Tutorial 4

ZHANG Jian

Outline

- More about Fortran
  - Function
- Common mistakes
  - mix up the names of functions and variables
  - string concatenation
  - the spaces after the date string
  - forget a GOTO statement
- Hints
  - Sorting
  - How to check the end of the file
  - How to read command line arguments
  - How to output \r\n on UNIX-like systems in Fortran
  - How to equivalently represent DO LOOP with IF GOTO
  - How to equivalently represent IF ELSE with IF GOTO
  - Sorting

More about Fortran

- Function

Function

- Function is similar to Subroutine. They both can be used to improve the codes' reusability.
- Differences:

Function	Subroutine
need to declare the function name in the variable declaration section of the caller	not need
Use the function name directly	Need to use a CALL statement to call
must use the function name in an assignment statement within the function. This is how the compiler knows which value to pass back to the main program	not need
Usually doesn't change the value of parameters	May change the valued of parameters
Can be used in an expression	Cannot

Function

```
PROGRAM test
 integer M,N,Sum
 M=5
 N=8
 WRITE(*,'(A,I)') 'M+N=', Sum(M,N)
END

Integer FUNCTION Sum(M,N)
  integer M,N
  Sum = M+N
END
```

- 1. declare the function name in the variable declaration section.
- 2. CALL statement isn't needed
- 3. use the function name in an assignment statement. This tells the program which value to be returned.

General structure of defining a function

- [type of the returned value] FUNCTION funcName(parameter list)

  function body

- END [FUNCTION funcName]

Common mistakes

- mix up the names of functions and variables
- string concatenation
- the spaces after the date string
- forget a GOTO statement

Never use intrinsic function name as variable name

- Fortran has no preserved words, which means you can use the name of some intrinsic functions as variables' names.

- For example, this is permitted in Fortran:

```
SIN = 3.5
PRINT *,  SIN
```

Never use intrinsic function name as variable name

- Wrong! Can’t pass the compilation!
- This also brings risks that you may mix up the variable and the function!

```
SIN = 5.0
A=SIN*SIN(2.0)
```

Format conversion of date

- In Ass1, you are required to convert the date like “2019-08-03” to another format: “August 3, 2019”.
- The idea to do that:
  - Firstly extract the value of the month 08 and then convert it to “August”
  - Secondly extract the value of the day 03 and then convert it to “3”
  - Finally, extract the value of the year and concatenate it with 2 strings above
- There is a possible mistake here!

String concatenation

```
PROGRAM atd
  character*20 date_string
  date_string='March'
  date_string=date_string//' 02'
  WRITE(*,'(A)') date_string
STOP
END
```

- Output:
- March_____ (use underlines to denote spaces here)

How to solve it

- Our purpose is to remove the spaces

- 1. substring and string concatenation

```
PROGRAM test
    character*20 a
    a="January"
    WRITE(*,'(A)') a
    a=a(1:7)//" 01"
    WRITE(*,'(A)') a
END
```

How to solve it

- 2. substring and WRITE statement

```
PROGRAM test
  character*20 a
  character*30 formatDate
  integer month
  month=3
  a="January"
  WRITE(formatDate,'(AX,I2)') a(1:7), month
  WRITE(*,'(A)') formatDate
END
```

the spaces after the date string

- different date strings have different lengths

```
Daily Attendance Summary
Date: January 4, 2019
Staff-ID Name                                Department Status
-------------------------------------------------------------
1009     CHAN       TAI MAN                  ITD        LATE
1077     WONG       ALICE                    ITD        PRESENT
1823     WONG       SIU MING                 HRD        SUSPICIOUS
-------------------------------------------------------------
Number of Presences:    1
```

- The specification requires that the length of the date string be 18. Be careful that you should pad some spaces after it.

- Be careful with all the spaces in other places. We may use diff instruction to compare your output with our answers.

Forget GOTO statement

- It’s quite difficult to debug when you can only use IF GOTO in your codes.
- My suggestion is that you may write codes with modern control structures firstly, like IF-THE-ELSE and DO-LOOP. After you test your codes and make sure that your codes are right, then you can try to translate the modern control structures into IF-GOTO structure.

Forget GOTO statement

```
      PROGRAM test
        INTEGER a
        IF(a .EQ. 3) THEN
10         a=a+1
        ELSE
11        a=a+2
        ENDIF

12      WRITE(*,'(I)') a
      END
```

```
      PROGRAM test
        INTEGER a
        IF(a .EQ. 3) GOTO 10
        GOTO 11
10      a=a+1
        GOTO 12

11      a=a+2
12      WRITE(*,'(I)') a
      END
```

A simpler way

- Negating the condition can saves a GOTO statement

```
      PROGRAM test
        INTEGER a
        IF(.NOT. (a .EQ. 3)) GOTO 11
10      a=a+1
        GOTO 12

11      a=a+2
12      WRITE(*,'(I5)') a
      END
```

Hints

- Sort the attendance.txt
- How to check the end of the file
- How to read command line arguments
- How to output \r\n on UNIX-like systems in Fortran
- How to equivalently represent DO LOOP with IF GOTO
- How to equivalently represent IF ELSE with IF GOTO
- Sorting

Sorting

- I didn’t find any intrinsic sort functions in Fortran77. I tried qsort and fsort, but they didn’t work on sparc1.
- If you can find, then you can use. But make sure that your codes can pass the compilation on sparc1, not on your own computer and not with another version of Fortran compiler.

- How to write our own sort subroutine?
  - bubble sort is enough
  - implement a swap subroutine first before the sort subroutine since bubble sort is dependent on swap operation
  - you can use 2 nested loops to implement your sort subroutine and then translate

Bubble sort

- A video about bubble sort

- n-1 passes in total
- In each pass, from the left to the right, compare every 2 elements, if the left one is bigger, then swap them.
- After i-th pass, the biggest element in the first (n+1-i) elements is moved to the rightest position
- If after the current pass, there isn’t any swap, then it’s already sorted.

Example of Bubble sort

- **First Pass:**
  (**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
  (1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4
  (1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2
  (1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.
- **Second Pass:**
  (**1 4** 2 5 8) –> (**1 4** 2 5 8)
  (1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2
  (1 2 **4 5** 8) –> (1 2 **4 5** 8)
  (1 2 4 **5 8**) –> (1 2 4 **5 8**)

How to check the end of the file

- In Ass1, it’s required to read files into memory. But how can we know whether it has already reached the end of the file?

- A optional parameter called IOSTAT can help with that.

- `READ (unit, label, IOSTAT=varname)`

How to check the end of the file

- Pass a IOSTAT parameter to READ statement.

- After the execution of READ statement, the value of this parameter will be set to some value according to the different circumstances of execution.

How to check the end of the file

- IOSTAT = 0: **READ** statement was executed successfully, all variables already have their values.

- IOSTAT > 0: **READ** has encountered some problem. For example, supplying a real number to an integer variable.

- IOSTAT < 0: the end of the input has reached.

How to read command line arguments?

- The specification of Ass1 requires that the program should be run with this statement:

  ./a employees.txt monthly-attendance.txt attendance.txt

- The names of the input files are passed to the program as command line arguments, like in C:

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

How to read command line arguments?

- How to get the values of names of the input files?

- an INTRINSIC function:
- CALL getarg(i, varname): read argument i to varname

instruction	./a	employees.txt	monthly-attendance.txt	attendance.txt
i-th argument	argument 0	argument 1	argument 2	argument 3
C Programming Language	argv[0]	argv[1]	argv[2]	argv[3]

\r\n

- In the past, it’s totally different between \r and \n.
- Imagine a write head on the printer, then:

- \r: Carriage Return(CR): move the write head to the left margin of this line
- \n: Line Feed(LF): move the write head to the next line, still in the same column

\r\n

- The designers of Windows and UNIX-like systems disagree with each other.
- UNIX-like: they think it’s wasteful to use 2 characters as the separator of lines, so they decided to use ‘\n’ as the separator.
- Windows: still use ‘\r\n’ as the separator of lines

\r\n

- The specification requires that we should use ‘\r\n’ as the separator. Then how to use ‘\r\n’ in Fortran on Solaris?

- WRITE statement will output the ‘\n’ automatically, you only need to append a ‘\r’ to the string.

- Example:
- `WRITE(4, '(A)') 'Daily Attendance Summary\r'`

Translate DO LOOP into IF GOTO

- Since all of us are familiar with modern advanced control structures, it may be more comfortable for you to write codes with those structures.

- You can use them to write codes first, and then translate them into IF-GOTO representation.

- Just a suggestion. It’s up to you.

Translate DO LOOP into IF GOTO

```
      PROGRAM test
        INTEGER a,b
        DO a=1, 10, 2
          b = b+1
        ENDDO
12      WRITE(*,'(A,I2,A,I2)') 'a=',a,';b=',b
      END
```

Translate DO LOOP into IF GOTO

- Equivalent representation of DO-LOOP with IF-GOTO

```
      PROGRAM test
        INTEGER a,b
        a=1
11      IF(a .GT. 10) GOTO 12
        b = b+1
        a = a+2
        GOTO 11

12      WRITE(*,'(A,I2,A,I2)') 'a=',a,';b=',b
      END
```

Translate DO LOOP into IF GOTO

- General Translation:

```
        DO initilization expression, upper bound, step size
          loop body
        ENDDO
label1  ANY_STATEMENT
```

```
        initilization expression
label2  IF(varname .GT. upper bound) GOTO label1
        loop body
        increment statement
        GOTO label2
label1  ANY_STATEMENT
```

Translate IF ELSE into IF GOTO

- Refer to the ‘Forget GOTO statement’ page

Translate IF ELSE into IF GOTO

- General Translation:

```
    IF(condition) THEN
      if section
    ELSE
l1    else section
    ENDIF
l2  ANY_STATEMENT
```

```
    IF(.NOT. condition) GOTO l1
    if section
    GOTO l2
l1  else section
l2  ANY_STATEMENT
```

Thank you