APS105F: Computer Fundamentals
Lab Quiz, November 22, 1996

Instructions

- Place your student card on the desk top, so that it can be checked by the examiner.
- The only aid allowed is an original edition of the "Turing Reference Manual", the "On-line Turing Reference Manual", or other book that has a manual for Turing. You may not bring or use a photocopy of that manual, any other books, notes, written materials, or a calculator. You may reuse any programs in your own directories in building your program for this quiz.
- You must not communicate in any way (oral, written, or electronic) with anyone other than the examiner and his or her assistants. You should only talk to them if you believe there is an error or other problem in the program specifications. They will not give you any advice.
- There are three parts to this exam. The answer to each part will consist of a Turing program which satisfies the specifications given below. You will compose these programs in files called "`part1.t`", "`part2.t`", and "`part3.t`", respectively.
- **Insert your name and student number as comments at the beginning of each file.**
- You will submit these programs for grading by typing the commands "`SUBMITPART1`", "`SUBMITPART2`", and "`SUBMITPART3`", respectively. Note that these commands are Unix commands and they are typed as one word (no blanks) in block capitals. If you are in "oot", you must save your Turing file and exit to UNIX before running this command. The command will check that you have an appropriate file in your current directory.
- You may submit revised versions of your programs at any time before the end of the quiz by again typing the submit command. However, if you run this command before your quiz starts, or after the examiner has announced your quiz is over, you will receive a grade of zero.
- You may use the back of the pages for your rough work and pseudo-code design.
- The examiner will announce when there are 10 minutes and 5 minutes remaining in the quiz. It is your responsibility to submit your program before the end of the two hour quiz period. It is strongly advised that you do not leave this submission to the last minute.
- It is highly recommended that you save the file you are working on periodically; i.e. as soon as you have typed more than you are willing to type in again.
- Marks will be taken off for **poor programming style** and **obviously inefficient algorithms**.
- Complete and sign this sheet, and hand in the complete paper at the end of the quiz.

Name (print)______________________________

Login ID (print)______________________________

Student Number______________________________

Signature______________________________

Marks:	1 [5]		2 [6]		3 [4]		Total [15]	

1 Printing Words

You are to complete a program that reads in a number of words as strings, such as a sentence, and then outputs the words in different orders. There are two procedures, called `ForwardPrint` and `ReversePrint`, that do the printing. Each procedure takes three parameters. The last parameter is the array of strings. The first two parameters indicate the part of the string array to print, i.e., print the strings from Strings(StartIndex) to Strings(EndIndex) or in reverse, depending on the procedure.

What to do

Copy the file "`/usr/copy/aps105/revstring.t`" into your working directory and rename it "`part1.t`". A listing of this file is given in Appendix A starting on page 5.

Complete the two procedures `ForwardPrint` and `ReversePrint`. You are to write `ForwardPrint` as a **recursive** procedure and `ReversePrint` **using a loop** in the procedure.

Assumptions

Assume that only legal inputs are entered.

Restrictions

You should only write the code for the procedures. Do not change any other part of the program when you submit your solution, though you are free to make modifications while testing. Just make sure that the program you submit has the original code.

Example Input Sequence

An example of the input sequence is:

```
Input String, end with 'end'
Be careful driving elephants in small flimsy garages. end
```

2 A Simple Spread Sheet

The program shown in Appendix B implements a very simple spread sheet based on an array of real numbers. There are two procedures that need to be completed. They are `SumARow` and `Compute`. The rest of the program will read in an array of numbers and do some computation on the elements (values) in the array using those procedures.

What to do

Copy the file "`/usr/copy/aps105/spread.t`" into your working directory and rename it "`part2.t`". A listing of this file is given in Appendix B on page 6.

Complete the two procedures `SumARow` and `Compute` according to the specifications given in the **More Information** section below.

Assumptions

No checking is done. For the existing main program, an array with 4 rows and 5 columns will work without error.

Restrictions

You should only write the code for the procedures. Do not change any other part of the program when you submit your solution, though you are free to make modifications while testing. Just make sure that the program you submit has the original code.

More Information

For the procedure `SumARow`, the order of the parameters is `RowNumber`, `StartCol`, `EndCol`, `DestCol`, and the 2-dimensional array. The procedure does the following: In row `RowNumber`, sum the values in the columns from `StartCol` to `EndCol` and put the result in `DestCol`.

For the procedure `Compute`, the order of the parameters is `Row1`, `Row2`, `Row3`, `RowR`, `StartCol`, `EndCol`, and the 2-dimensional array. The procedure does the following: For each column from `StartCol` to `EndCol`, inclusive, compute the function

```
A(RowR) := A(Row1) + A(Row2) * A(Row3)
```

where `A(x)` represents the element in column `x` of the row.

Example Input Sequence

An example of the input sequence is:

```
Number of rows?
4
Number of columns?
5

Input the elements by rows
Row 1:
1 2 3 4 5
Row 2:
6 7 8 9 10
Row 3:
11 12 13 14 15
Row 4:
16 17 18 19 20
```


3 Towards A Better Spread Sheet

As those of you who have used real spread sheet calculators will know, each element or *cell* in the spread sheet can hold an equation, a number, or some text. It is also possible to insert and delete rows and columns. This means that we will need to use records and some sort of data structure. A simple start is to have each row be a **doubly-linked** list of cells, where the cells are ordered according to the column number.

What to do

Copy the file "`/usr/copy/aps105/linkeda.t`" into your working directory and rename it "`part3.t`". A listing of this file is given in Appendix C. It is unlikely that you will need to understand all of it. Just realize that you are dealing with a simple doubly-linked list. If you do have problems, they will show up in the `PrintRow` procedure, which is described in more detail below.

Your task is to complete the `DeleteCell` procedure. Output an error message if the cell being deleted does not exist.

Assumptions

Inputs are always in the correct format and sequence.

Restrictions:

You should only write the code for the procedure. Do not change any other part of the program when you submit your solution, though you are free to make modifications while testing. Just make sure that the program you submit has the original code.

More Information

Each cell has the following fields:

```
Field                Use
-------------------------------------------------
Left            Pointer to the cell to the left
Right           Pointer to the cell to the right
Column          An int indicating the column that the cell is in
Value           A real number holding the value of the cell
HasValue        A boolean that is true when there is a number in Value
```

The use of the `Column` field means that it is not necessary to include a cell in the list if there is nothing in the cell for the corresponding column. The `HasValue` field is used to indicate whether the `Value` field has been assigned anything.

The `PrintRow` procedure prints the row both in the forward and reverse directions to check that all the links are correct. Note that the procedure assumes that the `Right` link of the cell at the right end of the row and the `Left` link of the left-end cell are set to `nil`.

Example Session

An example session is shown below. Note that the program first inputs a number of cells. At this point, it is assumed that the column numbers are **entered in increasing order**. A "`y`" response to the "`Value?`" prompt is asking whether the cell is to store a value. If you respond with "`y`", then you will be asked for the value. This first part allows for building a list quickly that you can then work on.

You are then asked whether you wish to delete a cell or not. After each deletion, the list is printed again. Remember to check all of the cases!

```
Enter a bunch of cells, exit with a Column Number < 1
Column? 3
Value? (y/n) n
Column? 7
Value? (y/n) y
Enter value 14
Column? 9
Value? (y/n) y
Enter value 18
Column? -1

[(3) No Value] --> [(7) 14] --> [(9) 18]
[(9) 18] --> [(7) 14] --> [(3) No Value]

Delete a cell? (y/n) y
Column? 7

[(3) No Value] --> [(9) 18]
[(9) 18] --> [(3) No Value]

Delete a cell? (y/n) n
```

A Listing for revstring.t

```
procedure ReversePrint(StartIndex, EndIndex: int, Strings : array 1 .. * of string)
        % ... your code here using some kind of loop
end ReversePrint

procedure ForwardPrint(StartIndex, EndIndex : int, Strings : array 1 .. * of string)
        % ... your code here using recursion
end ForwardPrint

var StringArray : array 1 .. 100 of string

var i : int := 0
var s : string

put "Input String, end with 'end' "

loop
        get s
        exit when s = "end"
        i := i + 1
        StringArray(i) := s
end loop

ForwardPrint(1,i,StringArray)
put ""
ReversePrint(1,i,StringArray)
put ""

ForwardPrint(1,i div 2, StringArray)
ReversePrint((i div 2) + 1, i,StringArray)
```


B Listing for spread.t

```
procedure SumARow(RowNumber,StartCol,EndCol,DestCol : int,
                        var DataArray : array 1 .. * , 1 .. * of real)
  %
  % Your code here
  %
end SumARow

procedure Compute(Row1,Row2,Row3,RowR : int,StartCol,EndCol : int,
                         var DataArray : array 1 .. * , 1 .. * of real)
  %
  % Your code here
  %
end Compute

procedure GetArray(Nrows,Ncols : int,
            var DataArray : array 1 .. * , 1 .. * of real)
    put "Input the elements by rows "

    for row : 1 .. Nrows
        put "Row ",row,": "
        for col : 1 .. Ncols
            get DataArray(row,col)
        end for
    end for
end GetArray

procedure PrintArray(Nrows,Ncols : int,
            DataArray : array 1 .. * , 1 .. * of real)
    for row : 1 .. Nrows
        for col : 1 .. Ncols
            put DataArray(row,col):8:3, " "..
        end for
        put ""
    end for
end PrintArray

var Nrows, Ncols : int

put "Number of rows? "
get Nrows
put "Number of columns? "
get Ncols

var Sheet : array 1 .. Nrows, 1 .. Ncols of real

put ""

GetArray(Nrows,Ncols,Sheet)
PrintArray(Nrows,Ncols,Sheet)

put ""
```

```
SumARow(2,1,3,4,Sheet)
PrintArray(Nrows,Ncols,Sheet)

put ""

SumARow(3,2,4,1,Sheet)
PrintArray(Nrows,Ncols,Sheet)

put ""

Compute(1,2,3,4,1,3,Sheet)
PrintArray(Nrows,Ncols,Sheet)
```

C Listing for linkeda.t

```
type cell:
  record
    Left : pointer to cell
    Right : pointer to cell
    Column : int
    Value : real
    HasValue : boolean
  end record

procedure DeleteCell(var RowPtr : pointer to cell, ColNum : int)
  % Your code here
  %   RowPtr is the pointer to the head (left end) of the list.
  %   ColNum is the number of the column to be deleted.
  % Print an error message for non-existent cells.
end DeleteCell

function GetCell() : pointer to cell

    var Col : int
    var Val : string
    var Cptr : pointer to cell

    put "Column? "..
    get Col
    if Col < 0 then
        result nil
    end if

    new Cptr
    Cptr->Column := Col

    put "Value? (y/n) "..
    get Val
    if Val = "y" then
        put "Enter value " ..
        get Cptr->Value
        Cptr->HasValue := true
```



```
    else
        Cptr->HasValue := false
    end if

    result Cptr
end GetCell

procedure GetRow(var RowPtr : pointer to cell)
    % Assume that cells are input in sequence and we just have to
    % add at the end of the row

    var Cptr : pointer to cell
    var Tptr : pointer to cell    % tail pointer

    put "Enter a bunch of cells, exit with a Column Number < 1"

    RowPtr := GetCell()
    RowPtr->Left := nil
    RowPtr->Right := nil
    Tptr := RowPtr

    loop
        Cptr := GetCell()
        exit when Cptr = nil
        Tptr->Right := Cptr
        Cptr->Left := Tptr
        Cptr->Right := nil
        Tptr := Cptr
    end loop
end GetRow

procedure PrintRow(RowPtr : pointer to cell)
    % Print front to back to front to check pointer connections

    var Cptr : pointer to cell

    Cptr := RowPtr

    if Cptr = nil then
        return
    end if

    put ""
    put ""

    loop    % go forward
        put "[(",Cptr->Column,") "..
        if Cptr->HasValue then
            put Cptr->Value..
        else
            put "No Value"..
        end if
```

```
        if Cptr->Right = nil then
            exit
        else
            Cptr := Cptr->Right
            put "] --> "..
        end if
    end loop

    put "]"
    put ""

    loop    % go backward
        put "[(",Cptr->Column,") "..
        if Cptr->HasValue then
            put Cptr->Value..
        else
            put "No Value"..
        end if

        if Cptr->Left = nil then
            exit
        else
            Cptr := Cptr->Left
            put "] --> "..
        end if
    end loop

    put "]"
    put ""

end PrintRow

% Main routine

var RowPtr : pointer to cell
var ColNum : int
var Ans : string

% Initialize the row first

GetRow(RowPtr)
PrintRow(RowPtr)

loop    % command loop
    put "Delete a cell? (y/n) "..
    get Ans
    exit when Ans not= "y"

    put "Column? "..
    get ColNum
    DeleteCell(RowPtr,ColNum)
    PrintRow(RowPtr)
end loop
```


Please:

- log on to the system
- start up oot
- do not turn this over until you are told to do so.