COSC 1030 — Fall 1996, Jeff Edmonds, Midterm Solutions

1. Running Time

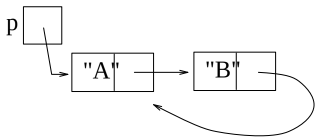
- Write a simple and natural non-recursive program that has running time $\mathcal{O}(nlogn)$. It should print "Hi" $n \log_2 n$ times. (Without using the math log function.)

```
for i : 1..n
  var j := n
  loop
    exit when j<=1
    put "Hi"
    j := j/2
  end loop
end for
```

- For the running times shown, give the associated time in Big O notation.
  - $5 \log_2 n$ $\qquad \mathcal{O}(\log n)$
  - $5 \times 2^{2n+3}$ $\qquad \mathcal{O}(4^n)$
  - $12 log_{10}(2n+3)$ $\qquad \mathcal{O}(\log n)$
  - $2^n + n^{1000}$ $\qquad \mathcal{O}(2^n)$
  - $1000 log_2 n + n^2 + 5n$ $\qquad \mathcal{O}(n^2)$
- Order the Big O running times you found previously. Indicate and explain if any two of the running times are identical in terms of Big O notation.
  $\mathcal{O}(4^n) \geq \mathcal{O}(2^n) \geq \mathcal{O}(n^2) \geq \mathcal{O}(\log n) = \mathcal{O}(\log n)$

2. Pointers

- Write the declarations and code required to construct the following data structure.



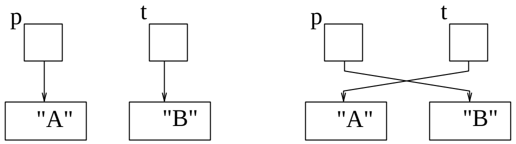

```
type nodeType:
  record
    item : string(1)
    next : ^nodeType
  end record

var p : ^nodeType
new p
p->item := "A"
new p->next
p->next->item := "B"
p->next->next := p
```


- Write the code required to deallocate all heap memory in the above figure.

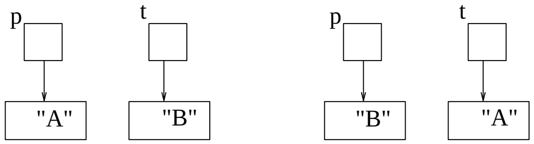
```
free p->next
free p
```

- Write the code required change the data structure from that on the left to that on the right, ie swap pointers.




```
var temp := t
t := p
p := temp
```

- Write the code required change the data structure from that on the left to that on the right, ie swap the record values.




```
var temp := ^t
^t := ^p
^p := temp
```

3. Classes

- Briefly describe the similarities and the differences between classes and records.
  - Similarities:
    * both have declarations that do not create an object but are templates for how an object would look when created.
    * both can be created by having a pointer p to the data type and saying "new p".
    * both can contain field containing data. These are accessed by "$p \rightarrow i$".
  - differences
    * You can have a variable R that is of type record. You access the fields of the record using $R.field$. This cannot be done with classes. A class can only be accessed via a pointer.
    * Classes can also contain fields that are routines.
    * Classes have inheritance.

- Describe Inheritance

  A class can be created by inheriting the properties from another class. New data structures and routines can be added. The body of previously defined routines can be changed as well. A pointer to a parent class, eg picture, is able to point at any derived class, eg square, circle.

- You are given the following Turing program.

```
type entry
   record
     name : string
     address : string
     next : ^entry
   end record

var e1, e2 : ^entry
new e1
new e2

e1->name := "Jeff"
e2->name := "Joshua"
```

  Is it possible to rewrite this program by using a class instead of a record? If your answer is "yes", do so by showing the class definition and making the required changes to the main program. If your answer is "no" explain why not.

```
class entry
   name : string
   address : string
   next : ^entry
end entry

  No changes to main
```

4. Recursion

- Write a simple recursive program that has running time $\mathcal{O}(2^n)$. It should print "Hi" exactly 2^n times. Hint: The base case for $n = 0$ prints "Hi" 2^0 times. It also helps to remember that the recursive call Rec(x) will print "Hi" 2^x times. Remembering my table of running times should also help.

```
procedure Rec( n : int )
  if n=0 then
    put "Hi"
  else
    Rec( n-1 )
    Rec( n-1 )
  end if
end Rec
```


- For $n = 2$, draw a diagram (as done in class) indicating each instance of the above recursive program that gets called and which calls which. Indicate which of the instances actually print "Hi".

5. Verification

   Below is the code that will compute the factorial of an integer using iteration.

```
function Fact ( n : int ) : int
   var i, f : int
   i := 1
   f := 1
   loop
       exit when i = n
       i += 1
       f := f * i
   end loop
   result f
end Fact
```

- What should the preconditions for this function be?

  $n \geq 1$
- What should the postconditions for this function be?

  returns n!
- What is the pre-loop assertion for this loop?

  i=1,f=1
- What is the loop invariant for this loop?

  f = i!
- What steps are needed to prove the correctness of the loop?
  - $\langle pre \rangle \rightarrow \langle loopInv \rangle$
  - $\langle loopInv \rangle \,\&\, \langle notexit \rangle \,\&code \rightarrow \langle loopInv \rangle$
  - $\langle loopInv \rangle \,\&\, \langle exit \rangle \rightarrow \langle post \rangle$
  - loop terminates.
- Sketch a proof of correctness for the loop. (Some math needed here)
  - $i = 1, f = 1 \rightarrow f = i!$
  - $f' = i'! \rightarrow f'' = i'' * f' = (i' + 1) * (i'!) = (i' + 1)! = i''!$
  - $f = i!$ & $i = n \rightarrow f = n!$ is returned.
  - i increases by one each time and hence will eventually reach n.