SOLUTIONS TO FIRST TERM TEST 148S 1997 FICH'S SECTION

1. (3 marks) Give three reasons why modularity is important when writing code.

   It is easier to divide work among different team members.
   It is easier to test and debug the code.
   The code is easier to understand.
   One part of the program can be changed without affecting other parts.
   It is easier to add new features.
   It is easier to change to another data structure.

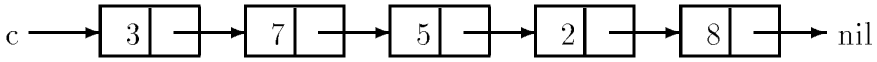
2. Suppose the following type declaration has been made:

```
type box :
    record
        stuff : int
        arrow : ^box
    end record
```

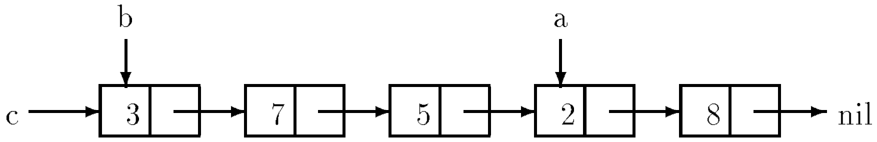
   Consider the following procedure. Note that executing `return` causes the procedure to return immediately.

```
procedure P (var c : ^box)
    var a, b : ^box
    var i : int
    a := c
    i := 0
    loop
        if a = nil then
            return
        end if
        a := a -> arrow
        i := i + 1
        exit when i = 3
    end loop
    if a = nil then
        c := c -> arrow
        return
    end if
    b := c
    loop
        a := a -> arrow
        exit when a = nil
        b := b -> arrow
    end loop
    b -> arrow := b -> arrow -> arrow
end P
```

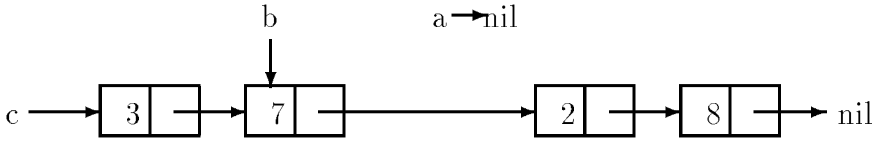
(a) (5 marks) Suppose that `P` is called with parameter `c`, where `c` is a pointer to the following linked list:



Draw a picture of the linked list and the variables `a`, `b`, and `c` immediately before the second loop is executed.



Draw a picture of the linked list and the variables `a`, `b`, and `c` at the end of procedure `P`.



(b) (5 marks) Suppose that `P` is called with parameter `c`, where `c` is a pointer to a linked list of length n. How many assignments are performed? Explain briefly. Be sure to include all cases.

If $n = 0, 1, 2$ then $2n + 2$ assignments are performed. This includes 2 assignments prior to the first loop and 2 assignments in each of the n complete iterations of that loop.

If $n \geq 3$, there are 3 complete iterations of the first loop performed. For $n = 3$, one additional assignment is performed after the first loop, for a total of $2 + 2 \cdot 3 + 1 = 9$.

When $n > 3$, one assignment is performed between the two loops, there are $n - 4$ complete iterations of the second loop (each containing 2 assignments), one assignment is performed in the last iteration of the second loop, and one assignment is performed after that. This results in a total of $2 + 2 \cdot 3 + 1 + 2 \cdot (n - 4) + 1 + 1 = 2n + 3$.

(c) (3 marks) What test cases would you include for testing this procedure? Explain.

I would consider the cases when `c` points to lists of length 0, 1, 2, 3, 4, and 5. There is no need to worry about the contents of the stuff field, because it is not referred to by the code in the procedure. It is important to test all execution paths. In particular, there should be cases in which `P` returns in the first loop, between the two loops, and after the second loop. For the loops, it is a good idea to have test cases that cause the loops to exit during their first iterations and during later iterations.

(d) (3 marks) Write appropriate comments for the top of procedure `P`.

Delete the third last element of the singly linked list pointed to by c, if c has at least 3 elements. Otherwise, the list pointed to by c remains unchanged.

(e) BONUS (2 marks) What few lines of code would you add to most improve the quality of the code in procedure `P`? Where would you put them? Explain briefly.

When the third last node of the list is deleted, it is not freed. This is bad programming style. There are two places where this occurs: in the `if` statement between the two loops and immediately after the second loop. Freeing the node can be accomplished by assigning `a` to point to it immediately before it is deleted and having `free a` immediately afterwards.

More specifically, the `if` statement becomes:

```
if a = nil
then
  a := c
  c := c->arrow
  free a
  return
end if
```

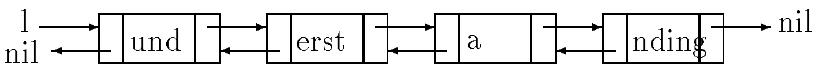
and after the second loop there is:

```
a := b->arrow
b->arrow := b->arrow->arrow
free a
end P
```

3. Consider a data structure for representing a long string of characters using a doubly linked list of short strings, each containing at most 20 characters. The following is a type declaration for ShortStringNode.

```
type ShortStringNode :
    record
        ShortString: string(20)
        fptr, bptr: ^ShortStringNode
    end record
```

(a) (1 mark) Draw a picture of a long string represented using this data structure.



(b) (5 marks) Write the signature for a module that uses this data structure to represent one long string and performs the following operations:

- initialize the long string to be empty
- determine the length of the long string
- delete the ith character of the long string
- append a character to the end of the long string

```
export opaque LongString
export Init, Length, Delete, Append
procedure Init (var l: LongString)
function Length (l: LongString): int
procedure Delete (var l: LongString, i: int)
procedure Append (var l: LongString, x: char)
```

Alternative Solution

```
export Init, Length, Delete, Append
procedure Init ()
function Length (): int
procedure Delete (i: int)
procedure Append (x: char)
```

(c) (5 marks) Write a function that returns the length of the long string. Recall that, if `s` is a variable of type `string`, the OOT function `length(s)` returns the number of characters in `s`.

```
function Length (): int
%assumes that l is a pointer to the LongString
var i: int := 0
var t: LongString := l
loop
   exit when t = nil
   i := i + length(t->ShortString)
   t := t->fptr
end loop
result i
```