ECU178 Computer Science:
210CT - Programming, Algorithms and Data Structures Portfolio

Due on Monday, December 15th, 2014

Dr James Shuttleworth

Robert Rigler : 4939377

Contents

Item 1: Week 3 - Linear Search and Duplicate Finder

1. Pseudocode for linear search

This Simple Algorithm demonstrates how to perform a linear search.
Input: This algorithm takes a populated array A and a value to search for v, as parameters.
Output: The Algorithm is a boolean type and returns either True or False respective of whether the v was found in the list or not.

Algorithm 1 LinearSearch

```
procedure BOOL LINEARSEARCH(v, A[ ])
    for each element i in A do
        if A[i] = v then
            return true
        end if
    end for
return false
end procedure
```

2. Pseudocode for finding duplicates in a list

This algorithm demonstrates how to examine if a list has duplicate entries using a linear search.
Input: This algorithm takes a populated array A as a parameter.
Output: This Algorithm is a boolean type and returns true or false respective of whether a duplicate value is found or not.

Algorithm 2 Examining for duplicates

```
procedure BOOL EXFORDUPES(A[ ])
    for each element i in A[ ] do
        for each element j in A[ ] do
            if A[i] = A[j] then
                return true
            end if
        end for
    end for
end procedure
```

Item 2: Week 4 - Time complexities and Big-O notation

1. Describe the runtime bounds of the linear search algorithm

Algorithm 3 LinearSearch

```
procedure BOOL LINEARSEARCH(item, list[ ])

    for each element i in list do          (n)
        if list[i] = list then t            (n)
            return true                     (n)
        end if
    end for
return false                                (1)
end procedure
```

Collecting the line-by-line runtime data from the algorithms gives: $n + n + n + 1$ which is equivalent to: $3n + 1$.
Therefore the time complexity of the algorithm is O(n).

2. Describe the runtime bounds of the duplicate finder algorithm

Algorithm 4 Examining for duplicates

```
procedure BOOL EXFORDUPES(list[ ])
    for each element i in list[ ] do                (n)
        for each element j in list[ ] do            (n*n)
            if list[i] = list[j] then               (n*n)
            return true                             (n*n)
            end if
        end for
    end for
return false                                        (1)
end procedure
```

Collecting the line-by-line runtime data from the algorithms gives: $n + (n * n) + (n * n) + (n * n) + 1$ which is equivalent to: $3n^2 + n + 1$.
Therefore the time complexity of the algorithm is O(n^2)

Item 2: Week 4 - Time complexities and Big-O notation continued on next page

Additional work: Critical values of relative runtimes

Write a function that determines the critical value at which the relative runtime of two linear algorithms swap.
For this algorithm, I am assuming that $k1 > k2$ (Expression 1 >Expression 2, when $n = 0$).
The Algorithm is very simple; While the value of Expression 1 is greater than Expression 2, increase the value of n.
When the Runtime of the algorithms swap, the while-loop exit condition is fulfilled and the current value of n is returned.

Algorithm 5 Relative runtime comparison algorithm

```
procedure CRITVAL(m1, k1, m2, k2)
    while (m1 * n + k1)>(m2 * n + k2) do
        n++
    end while
return n

end procedure
```

Item 3: Week 6 - Harmonic Series

1. Harmonic Series (Pseudocode)

Use pseudocode to specify a recursive algorithm to compute the nth value of the harmonic series, for some integer n.

The Harmonic series is as follows: $1 + 1/2 + 1/3 + 1/4 + 1/5 + ..1/n$
Input: This algorithm takes two parameters t and n which are the total sum of the algorithm and the number of repetitions, respectively.
Output: This algorithm outputs the value t which is the total sum of the harmonic series.

This procedure uses a while-loop to control the number recursive iterations.
While the number of iterations left is above 0, add the next value to t, decrease the number of iterations by 1 and recursively call the procedure with the new values of t and n.
When the number of iterations left is no longer above 0, the final value of t is returned and the procedure ends.

Algorithm 6 Computing nth value of harmonic series

```
procedure HARM(float t, float n)
    while n > 0 do
        t ← t + (1/n)
        n − −
        HARM(t, n)
    end while
return t
end procedure
```

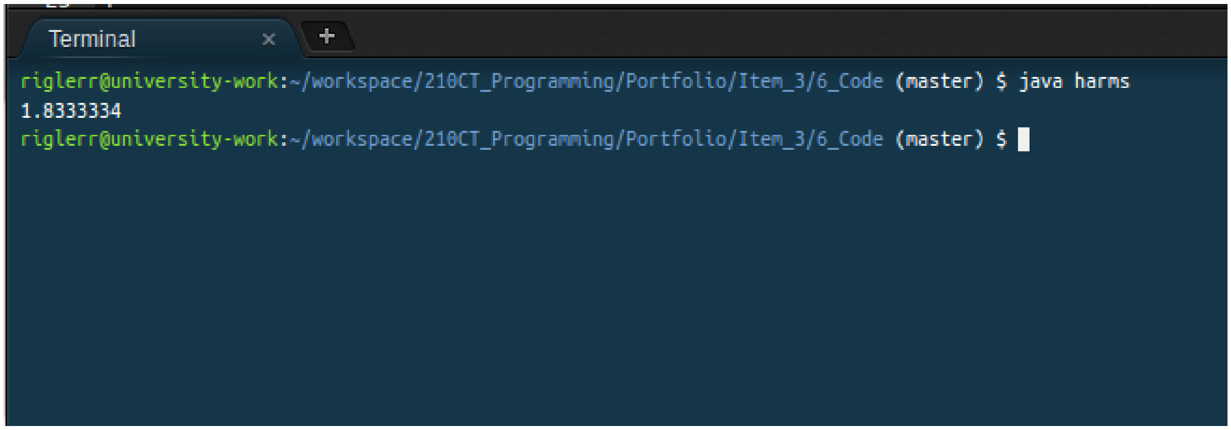
2. Harmonic Series (JAVA Implementation)

The Harmonic Series computation algorithm implemented in Java

Listing 1: harms java class file

```java
public class harms{

    public static void main(String[] args){
        /**/
        System.out.println(f(0,3));
    }

    public static float f(float t, float n){
        /*
        t always has a value of 0 on the initial method call.
        n is the nth term, which decreases by 1 each recursive call.

        When n = 0, stop recursive calling and return the value t.
        */
        while (n>0)
        {
            t+= (1/n);
            f(t,--n);
        }
        return t;
    }


}
```

Evidence of the Harmonic Series computation java implementation.
The nth value passed to the method was 3.



Item 4: Week 7 - RPN Calculator

1. Reverse Polish Notation Calculator

To implement this calculator I created three distinct classes:

- **InputString**: Which handles the string operations, such as: splitting the users string into an array.
- **RPN**: This class evaluates the input string and returns an answer.
- **MyStack**: This is the stack class that is capable of Pushing, Poping and Displaying values on the stack.
- **MathOps**: This class handles the mathematical operations.

Item 5: Week 8 - Linked List

1. Linked List Implementation

Below are the Linked List and Node classes that I have created as well as a Test class. This is a fully functional doubly Linked List capable of Inserting, Prepending, Deleting and Displaying items in the List. Please see the Evidence Screenshot below:

Listing 2: List Class Implementation: Java

```java
package com.company;
public class List {
      //Nodes to hold the head and tail of the list respectively
    Node head;
    Node tail;
    //-----------------------------------------------------------------
    //-----------List Constructor to init values to null----------
    //-----------------------------------------------------------------
    public List(){
        head = null;
        tail = null;
    }
    //-----------------------------------------------------------------
    //----Insert an Element After the specied element (N)---------
    //-----------------------------------------------------------------
    public void Insert(Node n, Node x){
        //If previous node exists
        if(n != null){
            //sWAP NODE POINTERS
            x.next = n.next;
            n.next = x;
            x.prev =n;

            // If new node isn't end of list
            if(x.next != null)
                x.next.prev =x;
        }
        //Prepend to existing list
        if (head != null & n == null) {
            n = head;
            n.prev = x;
            x.next = n;
            x.prev = null;
            head = x;
        }
```

```



        // If new list (First item to be placed)
       else  if(n == null & head == null)
        {
    //make new node equal to both head and tail
             head = tail =x;
             x.prev =x.next = null; // make head pointers null
        } //If all else exp
        else if(tail == n)
             tail =x;
    }
    //------------------------------------------------------------
    //---------------DELETE A NODE FROM THE LIST---------------
    //------------------------------------------------------------
    public void Delete(Node n){
        //create temporary node
        Node i = head;

        while (i != null){ // Loop until null (tail.next)

            if (i.value == n.value){

                if(i == head){ //If node to remove is head node
                    head = i.next;
                    i.prev = null;
                    break;
                }

          //if node to remove is tail node
                else if (i == tail){
                    tail = i.prev;
                    i.prev.next = null;
                }
                else { // If node to remove is not Head/Tail
                i.prev.next = i.next;
                i.next.prev = i.prev;
                break;}
            }
            i=i.next; // Increment
        }
    }
```

```java



    //--------------------------------------------------------
    //-----------DISPLAY ALL NODES IN LIST------------------
    //--------------------------------------------------------
    public void display(){
        Node i = head;

        while(i != null) {
            System.out.print(i.value + ", ");
            i = i.next;
        }
    }


}
```

Listing 3: Node Class Implementation: Java

```java
package com.company;

/**
 * Created by rob on 20/12/14.
 */

public class Node {
    int value;
    Node prev;
    Node next;

    public Node(int val){

            //Node Constructor
        value = val;
        next =null;
        prev =null;

    }


}
```

Listing 4: Test class

```java
package com.company;

public class Main {

    public static void main(String[] args) {
        List obj1 = new List();


        obj1.Insert(null, new Node(42));
        obj1.Insert(obj1.head, new Node(32));
        obj1.Insert(obj1.head.next, new Node(102));

        //------------------------------------------------------
        //----------------Test Inserting Tail-----------------
        //------------------------------------------------------
        System.out.println("Insert 3 items:");
        obj1.display();
        System.out.println();

        // ------------------------------------------------------
        //-----------------Test Insert Between-----------------
        //-------------------------------------------------------
        //Insert 12 between the head node and the head.next
        System.out.println("Insert 12 after head node: ");
        obj1.Insert(obj1.head, new Node(12));
        obj1.display();
        System.out.println();

        //-------------------------------------------------------
        //--------------------Test Prepend---------------------
        //-------------------------------------------------------
        //Insert 13 at start of pre-existing list
        System.out.println("Prepend 13: ");
           obj1.Insert(null,new Node(13));
           obj1.display();
          System.out.println();
```

```
        //---------------------------------------------------------
        //------------------Test Delete Head--------------------
        //---------------------------------------------------------
        //Delete the head node '13' from the list
        System.out.println("Delete Head: ");
        obj1.Delete(new Node(13));
        obj1.display();
        System.out.println();

        //----------------------------------------------------------
        //------------------Test Delete Norm---------------------
        //----------------------------------------------------------
        //Delete a middle node (32) from list an display
        System.out.println("Delete a Middle Node (32)");
        obj1.Delete(new Node(32));
        obj1.display();
        System.out.println();

        //--------------------------------------------------------
        //----------------Test Delete Tail---------------------
        //--------------------------------------------------------
        //Delete the tail node '42' from list an display
        System.out.println("Delete tail: ");
        obj1.Delete(new Node(102));
        obj1.display();
        System.out.println();


    }}
```

Evidence of working linked list using the Test class shown above:

```
/usr/lib/jvm/java-8-oracle/bin/java ...
Insert 3 items:
42, 32, 102,
Insert 12 after head node:
42, 12, 32, 102,
Prepend 13:
13, 42, 12, 32, 102,
Delete Head:
42, 12, 32, 102,
Delete a Middle Node (32)
42, 12, 102,
Delete tail:
42, 12,
```