.

The code for the module program is found here.
We now have a tool for performing a linear interpolation for floating point numbers.

Note that the subroutines for the double precision numbers are exactly the same with the exception that we use a double precision array and missing value variable.

The code for the main program is found here.

The code for the module program is found here.