How to THINK like a Programmer

Problem Solving for the Bewildered

paul vickers

Problem Solving 7

More on Indeterminate Loops

John Brosnan,
Computing Dept, ITT

Aim

- In this lesson we will continue our study of looping control structures
  - Input validation using indeterminate loops
  - Task-controlled loops

Indeterminate loops

- Recall that indeterminate loops are ones where we cannot know in advance the number of times the loop will iterate.
- Many programming situations require the use of such loops. The data-sentinel controlled while loop we examined last week is an example of one.
- Another very important situation that arises where an indeterminate loop can be used is for input validation.
- In this case, the user is asked to supply some information and we want to ensure that the value supplied is proper e.g. if you ask the user to input a number, then you should not be accepting arbitrary text values.

Input Validation

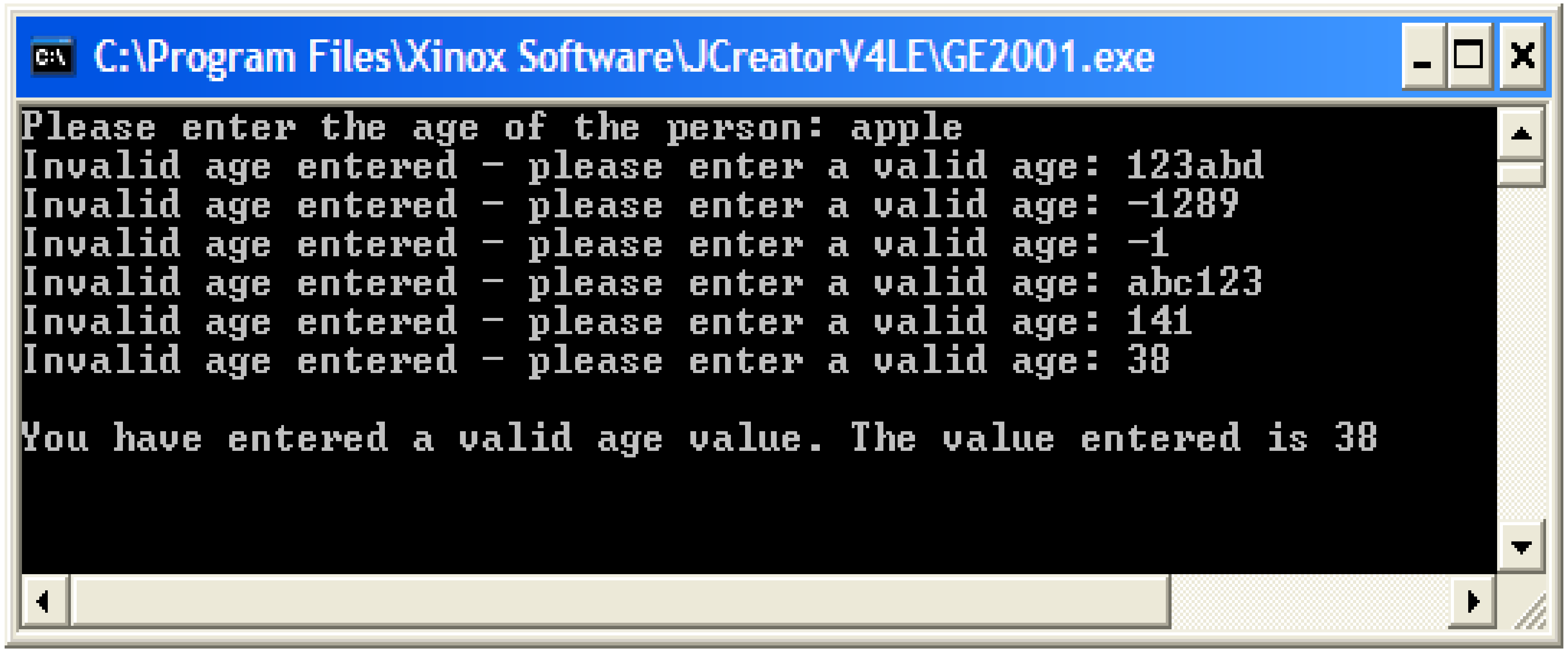
- Imagine that the user is required to supply their age.
- In this case, a little thought will tell us that a valid age must be a value over 0 but less than, say,140.
- Of course, a valid age must also be a valid integer number, so values such as 23.5 and “xyz” should be rejected also.
- In some languages, such as Java, a program will crash if an attempt is made to input text for a numeric variable, but in the case of others e.g. Just BASIC, the text will actually be accepted, creating a logical error

Using `while` for input validation

- A pseudocode solution for validating somebody's age might be:

```
1. Prompt for the user's age
2. Read in the user's age as a string
3. Set the value of a boolean-like variable called valid to false (gets us into the validation loop)
4. while (age is not a valid integer i.e. valid = false) //assume it is not valid to begin with
      4.1   Determine the number of characters in the age value entered
      4.2   Extract the first character in the age value entered
      4.3   Set index to 0
      4.4   while(index is less than number of characters in the age value and
                  the character at position index in the age string is a digit)
            4.4.1  Increment the value of index by 1
            4.4.2  Extract the next character in the age value entered
            endwhile
      4.5   if (index = length of age string)
              4.5.1   Convert the age string to an integer
              4.5.2   If (age>=0) and (age<=140)
                        4.5.2.1 Tell user age value is valid
                        4.5.2.2 Set the value of the boolean variable valid to true
                      otherwise
                        4.5.2.3 Display error message and ask user to enter a valid age
                        4.5.2.4 Read in the new age as a string
                        4.5.2.5 Reset the index to 0
                      endif
            otherwise
              4.5.3   Display error message and ask user to enter a valid age
              4.5.4   Read in the new age as a string
              4.5.5   Reset the index to 0
            endif
```

Now you should try to write the Java program that will use the pseudocode solution above as a basis to validate a person’s age. Be careful with step 4.4.2 of the pseudocode, you will have to do something extra (a test) at this point to prevent a potential runtime error from occurring. The program might run as indicated in the following sample screenshot:



```
Please enter the age of the person: apple
Invalid age entered - please enter a valid age: 123abd
Invalid age entered - please enter a valid age: -1289
Invalid age entered - please enter a valid age: -1
Invalid age entered - please enter a valid age: abc123
Invalid age entered - please enter a valid age: 141
Invalid age entered - please enter a valid age: 38

You have entered a valid age value. The value entered is 38
```

ACTIVITY

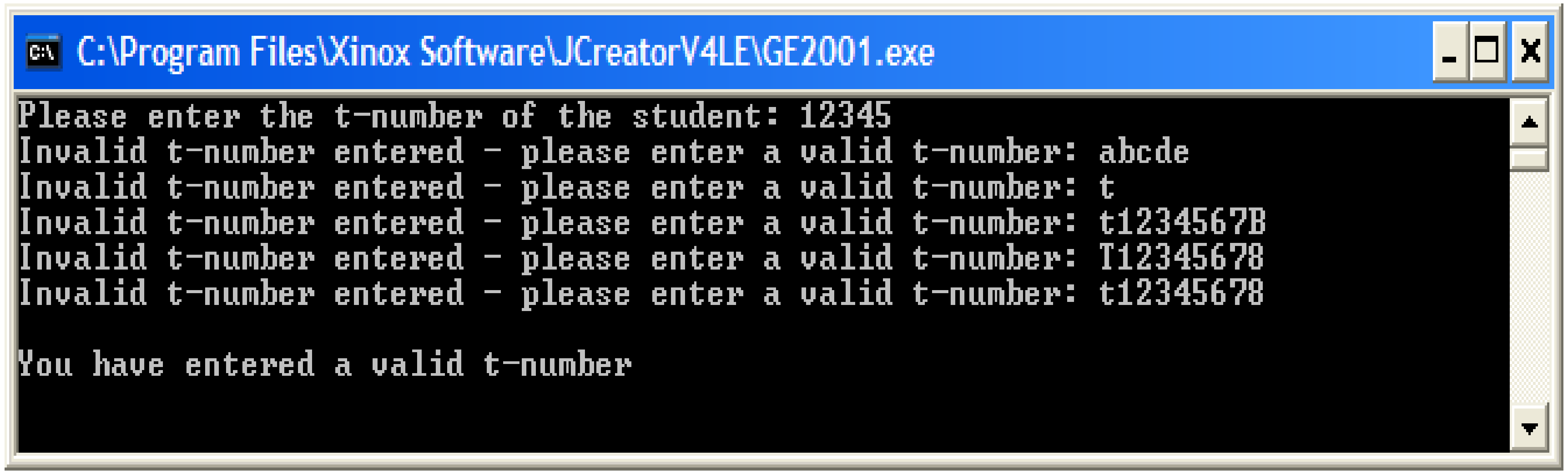
Write a 1st draft of pseudocode which will prompt for and read in a student's t-number and validate it perfectly. The t-number should have exactly 9 characters and begin with the letter 't'. Also the last 8 characters must all be digits.

Solution

```
1.Prompt for the user's t-number
2.Read in the user's t-number
3.while (first character in the t-number is not a 't') or
        (t-number does not contain 9 characters) or
        (last 8 characters of the t-number are not digits)
    3.1  Display an "Invalid t-number" error message
    3.2  Prompt for the user's t-number
    3.3  Read in the user's t-number
  endwhile
```

- From a programming perspective, various predefined functions/methods normally exist within the language to allow us to do things like determine the first character in the t-number, to find the number of characters in the t-number and check to see whether a particular character is a digit or not. You have met some of these functions in Java already.

Now you should try to write the Java program that will use the pseudocode solution above to validate a students t-number. Use the solution you wrote to the age validation question as a guide also. The program might run as indicated in the following sample screenshot:



```
Please enter the t-number of the student: 12345
Invalid t-number entered - please enter a valid t-number: abcde
Invalid t-number entered - please enter a valid t-number: t
Invalid t-number entered - please enter a valid t-number: t1234567B
Invalid t-number entered - please enter a valid t-number: T12345678
Invalid t-number entered - please enter a valid t-number: t12345678

You have entered a valid t-number
```

Although the solutions above to the age and t-number validations are very good efforts there is still an imperfection in them – see this by entering nothing when you run the programs. Don’t worry about this for now though, we will cover input validation in much more detail in semester 2 and then we will concentrate on perfecting our solutions.

Task-controlled Looping – understanding the problem

- Imagine that we need to write an algorithm that involves finding the smallest positive integer whose square exceeds 10 million and displaying it.

- Note that this problem statement tells us a few important things.
  - We know that there is no user input in this program which is quite unusual
  - We know that we want to find a positive whole number
  - We know that the square of the number we want to find exceeds 10 million
  - We know that in terms of output, there is only 1 value to be displayed

- Students typically dislike this kind of problem. For some, it is because it has a mathematical appearance. For most it is because they don't know where to start.

- In fact, it is a very short problem with few actions.

Task-controlled Looping – devising the plan

- A good way to approach this type of problem is to give it some thought. Think about what is being asked for again – “the smallest positive whole number whose square exceeds 10 million”.
- In this type of question you should plug in some values and see where it might take you:
  - 0 is the smallest positive integer and its square is 0
  - 1 is the next smallest positive integer and its square is 1
  - 2 is the next smallest positive integer and its square is 4
- We’re a long way from 10 million but we are definitely heading in the right direction!
- So, we should try to put our idea in pseudocode form

Task-controlled Looping – executing the plan

- A first draft here might be:

```
1. Initialise the value of targetNumber to 0
2. while (targetNumber squared ≤ 10 million)
      2.1  increase targetNumber by 1
   endwhile
3. display targetNumber
```

- This first draft looks pretty good. Now you should assess the solution and make sure it works according to plan – again plug in values to check it out.

- In a case like this where the test value is so large, you should reduce it to something more reasonable, say 55, and see if the algorithm works for this smaller value.

Task-controlled Looping - assessing the solution

- Set targetNumber to 0. Here the square is also 0 and this is ≤ 55, so we enter the loop. Then we increase the value of targetNumber to 1, perform the loop test condition again. Now 1 squared is 1 and this is still ≤ 55. Next, we increase targetNumber to 2, and do the loop test condition again. Now 2 squared is 4 and this is still ≤ 55. We continue on in the same manner as follows:

targetNumber	targetNumber Squared	≤ 55 ?
3	9	yes
4	16	yes
5	25	yes
6	36	yes
7	49	yes
8	64	no

- targetNumber does not get increased in the last case so 8 is our final result. This is the smallest integer whose square exceeds 55.

ACTIVITY

The mathematical series $1^2+2^2+3^2+4^2+$ ……..
goes on forever. Write an algorithm that finds the term in the series that puts the sum over 1000 and displays this term number along with the sum at that stage.

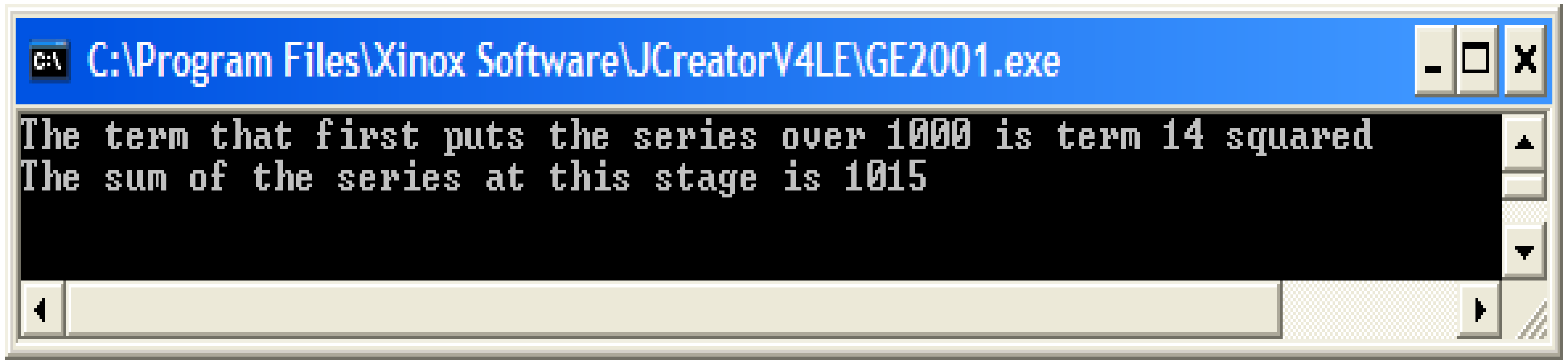
Note here that term 1 is 1^2, term 2 is 2^2 etc.

Solution

```
1. Initialise the value of termNumber to 0
2. Initialise the value of seriesSum to 0
2. while (seriesSum ≤ 1000)
      2.1  increase termNumber by 1
      2.2  set seriesSum to seriesSum + $(termNumber)^2$
   endwhile
3. display termNumber
4. display seriesSum
```

Both the activity above and the example earlier are both examples of what are often referred to as task-controlled iteration. In such cases, the user normally has no control over the termination of the loop (it is completely different from a data-sentinel controlled loop in that respect) and they are indeterminate (similar to data-sentinel in this respect).

Now you should try to write the Java program that will use the pseudocode solution above to solve the mathematical series problem. The program might run as indicated in the following sample screenshot:



```
The term that first puts the series over 1000 is term 14 squared
The sum of the series at this stage is 1015
```