AVLWindow (Window)

Constants
Integer cNr = 1000

AVLWindow.PB_Start.Action:
```
Sub Action()
  dim t as AVLTree
  dim value as integer
  dim values() as integer
  dim key as string
  dim found as integer

  t=new AVLTree
  if t=nil then
    MsgBox("Out of memory.")
    return
  end

  for value=0 to cNr
    values.Append(value)
  next

  values.Shuffle()

  for each value in values
    key=Str(value)
    if not t.Insert(key,value) then
      MsgBox("Insert fails.")
      return
    end
    if not t.Verify() then
      MsgBox("Verify fails.")
      return
    end
    if not t.Find(key,found) then
      MsgBox("Find fails.")
      return
    end
    if found<>value then
      MsgBox("found<>value.")
      return
    end
  next

  values.Shuffle()

  for each value in values
```

```
    key=Str(value)
    if not t.Find(key,found) then
      MsgBox("Find fails.")
      return
    end
  next

  values.Shuffle()

  for each value in values
    key=Str(value)
    if not t.Find(key,found) then
      MsgBox("Find fails.")
      return
    end
    if found<>value then
      MsgBox("found<>value.")
      return
    end
    if not t.Remove(key) then
      MsgBox("Remove fails.")
      return
    end
    if not t.Verify() then
      MsgBox("Verify fails.")
      return
    end
  next

  MsgBox("Everything OK.")
End Sub
```

AVLTree (Class)

Properties

m_top as AVLNode

AVLTree.Verify:

```
Function Verify() As boolean
  if m_top=nil then
    return true
  end

  if not m_top.Verify() then
    MsgBox("Verify top fails.")
    return false
  end

  return true
End Function
```

AVLTree.Find:

```
Function Find(key as string,byref value as integer) As boolean
  dim t as integer

  if m_top=nil then
    return false
  end

  if not m_top.Find(key,value) then
    return false
  end

  return true
End Function
```

AVLTree.Insert:

```
Function Insert(key as string,value as integer) As boolean
  if m_top=nil then
    m_top=new AVLNode(key,value)
    if m_top=nil then
      MsgBox("Out of memory.")
      return false
    end
    return true
  end
```

```
  if not AVLUtils.Insert(m_top,key,value) then
    MsgBox("Insert fails.")
    return false
  end

  return true
End Function
```

AVLTree.Remove:

```
Function Remove(key as string) As boolean
  if m_top=nil then
    MsgBox("The key wasn't found.")
    return false
  end

  if not AVLUtils.Remove(m_top,key) then
    MsgBox("Remove fails.")
    return false
  end

  return true
End Function
```

AVLNode (Class)

Properties

m_left as AVLNode

m_right as AVLNode

m_balance as integer

m_key as string

m_value as integer

m_height as integer

AVLNode.Verify:

```
Function Verify() As boolean
  dim height as integer
  dim balance as integer

  CalcHeightAndBalance(height,balance)

  if height<>m_height then
    MsgBox("Height is different.")
    return false
  end

  if balance<>m_balance then
    MsgBox("Balance is different.")
    return false
  end
  if balance<-1 then
    MsgBox("Balance < -1.")
    return false
  end
  if balance>1 then
```

```
    MsgBox("Balance > 1.")
    return false
  end

  if m_left<>nil then
    if m_key<=m_left.m_key then
      MsgBox("m_key<=m_left.m_key.")
      return false
    end
    if not m_left.Verify() then
      MsgBox("Verify left fails.")
      return false
    end
  end

  if m_right<>nil then
    if m_key>=m_right.m_key then
      MsgBox("m_key>=m_right.m_key.")
      return false
    end
    if not m_right.Verify() then
      MsgBox("Verify right fails.")
      return false
    end
  end

  return true
End Function
```

AVLNode.CalcHeightAndBalance:

```
Private Sub CalcHeightAndBalance(byref height as integer,byref balance as integer)
  dim left_height as integer
  dim right_height as integer

  if m_left<>nil then
    left_height=m_left.m_height
  end

  if m_right<>nil then
    right_height=m_right.m_height
  end

  if left_height>right_height then
    height=1+left_height
  else
    height=1+right_height
  end

  balance=right_height-left_height
```

```
End Sub
```

AVLNode.UpdateHeightAndBalance:

```
Sub UpdateHeightAndBalance()
  CalcHeightAndBalance(m_height,m_balance)
End Sub
```

AVLNode.Find:

```
Function Find(key as string,byref value as integer) As boolean
  if m_key=key then
    value=m_value
    return true
  end

  if key<m_key then
    if m_left<>nil then
      if m_left.Find(key,value) then
        return true
      end
    end
  else
    if m_right<>nil then
      if m_right.Find(key,value) then
        return true
      end
    end
  end

  return false
End Function
```

AVLNode.Constructor:

```
Sub Constructor(key as string,value as integer)
  m_height=1
  m_key=key
  m_value=value
End Sub
```

AVLNode.Get_key:

```
Function Get_key() As string
  return m_key
End Function
```

AVLNode.Get_left:

```
Function Get_left() As AVLNode
  return m_left
End Function
```

AVLNode.Set_left:

```
Sub Set_left(left as AVLNode)
  m_left=left
End Sub
```

AVLNode.Get_balance:

```
Function Get_balance() As integer
  return m_balance
End Function
```

AVLNode.Get_right:

```
Function Get_right() As AVLNode
  return m_right
End Function
```

AVLNode.Set_right:

```
Sub Set_right(right as AVLNode)
  m_right=right
End Sub
```

AVLNode.Copy:

```
Sub Copy(k as AVLNode)
  m_key=k.m_key
  m_value=k.m_value
End Sub
```

AVLUtils (Module)

Properties

m_chooseLeft as boolean

AVLUtils.Insert:

```
Protected Function Insert(byref top as AVLNode,key as string,value as integer) As boolean
  dim top_key as string
  dim balance as integer
  dim left as AVLNode
  dim right as AVLNode

  top_key=top.Get_key()

  if key=top_key then
    MsgBox("The key was found.")
    return false
  end

  if key<top_key then
    left=top.Get_left()
    if left<>nil then
      if not Insert(left,key,value) then
        MsgBox("Insert fails.")
        return false
      end
    else
      left=new AVLNode(key,value)
      if left=nil then
        MsgBox("Out of memory.")
        return false
      end
    end
    top.Set_left(left)
    top.UpdateHeightAndBalance()
    balance=top.Get_balance()
    if balance<-1 then
      BalanceLeft(top)
    end
  else
    right=top.Get_right()
    if right<>nil then
      if not Insert(right,key,value) then
        MsgBox("Insert fails.")
        return false
```

```
      end
    else
      right=new AVLNode(key,value)
      if right=nil then
        MsgBox("Out of memory.")
        return false
      end
    end
    top.Set_right(right)
    top.UpdateHeightAndBalance()
    balance=top.Get_balance()
    if balance>1 then
      BalanceRight(top)
    end
  end

  return true
End Function
```

AVLUtils.BalanceLeft:

```
Private Sub BalanceLeft(byref k as AVLNode)
  dim left as AVLNode
  dim left_balance as integer

  left=k.Get_left()
  left_balance=left.Get_balance()
  if left_balance<0 then
    RotateToTheRight(k)
  elseif left_balance>0 then
    RotateToTheLeft(left)
    k.Set_left(left)
    RotateToTheRight(k)
  end
End Sub
```

AVLUtils.BalanceRight:

```
Private Sub BalanceRight(byref k as AVLNode)
  dim right as AVLNode
  dim right_balance as integer

  right=k.Get_right()
  right_balance=right.Get_balance()
  if right_balance>0 then
    RotateToTheLeft(k)
  elseif right_balance<0 then
    RotateToTheRight(right)
    k.Set_right(right)
    RotateToTheLeft(k)
  end
```

```
End Sub
```

AVLUtils.RotateToTheLeft:

```
Private Sub RotateToTheLeft(byref k as AVLNode)
  dim r as AVLNode
  dim left as AVLNode
  dim right as AVLNode

  r=k
  k=k.Get_right()
  left=k.Get_left()
  r.Set_right(left)
  k.Set_left(r)

  left=k.Get_left()
  left.UpdateHeightAndBalance()

  right=k.Get_right()
  if right<>nil then
    right.UpdateHeightAndBalance()
  end

  k.UpdateHeightAndBalance()
End Sub
```

AVLUtils.RotateToTheRight:

```
Private Sub RotateToTheRight(byref k as AVLNode)
  dim r as AVLNode
  dim left as AVLNode
  dim right as AVLNode

  r=k
  k=k.Get_left()
  right=k.Get_right()
  r.Set_left(right)
  k.Set_right(r)

  right=k.Get_right()
  right.UpdateHeightAndBalance()

  left=k.Get_left()
  if left<>nil then
    left.UpdateHeightAndBalance()
  end

  k.UpdateHeightAndBalance()
End Sub
```

AVLUtils.Remove:

```
Protected Function Remove(byref top as AVLNode,key as string) As boolean
  dim top_key as string
  dim left as AVLNode
  dim right as AVLNode
  dim balance as integer

  top_key=top.Get_key()
  if top_key=key then
    left=top.Get_left()
    right=top.Get_right()
    if right=nil then
      top=left
    elseif left=nil then
      top=right
    else
      balance=top.Get_balance()
      if balance<0 then
        RemoveMostRightOfLeft(top)
      elseif balance>0 then
        RemoveMostLeftOfRight(top)
      elseif m_chooseLeft then
        RemoveMostRightOfLeft(top)
        m_chooseLeft=false
      else
        RemoveMostLeftOfRight(top)
        m_chooseLeft=true
      end
    end
  elseif top_key<key then
    right=top.Get_right()
    if right=nil then
      MsgBox("The key wasn't found.")
      return false
    end
    if not Remove(right,key) then
      MsgBox("The key wasn't found.")
      return false
    end
    top.Set_right(right)
    UpdateAndBalance(top)
  else
    left=top.Get_left()
    if left=nil then
      MsgBox("The key wasn't found.")
      return false
    end
    if not Remove(left,key) then
      MsgBox("The key wasn't found.")
```

```
      return false
    end
    top.Set_left(left)
    UpdateAndBalance(top)
  end
  return true
End Function
```

AVLUtils.RemoveMostRightOfLeft:

```
Private Sub RemoveMostRightOfLeft(byref k as AVLNode)
  dim left as AVLNode
  dim left_right as AVLNode
  dim left_left as AVLNode

  left=k.Get_left()
  left_right=left.Get_right()
  if left_right=nil then
    k.Copy(left)
    left_left=left.Get_left()
    k.Set_left(left_left)
  else
    RemoveMostRight(k,left,left_right)
    k.Set_left(left)
  end
  UpdateAndBalance(k)
End Sub
```

AVLUtils.UpdateAndBalance:

```
Private Sub UpdateAndBalance(byref k as AVLNode)
  dim balance as integer
  dim left as AVLNode
  dim left_balance as integer
  dim right as AVLNode
  dim right_balance as integer

  k.UpdateHeightAndBalance()
  balance=k.Get_balance()
  if balance<-1 then
    left=k.Get_left()
    left_balance=left.Get_balance()
    if left_balance<=0 then
      RotateToTheRight(k)
    elseif left_balance>0 then
      RotateToTheLeft(left)
      k.Set_left(left)
      RotateToTheRight(k)
    end
  elseif balance>1 then
    right=k.Get_right()
```

```
    right_balance=right.Get_balance()
    if right_balance>=0 then
      RotateToTheLeft(k)
    elseif right_balance<0 then
      RotateToTheRight(right)
      k.Set_right(right)
      RotateToTheLeft(k)
    end
  end
End Sub
```

AVLUtils.RemoveMostRight:

```
Private Sub RemoveMostRight(byref k as AVLNode,byref prev as AVLNode,mostRight as AVLNode
            )
  dim mostRight_right as AVLNode
  dim mostRight_left as AVLNode

  mostRight_right=mostRight.Get_right()
  if mostRight_right=nil then
    k.Copy(mostRight)
    mostRight_left=mostRight.Get_left()
    prev.Set_right(mostRight_left)
  else
    RemoveMostRight(k,mostRight,mostRight_right)
    prev.Set_right(mostRight)
  end
  UpdateAndBalance(prev)
End Sub
```

AVLUtils.RemoveMostLeft:

```
Private Sub RemoveMostLeft(byref k as AVLNode,byref prev as AVLNode,mostLeft as AVLNode)
  dim mostLeft_left as AVLNode
  dim mostLeft_right as AVLNode

  mostLeft_left=mostLeft.Get_left()
  if mostLeft_left=nil then
    k.Copy(mostLeft)
    mostLeft_right=mostLeft.Get_right()
    prev.Set_left(mostLeft_right)
  else
    RemoveMostLeft(k,mostLeft,mostLeft_left)
    prev.Set_left(mostLeft)
  end
  UpdateAndBalance(prev)
End Sub
```

AVLUtils.RemoveMostLeftOfRight:

```
Private Sub RemoveMostLeftOfRight(byref k as AVLNode)
  dim right as AVLNode
```

```
  dim right_left as AVLNode
  dim right_right as AVLNode

  right=k.Get_right()
  right_left=right.Get_left()
  if right_left=nil then
    k.Copy(right)
    right_right=right.Get_right()
    k.Set_right(right_right)
  else
    RemoveMostLeft(k,right,right_left)
    k.Set_right(right)
  end
  UpdateAndBalance(k)
End Sub
```

AVLUtils.Parameters:

Parameters
You can't pass an expression as a byref parameter.
x.m_left is an expression.