MET3220C
Meteorological Computations

Programming – week #1

Dr. Mark Bourassa

TA: David Moroni, room??, phone ??
Office Hours: TBA
dmoroni@met.fsu.edu

First Programming Assignment

- Goal #1: acquire familiarity with some simple UNIX commands
- Goal #2: acquire familiarity with a UNIX editor

- I suggest using the editor called emacs
  - It is available on most systems
  - It is reasonably easy to use
  - It has color coding that will cut down on mistakes

- The goal of the first assignment is to type in a simple program, compile it (without errors), and run it (without errors).

- But before we start that, we will go over how to start using the Meteorology Department’s computers and how to (and how not to) turn in assignments.

How to Log In

- Logging In to the machine that your monitor is attached to:
  - You need two pieces of information to log in
    - 1) A login ID
    - 2) The password associated with that login ID
    - Note that your password is something that you should keep secret so that someone else does not log into your account and do malicious things (e.g., deleting your assignments).
- When you sit down at the computer terminal, there should be a login prompt. After entering your login ID, you will typically be prompted for your password.
- Note regarding Metlab terminals: You are now logged into the best place for editing with emacs.
  - If you log into the metlab server you can also use emacs, but it will not have some of the cool functions.
- When your are done working, don't forget to log out!

Logging Into a Foreign Computer (using SSH)

- You can use SSH to log into a different computer.
- For example, to log into the metlab server, you would type
  - ssh metlab
  - This assumes that you logged into the original machine using the same login ID you have on the new machine.
- If you are logging in from a different ID, then type
  - ssh login_ID@metlab
  - Where login_ID is your login ID
- If you are logging into an offsite machine, you would type
  - ssh login_ID@name_or_ID_of_new_machine
  - Where name_or_ID_of_new_machine is the name of the new machine
  - For example:
    - ssh bourassa@huey.met.fsu.edu

Oddities of Meteorology’s System

- There are many metlab ‘terminals’ (e.g., metlab14), which are computers.
- There is also a more powerful metlab server (metlab.met.fsu.edu)
- You will probably want to
  - edit on the ‘terminals’, and
  - Compile and run code on the server.

Opening Multiple Windows

- You do not want to log in and log out between editing and trying to compile your code!
- Open another window, and use it to log into the server.
- How to do this varies a lot among computer system, but there is usual a button (on the window top, or screen top, or on a drop down menu after left clicking), that can be clicked to get another window.
- You can open more windows than you should need in this manner.

Working With Directories

- To create a new directory use the mkdir command
  - mkdir MET3220C
  - mkdir junk
  - Recall that UNIX commands ARE case sensitive
- To change directories use the cd command
  - cd MET3220C
  - cd .. Moves you up one directory level
  - cd MET3220C brings you back
  - cd ../junk takes you up one level, then down into the junk directory
  - cd ..
- To remove a directory use the rmdir command:
  - rmdir junk
- If you want to know what directory you are in type
  - pwd

Editing a File

- To edit a file using the emacs editor type
    - emacs filename
    - Where filename is the name of your file. E.g., my_great_code.f
        - If the file does not exist it will be created,
        - If it does exist, it will be opened for editing.
- You can enter text as you normally would on a really dumb word processor.
    - Please open a new file named my_great_code.f for editing.
    - In the file, write “I spent 100 hours on this amazing code!
    - Save the file: control-x control-s
    - Exit the file: control-x control-c
- Use a UNIX command to look at (but not edit) the file:
    - cat my_great_code.f
    - more my_great_code.f

Other UNIX commands

- The rm command can be used to delete (remove) files
  - First make a file: cp my_great_code.f oops.f
  - rm oops.f
  - Deleted files cannot be retrieved, so be cautious!
- If you want to see what files are in your directory, use the ls command
  - ls
  - If you just want to see files with an f90 extension type
  - ls *.f90
  - The * is a wild card representing any number of characters.
  - Similarly, a ? is a wild card representing one character.
- Note that typing ‘rm *’ should only be done with great caution.

Turning in Your Assignment

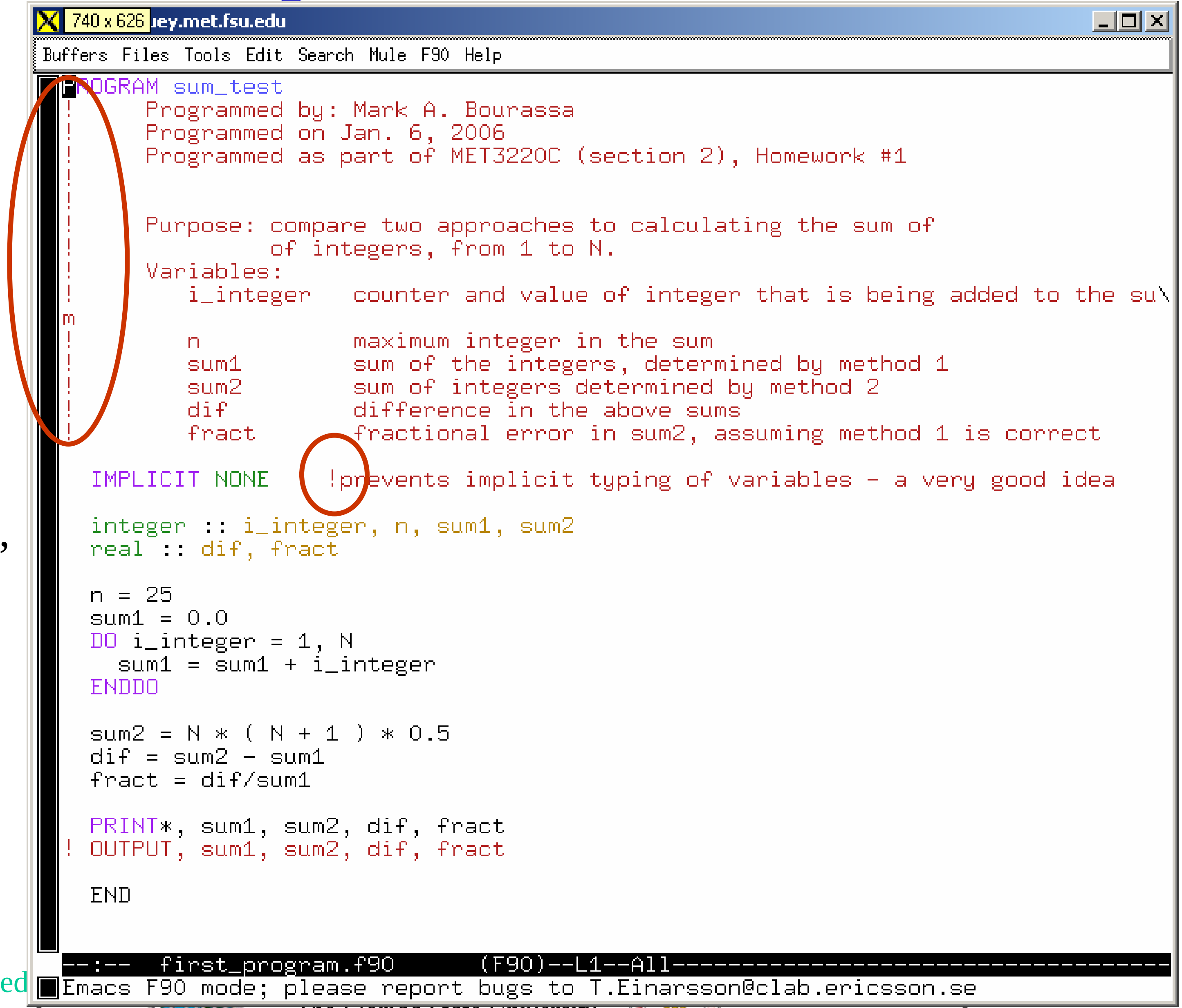
- This vary important step is where things can go horribly wrong!
- We will practice with the file that you just created.
- First, because bad things can happen, make a backup copy of your program:
  - cp is the UNIX command for copy
    - cp file1 file2   (don’t type cp file1 to file2)
    - Where file1 is the original, and file2 is the new file
  - Give it a try:
    - cp my_great_code.f my_great_code_bak.f
- To email the code use
  - /usr/lib/sendmail email_address < attachment
  - /usr/lib/sendmail your_ID@met.fsu.edu < my_great_code.f90
  - Now let’s make a horrible mistake:
  - /usr/lib/sendmail your_ID@met.fsu.edu > my_great_code.f90
  - You have sent nothing and have destroyed your great code!

Assignment #1: Enter, Compile, and Run a Simple Program

- 1) Create a MET3220 subdirectory
  - mkdir MET3220
- 2) Change directories to that directory
  - cd MET3220
- 3) Open a new file called AS1_your_last_name.f90, where ‘your_last_name’ is replaced by your last name.
  - emacs AS1_your_last_name.f90
- 4) Enter the program into that file.
- 5) Attempt to compile the program
  - f90 AS1_your_last_name.f90 –o AS1_your_last_name
  - Debug until it compiles.
- 6) Run the program
  - ./AS1_your_last_name
- 7) Email the working source code (AS1_your_last_name.f90) to the TA (dmoroni@met.fsu.edu).
- Due date: Tuesday, Jan. 17, before 5:00PM (local time)

The Program Seen In emacs

- The ! indicates comments. The text after the '!' is not converted to something the computer can interpret.
- It is good practice to define variables, and give some indication of what the code does.
- Well chosen variable names will help.

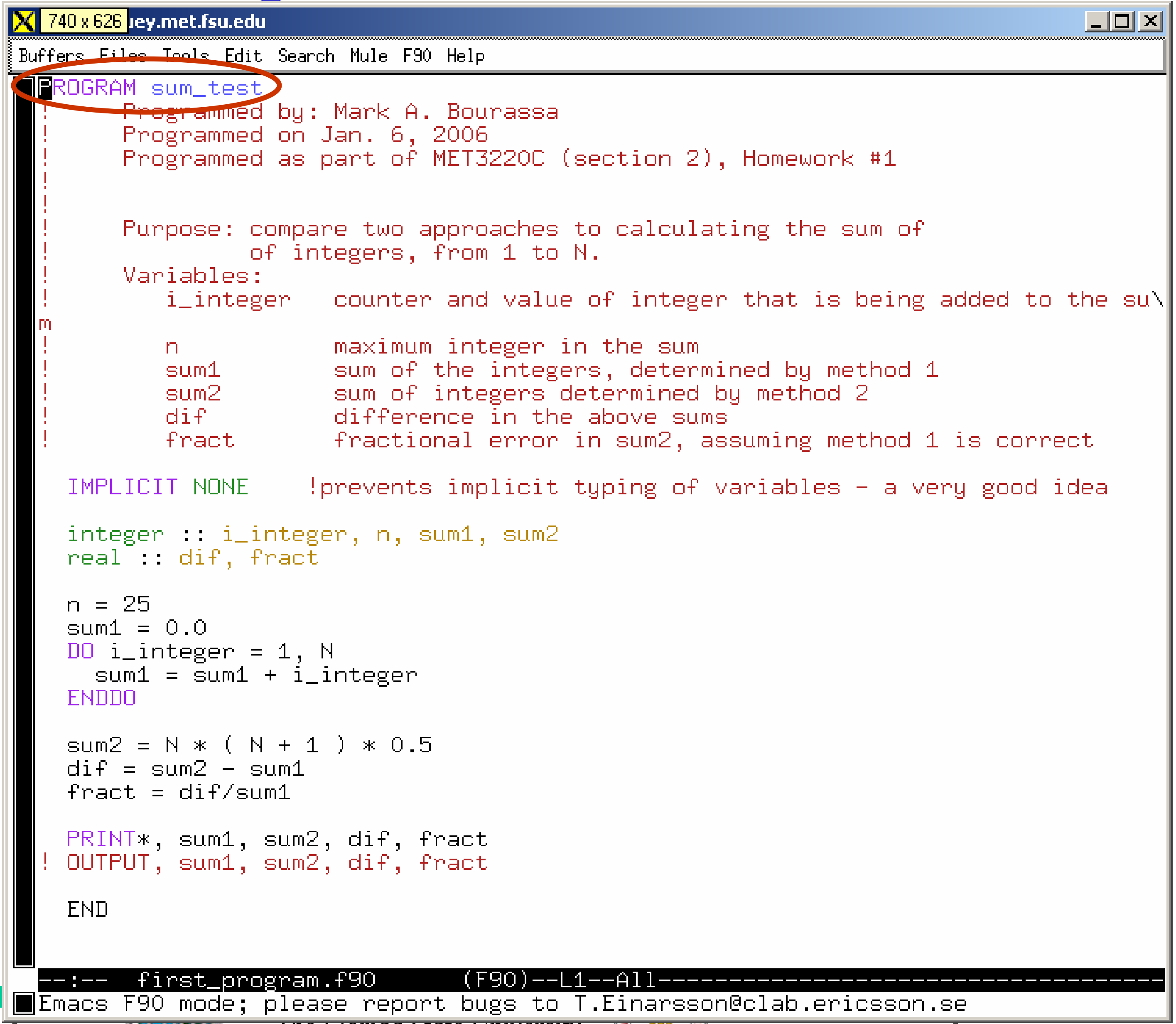


```
PROGRAM sum_test
!     Programmed by: Mark A. Bourassa
!     Programmed on Jan. 6, 2006
!     Programmed as part of MET3220C (section 2), Homework #1
!
!
!     Purpose: compare two approaches to calculating the sum of
!              of integers, from 1 to N.
!     Variables:
!        i_integer   counter and value of integer that is being added to the su\
m
!        n           maximum integer in the sum
!        sum1        sum of the integers, determined by method 1
!        sum2        sum of integers determined by method 2
!        dif         difference in the above sums
!        fract       fractional error in sum2, assuming method 1 is correct

  IMPLICIT NONE     !prevents implicit typing of variables - a very good idea

  integer :: i_integer, n, sum1, sum2
  real :: dif, fract

  n = 25
  sum1 = 0.0
  DO i_integer = 1, N
    sum1 = sum1 + i_integer
  ENDDO

  sum2 = N * ( N + 1 ) * 0.5
  dif = sum2 - sum1
  fract = dif/sum1

  PRINT*, sum1, sum2, dif, fract
! OUTPUT, sum1, sum2, dif, fract

  END
```

```
--:--  first_program.f90      (F90)--L1--All-----------------------------------
Emacs F90 mode; please report bugs to T.Einarsson@clab.ericsson.se
```

The Program Seen In emacs

- The program name.
- Chose something that makes sense!

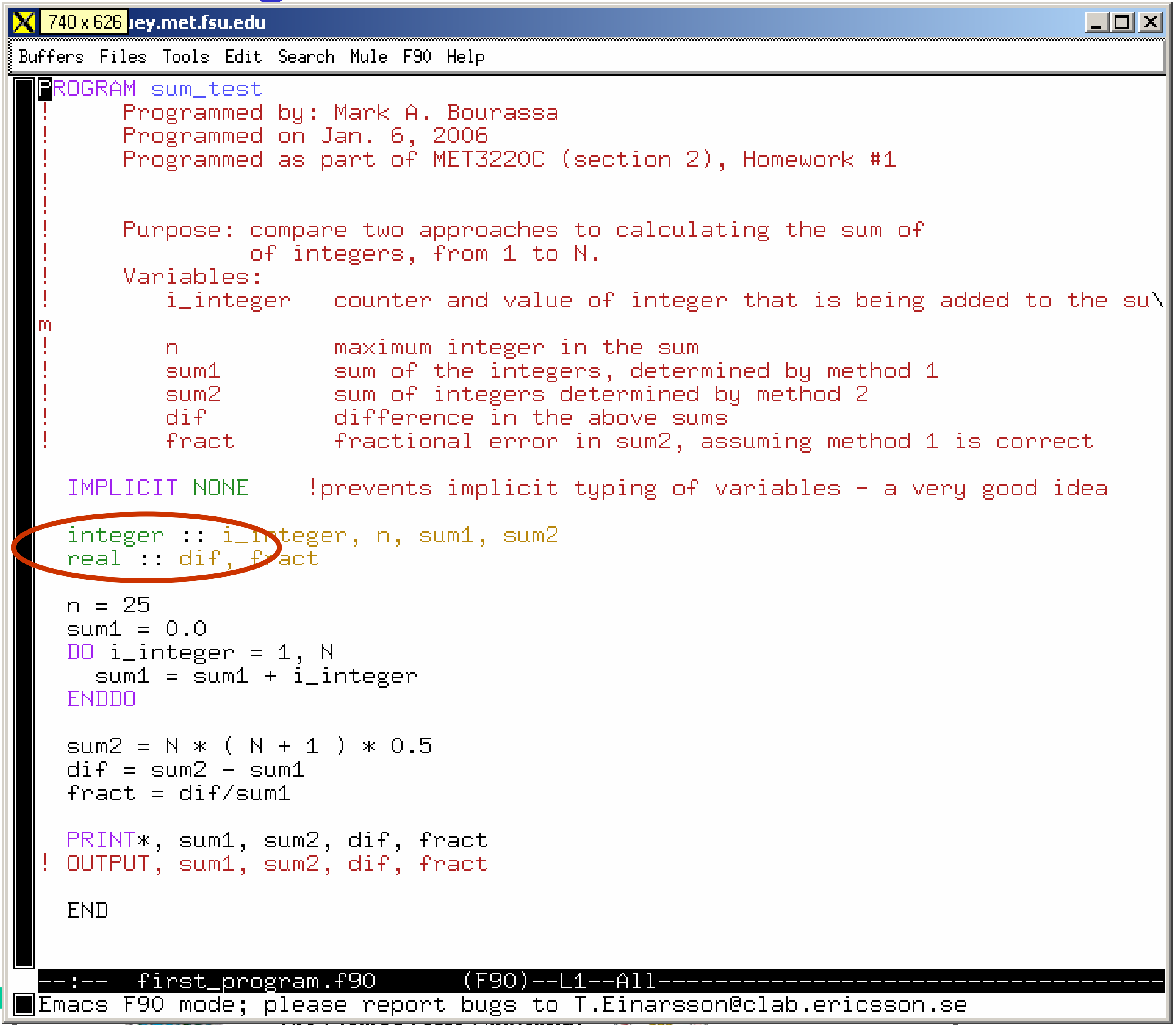


```
PROGRAM sum_test
      Programmed by: Mark A. Bourassa
!     Programmed on Jan. 6, 2006
!     Programmed as part of MET3220C (section 2), Homework #1
!
!
!     Purpose: compare two approaches to calculating the sum of
!              of integers, from 1 to N.
!     Variables:
!        i_integer   counter and value of integer that is being added to the su\
m
!        n           maximum integer in the sum
!        sum1        sum of the integers, determined by method 1
!        sum2        sum of integers determined by method 2
!        dif         difference in the above sums
!        fract       fractional error in sum2, assuming method 1 is correct

  IMPLICIT NONE    !prevents implicit typing of variables - a very good idea

  integer :: i_integer, n, sum1, sum2
  real :: dif, fract

  n = 25
  sum1 = 0.0
  DO i_integer = 1, N
    sum1 = sum1 + i_integer
  ENDDO

  sum2 = N * ( N + 1 ) * 0.5
  dif = sum2 - sum1
  fract = dif/sum1

  PRINT*, sum1, sum2, dif, fract
! OUTPUT, sum1, sum2, dif, fract

  END
```

```
--:--  first_program.f90      (F90)--L1--All------------------------------------
Emacs F90 mode; please report bugs to T.Einarsson@clab.ericsson.se
```

The Program Seen In emacs

- Declarations
- These specify the ‘nature’ of the variable.
- Integers are whole numbers, positive or negative.
- Reals can also be fractions.
- The distinction has to do with how memory is allocated when the code is compiled or run.

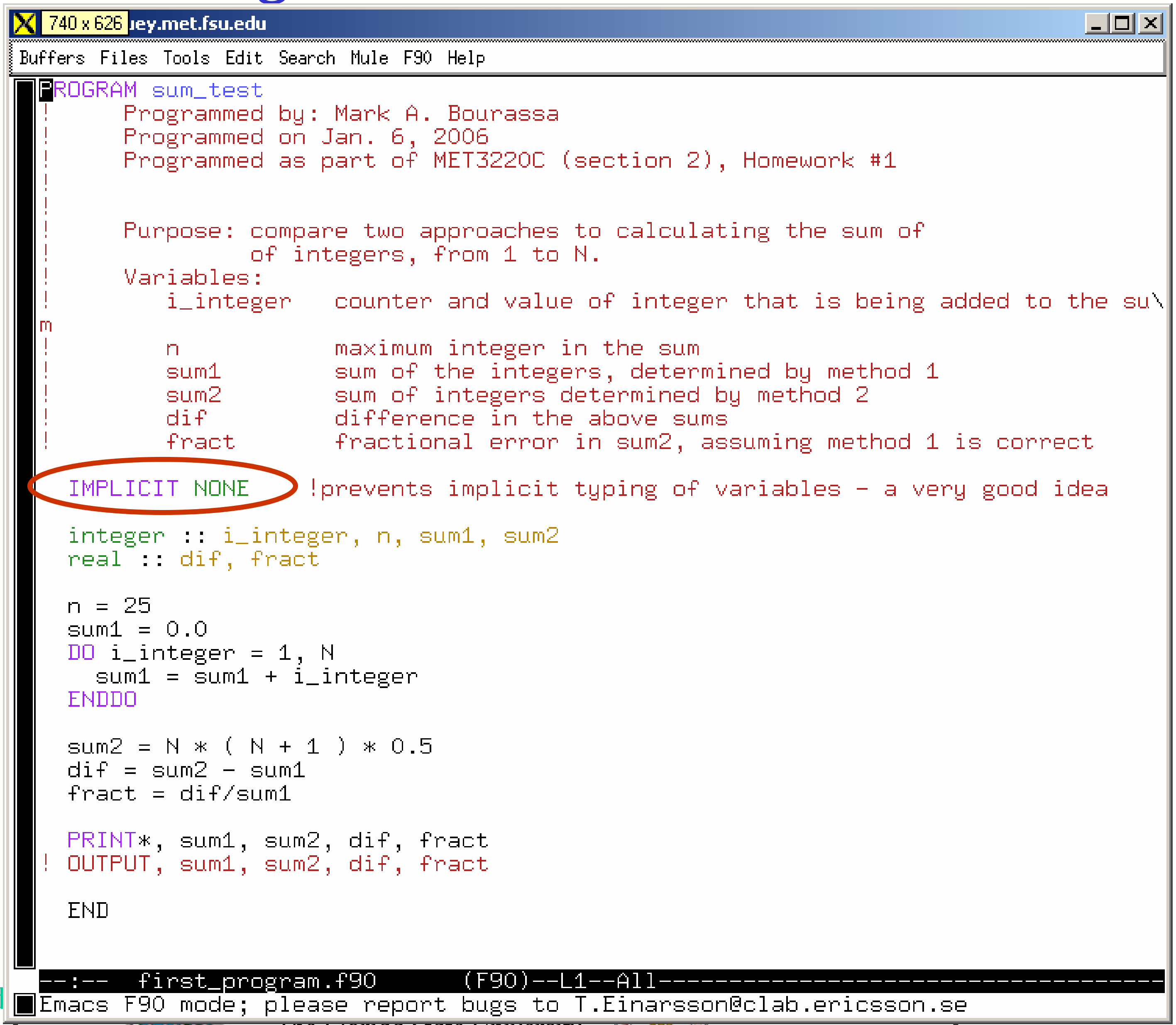


```
PROGRAM sum_test
!     Programmed by: Mark A. Bourassa
!     Programmed on Jan. 6, 2006
!     Programmed as part of MET3220C (section 2), Homework #1
!
!
!     Purpose: compare two approaches to calculating the sum of
!              of integers, from 1 to N.
!     Variables:
!        i_integer   counter and value of integer that is being added to the su\
m
!        n           maximum integer in the sum
!        sum1        sum of the integers, determined by method 1
!        sum2        sum of integers determined by method 2
!        dif         difference in the above sums
!        fract       fractional error in sum2, assuming method 1 is correct

  IMPLICIT NONE    !prevents implicit typing of variables - a very good idea

  integer :: i_integer, n, sum1, sum2
  real :: dif, fract

  n = 25
  sum1 = 0.0
  DO i_integer = 1, N
    sum1 = sum1 + i_integer
  ENDDO

  sum2 = N * ( N + 1 ) * 0.5
  dif = sum2 - sum1
  fract = dif/sum1

  PRINT*, sum1, sum2, dif, fract
! OUTPUT, sum1, sum2, dif, fract

  END

--:--  first_program.f90      (F90)--L1--All------------------------------------
Emacs F90 mode; please report bugs to T.Einarsson@clab.ericsson.se
```

The Program Seen In emacs

- The default naming convection for variables that are not declared is names starting with letter i to n are integers, and everything else is a real.

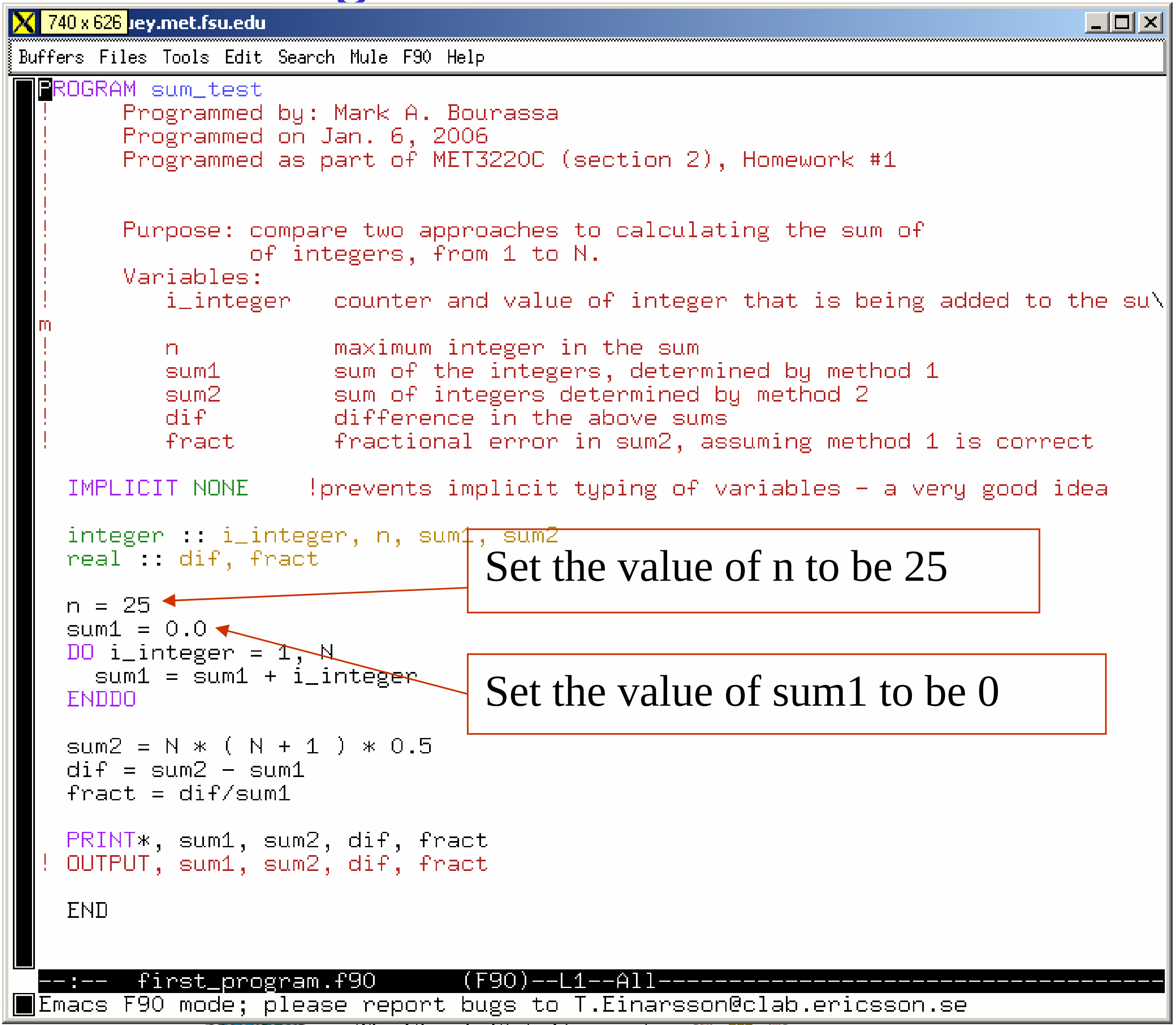
- The implicit none statement causes any undeclared variables to generate errors.



```
PROGRAM sum_test
!     Programmed by: Mark A. Bourassa
!     Programmed on Jan. 6, 2006
!     Programmed as part of MET3220C (section 2), Homework #1
!
!
!     Purpose: compare two approaches to calculating the sum of
!              of integers, from 1 to N.
!     Variables:
!        i_integer   counter and value of integer that is being added to the su\
m
!        n           maximum integer in the sum
!        sum1        sum of the integers, determined by method 1
!        sum2        sum of integers determined by method 2
!        dif         difference in the above sums
!        fract       fractional error in sum2, assuming method 1 is correct

  IMPLICIT NONE     !prevents implicit typing of variables - a very good idea

  integer :: i_integer, n, sum1, sum2
  real :: dif, fract

  n = 25
  sum1 = 0.0
  DO i_integer = 1, N
    sum1 = sum1 + i_integer
  ENDDO

  sum2 = N * ( N + 1 ) * 0.5
  dif = sum2 - sum1
  fract = dif/sum1

  PRINT*, sum1, sum2, dif, fract
! OUTPUT, sum1, sum2, dif, fract

  END
```

```
--:--  first_program.f90      (F90)--L1--All---------------------------------------
Emacs F90 mode; please report bugs to T.Einarsson@clab.ericsson.se
```

The Program Seen In emacs

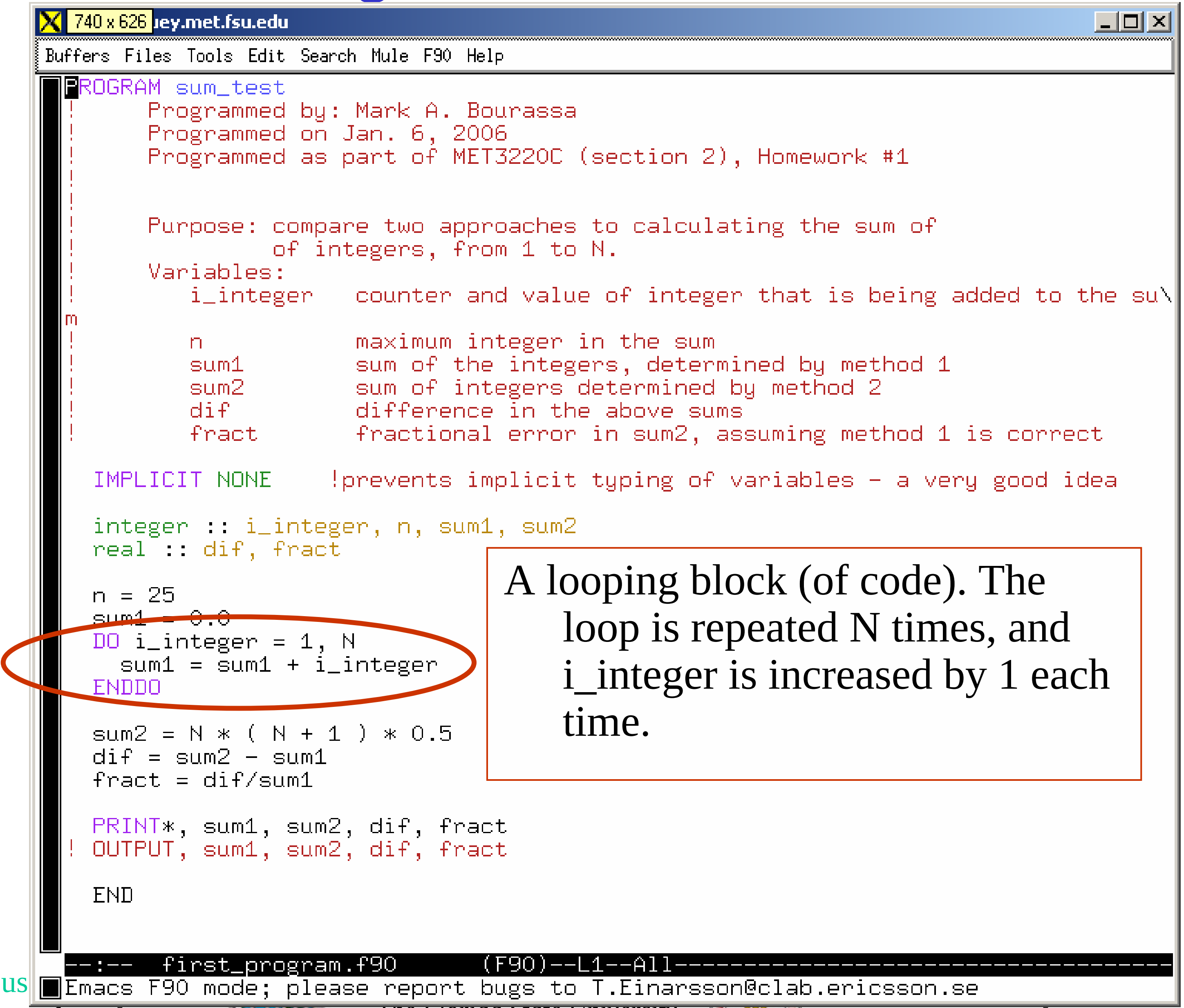


```
PROGRAM sum_test
!      Programmed by: Mark A. Bourassa
!      Programmed on Jan. 6, 2006
!      Programmed as part of MET3220C (section 2), Homework #1
!
!
!      Purpose: compare two approaches to calculating the sum of
!               of integers, from 1 to N.
!      Variables:
!         i_integer   counter and value of integer that is being added to the su\
m
!         n           maximum integer in the sum
!         sum1        sum of the integers, determined by method 1
!         sum2        sum of integers determined by method 2
!         dif         difference in the above sums
!         fract       fractional error in sum2, assuming method 1 is correct

  IMPLICIT NONE    !prevents implicit typing of variables - a very good idea

  integer :: i_integer, n, sum1, sum2
  real :: dif, fract

  n = 25
  sum1 = 0.0
  DO i_integer = 1, N
    sum1 = sum1 + i_integer
  ENDDO

  sum2 = N * ( N + 1 ) * 0.5
  dif = sum2 - sum1
  fract = dif/sum1

  PRINT*, sum1, sum2, dif, fract
! OUTPUT, sum1, sum2, dif, fract

  END

--:--  first_program.f90      (F90)--L1--All----------------------------------------
Emacs F90 mode; please report bugs to T.Einarsson@clab.ericsson.se
```

Set the value of n to be 25

Set the value of sum1 to be 0

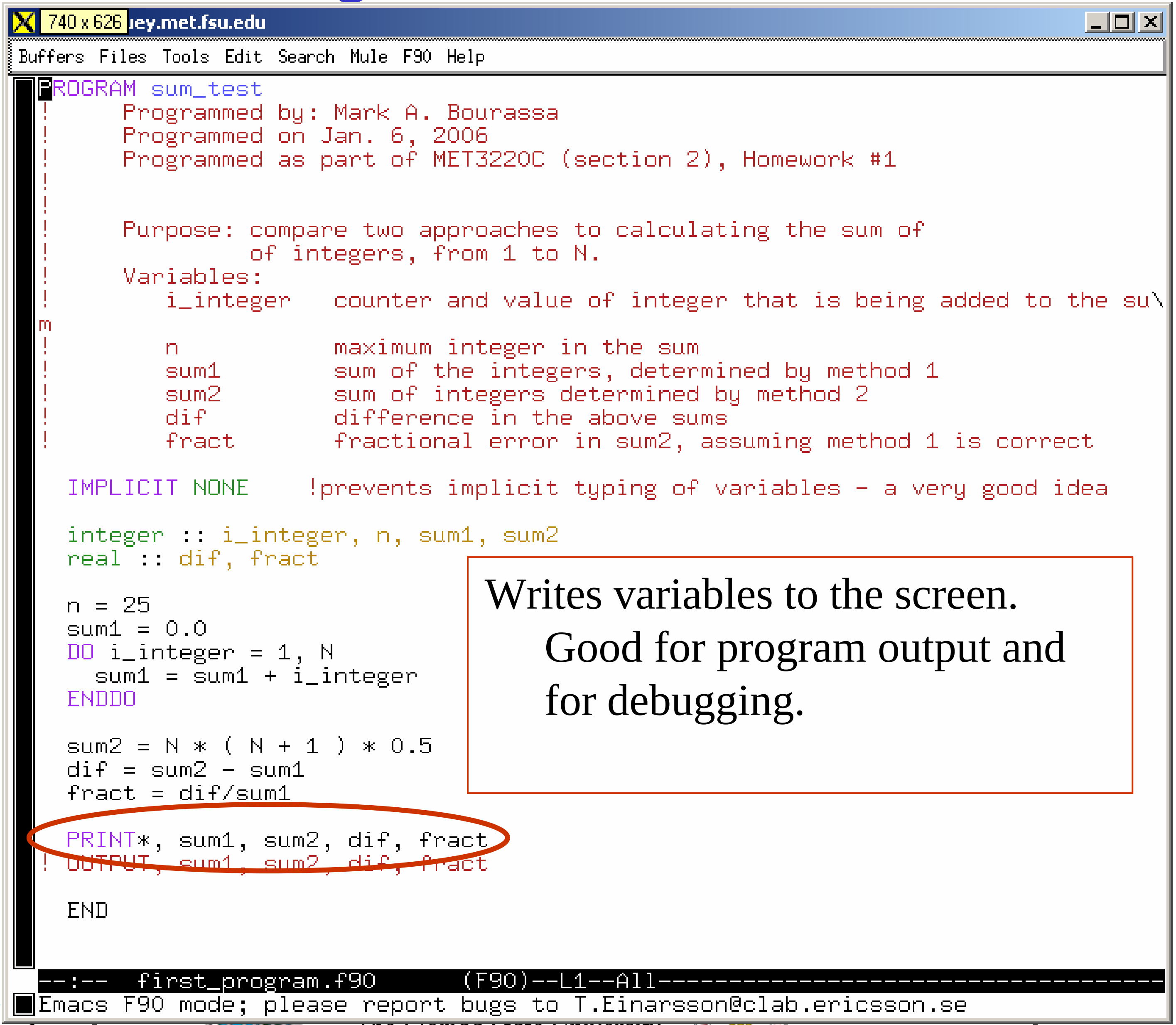
The Program Seen In emacs



```
PROGRAM sum_test
!       Programmed by: Mark A. Bourassa
!       Programmed on Jan. 6, 2006
!       Programmed as part of MET3220C (section 2), Homework #1
!
!
!       Purpose: compare two approaches to calculating the sum of
!                of integers, from 1 to N.
!       Variables:
!          i_integer   counter and value of integer that is being added to the su\
m
!          n           maximum integer in the sum
!          sum1        sum of the integers, determined by method 1
!          sum2        sum of integers determined by method 2
!          dif         difference in the above sums
!          fract       fractional error in sum2, assuming method 1 is correct

  IMPLICIT NONE     !prevents implicit typing of variables - a very good idea

  integer :: i_integer, n, sum1, sum2
  real :: dif, fract

  n = 25
  sum1 = 0.0
  DO i_integer = 1, N
    sum1 = sum1 + i_integer
  ENDDO

  sum2 = N * ( N + 1 ) * 0.5
  dif = sum2 - sum1
  fract = dif/sum1

  PRINT*, sum1, sum2, dif, fract
! OUTPUT, sum1, sum2, dif, fract

  END

--:--  first_program.f90      (F90)--L1--All------------------------------------
Emacs F90 mode; please report bugs to T.Einarsson@clab.ericsson.se
```

A looping block (of code). The loop is repeated N times, and i_integer is increased by 1 each time.

The Program Seen In emacs

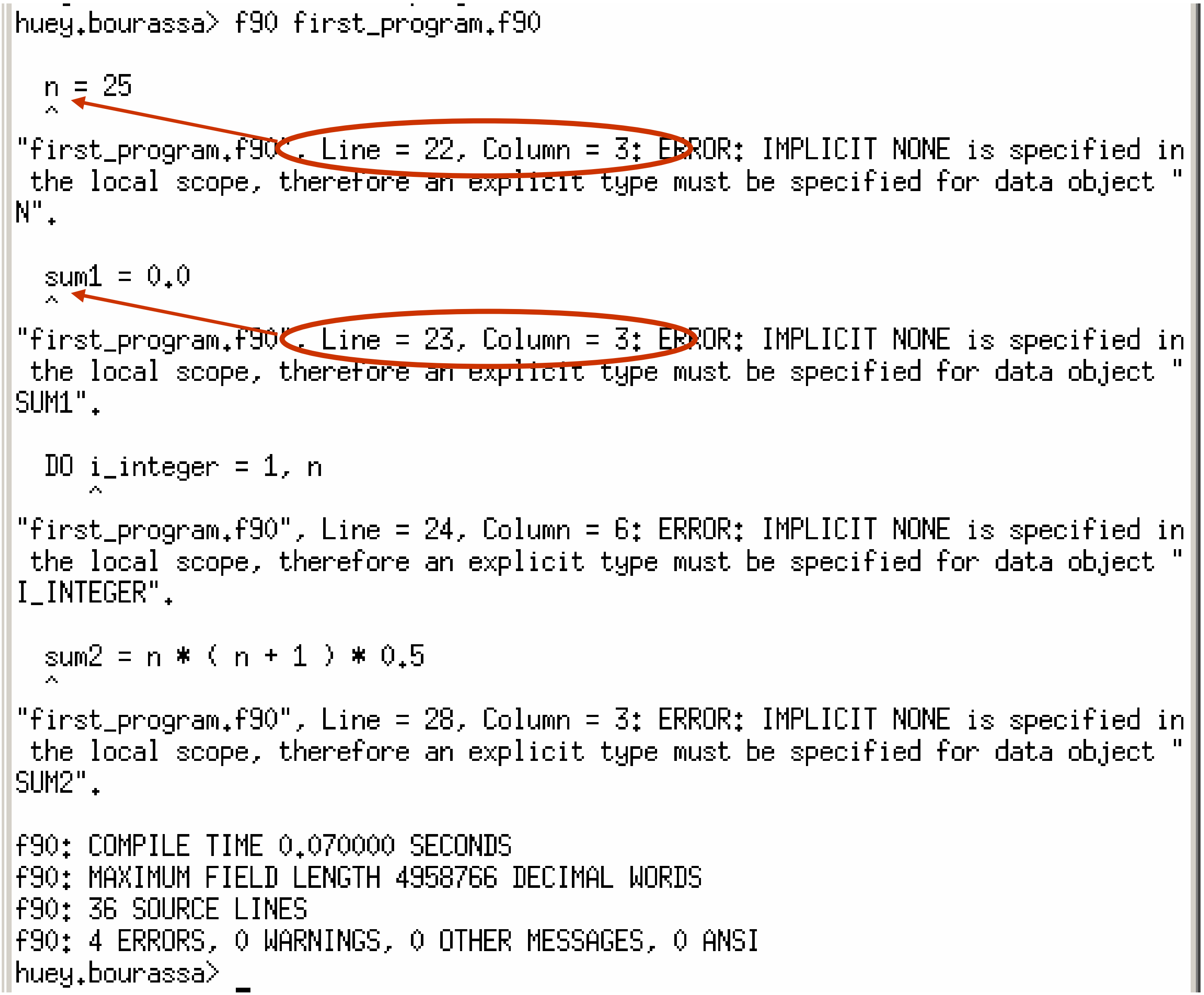


```
PROGRAM sum_test
!     Programmed by: Mark A. Bourassa
!     Programmed on Jan. 6, 2006
!     Programmed as part of MET3220C (section 2), Homework #1
!
!
!     Purpose: compare two approaches to calculating the sum of
!              of integers, from 1 to N.
!     Variables:
!        i_integer   counter and value of integer that is being added to the su\
m
!        n           maximum integer in the sum
!        sum1        sum of the integers, determined by method 1
!        sum2        sum of integers determined by method 2
!        dif         difference in the above sums
!        fract       fractional error in sum2, assuming method 1 is correct

  IMPLICIT NONE    !prevents implicit typing of variables - a very good idea

  integer :: i_integer, n, sum1, sum2
  real :: dif, fract

  n = 25
  sum1 = 0.0
  DO i_integer = 1, N
    sum1 = sum1 + i_integer
  ENDDO

  sum2 = N * ( N + 1 ) * 0.5
  dif = sum2 - sum1
  fract = dif/sum1

  PRINT*, sum1, sum2, dif, fract
! OUTPUT, sum1, sum2, dif, fract

  END

--:--  first_program.f90      (F90)--L1--All-----------------------------------
Emacs F90 mode; please report bugs to T.Einarsson@clab.ericsson.se
```

Writes variables to the screen.

Good for program output and for debugging.

Example Error Messages



```
huey.bourassa> f90 first_program.f90

  n = 25
  ^
"first_program.f90", Line = 22, Column = 3: ERROR: IMPLICIT NONE is specified in
 the local scope, therefore an explicit type must be specified for data object "
N".

  sum1 = 0.0
  ^
"first_program.f90", Line = 23, Column = 3: ERROR: IMPLICIT NONE is specified in
 the local scope, therefore an explicit type must be specified for data object "
SUM1".

  DO i_integer = 1, n
     ^
"first_program.f90", Line = 24, Column = 6: ERROR: IMPLICIT NONE is specified in
 the local scope, therefore an explicit type must be specified for data object "
I_INTEGER".

  sum2 = n * ( n + 1 ) * 0.5
  ^
"first_program.f90", Line = 28, Column = 3: ERROR: IMPLICIT NONE is specified in
 the local scope, therefore an explicit type must be specified for data object "
SUM2".

f90: COMPILE TIME 0.070000 SECONDS
f90: MAXIMUM FIELD LENGTH 4958766 DECIMAL WORDS
f90: 36 SOURCE LINES
f90: 4 ERRORS, 0 WARNINGS, 0 OTHER MESSAGES, 0 ANSI
huey.bourassa> _
```