Levi D. Smith CS1501 Intro to Computing

Class Syllabus – CS1501 Fall 1998

Instructor:	David Smith CoC119 404-894-9016	Office hours:	T & Th 1:30-2:30 & by appointment dmsmith1@cc.gatech.edu
Lecture:	Tuesday/Thursday	9:30-11:00 12:00-1:30	CoC 102 Architecture Auditorium
Recitation:	Tuesday Wednesday	6:00-8:00 6:00-8:00	various Skiles classrooms (use "viewsection" after Tuesday)

Find out at LAB

Labs: Labs are held in the French and Rich clusters beginning September 24. Labs meet this week – **GET YOUR GT ACCOUNT ACTIVATED**

1:1 Meetings: Schedule with TA in recitation next week. Mandatory EVERY week unless maintaining an "A" and TA approves

Help Sessions: To be announced

Texts and Readings:

Required: *Introduction to Computing and Algorithms, Second Edition* by Russell Shackelford, Addison-Wesley 1997

CS1501 Intro to Computing Lecture Notes

CS1501 Lab Manual

Optional: *Algorithmics, the Spirit of Computing* by David Harel, Addison-Wesley 1992

Newsgroups:	git.cc.class.1501.announce	REQUIRED DAILY – IMPORTANT!
	git.cc.class.1501.rec	optional, but strongly recommended
	git.cc.class.1501.lab	optional, but strongly recommended
	git.cc.class.150x	optional

Web Site: **http://www.cc.gatech.edu/classes/cs1501**

Due Dates/Times:

Homework:	due electronically Friday 9am
Projects:	due electronically Thursday 12 (noon); by Friday at noon for up to 60%
Buzzbacks:	start Thursday 00:00:01 – due by Monday 23:59:59 (midnight)

1501 – Two classes in one!

There are two ways in which you can take this course – high- and low-structured. Choose the option that will work best for you!

High-structure: homework is mandatory and must be turned in every week. You may correct mistakes and turn in a homework redo for each and get back half of the points lost on algorithm problems.

Best for students with little or no CS background and students who may not be self-motivated to study or do the homework.

Low-structure: you are **strongly** encouraged to do the homework and check your answers with your TA and with the published answers, but homework is not counted towards your overall grade. Quizzes count for more of your grade, and you can redo each quiz once (Friday at 3pm) the following week. The average of the quiz and the redo is your grade for that quiz.

Best for self-motivated students or students who have some CS background.

We've noticed that there is no difference in the percentage grades, but the letter-grade distribution among low-structure students shifted downward last quarter.

You choose which option you want in lab THIS week, and the choice is binding.

Grading Policies:

High-Structure		Low-Structure	
Quizzes	25% 90+	Quizzes	50%
Homework	25% 90+	Homework	---
Lab Projects	10% 90+	Lab Projects	10%
Final Exam	30%	Final Exam	30%
Lab Final	10% 100	Lab Final	10%
"Points"	+/- for Buzzback & 1:1s	"Points"	+/- for Buzzback & 1:1s

The final exam grade will replace quiz average if it improves your overall average.
You must have a passing grade on either the final exam or the quizzes to pass this course.
You will receive no more than one letter grade higher than the highest of your quiz and final exam grade.

Dec 3 Lab 6P - ~7:30P ispell 2 levels up
cd ../..

Bring ED
Draw Everything

MC - 14pts
Long - 20pts

CS 1501
Sept 24, 1998

- Paradigm - a model for a way of thinking
- Technology - any way to externalize an existing procedure.

- 1st Paradigm - Age of literal, preliterature culture
- Media Technology - Spoken word, drawing, sculpture
- World View - Theater of multiple gods, Divine is both good and evil
- Knowledge/Science - Appease the gods through ritual sacrifice, homage

- Alphabet
- Media Technology - Abstract symbol system. Technology externalizes human expression.
- World View - New ability: think abstractly
  Heresy: postulating abstract principles
- Knowledge/Science - Creation of abstract Philosophy as "the body of all knowledge."

- Age of Absolute Abstraction -
- Media Technology - Homemade documents
- World View - the Holy Battleground: Good and Evil Separation
- Knowlege/Science - Study of Divine Power

- Printing Press - Mechanical reproduction. Technology externalizes human body's work
- World View - Mechanistic thinking
  Heresy: laws of natural forces (Copernicus, Galileo)
- Knowledge/Science - Creation of Physical Sciences and Mathematics → they split from philosophy

- Age of Mechanistic thinking -
- Media Technology - Printed books papers.
- World View - God → Engineer of Universe
  World → Machine of Natural Force
- Knowledge / Science — Master the manipulation of matter by physical force

- Electronic Media -
- Media Technology - Instant sensory stimuli
  Technology externalizes human nervous system
- World View - New ability - process thinking
  Heresy: forces that we can't measure (Einstein, Freud, Curie)
- K/S = Creation of the Human Sciences; they split from Philosophy

New Paradigm - Age of Computational Models
Media Technology - Literal (non-abstract) audio/visual information
World View - A network of complex intangible processes
K/S - Master the manipulation of info by algorithmic processes

Rec

~~Bill Leahy~~ bleahy@cc.gatech.edu
6:05 - 8:00

T	R	F	W
L	L	Due	quiz

---

workon ␣ cs1501
getjob
~ hw
prj

getjob HW1

"rn" news group
emacs

Tue 1 on 1
2:30

[ESP]
- Estimate
- Design - think about it
- Code
- Check

GETJOB HWFORMAT
Redo problems due Next Friday
turnin hw2 <filename> getback hw2
turnin hw1r <filename>

RUN FTP
open

usually get HW back in 1 on 1
sometimes mailed to you

header

```
function square returns a Num ( N ISOFTYPE IN NUM)
  // Squares a number

    square returns N * N
End Function
```

```
PRINT (X, "Squared is", SQUARE (X))
```

```
Procedure PRINTASQUARE ( N ISOFTYPE IN Num)
// Square a Number and Print it
    PRINT ( SQUARE (N))
End Procedure
```

irwando@cc.gatech.edu
Irwin Dolobowsky
Wed 18:00 - 20:20
Location Skiles 169 Lvl: Intermediate

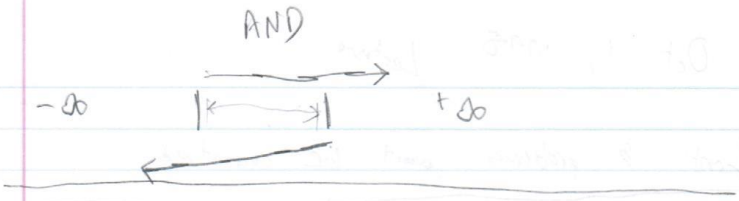
Skiles 169 @ 600 pm

Oct 1, 1998 Lecture

Last 3 problems won't be counted

"Can you guess my age within 5 years." Enter your guess
{ You guessed 47 - Sorry your grade is F
{ You guessed 52 - Good, your grade is A

```
algorithm Guess
// definitions
   guess isoftype Num ; LIMIT is 5
   comment isoftype String ; you.are.right isoftypeBoolean
   grade isoftype Char ; AGE is 56
// program starts here
   print (" can you guess my age within ", LIMIT, " year enter guess")
   read (guess)

// (AGE - LIMIT) < guess < (AGE + LIMIT)
you.are.right <- ((guess > (AGE-LIMIT) AND (
                    guess < (AGE+LIMIT)))


if (you.are.right) then
      comment <- "Good Job!" ; grade <- A
else  comment <- "Sorry" , grade <- F
print (Yg , "guess", "-", comment, "Yg" grade
end algorithm
```



NO CHEATING

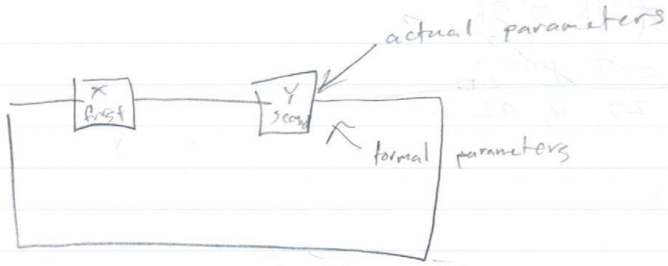
acme> chmod 600 <filename>

[p 8] 1:10 34 23 12

Pe One - On - One → 1:40 W

リバ゛ド スミス

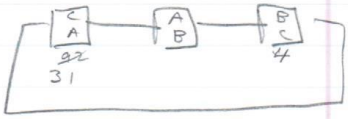
```
algorithm Guess
  AGE is 56
  LIMIT is 5
  guess isoftype num
  right isoftype Boolean
  // code starts here
    get_guess (guess)
  right → is it close (guess)
    write_result (right, guess)

    procedure get_guess (newage isoftype in/out Num)
      // Comments
        print (" Your last guess was ", new_age, " What now?" )
        read (new_age)

    end procedure // getguess

    function is_it_close returns a Boolean (guess isoftype in Num)
      // logic
      is_it_close returns ———

    end function

    procedure write_result (is_right isoftype Boolean,
                            age isoftype in Num)
      comment isoftype String
      grade isoftype char
      if (is_right) then
      else
      print (" ——— "); endprocedure
end algorithm
```



parameters → IN, OUT, IN/OUT
functions can only have "in" parameters

ホメヲルク

```
algorithm test
  A, B, C, isoftype Num
// code
  A <- 27
  B <- 4
  C <- 92
print (A, B, C)
juggle (C, A, B)
print (A, B, C)
B <- fiddle (C)
A <- fiddle (B)
print (A, B, C)
end algorithm
```




```
procedure juggle (A isoftype in Num, B isoftype out Num,
                  C isoftype in/out Num)
  A <- C + 27
  B <- A + 2
  C <- A
  print (A, B, C)
end procedure
```

おうとぷうと
オウトプウト
OUT PUT

27, 4, 92

October 8, 1998

STACKS
- Parameter Types

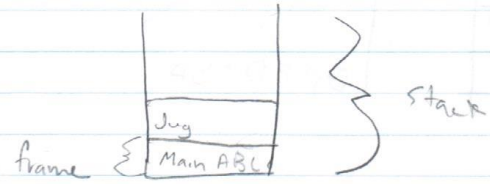
Recursion
- What it does
- Forms
- How to write
- Example

[not <u>what</u> to use it?] Documentation

```
algorithm Main
    A,B,C isoftype Num
    A <- 1
    B <- 2
    C <- 3
    print (A, B, C)
    Jug (C, A, B)
    print (A, B, C)

procedure Jug (x isoftype in Num,
               y isoftype out Num,
               z isoftype in/out Num)
    y <- x + z
    print (x, y, z)
    x <- z + 4
    z <- x - 2
end procedure
```

123
352
543

Stacks



Pringht 1st 7 letters of Alpha

```
procedure list_letters (
ch iot in Char)

if ( Not (ch < 'A')) then
    print (ch)
    list_letters (ch - 1)

end procedure
```

list letters ('G')

Wrap function

('A'-1)
'A'
'B'
'C'
'D'
'E'
L-L ch='F'
L-L ch='G'
Main 'G'

GFEDCBA

Fibonacci

fib (27, fred)

```
procedure fib (number isoftype in Num,
               answer isoftype out Num)
  if (number < 0) or ('____')
      print("go away")
  else
      answer <- real_fib (27)
  end if
end procedure
```

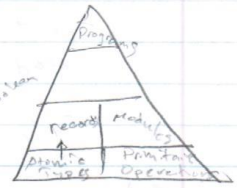
10 月 13 日 (月)

レクチャー ねくたいね

October 13, 1997 Lecture

Review - "Machinery"
Abstraction - ¿Dondé estamos?
Records - Defining
- Declaring
- Using
- Anonymous
- Combining Abstraction
Dynamic Data - Pointers
- Linked Lists

Party defines a record
address is String
start.time is Num
end.time is Num ← Place Cave is Boolean
endrecord // Party



// definition
// declaration
halloween.bash is Party

halloween.bash.adress ← "college of computing"

// declarations
This isoftype Record

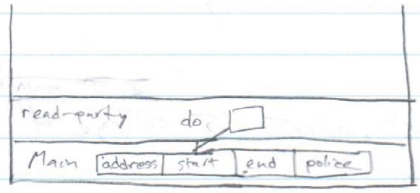
end record

read_Party (haloween.bash)
write_party (haloween.bash)

---

procedure read_party (do ;ot out Party)
print ("where is it?")
read (do, address)

end procedure // read.party
procedure print_party (do ;ot in Party)
print ("it is at", do.address)
end.procedure



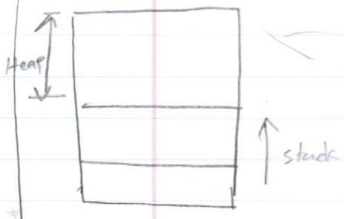
halloween.bash

thanks, starttime ← halloween bash. starttime

// definition
Student.Rec defines a record
name iot String
SSN iot num
grade iot char
end record // party
//declaration

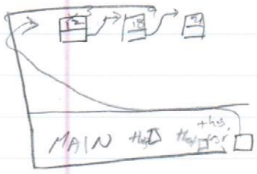
listhead^.data.name <- "Fred"

list.node defines a Record
data iot StudentRec
next iot Ptr to a list.node
end record
// declaration
list.head iot Ptr to a list.node
temp iot



"linked list"

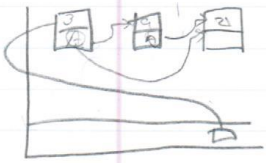
list.node defines a Record
data iot Num
next iot Pointer to a list node
end record
// declarations
this_list iot Ptr to a list node



this <- 4
this_list^

this_list^.next^.data <-9

this_list^.next <-
this_list^.next^.next
A



Recitation October 14, 1998

```
MyCar definesa record
    color isoftype string
    make isoftype string
    mileage isoftye Num
end record

Car1 isoftype MyCar
Car1.color <- "brown"
Car1.make <- "Toyota
Car1.mileage <- 30000
Car1.year <- 96
```

```
procedure fillcar(car1 iot out MyCar)
    Car1color <- "brown"
    Car1.make <- "Toyota"
    Car1.mileage <- 30000
    Car1.year <- 96
```

Pointers

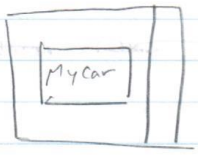
```
x, y, z iot Num
num.ptr iot Ptr toa Num
x <- 42
y <- 19
z <- 3
num.ptr <- x
y <- numptr ^
```

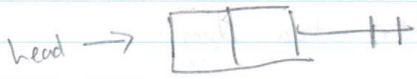
(x + numptr ^) = 84

Linked List

MyCar defines a record
    make is string
    mileage is Num
end record

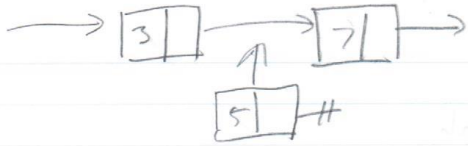
List_node defines a record
data is MyCar
next is Ptr to a List_Node
end record





```
algorithm blah
head iot Pointer toa List_Node
head <- new (List_Node)          // name iot Listnode
                                 // head.node
~~head^.data.make <- "Ford"~~
  head <- N.1

~~procedure fillcar~~


  fill_car (head^.data);
end alg
  procedure fillarcar (car1 iot out MyCar)
    car1.make <- "ford"
    car1.mileage <- 30000
  end prod
```

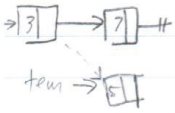
```
procedure fill list ( head iot inout ptr toa list_node)
  if (head = NIL) then
    head <- new (List_node)  //make a new block on end
    head
    fill_node (head)
  else
    fill_list (head^.next)
  end if
end procedure
procedure fill_node (head iot inout ptr toa list_node)
    fillcar (head^.data)
    head^.next <- NIL
```




```
LL defines a record
    x int Num
    next int ptr toa LL
end record

procedure Add_in_order (head int in/out ptr toa LL,
    x int in Num)
    if((head = NIL) OR (head^.data > x))
        temp isoftype Ptr to a LL
        temp <- new(LL)
        temp^.data <- x
        temp^.next <- head
        head <- temp
    else
        Add.In.Order (head^.next, x)
    endif
endprocedure
```

Cases

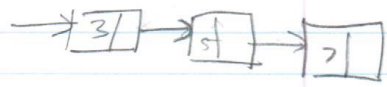
head —#



// Check for NIL first

→ ", pirate .of in Num)

→ OR (head ^.data. id-num > X. id_num)




```
if ( (head = NIL))
   // do nothing
```

DELETING

```
procedure                    target .of in Num)
   if (NOT (head = NIL))
   if (head ^.data. idnum = target)
    head <- head ^.next
   else
      Add_In Order (head ^. next, pirate)
   endif
   end if
   end procedure
```

Rec Oct 21

```
Tree_Node defined as Record
data : of Num
left_child : of Ptr to a tree_node
right_child : of Ptr to a tree_node
endrecord
```

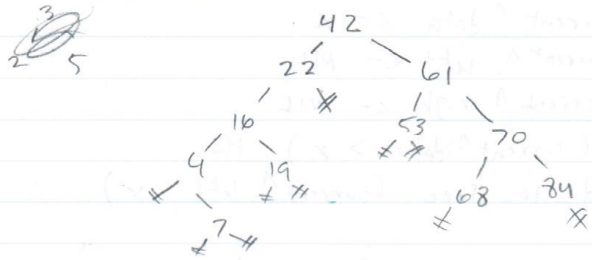
Always add at a NIL pointer

```
procedure Add_to_Tree (current : of ptr to a Tree_Node)
  if (current = NIL) then
      current <- new (Tree_Node)
      current^.data <- x
      current^.left <- NIL
      current^.right <- NIL
  else if (current^.data > x) then
      Add_to_Tree (current^.left, x)
  else
      Add_to_Tree (current^.right, x)
  endif
endprocedure // Add_to_Tree
```

```
procedure Delete_Node (current : of in/out Ptr to a TreeNode)
  if (current <> NIL) then
      if (current^.left_child = NIL) then
          current <- current^.right
      else
          Delete_Node (current^.left)
      endif
  endif
endprocedure
```

// tree traversals

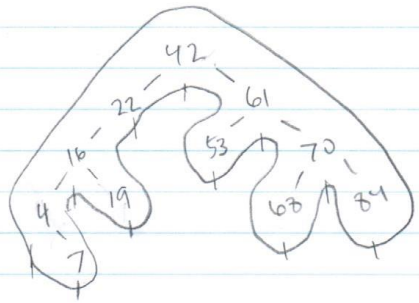
In Order - L, C, R
Pre Order - C, L, R
Post Order - L, R, C



In Order	if (current <> NIL)
	Left
	Something → Print
	Right
	endif
		4, 7, 16, 19, 22, 42, 53, 61, 68, 70, 84

Pre Order	Print Left Right	42, 22, 16, 4, 7, 19,
Post Order	Left Right Print	7, 4, 19, 16, 22, 53, last 42

In - Dash Down
Pre - Left
Post - Right

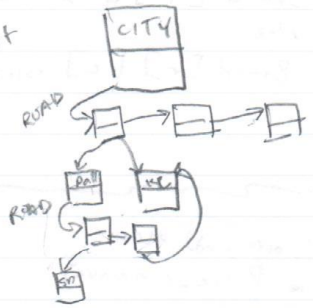


Lecture Oct 22, 1998

Homework - P4 Map ID is Unique
Revist Graphs
Arrays - Declaration
- Iteration
- Example
Traversal
Search

Trees - Acyclic - can't wrap around themselves

Root



Now it is cycle

MIN MAX (use constants)

Row_Type defines a array [1..8] of Num
Row isoftype Row_Type
i isoftype Num
//
i ← MIN
loop
exit if (i > Max)
print (Row [i])
i ← i+1
endloop

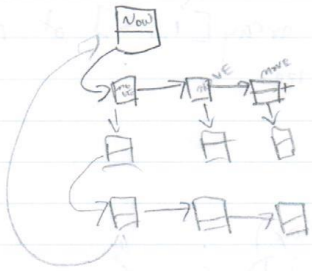
Row_type defines a array [1..8] of Square (r.size)
Board_type defines a array [1..8] of Row_type (c.size)
r, c iot Num
Board iot Board_Type

Square
Color iot Color

```
r=1
loop
  exit if (row > 8)
    c <- 1
    ~~color~~ loop
      c <- c+1
      end loop
    r <- r+1
endloop
```

```
if (r+c mod 2 = 0 then
  Board [r] [c] color <- Black
else
  Board [r] [c] color <- White
```

```
if (Board [r][c]_is_occupied) then
    if (Board [r][c]_Piece_can move)
        Add_to_Move_list (Board [r][c].piece])
```



Oct 22, 1998

Traversal ← Visit <u>All</u> Elements
Search ← Probably Stop

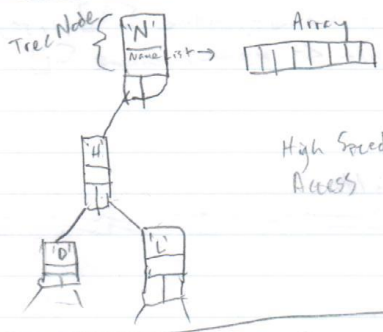
10 月 27 日 1998 年

October 27, 1998

October 27, 1998

```
Tree_Node defines a record
    data is Tree
    left, right Ptr to Tree Node
end Record
```

```
Tree_Block defines a Record
    letter is Char
    list is name_dist
end record
```




```
Name_List defines a record
    Size is Num
    count is Num
    list is array [1 .. SIZN]
            of name_block
end record
```

```
Name_Block defines a Record
    list is String
    first is String
    b
```

(last week search)

```
function L.S returns a Name_Block (here is in Ptr
        to Tree_Node, first is in Char, this_name is in String)
    if (here = NIL) then L.S returns EMPTY
    elseif (here^.data.letter = first) then
        L.S returns Search array( here^.data.list.name array)
    elseif (here^.data.letter > first) then
        L.S returns L.S (here^.left, ----)
    else                                 right, ---->
end function
```

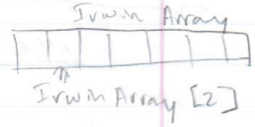
```
                (search array)
function S_A returns a Name_Block
        (names ist in Name_list, this_name ist
         in String)
  i ist Num
  reader ist Name_Block
  i <- 1
  reader <- empty
  loop exit if (i > Names_count)
      if (this_name = names.name[i].list) then
                result <- names.name[i]
    i <- i + 1
  endloop
   S_A returns result
```

Sorting

Insertion	50
Bubble Sort	45
Heap Sort	28
Exchange	17 (Similar to Merge Sort)
Shell	11
Quick	6

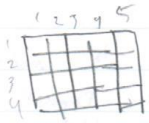
Rec Oct 28

MAX
My_Array defines Array [1..MAX] of Num
Irwin_Array is of type myArray



2D

Easy Way



ROW := 4
COL := 5

My_Array defines a Array [1..ROW] [1..COL] of Num
Row Array defines a Array [1..Row] of Num
My_Array defines a Array [1..Col] of Row Array

Searching through Array

My Array defines a Array [1..ROW] [1..COL] of Num
Irwin Array : of My Array
i, j : of Num
i <- 1
loop
exit if (i > ROW)
j <- 1
loop
exit if (j > COL)
Irwin Array [i][j] <- i*j
j <- j+1
endloop

i ← i+1
endloop

Classes

class ATM

user can see this → public

// headers to modules

headers

procedure Deposit (inmon iot in Num)
procedure Withdraw (outmon iot out Num, Amt iot in Num)
function GetBal returns a Num ()
procedure initalize ()

(private) protected

data goes here

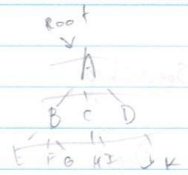
copy both exactly from top

balance iot Num
procedure Deposit (inmon iot in Num)
balance ← balance + inmon
end procedure
procedure Withdraw (outmon iot outNum, Amt iot inNum)
OutNum ← balance - Amt
balance ← balance - Amt
function Get returns a Num
GetBal returns balance
end function

also → procedure Initalize ()
(default values)

Lecture

October 29, 1998



queue

```
En queue (Root)
loop
   exit if (is_empty (q))
   dequeue (person
   enque ( all _children)
   printout (person
endloop
end prod
```

```
algorithm
   | DECLARATION { constants
                   definitions
                   variables
   | PROGRAM
end algorithm

algorithm
uses Door
```

```
class Door
public
   procedure Lock ( state , ot in Boolean)
   function is_locked returns Boolean
   initialize (
protected
   Door Rec defines a Record
      lock
   Ind Received
   the door ist Door
end class
```

guts of all methods

If data types are declared in protected, the scope is limited to class

Variables declared in class are scoped to whole class

```
algorithm SpaceShip
uses Door
inner, outer Door
inner.initalize (TRUE)
outer.initalize (TRUE)
  if (AirlockPres <> CabinPres) then
    inner.lock (TRUE)
  else inner.lock (FALSE)
  if (Ship<> ~) then outer.lock (TRUE)

this_door int Ptr to Door
  this_door <- new (Door)
  this_door ^.initalize
```

November 3, 1998, Lecture

```
function Fill returns a array [1 .. 26] of char
    Fred is Array_Type
    i is Num
    up is ~~Num~~ Char, down is Char
    i <- 1
    up <- 'A', down <- 'Z'
    loop
        exit if ( i > 26 )   MAX
        Fred [i] <- up,  Fred [i+1] <- down
        up <- up + 1
        i <- i + 2 , down <- down - 1
    endloop
    Fill returns Fred
```

i <- 1	
up <- A	
Fred 1	'A'
2	
3	'B'
4	
5	'C'
6	

Class abstraction

November 4, 1993 Rec

class Queue (qtype isoftype type)

procedure Enqueue (data isoftype in qtype) (Num) (chars)
// ~~~
procedure Dequeue (data isoftype out qtype) (Num) (chars)

---

algorithm stuff
uses Queue
Num Q int Queue (Num)
Char Q int Queue (Char)

Inheritance -

class Shape

sides

Sq rect tri ← can add or modify (add area)

Animal (eat, sleep,)

cat dog Sound (difference sounds)

Bear
sleep
(uses bear sleep)

- good idea to declare an empty sound in Animal

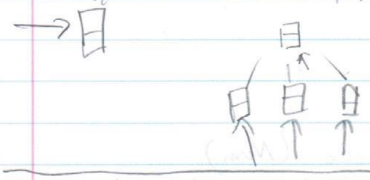
Class Dog
inherits Animal

Polymorphism

MyPtr iot Ptr toa Bear

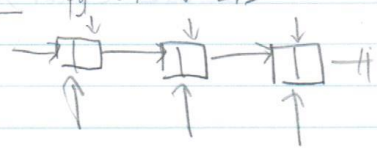
Bear

my ptr iot Ptr toa Animal

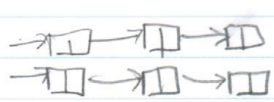


Copy / Clone pg 272 + 273

copy -



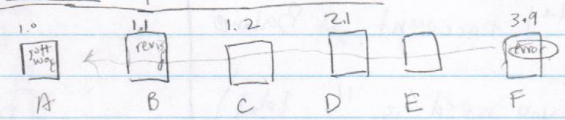
Clone →

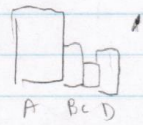


Nov 5, Lecture

- RE-USE
- EXAMPLES
- INHERITANCE
- POLYMORPHISM

1. You have written something you are proud of
2. It is documented
3. You have a different but similar problem

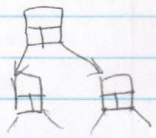




class BST (datatype .of type, indextype .of type)

public: Insert (Student_Rec) SSN — StudentRec → SSN

Delete (——— , SSN)



protected

Tree_Node (index .of Num)

data .of Ptr

left, right Ptr

end Rec

Students .of BST of (StudentRec, Num)

Cards .of BST of (Address Rec, String)

cards initialize

cards.Insert (my-address, "Smith")

Davids Cards ;ot BST (address, type . string)

D.C Initalize

Julies Cards ;ot BST ([illegible])

// copy David's cards

~~total = 0~~

loop

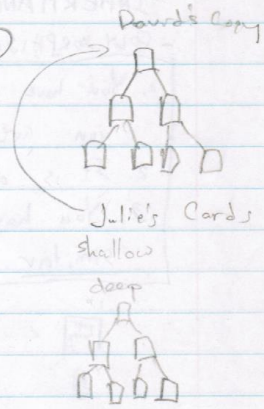
exit if (empty)

dequeue (a-account)

total = total + account, get Balance

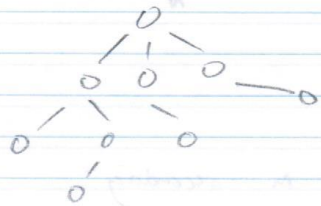
endloop

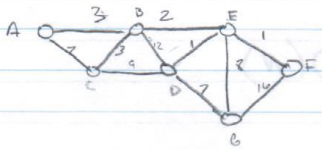
print ("All you need is " , total)



breath first search		depth first search
BFS	vs.	DFS
Slower		Faster
Optimal solution		A Solution

remembers visited nodes





BFS → 3, 2

A - B = 3

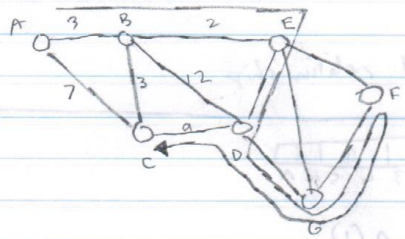
B - E = 2

E - F = 1

E - D = 1

B - C = 3

D - G = 7



A - B = 3

B - E = 2

E - F = 1

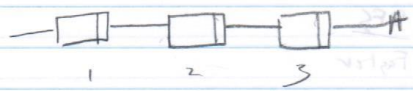
F - G = 16

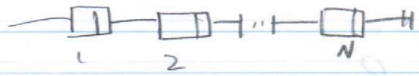
G - D = 7

D - C = 9

Big - Oh

search



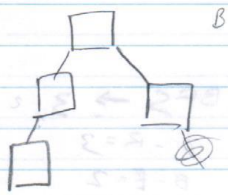


O(N) → have to go through all

O → drop constants

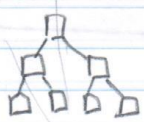
O(1) = smallest in ascending

Binary Tree



O(N)

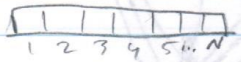
Full and Bal BST



O(log N)

log N relationship

Array



1) Seventh spot is O(1)
2) unsorted ~~checks~~ search are O(N)
3) ordered + search O(log N) - divide and conquer

loops → Nested vs. Sequential

```
m <- 1
loop   exit if (m = P)
    n <- 1
    loop   exit if (n = Q)
P   Q
    endloop
endloop
```

$O(PQ)$

Nested ⇒ Multiply

Sequential

```
loop
endloop   P times
loop
endloop   R times
```

$O(P+R)$

HW

LL = $O(P)$

FBST = $\log N$ ⇒ $P \log N$

Homework - Graph Problem - Don't Worry about ?

Measuring Work

Data Structures - Complexity

Compare $\log N$, N, N^2, 2^N

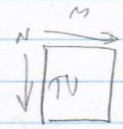
Performance Category

Nov 17, 1998 Lecture

Obj: solidify big oh ideas

drop slowest increasing

$O(N^2+N) = O(N^2)$



$O(M \times N)$

```
funct
f(data<1)
  fred ret 1
else
  fred returns fred(data-1) + fred(data-2)
end if
end funct
```

2^N

LL of BST

find something in structure

- find position in LL $O(N)$
- find position in BST $O(\log N)$ if balanced and indexed corr
  or $O(N)$
  $= O(N+N) = O(N)$

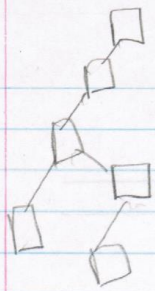
---

find + search BSTs $\rightarrow O(N \log N)$

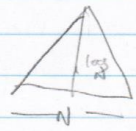
← More efficient

$O(N + \lg N) \rightarrow O(N)$

Balance BST



1. Traverse BST adding to LL at each node } N
2. Then take middle list entry and put on new BST } N
   - recurse on the left
   - recurse on the right } log N



Total : ~~N*~~ N log N

```
function Tile returns Boolean (data int in N num)
    if (data = 0) then
        tile returns true
    if (tile_cant_match (data)) then ret false
    i <- 1
    good <- true
    loop
        exit if (i = data or NOT good)
        good <- tile (data-1)
        i <- i+1
    exif
```

TRACE IT

⇒ tile(25)
24 times
23 times
22

O(N!)

tile (MAX)
MAX times
MAX times
⋮

O(N^N)

Inside Recursive Program

- one recursive call = O(N)
- two " " = O(2^N)
- helpers = O(helper x N)

loop ~~data~~ (data - 1) $= O(N!)$

loop (CONST - 1) $= O(N^N)$

USES

	USE	ALG	Freq
1.	Aircraft	Insert	15x/min
2.	Radar Update	N×Search	12x/min
3.	Coasting	Traversal	60x/min
4.	Query	Search	1x/min
5.	Conflict Probe	Traverse × Search	12x/min

	LL		Array		BST	
	Unsorted	Sorted	U	S		
1.)	$O(15)$	$O(15N)$	$O(15)$	$O(15N)$	$O(15 \log N)$	
2.	$12N^2$	$12N^2$	$12N^2$	$12N \log N$	$12N \log N$	
	N	N	N	$\log N$	$\log N$	
	$12N^2$	$12N^2$	$12N^2$	$12N \log N$	$12N \log N$	
	$24N^2$	$24N^2$	$24N^2$	$24N \log N$	$24N \log N$	$N = 100$
	240,000				4,800	

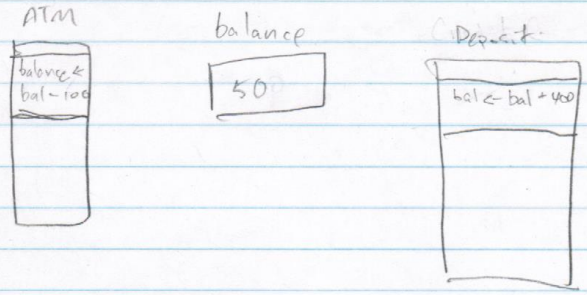
BST is faster for adding, but we have to balance it

	Bubble	Merge	Rec
p1]	N^2	$N \log N$	

p2] odd → $O(1)$
even → $O(1)$

$$\frac{MyAr[3] + MyAr[4]}{2}$$

p3]



Mutual exclusion makes only one data process at a time

Waits for one procedure to end before another one begins
"Data Integrity" - Insures this

Synchronization - each process wants to be on same page in order to communicate Ch 10 - Slide 24

Nov 19, 1998 Lect

Objective - Put computers in parallel to solve difficult problems

Slide #5 - Task ≡ concurrency number of tasks seem to be running at the same time

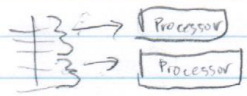
Processor -

Time slicing (UNIX) (slide 9) → Interleaving

2ms	2ms				

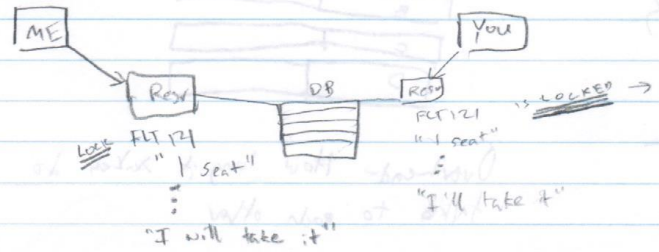
↑ your spot

Parallelism

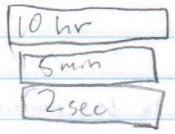


Tasks

	Tasks	Processors	
Close	Many	1	→ Multiprogramming
	1+	Many	→ Multiprocessing
Loose	Many	Don't Care	→ Multitasking
Distance	Many	Many	→ Distributing



Mutual Exclusion - Register for Classes



Win doesn't time slice

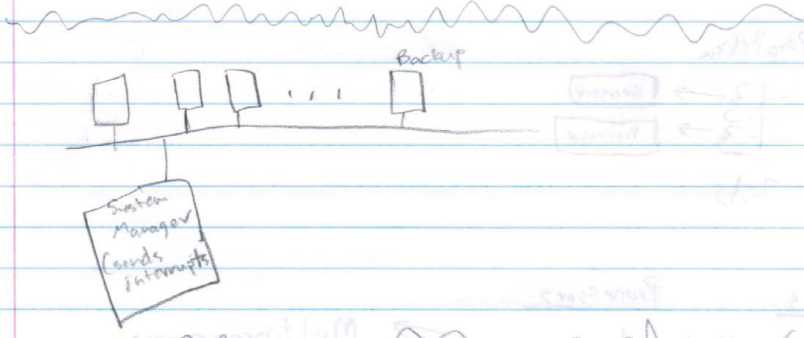
Fairness subdivide tasks into equal time chunks

ME		YOU	
121	LOCK	663	LOCK
663	WAIT	121	WAIT

Deadlock

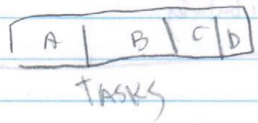
Must design software to avoid this

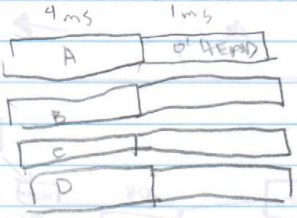
Treat query as a whole - lock only when you have all



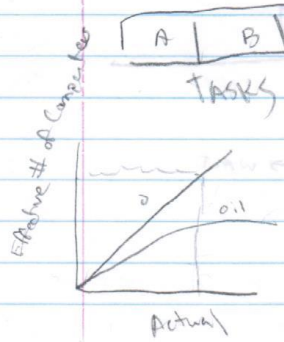
(Slide 26)

$\frac{\text{OVERHEAD}}{\text{Task Duration}} = R$





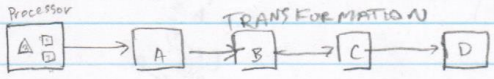
$\frac{1}{4} = 0.25$



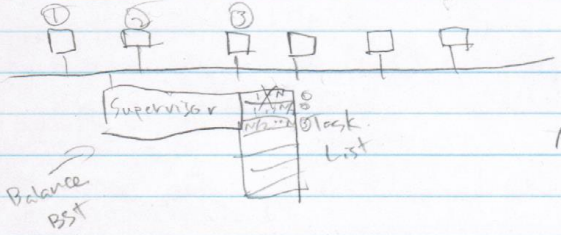
Overhead - How long it takes to talk to each other

How can computers be connected together?

Silicon Graphics - does drawing better (Slide 27)

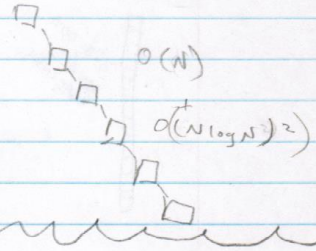


Flows through Pipeline - each box does a job
factor of ~~4~~ improvement - efficiency of tasks



(Slide #31)

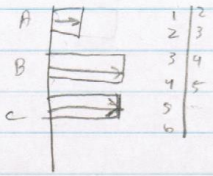
Make a LL



divides computing load by log N

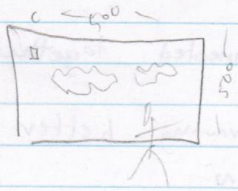
Improving Sorting

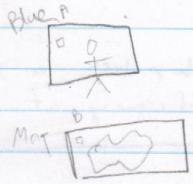
1 65 87 45 93 27 79 33 7
A B
2
3
N



loop
Do evens
wait while everyone finishes
Do odd number
wait until everyone finishes
endloop

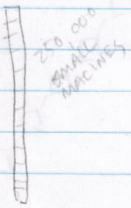
Small Tasks






```
loop 500
  loop 500
    if A = Blue
      C ← B
    else
      C ← A
    end if
  end loop
end loop
```

Single Instruction
— Multiple Data



Dec 12 – Lecture
FINAL Fri 8:00

<u>Upper Bound</u> – Bound where algorithm works with Big O
Bubble sort – $O(N)^2$ – defines what we know
<u>Lower Bound</u> – mathematically proven bound

- if upper = lower, then it is a closed problem
- if it is closed, you many still find ways to do a problem more efficiently (quickly)

Tractibility is an attribute of a problem – Tractable / Intractable

~~If lower bound is intractable the both bounds~~

CS people Intractable Upper
MATH people Tractable Lower
> NP-complete

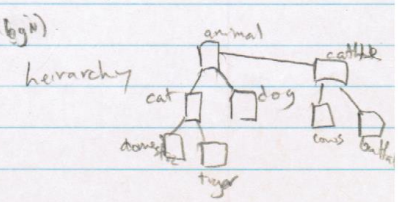
<u>D F F</u>
F F T
F F T
F T F – certificate

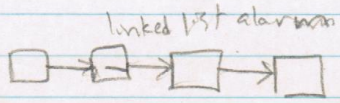
Non-Deterministic – can't figure out how to do the problem, but you can mathematically find a solution
= computer does guesswork

<u>EXAM STUFF</u>

- BST Questions – write code to add something to BST
  - what is BigO of BST? – balanced? – sorted?
    O(N) / O(log N)

- Polymorphism –





```
Animal list defines a record
    data int Animal
    next int Ptr to Animal list
end record

zoo int ptr to Animal list

animal int Animal
listptr int Ptr to Animal List
listptr <- zoo

loop
if buffalo then grunt OR
else if cat meow
:
endloop

loop
animal <- listptr^.data
animal.sound
listptr <- listptr^.next
exit if (listptr == NIL)

Class giraffe
    inherits tall
    public
    //no sound method
```

Dec. 10, 1998

- Every Thing will be on final

```
class Stack (Datatype iot type)
public
procedure push( in_data iot in DataType)   //add
procedure pop ( )                          // removes first item
                                           // no return

funct count returns a Num ()
pro initalize ()


protected
    ListNode Defines a record
        Data iot DataType
        Next iot Ptr to a List Node
    End Record
    head iot Ptr to a List Node
  count iot Num
  procedure Initalize ()
    count ← 0
    head ← NIL
  end procedure

  procedure push (In_data iot in Datatype)
      temp iot ptr to a ListNode
      temp ← new (ListNode)
      temp↑.data ← indata
      temp↑.next ← head
      head ← temp
      count ← count+1
end procedure
```

ALT (At end of LL)

```
procedure push (—)
    push-helper (head, in-data)
endprocedure

procedure push-helper (current : in/out ptr to
                        listnode, indata —)
  if (current = NIL)
      // add_stuff
  else
      push_helper (current^.next, indata)
  end if
endprocedure

procedure pop ()
    head <- head^.next
    count <- count - 1
end procedure

function counter returns a Num ( )
    counter returns count
end function
endclass
```

ALT pop

```
proc pop()
  pophelper (head)
end proc

proc pop-helper ( current : out in/out ptr to ListNode)
if (current^.next = NIL)
  current <- current^.next
else
  popHelper (current^.next)
end if
end proc
```

```
Algorithm Stacker
uses Stack
something iot stack (string)
something initialize ()
mynum iot num
temp iot string
loop
  menu (mynum)
  exit if (mynum = 4)
   if (my num = 1) then
     print ("what String do you want to add.")
     read (temp)
     something. push (temp)
  elseif (my_num = 2) then
     something. pop ()
  elseif (my_num = 3) then
     print (something. counter ())
  else
     print ("Invalid Option")
  endif
endloop
endalgorithm
```

```
procedure Menu (choice int out Num)
   print ("Choose one of the following.")
   print (" 1 Add a String")
   print (" 2 Pop a String")
   print (" 3 Count the Number of Strings")
   print (" 4 Quit")
  read (choice)
 endprocedure //Menu
```

```
procedure Add (My Class int in/out Stack)
   print ("String : ")
   read (mystring)
   myclass.push (mystring)
 endprocedure
```

```
ROWS is 7
COLS is 7
my array defines a array [1.. ROWS, 1..COL] of nums
this array is myarray
procedure convert ( arr is in myarray, current is
                    out ptr to a BST)
  i, j is Num
  i <- 1
  loop
    j <- 1
    exit if (i > ROWS)
      loop
        exit if (j > COL)
        Add_to_Tree ( current, arr [i,j])
      endloop
```

```
My BST defines a record
data is num
left is ptr to a My BST
right is ptr to a My BST
end record
root is ptr to a My BST
```

```
procedure Add_to_Tree (tree_root iot inlout ptr toa my BST,
        newdata iot in Num)
  temp iot ptr toa My BST
  if (tree_root <> NIL) then
    if (tree_root^.data > newdata) then
      Add_to_Tree (tree_root^.left, new_data)
    else
    Add_to_Tree (tree_root^.right, new data)
    endif
  else
    temp <- new (my BST)
    temp^.data <- newdata
    temp^.right <- NIL
    temp^.left <- NIL
    endif
endprocedure
```

cs1501 (100) Quiz 1 fall 1998 (Wednesday)

NAME : Levi D. Smith GTNUM : gte187k TA: Irwin

Please be sure to fill in the above information correctly. Any error will lead to a 5 point penalty.

Note: Answers which are ambiguous and/or illegible in the judgment of the grader will be penalized. For partial credit, show the details of your work. Comments are not required on quizzes. However, comments might allow your TA to understand your code better, _possibly_ resulting in partial credit.

The use of the word module is comprehensive and can be interpreted as 'module or group of modules'. Don't feel restricted to a single procedure or function if the problem asks you to write a module.

[p1]
40.0 points

Provide a constant or variable declaration to hold the desired data. Include a brief justification for each answer. Be sure to use descriptive names.

a) A person's age
b) The year the Magna Carta was signed (1215)
c) Whether or not an actor wears a wig
d) An identification number in the format xxx-xxxx-xxx where x represents an arbitrary alphanumeric symbol
e) The letter grade that you will get in any class at Tech
f) The distance, in kilometers, that light travels in space in two minutes (Assume exactly sixty seconds in one minute and c= 3x10^8m/s)
g) The name of a movie
h) The mileage on a car's odometer.

60
.2
120
3x10^8
3.0x10^8
3.6x10^10

a.) persons_age isoftype Num
// someone's age can only be a Number

b) MAGNA_CARTA_DATE is 1215
// the date will never change so it is constant

c.) actor_wears_wig isoftype Boolean
// the actor either does or does not wear
// a wig

d.) ident_numb isoftype String
// a series of Nums and Chars are required

e.) tech_class isoftype Char
// a grade can only be a Char (letter)

f.) LIGHT_DISTANCE is 3.6x10^10
// it is a constant since the speed of light
// and distance will never change.

g.) movie_name isoftype String
// a series of nums or chars will be required
// for the name of a movie

h.) car_milage isoftype Num
// the car's milage will change when it is
// driven

[p2]
15.0 points

Write an algorithm _in_ENGLISH_ to describe how to do groceries. The algorithm must have at least five steps and at least one conditional. DO NOT write any psuedocode for this algorithm.

Drive car to store.
Decide what you want for dinner.
Purchase necessary foods to make the dinner you decided to make.
If you need other groceries for following days then purchase those groceries.
Put groceries in car.
Drive car with groceries home.

[p3]

25.0 points

Levi D. Smith

Write an algorithm _in_PSUEDOCODE_ to determine the area of a trapezoid. Recall that the area of a trapezoid is defined as the one half the height times the sum of the upper side and the base. You should obtain any necessary input from the user and print the final answer back to the user. Be sure to prompt the user for any necessary information.

```
algorithm TrapezoidArea
// Purpose: to determine the area of a trapezoid
// specified by the user
trapezoid_height, trapezoid_top, trapezoid_bottom, trapezoid_area isoftype Num
print ("Enter the height of your trapezoid")
read (trapezoid_height)
print ("Enter the length of the upper side of your trapezoid")
read (trapezoid_top)
print ("Enter the length of the base of your trapezoid")
read(trapezoid_bottom)
trapezoid_area <- (trapezoid_height * (trapezoid_top + trapezoid_bottom))/ 2
print ("The area of your trapezoid is", trapezoid_area)
endalgorithm // TrapezoidArea
```

[p4]

40.0 points

Complete blanks (i.e. {___}) to complete the algorithm below.

```
Algorithm Multiple_Square
// this algorithm will compute the square of the multiple of the two
// inputted numbers and will print the final result back to the user.

        num1, num2 isoftype {___Num___}
        output isoftype num

        print("Enter the first number")
        {___read___} (num1)

        print("Enter the second number")
        read (num2)
        output <- (({___num1___} * num2) * (num1 * {___num2___}))

        print("The multiple squared is", {___output___})

endalgorithm
```

cs1501 (99) Quiz 2 fall 1998 (Wednesday)

NAME : Levi D. Smith GTNUM : gte187k TA: Irwin

Please be sure to fill in the above information correctly. Any error will lead to a 5 point penalty.

Note: Answers which are ambiguous and/or illegible in the judgment of the grader will be penalized. For partial credit, show the details of your work. Comments are not required on quizzes. However, comments might allow your TA to understand your code better, _possibly_ resulting in partial credit.

The use of the word module is comprehensive and can be interpreted as 'module or group of modules'. Don't feel restricted to a single procedure or function if the problem asks you to write a module.

[p1]
25.0 points

Given the length and width, write a module that will calculate and return the area of a rectangle. The area of a rectangle is defined to be:
area = length * width

```
function Rectangle_Area returns a Num ( Rectangle-Length, Rectangle-Width
    is of type in Num)
// Purpose: the two given values (length and height) are passed
// into this function and the function returns the area of
// the rectangle

    // start processing
Rectangle-Area returns (Rectangle-Length * Rectangle-Width)

end function // Rectangle-Area
```

Levi D. Smith

[p2]
25.0 points

Write a module that will be passed a person's GPA for a particular quarter and will print a message telling whether they are on academic probation (probation is a GPA less than a 2.0).

```
procedure Probation ( persons_GPA isoftype in Num)
// Purpose: this procedure is passed someone's GPA and it
// outputs wether the person is on probation or not

    // Constants
    PROBATION_GPA is 2

    // start processing
    if (persons_GPA < PROBATION_GPA) then
        print ("Haha!! You are on probation")
    else
        print ("Yay!! You are not on probation")
    end if

end procedure // Probation
```

Levi D. Smit

[p3]
25.0 points

Write a module that, given an age, will return whether or not the person is able to vote. (Voting age is 18 and older, in case you didn't know).

```
function Can_Vote returns a Boolean (persons_age is of type
    in Num)
// Purpose: this function gets an age from the algorithm
// and it returns wether the person can vote or not
  // Constant Declaration (although voting age probably won't
  //   change unless a Constitutional Amendment is
  //   passed)
  VOTING_AGE is 18
  // Begin processing
  if (persons_age >= 18) then
    Can_Vote returns TRUE
  else
    Can_Vote returns FALSE
  endif
end function // Can_Vote
```

-1.ft

[p4]

Levi D. Smith

25.0 points

Find and correct all errors in the following module. You may assume all comments are correct. Note: There may not be any errors.

*** DO NOT re-write the modules. Simply correct the errors. ***

a. ~~function~~ procedure PrintNum (new-num isoftype ~~out~~ in num)
   //takes in a number and prints that number out

   ~~read~~ print(new-num)

endprocedure // Print Num

b. function IsEven returns a boolean (in_num isoftype in ~~boolean~~ Num)
   //takes in a number via parameter and returns whether or not the
   //number is even. (MOD returns remainder from division,
   // EX: 4 MOD 2 = 0)

   if (in_num MOD 2 = 1) then
         IsEven returns ~~TRUE~~ FALSE
   else
         IsEven returns ~~FALSE~~ TRUE
   endif
endfunction // Is Even

c. procedure PrintNum (the_num isoftype in~~/out~~ num)
   // will print TRUE if a given number is greater than zero, FALSE if not

   if (the_num ~~<~~ > 0) then
         print ("TRUE")
   else
         print ("~~TRUE~~ FALSE")
   endif
endprocedure // Print Num

cs1501 Quiz 3 (94) fall 1998 (Wednesday)

NAME : Levi D. Smith GTNUM : gte187k TA: Irwin

Please be sure to fill in the above information correctly. Any error will lead to a 5 point penalty.

Note: Answers which are ambiguous and/or illegible in the judgment of the grader will be penalized. For partial credit, show the details of your work. Comments are not required on quizzes. However, comments might allow your TA to understand your code better, _possibly_ resulting in partial credit.

The use of the word module is comprehensive and can be interpreted as 'module or group of modules'. Don't feel restricted to a single procedure or function if the problem asks you to write a module.

[p1]
25.0 points

What is the output of the following algorithm? Please use one line per print and place your answer where directed (after algorithm).

```
Algorithm AlmostTraceable

  First,Second,Third isoftype num

  First <- 3
  Third <- 4
  Second <- 1
  print (Third,Second,First)
  WatchIt (First,Second,Third)
  print (First,Third,Second)
endalgorithm

Procedure WatchIt (First,Third,Second isoftype in/out num)
  LookOut (First,Second, Third)
  print (Second,First, Third)
  LookOut (Second,Third, First)
enprocedure

Procedure LookOut (First isoftype in/out num, Second, Third isoftype in num)
  First <- Second * 5
  Second <- First - 4
  print (First,Second, Third)
endprocedure
```

First <- 3
Second <- 1
Third <- 4

(x, z, y)
(z, x, y)
(z, y, x)

A <- 5
B <- 1

(A, B, C)

Place output here:

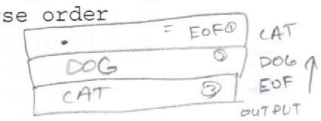
4	1	3
20	16	1
4	20	1
5	1	20
20	5	1

Levi D. Smith

[p2]
20.0 points

Find and correct any errors in the following code. You may assume that the comments are correct. Please note that there may not be any errors.

```
procedure
~~function~~ Print_Backwards()

   //this module will ask the user to enter words until a period is entered
   //When the period is entered, the module will print "EOF" and then procede
   //to display all the words the user has entered in reverse order

   word isoftype string        NOT NECESSARY
   print("Enter a word. (~~"."~~ to quit)")
   read(word)
   if(word = ".") then
     ~~Print_Backwards()~~  print ("EOF")
   else
SWITCH print(word)
       Print_Backwards()
   endif
~~endfunction~~ end procedure // PrintBackwards
```



[p3] Assume input is positive
25.0 points

Write a module that, when passed a number, will calculate the sum of all the numbers between 0 and that number (inclusive if applicable)

Hint: consider the following patterns

```
8  =>   8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
7  =>   7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
```

```
function Factorial returns a Num (number isoftye in Num)
// this function calculates the sum of all of the numbers between
// the passed in number and 0,

   current_sum isoftype Num   // holds the current sum
   ENDING_POINT is 0          //constant
      if (number = ENDING_POINT)
         Factorial returns ~~current_sum~~ 0   -> ret
      else
         current_sum <- current_sum + number
         Factorial (number - 1)
      endif                    Fac ret num + Fac (num - 1)
                               -4,5r
end function // Factorial
```

[p4]
30.0 points

Write a data structure that will hold the following information about a car:

Owner's Name
Model Name
Year Made
How many passangers it can hold
Whether or not it's insured

```
CarRec defines a Record
    owner_name  isoftype string
    model_name  isoftype string
    year_made   isoftype Num
    max_passangers isoftype Num
    is_insured  isoftype Boolean
end record // CarRec
```

Now write a module that will be passed a filled in data type (declared above). If the car is already insured, the module will print out a message telling the user that the car is insured. If the car is not insured, the module will ask the user if he wants to insure the car and change the field accordingly. The module will return the datatype with any changes that may have been made.

```
procedure Add_Insurance (ACar isoftype in/out CarRec)
// this procedure is passed a CarRec and then prints out ~~that~~ the if
// car is insured, or if it is not insured, asks the user if he
// wants it insured and makes appropriate changes
    yn isoftype char // for the Yes/No question
    if (ACar.is_insured = TRUE) then
        print("Your car is insured")
    else
        print("Your car is not insured. Would you like to insure it? y/n")
        read (yn)
        if (yn = 'y') then
            print("Great, I'll make the changes right away.")
            ACar.is_insured <- TRUE
        else
            print("Too bad. Maybe next time")
        end if
    end if
end procedure // Add_Insurance
```

(86)

cs1501 Quiz 4 fall 1998 (Wednesday)

NAME : Levi D. Smith GTNUM : gte187k TA: Irwin

Please be sure to fill in the above information correctly. Any error will lead to a 5 point penalty.

Note: Answers which are ambiguous and/or illegible in the judgment of the grader will be penalized. For partial credit, show the details of your work. Comments are not required on quizzes. However, comments might allow your TA to understand your code better, _possibly_ resulting in partial credit.

The use of the word module is comprehensive and can be interpreted as 'module or group of modules'. Don't feel restricted to a single procedure or function if the problem asks you to write a module.

The administrative department needs your help. They need you to help them keep track of all students at Tech with the following information.

Name
GradeLevel (1 = FirstYear, 2 = SecondYear, etc)
GPA
Social Security Number (XXX-XX-XXXX)
In-state resident or not

[p1]
15.0 points

Write the type declaration for the data structure that will be used to store the information of _ALL_ the students at Tech.

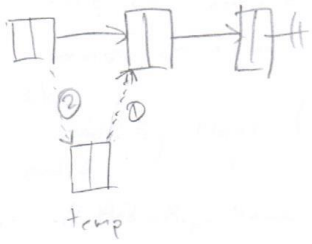
```
A_Student defines a record
    name isoftype string
    grade-level isoftype num
    gpa isoftype num
    soc-sec-num isoftype string
    is-a-resident isoftype boolean
end record

Student_Node defines a record
    data isoftype A-Student
    next isoftype Ptr to a Student_Node
end record
```

Levi D. Smith

[p2]
30.0 points

The administrative office now needs you to write a module that will allow for the addition of students to the database. Use your structure declared above and write a module that will take in the structure and a new student to be added and will add that student according to social security number. Remember, the comparison of strings is allowed. (i.e. "alpha" < "beta")

```
procedure Add_A_Student (currentptr isoftype in/out Ptr toa Student_Node,
        find_this_student isoftype A_Student)
// This procedure is passed a student's record and it will place it
// in the correct position in the linked list according to Social security
// Number. It is also passed the head of the linked list (currentptr).
        temp isoftype Ptr toa Student_Node
        if ( (current ptr = NIL) OR (currentptr^.data.soc_sec_num > find_this_
            student~~.data~~.soc_sec_num)) then
                                    -1 wrt
            temp <- new (Student_Node)
                                                    -2 log
            temp^.data <- find_this_student ~~^.data~~ // All in data
            temp <- currentptr^.next                // hook temp up to next node
            currentptr <- temp                      // hook previous node to temp
        else
            Add_A_Student (currentptr^.next, find_this_student)
        end if
end procedure // Add_A_Student
```



[p3]
40.0 points

Georgia Tech now needs a list of students that correspond to a particular grade level. It is your job to write a module that will take in a current student list and a grade level and will return a new list with all those students who are in that grade level. The new list should be sorted by name, alphabetical. Once again you can compare strings using boolean operators.

```
procedure Grade_Level_List ( oldptr, isoftype in Ptr toa Student_Node,
    newptr isoftype in/out Ptr toa Student_Node, target_grade isoftype in Num)
// prints all students in a specified grade
    ~~temp isoftype Ptr toa Student_Node~~
    if (oldptr <> NIL) then
        if (oldptr^.data.grade_level = target_grade) then
            Add_By_Name (oldptr, newptr)
            Grade_Level_List (oldptr^.next, newptr, target_grade)
        else
            Grade_Level_List (oldptr^.next, newptr, target_grade)
        end if
endprocedure // Grade_Level_List


procedure Add_By_Name (oldptr isoftype in Ptr toa StudentNode,
    newptr isoftype in/out Ptr toa Student Node)
// Adds the name to the new node by name (alphabetically)
    temp isoftype Ptr toa Student Node
    if ((newptr = NIL) OR (oldptr^.data.name < newptr^.data.name)) then
        temp <- new (Student_Node)
        temp^.data <- oldptr^.data
        temp <- newptr^.next
        newptr <- temp
    else
        Add_By_Name (oldptr, newptr)
    endif
endprocedure // Add_By_Name
```

-1 no

Levi D. Smith.

[p4]
15.0 points

Given the following module (for a linked list of numbers):

```
procedure Print_It(head isoftype in ptr toa ListNode)
        if(head <> NIL) then
                if(head^.next <> NIL) then
                        Print_It(head^.next^.next)

                        if (head^.data mod 2 =1) then
                                print(head^.next^.data)
                        else
                                print (head^.data)
                        endif
                endif
        endif
endprocedure
```

head -> 12 -> 16 -> 3 -> 1 -> 8 -> 11 -> 17 -> 22 -> 6 -> 19 ->nil

What is the output of Print_It when applied to the above linked list?

(Hint: draw a picture of the list and use arrows to keep track of where head is currently pointing)

~~NIL~~

22

~~17~~

1

~~16~~

- 10 wrong

cs1501 (97) Quiz 5 fall 1998 (Wednesday)

NAME : Levi D. Smith GTNUM : gte187k TA: Irwin

Please be sure to fill in the above information correctly. Any error will lead to a 5 point penalty.

Note: Answers which are ambiguous and/or illegible in the judgment of the grader will be penalized. For partial credit, show the details of your work. Comments are not required on quizzes. However, comments might allow your TA to understand your code better, _possibly_ resulting in partial credit.

The use of the word module is comprehensive and can be interpreted as 'module or group of modules'. Don't feel restricted to a single procedure or function if the problem asks you to write a module.

[p1]
10.0 points

Find and correct any errors in the following module:

```
*** You may assume that all type declarations,          ***
*** and comments are correct.                           ***

  TreeNode definesa record
    Data isoftype String
    LeftChild isoftype ptr toa TreeNode
    RightChild isoftype ptr toa TreeNode
  endrecord
function                    returns a Boolean
~~procedure~~ BST_Search ( root isoftype in ptr toa TreeNode,
                           element isoftype in String )
//Purpose: Returns TRUE if the element is found in the given BST, else
           returns FALSE
//NOTE: Strings can be compared like numbers, "alpha" < "zeta"
 if (root <> NIL) then
   if (element = root^.Data) then
      BST_Search returns TRUE
   elseif (element > root^.Data) then
      BST_Search returns BST_Search ( root^.RightChild, element )
   else
      BST_Search returns BST_Search ( root^.LeftChild, element )
   endif
~~endprocedure~~
~~endif~~
else
BST_Search returns FALSE
endif
end function // BST-Search
```

- lura

Levi D. Smith

[p2]
30.0 points

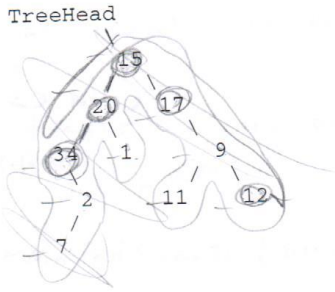
Given the following code:

```
    Tree definesa record
        Data isoftype num
        LeftChild isoftype ptr toa Tree
        RightChild isoftype ptr toa Tree
    endrecord

procedure Print_Numbers ( root isoftype in ptr toa TreeNode )

   LIMIT is 10
   if ( root <> NIL ) then
      if ( root^.Data > LIMIT ) then
         print( root^.Data )
         Print_Numbers( root^.LeftChild )
      endif
      Print_Numbers ( root^.RightChild )
   endif
endprocedure
```

In Order

P L R
L P R
L R P

What would be the output of PrintNumbers(TreeHead) for the following tree? It is highly advisable to draw a stack trace of the procedure.



15
20
34
17
12

Levi D. Smith

[p3]
30.0 points

The Bursar's Office wants to find out which registered students have not paid next quarter's matriculation fees. They need you to write a module which will take in a pointer to a binary search tree and will print the name and id number of all students who have not paid next quarter's fees.

They have given you the following data types to work with.

```
TreeNode definesa record
    Data isoftype Student_Rec
    LeftChild isoftype ptr toa TreeNode
    RightChild isoftype ptr toa TreeNode
endrecord

Student_Rec definesa record
    Name isoftype String
    ID isoftype Num
    Paid isoftype Boolean
    Address isoftype String
endrecord
```

```
procedure  Print NonPaid Students  (tree ptr  in Ptr toa TreeNode)
    ~~Students_to_find isoftype in Student_Rec~~)
// Purpose: to find all students who have paid their fees and
// print their names
    if (treeptr <> NIL)
      if ( treeptr^.data. paid = FALSE) then
          print (Name)
      endif
          Print Non Paid Students (treeptr^. leftchild)
          Print Non Paid Students ( treeptr^ . rightchild)
    endif

endprocedure // PrintNonPaid Students
```

~~Next Non Paid Student (Students_to find~~
~~// call this one FIRST~~

Levi D. Smith

[p4]
30.0 points

Write a module that will return a 20 element array of numbers. Your module should fill the array with the first 20 even numbers (starting with 2). Be sure to declare all necessary data structures.

Use iteration to solve this problem.

```
function FillArray returnsa Array ()
// Purpose: to return an array filled with the first 20
// even numbers
    ARRAYMAX is 20
    newarray  definesa array [1.. ARRAYMAX] of num
    arraytofill isoftype newarray
    STOP is 20
    i isotype num

    i <- 1

    loop
        exitif (i > stop)

            newarray[i] <- i * 2
            i <- i + 1
    endloop

    FillArray returns newarray

endfunction // FillArray
```

cs1501 (96) Quiz 6 fall 1998 (Wednesday)

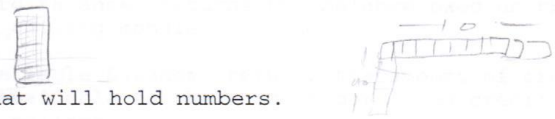
NAME : Levi D. Smith GTNUM : gtel87k TA: Irwin

Please be sure to fill in the above information correctly. Any error will lead to a 5 point penalty.

Note: Answers which are ambiguous and/or illegible in the judgment of the grader will be penalized. For partial credit, show the details of your work. Comments are not required on quizzes. However, comments might allow your TA to understand your code better, _possibly_ resulting in partial credit.

The use of the word module is comprehensive and can be interpreted as 'module or group of modules'. Don't feel restricted to a single procedure or function if the problem asks you to write a module.

[p1]
10.0 points



Declare a 10x40 array that will hold numbers.

```
ROWMAX is 40
COL MAX is 10

Row Array defines a array [1.. ROWMAX] of Col Array
Col Array defines a array [1.. COLMAX] of Num
```

[p2]
25.0 points

Write a module that, when passed a 2D array of numbers (of the type you declared in p1), will return the sum of all the values stored in the indices in the array. You may assume that the array is already filled.

```
function Find_Sum returns a Num ( This_Array is of type in Col Array)
// Purpose: this function returns the sum of all of the cells ~~values~~ in the passed in
// array
  sum is of type Num
  i, j is of type Num
  ROWMAX is 40
  COL MAX is 10
  sum <- 0
  i <- 1
    loop
      exit if (i > ROWMAX)
      j <- 1
          loop
            exit if (j > COLMAX)
            sum <- sum + This_Array[i][j]
            j <- j+1
          end loop
      i <- i+1
    endloop
  Find Sum returns sum
end Function // Find_Sum
```

Lori D. Smith

[p3]
30.0 points

You are going to write a class that models a credit card. Your class must have the following functionality:

- Make a Purchase: increases the balance owed on the card by the amount specified by the user. The card starts with a balance of zero dollars.

- Make a Payment: decreases the balance owed on the card by the amount specified by the user.

- Get Card Balance: returns the balance owed on the card to the calling module.

- Get Available Balance: returns the amount of credit still available to the card holder. The credit limit is 500 dollars.

You do not need to worry about going over the limit or calculating interest.

For p3, you need to write the public section of this class.

YOU MUST INCLUDE ALL PURPOSE, PRE, AND POST CONDITIONS IN THE PUBLIC SECTION

Class Credit Card

public

~~function~~ Purchase returns a Num (n)
// Pre: Must pass in purchase amount
-3pre // Post: withdraws money
// Purpose: Returns the balance of the card -3pro

~~function~~ Payment returns a Num (n)
// Pre: Must pass in payment amount
// Post: Money in account goes up
// Purpose: to add to the credit card balance

function Owed returns a Num ()
// Pre: ~~None~~ Must have an account
// Post: Returns the amount of money ~~you~~ the user owes
// Purpose: to return the amount of money the user owes

function Availablecredit returns a Num ()
// Pre: must have an account
// Post: Returns the available credit
// Purpose: to let the user know how much money he can spend

procedure Initalize
// pre: must have a class
// post: initalizes the class
// Purpose: to initalize the class

Levi D. Smith

[p4]
35.0 points

For p4, write the protected section for the class described in p3.

You do not need to rewrite the comments from the public section.

```
protected
    balance isoftype Num
    CARDINIT is ∅
    CARDLIMIT is 500

    purchaseamount   isoftype Num
    depositamount    isoftype Num
    owed isoftype Num
    creditleft isoftype Num

    ~~function~~ Purchase ~~returns a~~ Num ()

        balance <- balance - purchase amount

        ~~Purchase returns~~ balance
    endfunction // Purchase

    function Payment ~~returns a Num~~ ()

        balance <- balance + depositamount
        Payment ~~returns~~ balance
    endfunction // Payment

    function Owed returns a Num()
        if (balance < 0) then
            owed <- ∅ - balance // make the negative balance positive since
            Balance returns owed // the user wants to know how much he OWES
        else
            Balance returns ∅ // if he has a positive amount on the card, then
        end if                // he doesn't owe anything
    end function

    function Availablecredit

        card limit <- 500 + balance // if the balance is negative, then it will be subtracted
    Availablecredit returns cardlimit // from 500
    end function

    procedure Initalize
        balance <- CARDINIT
    end procedure
end class
```

// I'm assuming that if he owes money
// he has a negative balance, and if he
// has extra money on the card he has
// a positive balance (relates to last
// two functions)

cs1501 96 Quiz 7 fall 1998 (Wednesday)

NAME : Levi D. Smith GTNUM : gtc187k TA: Irwin

Please be sure to fill in the above information correctly. Any error will lead to a 5 point penalty.

Note: Answers which are ambiguous and/or illegible in the judgment of the grader will be penalized. For partial credit, show the details of your work. Comments are not required on quizzes. However, comments might allow your TA to understand your code better, _possibly_ resulting in partial credit.

The use of the word module is comprehensive and can be interpreted as 'module or group of modules'. Don't feel restricted to a single procedure or function if the problem asks you to write a module.

You are going to be designing and implementing a class for a linked list that can hold any type of data. It will have the following functionality:

AddToEnd: Adds datatype to the end of the linked list.
ReturnFront: Returns the data that is located in the first node of the linked list. The node will not be deleted after the data is returned.

The following code is the given code for the public section of this class:

```
Class LinkedList (DataType isoftype Type)
public
        procedure AddToEnd(NewData isoftype in DataType)
        //purpose: will add the passed data to the end of the list
        //pre: head must be initialized
        //post: the passed data is added to the end of the list

        function ReturnFront returnsa DataType()
        //purpose: will return the data at the front of the list
        //pre: the list contains data
        //post: the data returned is the data at the front of the list

        procedure Initialize ()
        //purpose: initializes global variables
        //pre: none
        //post: none
```

Levi D. Smith

[p1]
30.0 points

Write the protected section for the class described above. Be sure to include any variables, types, helper modules and initializations needed.

YOU DO NOT NEED TO RECOPY THE COMMENTS IN THE PROTECTED SECTION

```
protected

  Node  defines a  Record
     data  isoftype  DataType
     next  isoftype  Ptr  toa Node
  end record
  currentptr  isoftype  Ptr  toa Node

  procedure  AddTo End  (New Data  isoftype  in  DataType)
       AddTo End 2  ( New Data , currentptr)
  end procedure
  procedure  AddTo End 2  ( Data  isoftype in DataType,  currentptr  isoftype  in/out  Ptr toa Node
// Purpose: Adds  new node  to  end  of  LL  // Pre: Something to be helped  // Post: It has helped
     temp  isoftype  Ptr  toa Node
     if (currentptr = NIL) then
     temp <-  new (Node)
     temp^.data <- Data
     temp^.next <- currentptr
     currentptr <- temp
     else
       AddTo End 2 ( Data, currentptr^.next)
     end if
  end procedure
  function  Return Front  returns a  DataType()
     Return Front  returns  Return Front 2 ( currentptr)   OK
  end function

  function  Return Front 2  return s a  DataType (currentptr)
// Purpose: helper function  // Pre: something to help  // Post: It has helped
     Return Front 2  returns  currentptr ^.data
  end function

  procedure  Initalize ( )
     currentptr <- NIL
  end procedure
```

Levi. D. Smith

[p2]
30.0 points

Now, write an algorithm that will use the class you created. It will be implementing a linked list of numbers. The algorithm will prompt the user with a menu:

1) Add item to the end of the linked list object.
2) Return item from the front of the linked list and print to user.
3) Quit

After an option is chosen and the task accomplished, this menu should be redisplayed until the Quit option is chosen.

```
algorithm Manage_List
uses Linked List

My Linked List isoftype Linked List (Num)


Menu ()

procedure Menu () // Pre: None // Post: Decision Made // Purpose: lets the user make decision
  print("Make your decision.")
  print(" 1. Add Item to the end of the linked list.")
  print(" 2. Return item from the front of the linked list and print to user.")
  print(" 3. Quit")

  choice isoftype num
    loop
      exit if (choice = 3)
        if (choice = 1) then
          Add Item ()
        elseif (choice = 2) then
          Print Front ()
        elseif (choice = 3)
          print("Exiting")
        else
          print("Wrong choice")
      endif
    endloop
end procedure
```

If you do it like this, need to pass in "My Linked List".

-4pts

CONTINUED

```
procedure Add Item ()
//Purpose: adds something to the linked list   // Pre: must have LL  //Post: added to LL
    data isoftype DataType
    print ("Enter data to add to linked list")
    read (data)
  MyLinkedList. Add To End (data)

  end procedure


  procedure Print Front ()
// Pre: Something to print //Post: Prints
//Purpose: prints out the data of the first node in the linked list
    print ("The First Node holds the value of ... ")
    print ( My Linked List. Return Front ())
    end procedure

end algorithm
```

Levi D. Smith

[p3]
40.0 points

For each of the following situations, give the Big-O and an explanation. ANSWERS WITHOUT EXPLANATIONS WILL RECIEVE NO CREDIT.

Note: For questions that refer to an ordered data structure, assume it is in ASCENDING order. - I'm assuming that they are ordered by the value we are looking for

a. Finding the smallest value in an ordered linked list with Q nodes.

O(1) - because the smallest value will be in the first node.

b. Finding a value in an ordered array of size L.

O(log L) - because you can keep dividing it until you find the value

c. Finding the largest value in a full and balanced binary tree of W nodes.

O(W) - because you have to search all nodes since it is not a BST

d. Find the Big-O of the following segment of code. You may assume that both A and B are positive integers.

```
.
.
.
i, j isoftype num
i <- A
j <- B
loop
  exitif (i < 1)
  loop
     exitif (j > B)
     MyArray [i,j] <- i*j
     j <- j + 1
  endloop
  i <- i - 1
endloop
.
.
.
```

O(A) - it takes A·1 times to run

It only takes one time for it to become greater than B since it starts starts as B

CS 1501 - Fall 1998

Skiles 270

Homework 1

126.0 Points

Due: Friday, October 2, 1998 - 9:00 A.M.

TURN IN THIS ASSIGNMENT ELECTRONICALLY USING TURNIN

BE SURE TO DOCUMENT ANY REQUIRED ESP INFORMATION (as noted for each problem)

FOLLOW THE CORRECT FORMAT WHEN SUBMITTING ALL HOMEWORKS

[p1]
16.0 points

List the four atomic data types, and give an example of a value each could hold.

[p2]
10.0 points

What is a constant? What is a variable? Describe when each should be used.

[p3]
20.0 points
REDOABLE ESP

For each of the following, provide a complete variable or constant declaration as appropriate.

a) The amount of money in your checking account
b) Your birthday
c) A letter grade for a class
d) Someone's social security number, including the dashes
   i.e 123-45-6789
e) Whether or not someone in college

[p4]
20.0 points
REDOABLE ESP

Write an algorithm to do the following:

1) Prompt the user for two numbers
2) Read in two numbers
3) Ask the user if they want the numbers averaged
4) Read in the user's choice
5) If the user wants them averaged, print the average

6) Otherwise, print the sum

[p5]
20.0 points

Briefly describe the following:

a) Formal parameters
b) Actual parameters
c) In parameters
d) Out parameters
e) In/Out parameters

[p6]
25.0 points
REDOABLE ESP

Write the headers for the following modules:
(DO NOT WRITE THE MODULES THEMSELVES!)

a) A module that takes in two numbers and returns their average.
b) A module that takes in two numbers and prints their average.
c) A module that takes in three numbers, and returns both their average and their sum.
d) A module that is passed two numbers, divides both numbers in half, and returns these two values.
e) A module that prompts the user to enter three numbers and returns their sum.

[p7]
15.0 points
REDOABLE ESP

Some pirates are worried that the seas may be too rough for them to sail today. They have decided that if the winds are greater than 50 MPH, then they will not sail. Write a module that takes in the current wind speed and returns whether or not the pirates may safely sail.

--
Hi.

```
#################################################################
# Name          : Levi Smith                                    #
# GTNum         : gtel87k                                       #
# Sections      : IW15a,A14                                     #
# TA Name       : Irwin Dolobowsky                              #
# Assignment    : hw1                                           #
# Size          : 2939 bytes                                    #
# Submitted     : 00:25:20(Fri)10-02-98                         #
# Original File : /fuse3/41/gtel87k/ldsmith1.txt                #
# Last Modified : 00:24:57(Fri)10-02-98                         #
#################################################################
```

64/66

[p1]

Numbers, Characters, Booleans, and Pointers

Number: 42
Character: 'c'
Boolean: TRUE
Pointer: Can hold the value of any of the other data types

[p2]

Constant - A constant is a value that remains the same throughout an algorithm.
- Constants make readability much easier and make some values easier to remember (PI is easier to remember than 3.14159265359).
- They should be used for a value that will not change in the algorithm.

Variable - A variable contains a value that can change throughout an algorithm.
- Variables are used for values that are read into the program or values that are dependent on the output of an operation.

[p3] 15 10 3 2

```
checking_account isoftype Num
my_birthday isoftype String
class_grade isoftype Char
soc_sec isoftype String
in_college isoftype Boolean
```

-3 cpu

[p4] 30 15 10 5

```
algorithm TwoNumbers
//Purpose:  to prompt the user for two numbers then
//either average or add the two numbers depending
//on user input
```



```
        //Variable Declarations
        value_one isoftype Num
        value_two isoftype Num
        answer isoftype Num
        user_choice isoftype Char
        sum_or_avg isoftype String

        //Begin Prompting User
        print ("Enter first number:")
        read (value_one)
        print ("Enter second number:")
        read (value_two)

        //Ask the user what shall be done with the
        //values then compute
        print ("Shall I average the two numbers?? Y/N")
        read (user_choice)
        if (user_choice = 'Y') then
                answer <- (value_one + value_two) / 2
                sum_or_avg <- "Average"
        else
                answer <- value_one + value_two
                sum_or_avg <- "Sum"
        endif

        print ("The ", sum_or_avg, " of the two numbers is ", answer)

endalgorithm //TwoNumbers
```

[p5]

[p6] 20 15 3 2

function Average returnsa Num (value_one isoftype in Num, value_two isoftype in Num)

procedure Print_Average (value_one isoftype in Num, value_two isoftype in Num)

procedure Avg_and_Sum (num_avg isoftype out Num, num_sum isoftype out Num, num_one isoftype in Num, num_two isoftype in Num, num_three isoftype in Num)

procedure Divide_by_Two (num_one isoftype in Num, num_two isoftype in Num, half_one isoftype out Num, half_two isoftype out Num)

procedure Sum_of_Three (num_one isoftype in Num, num_two isoftype in Num, num_three isoftype in Num, num_sum isoftype out Num)

[p7] 20 10 7 3

```
function Safe_to_Sail returnsa Boolean (wind_speed isoftype Num)
        SAFESPEED is 50
        if (wind_speed < SAFESPEED) then
                Safe_to_Sail returns TRUE
        else
                Safe_to_Sail returns FALSE
        endif
endfunction //Safe_to_Sail
```


[This page intentionally left non-blank]

```
#################################################################
# Name          : Levi Smith                                    #
# GTNum         : gte187k                                       #
# Sections      : IW15a,A14                                     #
# TA Name       : Irwin Dolobowsky                              #
# Assignment    : hw1r                                          #
# Size          : 351 bytes                                     #
# Submitted     : 03:29:51(Fri)10-09-98                         #
# Original File : /fuse3/41/gte187k/ldsmitr1.txt                #
# Last Modified : 03:29:08(Fri)10-09-98                         #
#################################################################
```

[p1]

[p2]

[p3] 15 10 3 2

checking_account isoftype Num
MY_BIRTHDAY is
class_grade isoftype C
SOC_SEC is 123-45-6789
in_college isoftype Boolean

-1pt

[p4]

[p5]

[p6]

[p7]

ESP 40 min

CS 1501 - Fall 1998

Homework 2

135.0 Points

Due: Friday, October 9, 1998 - 9:00am

TURN IN THIS ASSIGNMENT ELECTRONICALLY USING TURNIN

BE SURE TO DOCUMENT ANY REQUIRED ESP INFORMATION (as noted for each problem)

FOLLOW THE CORRECT FORMAT WHEN SUBMITTING ALL HOMEWORKS

[p1]
20.0 points

In your own words, briefly describe the following:

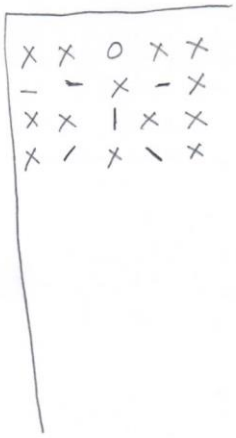
a) Formal parameters
b) Actual parameters
c) In parameters
d) Out parameters
e) In/Out parameters

[p2]
20.0 points

Give the output of the following algorithm:

```
procedure Confusion(A, B isoftype in char
                    C, D isoftype in/out char
                    E isoftype out char)

   B <- A
   D <- '-'
   D <- B

   Print(A, B, C, D, E)

   D <- 'X'
   E <- '|'
endprocedure

function Vague returnsa char (A isoftype in char)

   if (A = '|') then
        Vague returns '\'
   else
        Vague returns '/'
   endif
```

```
endfunction

algorithm Trace

   A, B, C, D, E isoftype char

   A <- 'X'
   B <- 'X'
   C <- 'O'
   D <- 'X'
   E <- 'X'

   Print (A, B, C, D, E)
   Confusion(B, D, E, A, C)
   Print (A, B, C, D, E)
   B <- Vague(D)
   D <- Vague(C)
   C <- A
   Print (A, B, C, D, E)

endalgorithm
```



[p3]
25.0 points
REDOABLE ESP

Write the headers for the following modules:
(DO NOT WRITE THE MODULES THEMSELVES!)

a) A module that takes in two numbers and returns their average.
b) A module that takes in two numbers and prints their average.
c) A module that takes in three numbers, and returns both their average and their sum.
d) A module that is passed two numbers, divides both numbers in half, and returns these two values.
e) A module that prompts the user to enter three numbers and returns their sum.

[p4]
10.0 points
REDOABLE ESP

Write a module that prompts the user to enter three grades, and returns the average.

[p5]
15.0 points
REDOABLE ESP

Write a module that is passed a lab average, a test average, and a final exam grade, and returns the overall average. The three grades have the following weights:

Labs are 20% of the overall average
Tests are 30% of the overall average
The final exam is 50% of the overall average

[p6]
15.0 points
REDOABLE ESP

Write a module that is passed an overall grade and returns the corresponding letter grade. Assume 90% for an A, 80% for a B, and so on.

[p7]
15.0 points
REDOABLE ESP

Write an algorithm to calculate and print out a student's letter grade. The student has a total of seven grades: three lab grades, three test grades, and a final exam grade. Be sure to prompt the user for any necessary values, and to use good modularity.

[p8]
15.0 points
REDOABLE ESP

Write a recursive module called Falling that is passed a number. This module will then print out all the numbers between the one passed in and zero, inclusively and in reverse order. For example, Falling(3) would print out:

```
3
2
1
0
```

```
#############################################################
# Name          : Levi Smith                                #
# GTNum         : gte187k                                   #
# Sections      : IW15a,A14                                 #
# TA Name       : Irwin Dolobowsky                          #
# Assignment    : hw2                                       #
# Size          : 5109 bytes                                #
# Submitted     : 03:25:02(Fri)10-09-98                     #
# Original File : /fuse3/41/gte187k/ldsmith2.txt            #
# Last Modified : 03:23:56(Fri)10-09-98                     #
#############################################################
```

130

[p1]

a) A formal parameter is the name that is given to a variable once it enters a module.

b) An actual parameter is the name of a variable that is used in the main algorithm

c) An In parameter lets the data from a variable in an algorithm enter a module. Any change to the variable in the module will not be sent back to the main algorithm.

d) An out parameter only allows changes to a variable in a procedure to be sent back to the main algorithm. Values for variables of this type cannot be read form the main algorithm.

e) An In/Out parameter lets values for a variable to be read from the main algorithm to a module, and they also can be sent back to the main algorithm from a procedure.

[p2]

```
X X O X X
X - X - X
X X | X X
X / X \ X
```

[p3] 5 10 10 3

```
function Average returnsa Num (value_one, value_two isoftype in Num)
//This function returns the average of two numbers

procedure Print_Average (value_one, value_two isoftype in Num)
//This procedure prints the average of two numbers

procedure Avg_and_Sum (num_one, num_two, num_three isoftype in Num,
   num_avg, num_sum isoftype out Num)
//This procedure sends out the average and sum of three numbers
```



```
procedure Divide_by_Two (num_one, num_two isoftype in Num, half_one,
   half_two isoftype out Num)
//This procedure takes in two numbers then passes out both numbers
//divided by two

procedure Sum_of_Three (num_one, num_two, num_three isoftype in Num,
   num_sum isoftype out Num)
//This procedure prompts the user for three numbers and then passes
//out the sum of these three numbers
```

[p4] 5 10 5 3

```
procedure Grade_Average (test_or_lab isoftype in String, grade_avg isoftype
   out Num)
//Purpose: to read in three grades and return their average
//Note: the 'test_or_average' variable is needed for [p7]

   //Local Variables: The three grades
   grade_one, grade_two, grade_three isoftype Num
   NUMBER_OF_GRADES is 3

   print ("Enter the first ", test_or_lab, " grade.")
   read (grade_one)
   print ("Enter the second ", test_or_lab, " grade.")
   read (grade_two)
   print ("Enter the third ", test_or_lab, " grade.")
   read (grade_three)

   grade_avg <- (grade_one + grade_two + grade_three) / NUMBER_OF_GRADES
end procedure //Grade_Average
```

[p5] 5 15 7 2

```
function Overall_Avg returnsa Num (avg_lab, avg_test, exam isoftype in Num)
//Purpose:  takes in three averages, weights them, and returns an overall
//average

   //The weighting of each average
   WEIGHTLAB is 0.2
   WEIGHTTEST is 0.3
   WEIGHTEXAM is 0.5

   Overall_Avg returns (avg_lab * WEIGHTLAB) + (avg_test * WEIGHTTEST) +
      (exam * WEIGHTEXAM)
endfunction //Overall_Avg
```

[p6] 5 15 7 5

```
function Letter_Grade returnsa Char (number_grade isoftype in Num)
//Purpose:  takes in a number grade and returns the equivalent letter grade
```

```
   //Lowest possible value for each grade
   A_LOW is 90
   B_LOW is 80
   C_LOW is 70
   D_LOW is 60
   result isoftype Char //temporarily holds letter grade

   if (number_grade >= A_LOW) then
      result <- 'A'
   elseif (number_grade < A_LOW AND number_grade >= B_LOW) then
      result <- 'B'
   elseif (number_grade < B_LOW AND number_grade >= C_LOW) then
      result <- 'C'
   elseif (number_grade < C_LOW AND number_grade >= D_LOW) then
      result <- 'D'
   else (number_grade < D_LOW) then
      result <- 'F'
   endif

   Letter_Grade returns result

endfunction //Letter_Grade


[p7] 5 15 10 5

algorithm Get_Grade
//Purpose:  to calculate and print a letter grade from three lab grades,
//three test grades, and a final exam grade

//Variables:
//grade_avg is used to hold the result of Grade_Average procedure
//avg_test and avg_lab hold the averages of the three grades from
   //Grade_Average
//exam is the final exam grade
//final_num is used to hold the final numerical grade so it can be passed
   //to function Letter_Grade
//final_char is used to hold the final letter grade so it can be printed
//test_or_lab is used in Grade_Average's prompts

grade_avg, avg_test, avg_lab, exam, final_num isoftype Num
final_char isoftype Char
test_or_lab isoftype String

//Find out what the test average is

//The next line is used by the procedure's prompt so the user will know
//wether a test or lab grade is being entered (see p4)
test_or_lab <- "test"
Grade_Average (test_or_lab, grade_avg)
test_avg <- grade_avg

//Find out what the lab average is
test_or_lab <- "lab"
Grade_Average (test_or_lab, grade_avg)
lab_avg <- grade_avg
```



```
print ("Enter the final exam grade")
read (exam)

final_num <- Overall_Avg (avg_lab, avg_test, exam)
final_char <- Letter_Grade (final_num)
print ("The final grade for this class is a(n)", final_char)

end algorithm //Get_Grade


[p8] 10 5 5 2

procedure Falling returnsa Num (number isoftype in Num)
//Purpose:  Prints out numbers descending from a passed in number to 0

   if (number = 0) then
      print (number)
   else
      print (number)
      Falling (number - 1)
   endif
endprocedure //Falling
```

```
#############################################################
# Name           : Levi Smith                               #
# GTNum          : gte187k                                  #
# Sections       : IW15a,A14                                #
# TA Name        : Irwin Dolobowsky                         #
# Assignment     : hw2r                                     #
# Size           : 963 bytes                                #
# Submitted      : 02:32:22(Fri)10-16-98                    #
# Original File  : /fuse3/41/gte187k/ldshw02r.txt           #
# Last Modified  : 02:31:46(Fri)10-16-98                    #
#############################################################

[p1]

[p2]

[p3] 5 3 2 1

function Average returnsa Num (value_one, value_two isoftype in Num)
//This function returns the average of two numbers

procedure Print_Average (value_one, value_two isoftype in Num)
//This procedure prints the average of two numbers

procedure Avg_and_Sum (num_one, num_two, num_three isoftype in Num,
   num_avg, num_sum isoftype out Num)
//This procedure sends out the average and sum of three numbers


//The Redoable problems start here

procedure Divide_by_Two (num_one, num_two isoftype in/out Num)
//This procedure takes in two numbers then passes out both numbers
//divided by two

procedure Sum_of_Three (num_sum isoftype out Num)
//This procedure prompts the user for three numbers and then passes
//out the sum of these three numbers

[p4]

[p5]

[p6]

[p7]

[p8]
```


[This page intentionally left non-blank]

```
#############################################################
# Name           : Levi Smith                               #
# GTNum          : gte187k                                  #
# Sections       : IW15a,A14                                #
# TA Name        : Irwin Dolobowsky                         #
# Assignment     : hw3                                      #
# Size           : 4332 bytes                               #
# Submitted      : 02:22:28(Fri)10-16-98                    #
# Original File  : /fuse3/41/gte187k/ldshw03.txt            #
# Last Modified  : 02:21:32(Fri)10-16-98                    #
#############################################################
```

132/140

[p1] 5 1 2 2

```
Intrepid_Code definesa record
        encoded_message isoftype String
        numeric_key isoftype Num
        this_month isoftype Num
        this_day isoftype Num
        this_year isoftype Num
endrecord //Intrepid_Code
```

[p2] 20 5 5 2

```
function Decoding_Key returnsa Boolean (key isoftype in Num, New_Code
   isoftype in Intrepid_Code)
//Purpose: After a sample key and a record is passed into the function, the
//function returns wether the two keys are the same or not.

     if (key = New_Code.numeric_key) then
          Decoding_Key returns TRUE
     else
          Decoding_Key returns FALSE
     endif
endfunction //Decoding_Key
```

[p3]

A pointer only holds the address of another data cell.

[p4] 15 5 8 2

```
PirateRec definesa record
     pirate_name isoftype String
     date_joined isoftype Date
     pirate_rank isoftype String
     segment_num isoftype Num
     pirate_dead isoftype Boolean
endrecord //PirateRec
```



```
Date definesa record
     month isoftype Num
     day isoftype Num
     year isoftype Num
endrecord //Date
```

[p5] 7 3 2 2

```
PirateNode definesa record
     data isoftype PirateRec
     next isoftype Ptr toa PirateNode
endrecord //PirateNode

nodepointer isoftype Ptr toa PirateNode
nodepointer <- NIL
```

[p6] 20 3 13 3

```
procedure PrintPirates (pirate_data isoftype in PirateRec)
//Purpose:  To take in the PirateRec record and print all of the data which
//is contained within it.

     print ("This pirate's name is ", pirate_data.pirate_name)
     //Slashes are used to separate the date entries.  A sample output would
     //be: This pirate joined on 3 / 20 / 79
     print ("This pirate joined on ", pirate_data.date_joined.month, " / ",
          pirate_data.date_joined.day, " / ", pirate_data.date_joined.year)
     print ("This pirate's rank is ", pirate_data.pirate_rank)
     print ("This pirate's tatooed map number is ", pirate_data.segment_num)
     print ("This pirate is dead. ", pirate_data.pirate_dead)

endprocedure //Print Pirates
```

sel. pro.

-2 pp

-3 pa

[p7] 20 8 8 3

```
procedure Add_A_Pirate (pirate_data isoftype in PirateRec, nodepointer
     isoftype in/out Ptr toa PirateRec)
//Purpose:  this procedure is passed in a PirateRec and a pointer to the
//head of the list, and with that data it will successfully put the PirateRec
//data in the correct spot in the list.

     temp isoftype Pointer toa PirateRec

     if ((nodepointer = NIL) OR (nodepointer^.data.segment_num > pirate_data.
        segment_num)) then
```

```
        temp <- new(PirateRec)
        temp^.data <- pirate_data
        temp^.next <- nodepointer
        nodepointer <- temp
    else
        Add_A_Pirate (pirate_data, nodepointer^.next)
    endif
endprocedure //Add_A_Pirate
```

[p8] 20 7 5 3

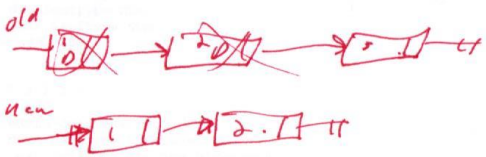
```
procedure Remove_A_Pirate (dead_pirate_name isoftype in String,
    nodepointer isoftype in/out Ptr toa PirateRec)
//Purpose:  This procedure is passed in a dead pirate name and a pointer
//to the head of this list, so it sorts through all of the names and
//removes the dead pirate.

    if (NOT (nodepointer = NIL))
        if (nodepointer^.data.pirate_name = dead_pirate_name) then
            nodepointer <- nodepointer^.next
        else
            Remove_A_Pirate (dead_pirate_name, nodepointer^.next)
        endif
    endif
endprocedure //Remove_A_Pirate
```

[p9] 25 40 20 10

```
procedure Two_Pirate_Lists (oldnodepointer isoftype in/out Ptr toa PirateRec,
    newnodepointer isoftype in/out Ptr toa PirateRec)
//Purpose:  To put all of the dead pirate data in the list with the new
//pointer, and to remove all of the dead pirates from the old list.

    //I'm trying to make the pointer follow down the list until it finds a
    //dead pirate.  When it finds a dead pirate, it sends it to the new
    //list (hopefully)

    if ((oldnodepointer = NIL) OR (oldnodepointer^.data.pirate_dead =
        TRUE)) then

        newnodepointer <- new(PirateRec)
        newnodepointer^.data <- oldpointer^.data
        oldpointer <- oldpointer^.next
    else
        Two_Pirate_Lists (oldnodepointer^.next, newnodepointer)
    endif

endprocedure //Two_Pirate_Lists
```

-10

[This page intentionally left non-blank]



if (old <> NIL) then
    if (old ^.data.pirate_dead = TRUE) then
        Add_Pirate (old ^.data, new)
        Remove_A_Pirate (old^.data.name, old)

else

endif

```
#################################################################
# Name           : Levi Smith                                   #
# GTNum          : gte187k                                      #
# Sections       : IW15a,A14                                    #
# TA Name        : Irwin Dolobowsky                             #
# Assignment     : hw3r                                         #
# Size           : 1975 bytes                                   #
# Submitted      : 02:55:14(Fri)10-23-98                        #
# Original File  : /fuse3/41/gte187k/ldshw03r.txt               #
# Last Modified  : 02:54:54(Fri)10-23-98                        #
#################################################################
```

[p1]

[p2]

[p3]

[p4]

[p5]

[p6] 20 3 13 3

```
procedure PrintPirates (pirate_data isoftype in PirateRec)
//Purpose:  To take in the PirateRec record and print all of the data which
//is contained within it.

     print ("This pirate's name is ", pirate_data.pirate_name)
     PrintDate (pirate_data)
     print ("This pirate's rank is ", pirate_data.pirate_rank)
     print ("This pirate's tatooed map number is ", pirate_data.segment_num)
     print ("This pirate is dead. ", pirate_data.pirate_dead)

endprocedure //Print Pirates

procedure PrintDate (pirate_data isoftype in PirateRec)
//Purpose: print the date in the PirateRec; added this procedure to make it
//more modular

     //Slashes are used to separate the date entries.  A sample output would
     //be: This pirate joined on 3 / 20 / 79

     print ("This pirate joined on ", pirate_data.date_joined.month, " / ",
            pirate_data.date_joined.day, " / ", pirate_data.date_joined.year)
endprocedure //PrintDate
```

[p7]

[p8]

[p9] 25 40 20 10


```
procedure Two_Pirate_Lists (oldnodepointer isoftype in/out Ptr toa PirateRec,
     newnodepointer isoftype in/out Ptr toa PirateRec)
//Purpose:  To put all of the dead pirate data in the list with the new
//pointer, and to remove all of the dead pirates from the old list.

     //The procedures, Add_A_Pirate and Remove_A_Pirate, are from [p7] and
     //[p8] on HW3, respectively

     if (oldnodepointer <> NIL) then
        if(oldnodepointer^.data.pirate_dead =TRUE) then
           Add_A_Pirate (oldnodepointer^.data, newnodepointer)
           Remove_A_Pirate (oldnodepointer^.data.name, oldnodepointer)
           Two_Pirate_Lists (oldnodepointer^.next, newnodepointer)
     else
           Two_Pirate_Lists (oldnodepointer^.next, newnodepointer)

     endif

endprocedure //Two_Pirate_Lists
```

Levi D. Smith

CS 1501 - Fall 1998

Homework 4

160.0 Points

Due: Friday, October 23, 1998 - 9:00am

TURN IN THIS ASSIGNMENT ELECTRONICALLY USING TURNIN

BE SURE TO DOCUMENT ANY REQUIRED ESP INFORMATION (as noted for each problem)

FOLLOW THE CORRECT FORMAT WHEN SUBMITTING ALL HOMEWORKS

After their fierce sea battle, the pirates needed to hire many new crew members. One new crew member, Inept Melvin, was placed in charge of the linked list which stored the segments of the treasure map. Unfortunately, Inept Melvin accidentally deleted the entire list (during a fit of ineptitude). Thorough Bill takes over the project of rebuilding the list.

Declare a Pirate tree Node

[p1]
REDOABLE ESP
15.0 points

Thorough Bill decides that searching a linked list to find segments of the map is too inefficient. He decides to store the information in a binary search tree instead. Declare a data structure called PirateTreeNode that will be a binary search tree of PirateRecs (see hw3). Be sure to declare any necessary data types.

ADD Pirate Rec to tree

[p2]
15.0 points
REDOABLE ESP

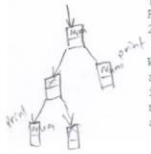
Write a module that will add a new (and already filled) PirateRec to the binary search tree structure you declared in p1. The tree should be sorted by map ID number.

Print all pirate names in tree

[p3]
REDOABLE ESP
20.0 points

Thorough Bill needs to be able to print information about the pirates. Write a module that will be passed a pointer to your binary search tree, and will print out the name of every pirate in the tree. Be sure to use good modularity, and you may re-use code from other homeworks.

any order



[p4]
REDOABLE ESP
20.0 points

Write a module that will be passed a pointer to your binary search tree, and map ID. The module should search through the tree and print out all information for any node with the same map ID. If a map with the given ID number is not in the tree, your module should print an error message. Be sure to use good modularity.

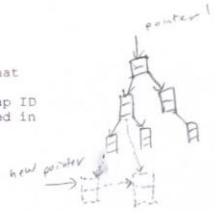
REDOABLE ESP
20.0 points

Inept Melvin accidentally included a erroneous map segment in the tree. However, it turns out that this map segment has the lowest ID number. Write a module that will delete the node with the smallest map ID number from a binary search tree of PirateRecs.

Delete Smallest Node

[p6]
REDOABLE ESP
30.0 points

Unfortunately, Thoughtless Earl wrote a few modules that require that the PirateRecs be stored as a linked list. Write a module that is passed a pointer to a binary seach tree of PirateRecs (sorted by map ID number), and returns a pointer to a linked list of PirateRecs sorted in ascending order. You may re-use your linked list from homework 3.



[p7]
30.0 points

Given the following tree:

What would be the output of the following:

A) Pre-order traversal 10, 20, 40, 70, 50, 30, 60, 80, 90
B) Inorder traversal 40, 70, 20, 50, 10, 30, 80, 60, 90
C) Post-order traversal 70, 40, 50, 20, 80, 90, 60, 30, 10

[p8]
REDOABLE ESP
10.0 points

Declare an array that holds exactly 30 numbers.

ARRAY MAX is 30
ARRAY MIN is 1

My_Array defines a array [ARRAY MIN..ARRAY MAX] of Num

```
#################################################################
# Name          : Levi Smith                                    #
# GTNum         : gte187k                                       #
# Sections      : IW15a,A14                                     #
# TA Name       : Irwin Dolobowsky                              #
# Assignment    : hw4                                           #
# Size          : 5688 bytes                                    #
# Submitted     : 02:54:13(Fri)10-23-98                         #
# Original File : /fuse3/41/gte187k/ldshw04.txt                 #
# Last Modified : 02:53:44(Fri)10-23-98                         #
#################################################################
```

160/160

```
[p1] 15 5 5 2

PirateTreeNode definesa record
   data isoftype PirateRec
   left_child isoftype Ptr toa PirateTreeNode
   right_child isoftype Ptr toa PirateTreeNode
endrecord //PirateTreeNode


//Other necessary data types
PirateRec definesa record
     pirate_name isoftype String
     date_joined isoftype Date
     pirate_rank isoftype String
     pirate_id isoftype Num
     pirate_dead isoftype Boolean
endrecord //PirateRec

Date definesa record
     month isoftype Num
     day isoftype Num
     year isoftype Num
endrecord //Date


[p2] 20 10 5 5

procedure Add_Node_to_BST (currentptr isoftype Ptr toa PirateTreeNode,
   PirateRec_to_Add isoftype in PirateRec)
//Purpose:  this procedure will take in add a filled PirateRec and add
//it to the BST in the proper spot.  The new data will only be added at a
//NIL pointer (as according to instruction.

   if (currentptr = NIL) then
      currentptr <- new(PirateRec)
      currentptr^.data <- PirateRec^.data
```



```
      currentptr^.left_child <- NIL
      currentptr^.right_child <- NIL
   elseif (currentptr^.data.pirate_id > PirateRec_to_Add^.data.pirate_id)
      then
      Add_Node_to_BST (currentptr^.left_child, PirateRec_to_Add)
   else
      Add_Node_to_BST (currentptr^.right_child, PirateRec_to_Add)
   endif
endprocedure //Add_Node_to_BST

[p3] 15 10 5 5

procedure FindPirateNames (currentptr isoftype in/out Ptr toa
PirateTreeNode)
//Purpose:  this procedure will take in a pointer to the BST and will
//find all of the pirate names.
//The names will be printed with the helping procedure
//BTW: I will be finding (and then printing)  the names 'In Order'

   if (currentptr <> NIL) then
      FindPirateNames (currentptr^.left_child)
      PrintPirateName (currentptr)
      FindPirateNames (currentptr^.right_child)
   endif
endprocedure //FindPirateNames

procedure PrintPirateName (currentptr isoftype in Ptr toa PirateTreeNode)
//Purpose:  the procedure will print the pirate name that was found by the
//procedure FindPirateNames.

   print (currentptr^.data.pirate_name)
endprocedure //PrintPirateName


[p4] 20 15 10 10

procedure FindPirateID (currentptr isoftype in/out Ptr toa PirateTreeNode,
MapID isoftype in Num)
//Purpose:  to find the record of the pirate who's ID has been passed in and
//print it with the helping procedure which was made in HW3
//Prints error message if not found

   if (currentptr = NIL) then
      print ("Error:  PirateRec not found")
   elseif (currentptr^.data.pirate_id = MapID) then
      PrintPirates (currentptr^.data)
   elseif (MapID < currentptr^.data.pirate_id) then
      FindPirateID (currentptr^.left_child, MapID)
   else
      FindPirateID (currentptr^.right_child, MapID)
   endif
endprocedure //FindPirateID
```

```
//from HW3
procedure PrintPirates (pirate_data isoftype in PirateRec)
//Purpose:  To take in the PirateRec record and print all of the data which
//is contained within it.

     print ("This pirate's name is ", pirate_data.pirate_name)
     PrintDate (pirate_data)
     print ("This pirate's rank is ", pirate_data.pirate_rank)
     print ("This pirate's tatooed map number is ", pirate_data.segment_num)
     print ("This pirate is dead. ", pirate_data.pirate_dead)

endprocedure //Print Pirates

procedure PrintDate (pirate_data isoftype in PirateRec)
//Purpose: print the date in the PirateRec
     //Slashes are used to separate the date entries.  A sample output would
     //be: This pirate joined on 3 / 20 / 79
     print ("This pirate joined on ", pirate_data.date_joined.month, " / ",
          pirate_data.date_joined.day, " / ", pirate_data.date_joined.year)
endprocedure //PrintDate
```

[p5] 10 5 3 5

```
procedure DeletePirateTreeNode (currentptr isoftype in/out Ptr toa
   PirateTreeNode)
//Purpose:  this procedure goes through a BST, finds the Node with the
//lowest ID number, then deletes it

   if (currentptr <> NIL) then
      if (currentptr^.left_child = NIL) then
         currentptr <- currentptr^.right_child
      else
         DeletePirateTreeNode (current^.left_child)
      endif
   endif
endprocedure //DeletePirateTreeNode
```

[p6] 30 15 10 10

```
procedure MakeListFromTree (treeptr isoftype in/out Ptr toa
   PirateTreeNode, listptr isoftype in/out Ptr toa PirateListNode)
//Purpose:  to search through a BST and add all entries to a linked
//list (we are assuming that we are given a pointer to a linked list
//which is set to NIL)

  if (treeptr^.data.pirate_id <> NIL) then
        MakeListFromTree (treeptr^.left_child, listptr)
        Add_A_Pirate (treeptr^.data, listptr)
        MakeListFromTree (treeptr^.right_child, listptr)
  endif
endprocedure //MakeListFromTree
```



```
//from HW3
procedure Add_A_Pirate (pirate_data isoftype in PirateRec, nodepointer
     isoftype in/out Ptr toa PirateRec)
//Purpose:  this procedure is passed in a PirateRec and a pointer to the
//head of the list, and with that data it will successfully put the PirateRec
//data in the correct spot in the list.
   temp isoftype Pointer toa PirateRec
   if ((nodepointer = NIL) OR (nodepointer^.data.segment_num > pirate_data.
      segment_num)) then
      temp <- new(PirateRec)
      temp^.data <- pirate_data
      temp^.next <- nodepointer
      nodepointer <- temp
   else
      Add_A_Pirate (pirate_data, nodepointer^.next)
   endif
endprocedure //Add_A_Pirate
```

[p7]

A) 10, 20, 40, 70, 50, 30, 60, 80, 90
B) 40, 70, 20, 50, 10, 30, 80, 60, 90
C) 70, 40, 50, 20, 80, 90, 60, 30, 10

[p8] 1 2 1 1

ARRAYMAX is 30
ARRAYMIN is 1

My_Array definesa array [ARRAYMIN..ARRAYMAX] of Num

CS 1501 - Fall 1998

Homework 5

145.0 Points

Due: Friday, October 30, 1998 - 9:00am

TURN IN THIS ASSIGNMENT ELECTRONICALLY USING TURNIN

BE SURE TO DOCUMENT ANY REQUIRED ESP INFORMATION (as noted for each problem)

FOLLOW THE CORRECT FORMAT WHEN SUBMITTING ALL HOMEWORKS

[p1]
REDOABLE ESP
10.0 points

Declare the necessary data type to hold an array of 100 numbers.

[p2]
REDOABLE ESP
20.0 points

Write a module that fills and returns an array of the type declared in problem 1. The array should be filled with the numbers from 100 down to 1. In other words, the first array slot should have 100, the second 99, and so on until the last one is filled with 1.

[p3]
REDOABLE ESP
15.0 points

Write a module that takes in an array of the type from problem 1 and prints out all the values in the array that are less than 50.

[p4]
REDOABLE ESP
10.0 points

Declare the necessary data type to hold a two dimensional array of strings. The array will be used by a theatre to keep track of the customer name associated with each seat. There are 70 rows, 50 seats per row. There will be only one customer per seat.

For example, Row 1 Seat 1 would be reserved for Scott, and Row 70 Seat 50 would be reserved for Thoughtless Earl.

[p5]
REDOABLE ESP
20.0 points

Write a module that when passed a customer name and an array of theatre seats as defined in problem 4, prints out that customer's seat. For example, if passed "Scott", it will print out "Row 1, Seat 1".

You may assume that every name in the array is unique.

[p6]
20.0 points

Given the following:

1, 9, 18, 4, 33, 5, 6

What would be the output after one pass of the OUTER loop of Bubblesort?

[p7]
10.0 points

Briefly describe both Mergesort and Bubblesort.

[p8]
REDOABLE ESP
40.0 points

A television set has the following properties:

A) The volume can be set in a range from 0 to 50. 50 is the loudest, and 0 indicates no sound. When the TV is first turned on, the volume is set to 20.

B) The channels range from 3 to 99. If the TV is on channel 99 and the channel is increased by 1, it wraps around to 3. Conversely, if the TV is on channel 3 and the channel is decreased by 1, it wraps around to 99.

Your task is to write a class that implements this television set. In particular, you must implement the following methods:

1) A method that increments the volume.
2) A method that decrements the volume.
3) A method that implements the mute button.
4) A method that increments the channel.
5) A method that decrements the channel.

```
#################################################################
# Name          : Levi Smith                                    #
# GTNum         : gte187k                                       #
# Sections      : IW15a,A14                                     #
# TA Name       : Irwin Dolobowsky                              #
# Assignment    : hw5                                           #
# Size          : 5439 bytes                                    #
# Submitted     : 03:49:59(Fri)10-30-98                         #
# Original File : /fuse3/41/gte187k/ldshw05.txt                 #
# Last Modified : 03:48:36(Fri)10-30-98                         #
#################################################################
```

130/145

[p1] 2 1 1 1

```
ARRAYMAX is 100
problem_one_array definesa array [1..ARRAYMAX] of num
```

[p2] 10 5 3 2

fun returns prob_one_array

```
procedure Fill_Array (array_to_fill isoftype in/out problem_one_array)
//Fills in an array with numbers from 100 through 1
```

array_to_fill isn't prob...

```
   //MAGICNUMBER is the number that variable i is subtracted from to get
   //desired cell data.  EX: cell one gets 100 (101 - 1), cell two gets
   //99 (101 - 2), etc.
   MAGICNUMBER is 101
   i isoftype num
   i <- 1

   loop
      exitif (i > ARRAYMAX)
      array_to_fill[i] <- MAGICNUMBER - 1
      i <- i + 1
   endloop
endprocedure //Fill_Array
```

set array_to_fill

-3 fun

-doc

[p3] 12 5 5 2

```
procedure Print_LessThan_50 (array_to_print isoftype in problem_one_array)
//This procedure prints all of the numbers in the array that are less than
//50

   i isoftype num
   i <- 1

   loop
      exitif (i > ARRAYMAX)
         if (array_to_print[i] < 50) then
            print (array_to_print[i])
         endif
      i <- i + 1
   endloop
endprocedure //Print_LessThan_50
```


[p4] 2 1 1 1

```
ROWMAX is 70
SEATMAX is 50

Row_Array definesa array[1..ROWMAX] of string
Seat_Array definesa array [1..SEATMAX] of Row_Array
```

[p5] 20 10 5 5

```
procedure Print_Out_Seat (search_array isoftype in Seat_Array, name
   isoftype in string)
//This procedure is passed in a Seat_Array and a name.  It then finds the
//name in the Seat_Array and prints out the person's row and seat.

   i, j isoftype num
   i <- 1

   loop
      //Exits the loop if it does not find the name
      //after searching through all of the rows
      exitif (i > ROWMAX)
         j <- 1

         loop
         //Exits the loop if it runs out of seats to
         //search, or prints the seat number if it finds the name
            exitif (i > SEATMAX)

            if (name = search_array[i][j])
               print (name, "'s seat is Row ", i, " Seat ", j)
            endif

            j <- j + 1
         endloop

      i <- i + 1
   endloop

endprocedure //Print_Out_Seat
```

[p6]

1, 9, 4, 18, 5, 6, 33

[p7]

ubble Sort takes the first two numbers and puts or leaves the larger number in the second number position. Then it takes the second and third number and puts or leaves the larger number in the third number position. It continues this process until it gets to the end of the array. Overall, the largest number ends up (or "bubbles") to the last position. This process must be repeated over and over if all of the numbers in the array are to be in the correct order.

Merge Sort is where the array is divided in half (making two subarrays), and the two subarrays are divided into half (making four subarrays), and this process continues each subarray only contians one cell. Then the first smallest subarray and second smallest array are compared and put in correct order. They then reform (or "merge") into an subarray of two cells. This continues until all of the subarrays containing only one cell are merged. Then the first subarray of two cells and the second subarray of two cells are compared, and put into correct order. This process is continued for the rest of the arrays of two cells are merged. Then all of the other layers are merged according to this pattern until only one array remains. If there is an off number of cells (not a power of 2) then extra cells are added with the value of 0 so Merge Sort can work properly (of corse the 0's are removed once you get your final array).

[p8] 18 10 5 3

```
class TelevisionControls

   public
      procedure VolumeUp (volume isoftype in/out num)
      //increases the volume by one

      procedure VolumeDown (volume isoftype in/out num)
      //decreases the volume by one

      procedure Mute (volume isoftype in/out num)
      //sets the volume to zero

      procedure ChannelUp (channel isoftype in/out num)
      //increases the channel by one

      procedure ChannelDown (channel isoftype in/out num)
      //decreases the channel by one

      procedure initalize (volume isoftype in/out num)
      //initializes the television when it is turned on

   private
      variable :=
      procedure VolumeUp (volume isoftype in/out num)
      //increases the volume by one
         if (volume < 50)
            volume <- volume + 1
         endif
      endprocedure //VolumeUp
```

-5 pro

-5 con


```
      procedure VolumeDown (volume isoftype in/out num)
      //decreases the volume by one
         if (volume > 0)
            volume <- volume - 1
         endif
      endprocedure //VolumeDown

      procedure Mute (volume isoftype in/out num)
      //sets the volume to zero
         volume <- 0
      endprocedure //Mute

      procedure ChannelUp (channel isoftype in/out num)
      //increases the channel by one
         if (channel < 99) then
            channel <- channel + 1
         else
            channel <- 3
         endif
      endproedure //ChannelUp

      procedure ChannelDown (channel isoftype in/out num)
      //decreases the channel by one
         if (channel > 3) then
            channel <- channel - 1
         else
            channel <- 99
         endif
      endproedure //ChannelDown

      procedure initalize (volume isoftype in/out num)
      //initializes the television when it is turned on
         volume <- 20
      enprocedure //initalize

endclass //Television Controls
```

```
#############################################################
# Name          : Levi Smith                                #
# GTNum         : gte187k                                   #
# Sections      : IW15a,A14                                 #
# TA Name       : Irwin Dolobowsky                          #
# Assignment    : hw5r                                      #
# Size          : 2591 bytes                                #
# Submitted     : 01:46:20(Fri)11-06-98                     #
# Original File : /fuse3/41/gte187k/ldshw05r.txt            #
# Last Modified : 01:45:50(Fri)11-06-98                     #
#############################################################

[p1]

[p2] 10 5 3 2

function Fill_Array returnsa problem_one_array ()
//Fills in an array with numbers from 100 through 1

   //MAGICNUMBER is the number that variable i is subtracted from to get
   //desired cell data.  EX: cell one gets 100 (101 - 1), cell two gets
   //99 (101 - 2), etc.
   MAGICNUMBER is 101
   i isoftype num
   array_to_fill isoftype problem_one_array

   i <- 1

   loop
      exitif (i > ARRAYMAX)
      array_to_fill[i] <- MAGICNUMBER - i
      i <- i + 1
   endloop

Fill_Array return array_to_fill
endfunction //Fill_Array

[p3]
[p4]
[p5]
[p6]
[p7]

[p8] 18 10 5 3

class TelevisionControls

   public
      procedure VolumeUp ()
      //increases the volume by one

      procedure VolumeDown ()
      //decreases the volume by one

      procedure Mute ()
      //sets the volume to zero
```

-> (0)


```
      procedure ChannelUp ()
      //increases the channel by one

      procedure ChannelDown ()
      //decreases the channel by one

      procedure initalize ()
      //initializes the television when it is turned on

   private

   volume isoftype num
   VOLMAX = 50
   VOLMIN = 0
   VOLINIT = 20
   CHANMAX = 99
   CHANMIN = 3

      procedure VolumeUp (volume isoftype in/out num)
      //increases the volume by one
         if (volume < VOLMAX)
            volume <- volume + 1
         endif
      endprocedure //VolumeUp

      procedure VolumeDown (volume isoftype in/out num)
      //decreases the volume by one
         if (volume > VOLMIN)
            volume <- volume - 1
         endif
      endprocedure //VolumeDown

      procedure Mute (volume isoftype in/out num)
      //sets the volume to zero
         volume <- VOLMIN
      endprocedure //Mute

      procedure ChannelUp (channel isoftype in/out num)
      //increases the channel by one
         if (channel < CHANMAX) then
            channel <- channel + 1
         else
            channel <- CHANMIN
         endif
      endproedure //ChannelUp

      procedure ChannelDown (channel isoftype in/out num)
      //decreases the channel by one
         if (channel > CHANMIN) then
            channel <- channel - 1
         else
            channel <- CHANMAX
         endif
      endproedure //ChannelDown
```

- Spa

```
    procedure initalize (volume isoftype in/out num)
    //initializes the television when it is turned on
       volume <- VOLINIT
    enprocedure //initalize

endclass //Television Controls
```


[This page intentionally left non-blank]

CS 1501 - Fall 1998

Homework 6

165.0 Points

Due: Friday, November 6, 1998 - 9:00am

TURN IN THIS ASSIGNMENT ELECTRONICALLY USING TURNIN

BE SURE TO DOCUMENT ANY REQUIRED ESP INFORMATION (as noted for each problem)

FOLLOW THE CORRECT FORMAT WHEN SUBMITTING ALL HOMEWORKS

[p1]
20.0 points
REDOABLE ESP

Thoughtless Earl has once again made a mess of his data. He is supposed to be keeping track of all of the ships in the pirate fleet as well as all of the pirates in the fleet. For each ship, Thoughtless Earl needs to know the following:

The name of the ship
The captain of the ship
The number of pirates on board
Whether or not the ship is seaworthy

For each pirate, Thoughtless Earl needs to know the following:

The pirate's name
How old they are
How long they have been in the fleet
Whether or not they have scurvy

Declare two records, called ShipRec and PirateRec that hold this information.

[p2]
70.0 points
REDOABLE ESP

Salty Sarah would like to store the information from p1 in linked lists. Specifically, she would like to be able to add a node, delete a node, and calculate the number of nodes that are in the list. She would like to perform these actions on lists that contain other types of data.

After consulting with First Mate Carey, it is decided that writing a class that will work for ANY datatype is the best solution.

Write a class that implements the following methods:

1. Add a node to a linked list. This method will be passed a generic datatype, and will add it to a linked list of datatypes.

2. Delete a node from a linked list. This method will be passed a generic datatype, and will delete any node in the list with the exact same data.

3. Return the number of nodes in the list

The class should work for ANY datatype that is passed to it.

[p3]
35.0 points
REDOABLE ESP

Write an algorithm that uses the class from p2. The algorithm should have a menu that gives the user the following options:

7. Quit

The algorithm should utilize TWO different objects: one for the PirateRecs and one for the ShipRecs. Each selection will call the appropriate method from the appropriate object. The user should be able to make as many selection as they wish, until they select quit. Be sure to use good modularity.

[p4]
10.0 points

Explain the concept of generic parameters. Why are they used?

[p5]
10.0 points

Explain the idea of inheritance, and give at least one original example of when you would use it.

[p6]
10.0 points

Explain the idea of polymorphism.

[p7]
10.0 points

Explain the difference between the copy and clone operations.

```
#############################################################
# Name          : Levi Smith                                #
# GTNum         : gte187k                                   #
# Sections      : IW15a,A14                                 #
# TA Name       : Irwin Dolobowsky                          #
# Assignment    : hw6                                       #
# Size          : 7486 bytes                                #
# Submitted     : 01:42:46(Fri)11-06-98                     #
# Original File : /fuse3/41/gte187k/ldshw06.txt             #
# Last Modified : 01:41:58(Fri)11-06-98                     #
#############################################################
```

144/165

[p1] 16 7 7 2

```
ShipRec definesa record
   shipname isoftype string
   shipcaptainname isoftype string
   piratesonboard isoftype num
   shipisseaworthy isoftype boolean
endrecord //ShipRec

PirateRec definesa record
   piratename isoftype string
   pirateage isoftype num
   piratefleettenure isoftype num
   piratehasscurvy isoftype boolean
endrecord //PirateRec
```

[p2] 60 20 30 10

```
class Manage_List (record_Type isoftype Type)
//Purpose: to be able to add, delete, and count nodes of a linked list of
//any datatype
//Precondition: Must be passed in a datatype
//Postcondition: all nodes must be added, deleted, and counted correctly

public

   procedure Add_to_LL (record isoftype in Record_Type)
   //Purpose: to add a node of any type to a Linked List (LL)
   //Precondition: you must have a LL and a generic parameter
   //Postcondition: a LL with an added node


   procedure Remove_From_LL (record isoftype in Record_Type)
   //Purpose: to remove a node of any type from a Linked List (LL) containing
   //the same data that is passed in with the record
   //Precondition: you must have a LL and a generic parameter
   //Postcondition: a LL with an deleted node


   function Count_LL_Nodes ()
   //Purpose: to count the number of nodes in the Linked List (LL)
   //Precondition: you must have a LL
   //Postcondition: the function return the number of nodes in the LL
```

pass in data

pass in data + pointer


```
   procedure Initalize()
   //Purpose: to initalize the data
   //Precondition: has to have a data type to initalize
   //Postcondition: it initializes the data


protected

   //can't forget the LL, head pointer, and counter!!
   RecordNode definesa record
      data isoftype Record_Type
      next isoftype Ptr toa RecordNode
   endrecord

   headptr isoftype Ptr toa RecordNode

   counter isoftype num

   procedure Add_to_LL (record isoftype in Record_Type)
   //Purpose: to add a node of any type to a Linked List (LL)
   //Precondition: you must have a LL and a generic parameter
   //Postcondition: a LL with an added node
   //Note:  Since it did not say to place the data in any specific order, I
   //will put all new nodes at the end of the list

   if (headptr = NIL) then
      headptr <- new(RecordNode)
      headptr^.data <- Record_Type
      headptr^.next <- NIL
   else
      Move_Down_LL_helper (headptr, record)
      Add_to_LL (record)
   endif
   endprocedure //Add_to_LL


   procedure Move_Down_LL_helper (headptr isoftype in/out Ptr toa RecordNode,
      record isoftype in/out Record_Type)
   //Purpose: moves headptr down the list: it must be done this way since
   //headptr is global to the private section
   //Precondition: must be called from another procedure (for it to do
   //something productive)
   //Postcondition: moves the headptr down one Node on the LL

   headptr <- headptr^.next

   endprocedure //Move_Down_LL_helper


   procedure Remove_From_LL (record isoftype in Record_Type)
   //Purpose: to remove a node of any type from a Linked List (LL) containing
   //the same data that is passed in with the record
   //Precondition: you must have a LL and a generic parameter
   //Postcondition: a LL with an deleted node
```

temp := ptr
temp <- new(node)
temp^.data <- data
temp^.next <- head
head <- temp

-3 (g

pass

```
    if (headptr <> NIL)
       if (headptr^.data = record.data)
          headptr <- headptr^.next
       else
          Move_Down_LL_helper (headptr, record)
          Remove_From_LL (record)
       endif
    endif
    endprocedure //Remove_From_LL


    function Count_LL_Nodes ()
    //Purpose: to count the number of nodes in the Linked List (LL)
    //Precondition: you must have a LL
    //Postcondition: the function return the number of nodes in the LL

    loop
       exitif (headptr = NIL)
       Move_Down_LL_helper (headptr isoftype in Ptr toa Record_Node)
       count <- count + 1
    endloop

    Count_LL_Nodes returns count

    endfunction //Count_LL_Nodes

    procedure Initalize()
    //Purpose: to initalize the data
    //Precondition: has to have a data type to initalize
    //Postcondition: it initalizes the data

    //I'm assuing we are starting from scratch, so I will make the LL empty
    //The counter needs to be set at 0
    headptr <- NIL
    counter <- 0
    endprocedure //Initalize

endclass //Manage_List


[p3] 40 15 20 5

algorithm Manage_Pirates_and_Ships
uses Manage_List

//[p1] Record declarations
ShipRec definesa record
   shipname isoftype string
   shipcaptainname isoftype string
   piratesonboard isoftype num
   shipisseaworthy isoftype boolean
endrecord //ShipRec
```

pass in data + pointer

~3pp ~3hr

if(current = NIL)
ret 0
else ret 1 + Count(current^.next)

Delete (, ptr

end procc

DeleteHlpr (pointer

head <- head^.next

end


```
PirateRec definesa record
   piratename isoftype string
   pirateage isoftype num
   piratefleettenure isoftype num
   piratehasscurvy isoftype boolean
endrecord //PirateRec

Pirate_Manage_List isoftype Manage_List (PirateRec)
Pirate_Manage_List.initialize

Ship_Manage_List isoftype Manage_List (ShipRec)
Ship_Manage_List.initialize


procedure Get_Input ()
//Purpose: To find out what the user wants to do
//Precondition: None
//Postcondition: does what the user wants it to do
Print ("What do you want to do??")
Print ("1. Insert a PirateRec into the list")
Print ("2. Insert a ShipRec into the list")
Print ("3. Delete a PirateRec from the list")
Print ("4. Delete a ShipRec from the list")
Print ("5. See how many PirateRecs are in the list")
Print ("6. See how many ShipRecs are in the list")
Print ("7. Quit")


choice isoftype num

loop
  exitif (choice=7)
   Get Input ()
   read (choice)

   if (choice = 1) then
      Pirate_Manage_List.Add_to_LL (PirateRec)
   elseif (choice = 2) then
      Ship_Manage_List.Add_to_LL (ShipRec)
   elseif (choice = 3) then
      Pirate_Manage_List.Remove_From_LL (PirateRec)
   elseif (choice = 4) then
      Ship_Manage_List.Remove_From_LL (ShipRec)
   elseif (choice = 5) then
      Pirate_Manage_List.Count_LL_Nodes ()
   elseif (choice = 6) then
      Ship_Manage_List.Count_LL_Nodes ()
   elseif (choice = 7) then
      Print ("Exiting...")
   else
      Print ("Invalid Choice")

   endif

endloop
```

~6 pp
~4hr
Fill it

```
public
function count returns a Num()
    private
        function Count2 returns a Num ( current ,out ,inOtr to a LinkNode)

            if (current = NIL) then
                count returns = 0
            else
                count returns 1 + Count ( current ^.next )
            endif
        end funct

        function Count returns a Num ()
        //helper

            Count Returns Count2 (head)

        end function
```

```
endprocedure //Get_Input

endalgorithm //Manage_Pirates_and_Ships
```

[p4]

A generic parameter is a parameter that is not specified, but left open to any data type. The algorithm has to tell the class which kind of data structure is to be used. This allows variables in a class to be not restricted to one single data type.

[p5]

Inheritance is where one or more classes use modified properties (code) from another class. Instead of writing the code over and over, the "Inherit" command is used to "copy" the original code.
Example: A video store has a class to keep track of all of the video rentals in the store. The store then decides to start renting DVD movies. Instead of writing all of the code over agian, the DVD rental class would inherit all of the code from the video rental class (due dates, late fees, etc.), but also change a few properties (the rewind check would not be needed and "needs to be cleaned" or something will have to be added). The original video rental class could also be inherited if the store began renting videogames. The necessary adjustments to the class would be made in the videogame rental class.

[p6]

Polymorphism is where a linked list or tree can hold data for various data types of data so one will not have to make a new list/tree for each data type.

[p7]

Copy just places new pointer(s) at the positions of the pointers in an existing list/tree. Clone makes an entire new list/tree.

[This page intentionally left non-blank]

```
}
}

Add_to_LL (pointer, data)
    AddHelper (
end procedure
```

```
class Circle
: inherits Shape ?
```

```
#############################################################
# Name           : Levi Smith                               #
# GTNum          : gte187k                                  #
# Sections       : IW15a,A14                                #
# TA Name        : Irwin Dolobowsky                         #
# Assignment     : hw6r                                     #
# Size           : 6619 bytes                               #
# Submitted      : 07:17:22(Fri)11-13-98                    #
# Original File  : /fuse3/41/gte187k/ldshw06r.txt           #
# Last Modified  : 07:16:39(Fri)11-13-98                    #
#############################################################


[p2] 60 20 30 10

class Manage_List (record_Type isoftype Type)
//Purpose: to be able to add, delete, and count nodes of a linked list of
//any datatype
//Precondition: Must be passed in a datatype
//Postcondition: all nodes must be added, deleted, and counted correctly

public

   procedure Add_to_LL (record isoftype in Record_Type)
   //Purpose: to add a node of any type to a Linked List (LL)
   //Precondition: you must have a LL and a generic parameter
   //Postcondition: a LL with an added node


   procedure Remove_From_LL (currentptr isoftype Ptr toa RecordNode, data
      isoftype in Record_Type)
   //Purpose: to remove a node of any type from a Linked List (LL) containing
   //the same data that is passed in with the record
   //Precondition: you must have a LL and a generic parameter
   //Postcondition: a LL with an deleted node


   function Count_LL_Nodes returnsa Num()
   //Purpose: to count the number of nodes in the Linked List (LL)
   //Precondition: you must have a LL
   //Postcondition: the function return the number of nodes in the LL

   procedure Initalize()
   //Purpose: to initalize the data
   //Precondition: has to have a data type to initalize
   //Postcondition: it initalizes the data


protected

   //can't forget the LL, head pointer, and counter!!
   RecordNode definesa record
      data isoftype Record_Type
      next isoftype Ptr toa RecordNode
   endrecord
```



```
   headptr isoftype Ptr toa RecordNode
   counter isoftype num

   procedure Add_to_LL (data isoftype in Record_Type)
   //Purpose: to add a node of any type to a Linked List
      Add_to_LL2 (data)
   endprocedure //Add_to_LL

   procedure Add_to_LL2 (data isoftype in Record_Type)
   //Purpose: Helper module to Add_to_LL
      temp isoftype Ptr toa Node
      temp <- new(Node)
      temp^.data <- data
      temp^.next <- headptr
      headptr <- temp
   endprocedure //Add_to_LL2

   procedure Remove_From_LL (currentptr isoftype Ptr toa RecordNode, data
      isoftype in Record_Type)
   //Purpose: to remove a node of any type from a Linked List containing
   //the same data that is passed in with the record

      Remove_From_LL2(currentptr,data)
   endprocedure //Remove_From_LL

   procedure Remove_From_LL2 (currentptr isoftype in/out Ptr toa RecordNode,
      data isoftype in Record_Type)
   //Purpose: Helps Remove_From_LL

      if (currentptr <> NIL)
         if (currentptr^.data = record.data)
            currentptr <- currentptr^.next
         else
            Remove_From_LL2 (currentptr, data)
         endif
   endprocedure //Remove_From_LL2



   function Count_LL_Nodes returnsa Num()
   //Purpose: to count the number of nodes in the Linked List

   Count_LL_Nodes returns Count_LL_Nodes2 (currentptr)
   endfunction //Count_LL_Nodes

   function Count_LL_Nodes2 returnsa Num(currentptr isoftype in Ptr toa
      RecordNode)
   //Purpose: Helps the counter module

      if (current = NIL) then
         count returns 0
      else
         count returns 1 + Count_LL_Nodes2 (currentptr^.next)
      endif
   endfunction //Count_LL_Nodes2
```

```
procedure Initalize()
//Purpose: to initalize the data
//I'm assuing we are starting from scratch, so I will make the LL empty
//The counter needs to be set at 0
headptr <- NIL
counter <- 0
endprocedure //Initalize

endclass //Manage_List


[p3] 40 15 20 5

algorithm Manage_Pirates_and_Ships
uses Manage_List

//[p1] Record declarations
ShipRec definesa record
   shipname isoftype string
   shipcaptainname isoftype string
   piratesonboard isoftype num
   shipisseaworthy isoftype boolean
endrecord //ShipRec

PirateRec definesa record
   piratename isoftype string
   pirateage isoftype num
   piratefleettenure isoftype num
   piratehasscurvy isoftype boolean
endrecord //PirateRec

Pirate_Manage_List isoftype Manage_List (PirateRec)
Pirate_Manage_List.initialize

Ship_Manage_List isoftype Manage_List (ShipRec)
Ship_Manage_List.initialize


procedure Get_Input ()
//Purpose: To find out what the user wants to do
//Precondition: None
//Postcondition: does what the user wants it to do
Print ("What do you want to do??")
Print ("1. Insert a PirateRec into the list")
Print ("2. Insert a ShipRec into the list")
Print ("3. Delete a PirateRec from the list")
Print ("4. Delete a ShipRec from the list")
Print ("5. See how many PirateRecs are in the list")
Print ("6. See how many ShipRecs are in the list")
Print ("7. Quit")


choice isoftype num
```



```
loop
  exitif (choice=7)

  Get_Input()

   read (choice)

   if (choice = 1) then
      Fill_A_Pirate (PirateRec)
      Pirate_Manage_List.Add_to_LL (PirateRec)
   elseif (choice = 2) then
      Fill_A_Ship (ShipRec)
      Ship_Manage_List.Add_to_LL (ShipRec)
   elseif (choice = 3) then
      Pirate_Manage_List.Remove_From_LL (PirateRec)
   elseif (choice = 4) then
      Ship_Manage_List.Remove_From_LL (ShipRec)
   elseif (choice = 5) then
      Pirate_Manage_List.Count_LL_Nodes ()
   elseif (choice = 6) then
      Ship_Manage_List.Count_LL_Nodes ()
   elseif (choice = 7) then
      Print ("Exiting...")
   else
      Print ("Invalid Choice")

   endif

endloop

endprocedure //Get_Input

procedure Fill_A_Pirate (record isoftype in/out PirateRec)
//Purpose: to fill in pirate data
//Precondition: need a PirateRec
//Postcondition: a filled PirateRec

name isoftype string
age isoftype num
tenure isoftype num
scurvy isoftype boolean

print("What is the pirate's name??")
read(name)
print("What is the pirate's age??")
read(age)
print("How long has the pirate been on the ship??")
read(tenure)
print("Does the pirate have scurvy??")
read(scurvy)

record.piratename <- name
record.pirateage <- age
record.piratefleettenure <- tenure
record.piratehasscurvy <- scurvy
```

```
cedure

procedure Fill_A_Ship (record isoftype in/out ShipRec)
//Purpose: to fill in ship data
//Precondition: need a ShipRec
//Postcondition: a filled ShipRec

name isoftype string
capname isoftype string
piratesonboard isoftype num
seaworthy isoftype boolean

print("What is the ship's name??")
read(name)
print("What is the ship captain's age??")
read(capname)
print("How many pirates are on board??")
read(piratesonboard)
print("Is the ship seaworthy??")
read(seaworthy)

record.shipname <- name
record.captainname <- capname
record.piratesonboard <- piratesonboard
record.shipisseaworthy <- seaworthy

endprocedure


endalgorithm //Manage_Pirates_and_Ships



[p4]

[p5]

[p6]

[p7]
```


[This page intentionally left non-blank]

CS 1501 - Fall 1998

Homework 7

165.0 Points

Due: Friday, November 13, 1998 - 9:00 A.M.

TURN IN THIS ASSIGNMENT ELECTRONICALLY USING TURNIN

BE SURE TO DOCUMENT ANY REQUIRED ESP INFORMATION (as noted for each problem)

FOLLOW THE CORRECT FORMAT WHEN SUBMITTING ALL HOMEWORKS

For p1 and p2, use the following class definition:

```
class Shape

   public
          function Get_Area returnsa num()
          // Purpose: Returns the area
          // Preconditions: All values have been initialized
          // Postconditions: The area is returned

         function Get_Circumference returnsa num()
         // Purpose: Returns the circumference
         // Preconditions: All values have been initialized
         // Postconditions: The circumference is returned

          procedure Initialize()
          // Purpose: Initializes all variables
          // Preconditions: None
          // Postconditions: All values have been initialized

   protected
          function Get_Area returnsa num()
             deferred
          endfunction

          function Get_Circumference returnsa num()
             deferred
          endfunction

          procedure Initialize()
             deferred
          endprocedure

endclass
```

shape

have to write procedure in child when says "deferred"

need to have all modules

Class Triangle

inherits Shape

Shape_to_add ← Circle/Rectangle

~~head^.data ← shape~~

Shape_to_add. Init

Shape_to_add. Get_data

current^.data ← shape_to_add

[p1]
REDOABLE ESP
50.0 points

Write two classes, called Circle and Rectangle that extend the Shape class. These classes must implement all of the methods of Shape, and must also implement a method for setting the dimensions. I.e. Circle needs to know the radius, and Rectangle needs to know the height and width.

[p2]
REDOABLE ESP
40.0 points

Write an algorithm that will allow the user to build a linked list of Shapes. The algorithm should display the following menu of options:

1. Add either a Circle or Rectangle to the list.
2. Print out the area and circumference of every Shape in the list.
3. Quit

The user should be able to make as many selections as they wish until they choose quit. Each selection should executed before the next selection is entered.

Your solution should involve polymorphism where appropriate.

For problems 3 - 10, give the Big-O of the given algorithms, along with a brief explanation of your answer. YOU MUST PROVIDE AN EXPLANATION IN ORDER TO GET CREDIT. On questions which refer to an "ordered" data structure, assume the data is in ascending order.

[p3]
5.0 points

Finding the smallest node in an unsorted array of size N. O(N)

Not Sorted - Have to search them all

EXPLAIN

[p4]
5.0 points

Divide and Conquer

O(log N)

Finding the largest node in a full and balanced binary search tree of size N.

The amount of time to find the largest node changes as the BST increases and decreases in size

[p5]
5.0 points

~~O(N)~~ ⇒ O(log N)

Finding the largest node in an array of size N, sorted in ascending order.

The largest node will always be the last node in an ascending array

[p6]
5.0 points

Use divide and conquer

Finding the smallest node in a linked list of size N, sorted in ascending order.

O(1)

The smallest node will always be the first node in an ascending LL,

[p7]
5.0 points

Finding the smallest node in a binary search tree of size N.

O(N)

The amount of time to find N depends on how large the tree is. Also it is not full and balanced, so we can't use O(log N)

[p8]
5.0 points

Finding a node in a full and balanced binary search tree (size N) of linked lists (size P).

O(P log N)

(assuming that it is indexed on what we are looking for)

[p9]
5.0 points

i, j, k isoftype num

```
i <- 1
loop
   exitif (i > P)
   j <- 1

   loop
        exitif (j > N)
        k <- N

        loop
           exitif (k < 1)
           k <- k - 1
        endloop

        j <- j + 1
   endloop

   i <- i + 1
endloop
```

O(N^3)

Three nested loops, each accounting for one O(N); when these are multiplied together, the result is O(N^3).

AGH BCDEF KJI = 82

AGH BCIJDKF = 83

ABCDEFKJIAG = 89

[p10]
5.0 points

Deleting a node from a linked list of size N.

N is exponent or ~~unreasonable ends in exponent or~~ factorial

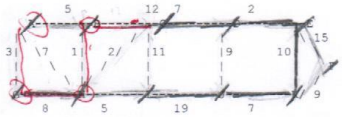
[p11]
20.0 points

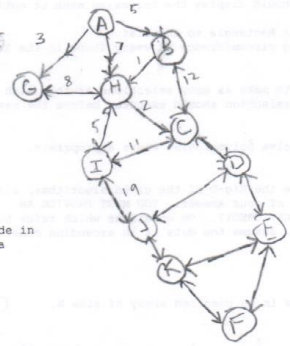
Compare and contrast reasonable and unreasonable algorithms. Give an example of a each, and list some examples of Big-O's which indicate that an algorithm is reasonable or unreasonable.

[p12]
15.0 points

A, G, H, B, C, D, E, F, K, J, I

Given the following graph:





3
8 11
12 13
7 20
2 22
15 27
9 36
7 43
19 52
82

Suppose you started at point A, and were required to visit every node in the tree once and ONLY once. What path would you take if you used a depth first search?

A, G,

AGHBCDEKFKJI

3 55
10 12
11 7 14
12 2 16
11 5 21
19 9 30
9 7 37
2 19 46
9 5 51
3 59

A G H B C D E F
K J I

```
#############################################################
# Name          : Levi Smith                                #
# GTNum         : gte187k                                   #
# Sections      : IW15a,A14                                 #
# TA Name       : Irwin Dolobowsky                          #
# Assignment    : hw7                                       #
# Size          : 7019 bytes                                #
# Submitted     : 01:40:35(Fri)11-13-98                     #
# Original File : /fuse3/41/gte187k/ldshw07.txt             #
# Last Modified : 01:38:34(Fri)11-13-98                     #
#############################################################
```

147/165

```
[p1] 45 20 20 5

Class Circle
inherits Shape

public

     function Get_Area returnsa num()
     //Purpose: returns the area
     //Preconditions: all values have been intialized
     //Postconditions: the area is returned

     function Get_Circumference returnsa num()
     //Purpose: returns the circumference
     //Preconditions: all values have been intialized
     //Postconditions: the circumference is returned

     procedure Get_Data()
     //Purpose: to set the shape dimensions
     //Preconditions: all values have been initalized
     //Postconditions: returns the new shape values specified by the user

     procedure Initalize()
     //Purpose: initalizes all variables
     //Preconditions: none
     //Postconditions: all values have been initalized

protected

PI is 3.14159265359
SQUARED is 2
CIRCUM_CONST is 2
radius isoftype num

     function Get_Area returnsa num()
        Get_Area returns (PI * radius * SQUARED)
     endfunction

     function Get_Circumference returnsa num()
        Get_Circumference returns (CIRCUM_CONST * PI * radius)
     endfunction
```

-1 lag


```
     procedure Get_Data()
        print("What is the radius of the circle??")
        read(radius)
     endprocedure

     procedure Initalize()
        radius <- 0
     endprocedure
```

-1 lag

```
endclass //Circle

Class Rectangle
inherits Shape

public

     function Get_Area returnsa num()
     //Purpose: returns the area
     //Preconditions: all values have been intialized
     //Postconditions: the area is returned

     function Get_Circumference returnsa num()
     //Purpose: returns the perimeter
     //Preconditions: all values have been intialized
     //Postconditions: the perimeter is returned

     procedure Get_Data()
     //Purpose: to set the shape dimensions
     //Preconditions: all values have been initalized
     //Postconditions: returns the new shape values specified by the user

     procedure Initalize()
     //Purpose: initalizes all variables
     //Preconditions: none
     //Postconditions: all values have been initalized

protected

DOUBLE is 2
height isoftype num
width isoftype Num

     function Get_Area returnsa num()
        Get_Are returns (height * width)
     endfunction

     function Get_Circumference returnsa num()
        Get_Circumference returns ((DOUBLE * height) + (DOUBLE * width))
     endfunction

     procedure Get_Data()
        print("What is the height of the rectangle??")
        read(height)
        print("What is the width of the rectangle??")
        read(width)
     endprocedure
```

```
    procedure Initalize()
      height <- 0
      width <- 0
    endprocedure

endclass //Circle

[p2] 45 20 20 5

algorithm Manage_Shapes
uses Shape
ShapeNode definesa record
   data isoftype Shape
   next isoftype Ptr toa ShapeNode
endrecord

currentptr isoftype Ptr toa ShapeNode
currentptr <- NIL

Menu ()

procedure Menu()
choice isoftype Char
     loop
          print("Welcome to Shape Manager")
          print("Choose an option")
          print("C: Add a circle to the list")
          print("R: Add a rectangle to the list")
          print("P: Print out the area and circumference of every shape in
               the list")
          print("Q: Quit")
          read(choice)
          exitif((choice = "Q") or (choice = "q"))
          if ((choice = "C") or (choice = "c")) then
             Add_Circle(currentptr)
          elseif ((choice = "R") or (choice = "r")) then
             Add_Rectangle(currentptr)
          elseif ((choice = "P") or (choice = "p")) then
             Print_List(currentptr)
          else
             print("That choice is not avaiable")
          endif
     endloop
endprocedure //Menu

procedure Add_Circle (currentptr isoftype in/out Ptr toa ShapeNode)
//Purpose: to add a circle to the Shape list
//Preconditions: need a list
//Postconditions: a new circle will be on the Shape list

     new_circle isoftype Circle
     new_circle.Initalize
     new_circle.Get_Data
```



```
     temp isoftype Ptr toa ShapeNode
     temp <- new(ShapeNode)
     temp^.data <- new_circle
     temp^.next <- currentptr
     currentptr <- temp
endprocedure

procedure Add_Rectangle (currentptr isoftype in/out Ptr toa ShapeNode)
//Purpose: to add a rectangle to the Shape list
//Preconditions: need a list
//Postconditions: a new rectangle will be on the Shape list

     new_rect isoftype Rectangle
     new_rect.Initalize
     new_rect.Get_Data

     temp isoftype Ptr toa ShapeNode
     temp <- new(ShapeNode)
     temp^.data <- new_rect
     temp^.next <- currentptr
     currentptr <- temp
endprocedure

procedure Print_List (currentptr isoftype in Ptr toa ShapeNode)
//Purpose: Print all of the data in the Shape list
//Preconditions: need a linked list
//Postconditions: all of the data will be displayed on the screen

if (currentptr <> NIL) then
   print ("Shape data is:")
   print ("Circumference: ", shapedata.Get_Circumference)
   print ("Area: ", shapedata.Get_Area)
   Print_List (currentptr^.next)
endif
endprocedure

endalgorithm


Note:  in problems 3 through 10, I am assuming that the linked list, BST,
or array is indexed on the value that we are searching.

[p3]

O(N)
This array is not sorted so the program will have to search all of the cells.

[p4]

O(log N)
A divide and conquer approach can be taken for this problem.  Each search
"throws away" half of the full and balanced BST.

[p5]
```

same diving approach can be used for this array since it is ordered.

[p6]

O(1)
The smallest node will always be the first node in an ascending linked list.

[p7]

O(N)
Since the tree is not full and balanced, we cannot use the dividing approach.

[p8]

O(P log N) work NP
It takes O(P) to search the linked list (the LL is not ordered so all of the nodes have to be searched), and it takes O(log N) to find the node in the full and balanced BST (which allows us to use the diving method). These two results are then multiplied together to get the total order notation. O(P) * O(log N)

[p9]

O(N^2 * P)
It takes P times to go through the outer loop. It takes N times to go through the second loop. It also takes N times to go through the inner loop. These three values are then multiplied together to get the Big Oh.

[p10]

O(1) work N
Since a specific node is not specified, we will just delete the first one.

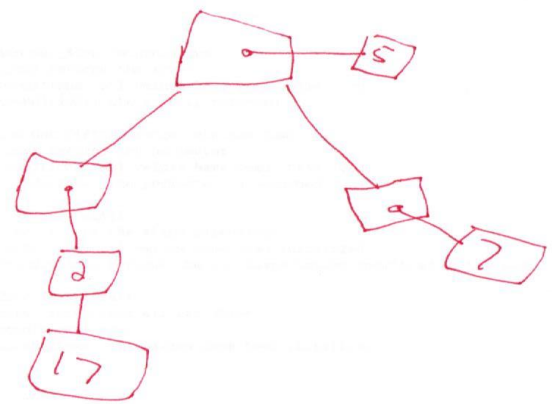
[p11]

Reasonable algorithms are ones that can be completed in a reasonable amount of time. They do take some time to process, but you will not be waiting an unreasonable time such as multiple years.
Big Os: O(N); O(log N); O(N^C) -->C must be a constant

Unreasonable algorithms are algorithms that take months, years, decades, or even longer to compute. It is not a very good idea to use them.
Big Os: O(C^N); O(N!); O(N^N)

[p12]

A - G = 3
G - H = 8
H - B = 1
B - C = 12
C - D = 7
D - E = 2
E - K = 10
K - F = 9
K - J = 7
J - I = 19
TOTAL = 54



```
#################################################################
# Name          : Levi Smith                                    #
# GTNum         : gte187k                                       #
# Sections      : IW15a,A14                                     #
# TA Name       : Irwin Dolobowsky                              #
# Assignment    : hw7r                                          #
# Size          : 5209 bytes                                    #
# Submitted     : 02:15:19(Fri)11-20-98                         #
# Original File : /fuse3/41/gte187k/ldshw07r.txt                #
# Last Modified : 02:14:52(Fri)11-20-98                         #
#################################################################

[p1] 45 20 20 5

Class Circle
inherits Shape

public

     function Get_Area returnsa num()
     //Purpose: returns the area
     //Preconditions: all values have been intialized
     //Postconditions: the area is returned

     function Get_Circumference returnsa num()
     //Purpose: returns the circumference
     //Preconditions: all values have been intialized
     //Postconditions: the circumference is returned

     procedure Get_Data()
     //Purpose: to set the shape dimensions
     //Preconditions: all values have been initalized
     //Postconditions: returns the new shape values specified by the user

     procedure Initalize()
     //Purpose: initalizes all variables
     //Preconditions: none
     //Postconditions: all values have been initalized

protected

PI is 3.14159265359
SQUARED is 2
CIRCUM_CONST is 2
radius isoftype num

     function Get_Area returnsa num()
        Get_Area returns ((PI * radius)*(PI * radius))
     endfunction

     function Get_Circumference returnsa num()
        Get_Circumference returns (CIRCUM_CONST * PI * radius)
     endfunction
```



```
     procedure Get_Data()
        print("What is the radius of the circle??")
        read(radius)
     endprocedure

     procedure Initalize()
        radius <- 0
     endprocedure

endclass //Circle

Class Rectangle
inherits Shape

public

     function Get_Area returnsa num()
     //Purpose: returns the area
     //Preconditions: all values have been intialized
     //Postconditions: the area is returned

     function Get_Circumference returnsa num()
     //Purpose: returns the perimeter
     //Preconditions: all values have been intialized
     //Postconditions: the perimeter is returned

     procedure Get_Data()
     //Purpose: to set the shape dimensions
     //Preconditions: all values have been initalized
     //Postconditions: returns the new shape values specified by the user

     procedure Initalize()
     //Purpose: initalizes all variables
     //Preconditions: none
     //Postconditions: all values have been initalized

protected

DOUBLE is 2
height isoftype num
width isoftype Num

     function Get_Area returnsa num()
        Get_Are returns (height * width)
     endfunction

     function Get_Circumference returnsa num()
        Get_Circumference returns ((DOUBLE * height) + (DOUBLE * width))
     endfunction

     procedure Get_Data()
        print("What is the height of the rectangle??")
        read(height)
        print("What is the width of the rectangle??")
        read(width)
     endprocedure
```

```
      procedure Initialize()
         height <- 0
         width <- 0
      endprocedure

endclass //Circle

[p2] 45 20 20 5

algorithm Manage_Shapes
uses Shape
uses Rectangle
uses Circle

ShapeNode definesa record
   data isoftype Shape
   next isoftype Ptr toa ShapeNode
endrecord

currentptr isoftype Ptr toa ShapeNode
currentptr <- NIL

Menu ()

procedure Menu()
choice isoftype Char
          print("Welcome to Shape Manager")
          print("Choose an option")
          print("C: Add a circle to the list")
          print("R: Add a rectangle to the list")
          print("P: Print out the area and circumference of every shape in
               the list")
          print("Q: Quit")
endprocedure //Menu

      loop
          read(choice)
          exitif((choice = "Q") or (choice = "q"))
          if ((choice = "C") or (choice = "c")) then
             Add_Circle(currentptr)
          elseif ((choice = "R") or (choice = "r")) then
             Add_Rectangle(currentptr)
          elseif ((choice = "P") or (choice = "p")) then
             Print_List(currentptr)
          else
             print("That choice is not avaiable")
          endif
      endloop

procedure Add_Circle (currentptr isoftype in/out Ptr toa ShapeNode)
//Purpose: to add a circle to the Shape list
//Preconditions: need a list
//Postconditions: a new circle will be on the Shape list
```



```
      new_circle isoftype Circle
      new_circle.Initalize
      new_circle.Get_Data

      temp isoftype Ptr toa ShapeNode
      temp <- new(ShapeNode)
      temp^.data <- new_circle
      temp^.next <- currentptr
      currentptr <- temp
endprocedure

procedure Add_Rectangle (currentptr isoftype in/out Ptr toa ShapeNode)
//Purpose: to add a rectangle to the Shape list
//Preconditions: need a list
//Postconditions: a new rectangle will be on the Shape list

      new_rect isoftype Rectangle
      new_rect.Initalize
      new_rect.Get_Data

      temp isoftype Ptr toa ShapeNode
      temp <- new(ShapeNode)
      temp^.data <- new_rect
      temp^.next <- currentptr
      currentptr <- temp
endprocedure

procedure Print_List (currentptr isoftype in Ptr toa ShapeNode)
//Purpose: Print all of the data in the Shape list
//Preconditions: need a linked list
//Postconditions: all of the data will be displayed on the screen

if (currentptr <> NIL) then
   print ("Shape data is:")
   print ("Circumference: ", shapedata.Get_Circumference)
   print ("Area: ", shapedata.Get_Area)
   Print_List (currentptr^.next)
endif
endprocedure

endalgorithm
```

CS 1501 - Fall 1998

Homework 9

70.0 Points

Due: Monday, November 30, 1998 - 9:00am

TURN IN THIS ASSIGNMENT ELECTRONICALLY USING TURNIN

BE SURE TO DOCUMENT ANY REQUIRED ESP INFORMATION (as noted for each problem)

FOLLOW THE CORRECT FORMAT WHEN SUBMITTING ALL HOMEWORKS

[p1]
10.0 points

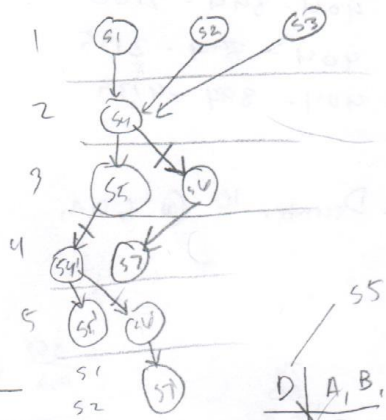
In your own words, define "precedence" and "dependence", and describe the difference between them.

For the next two problems, you need to provide textual representations of precedence graphs. Examples of this are available via the class web page: http://www.cc.gatech.edu/classes/cs1501/precexam/prec.html. You should first draw your precedence graphs on paper, then convert them to text.

[p2]
15.0 points

Draw a precedence/dependency graph for the following code:

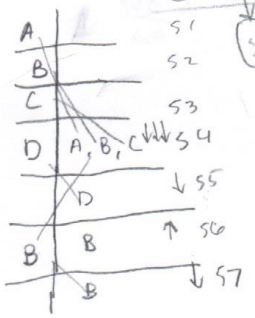
```
read(A)                 // s1
read(B)                 // s2
print(A+B)              // s3
C <- A - B              // s4
D <- C + A              // s5
E <- B / C              // s6
print(E)                // s7
A <- 5 * C              // s8
```

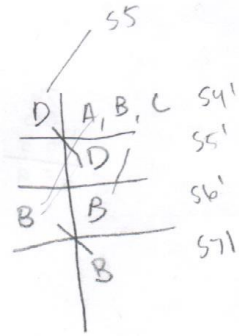


[p3]
15.0 points

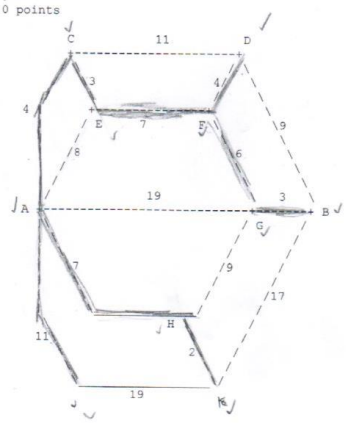
Draw a precedence/dependency graph for the following code:

```
read(A)                 // s1
B <- 10                 // s2
C <- 8                  // s3
loop
   exitif(B<C)
   D <- A+B+C   // s4
   print(D)     // s5
   B <- B-1     // s6
   print(B)     // s7
endloop
```





[p4]
30.0 points



A.) A,H,G,B = 19

B.) CEFDB

C)

J K

a.) On the above undirected graph, what is the shortest path from A to B?

b.) What path would a greedy algorithm find from A to B? Note: The algorithm must make its choice from all edges connected to its current node but is not allowed to backtrack.

c.) Suppose each lettered node represents a city and the numbers represent the cost of connecting the city. What is the minimum cost to connect all cities with a least one line.

404-894-2100
404-894-2195
404-894-4139

R December 10 @ 6PM CoC RM 16 or 17

76
80
103
259

359
400

CS 1501 - Fall 1998

Homework 8

75.0 Points

Due: Friday, November 20, 1998 - 9:00am

TURN IN THIS ASSIGNMENT ELECTRONICALLY USING TURNIN

BE SURE TO DOCUMENT ANY REQUIRED ESP INFORMATION (as noted for each problem)

FOLLOW THE CORRECT FORMAT WHEN SUBMITTING ALL HOMEWORKS

[p1]
10.0 points

Explain exactly why MergeSort is more efficient than BubbleSort.

N^2 vs. $N \log N$

For problems 2 - 3, give the Big-O of the given algorithms, along with a brief explanation of your answer. YOU MUST PROVIDE AN EXPLANATION IN ORDER TO GET CREDIT. On questions which refer to an "ordered" data structure, assume the data is in ascending order.

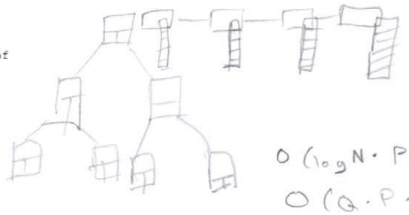
[p2]
10.0 points

Determining the median of all the values in a sorted array of numbers. The median for a collection of numbers is the middle value. ex: for -1, -1, 1, 4, 5 the median would be 1. If there are an even number of numbers then the average of the two numbers is the median. ex: for 1, 2, 3, 5, 6, 7 the median would be (3+5)/2 = 4. The array is of size N.

$O(1)$

[p3]
10.0 points

Finding a node in a full and balanced binary search trees (size N), of linked lists of unsorted arrays (size P). That is: the tree is made up of lists, and the lists are made up of arrays (size Q).



$O(\log N \cdot P \cdot Q)$

$O(Q \cdot P \cdot \log N)$

[p4]
10.0 points

Define "concurrency" and "parallelism" and describe the difference between them, in your own words.

[p5]
10.0 points

Compare and contrast the four different types of concurrent systems.

[p6]
20.0 points

Define each of the following, and explain how they relate to concurrency:

A) Mutual exclusion
B) Fairness
C) Deadlock
D) Synchronization

[p7]
5.0 points

Explain the concept of "pipeline processing".

p4]

concurrency - where many tasks seem to be running at the same time, but really each time slicing is taking place, where each task gets an assigned processor usage spot. Only one processor is used

parallelism - similar to concurrency, but multiple processors are used to complete the tasks

p5] Multiprogramming is where many tasks are completed by only one processor

```
#############################################################
# Name          : Levi Smith                                #
# GTNum         : gte187k                                   #
# Sections      : IW15a,A14                                 #
# TA Name       : Irwin Dolobowsky                          #
# Assignment    : hw8                                       #
# Size          : 3055 bytes                                #
# Submitted     : 02:22:01(Fri)11-20-98                     #
# Original File : /fuse3/41/gte187k/ldshw08.txt             #
# Last Modified : 02:21:37(Fri)11-20-98                     #
#############################################################
```

70/75

[p1]

MergeSort, which has O(N*log N), is more efficient than BubbleSort, which has O(N^2), because it's performance is much faster (as indicated by the Big O). In other words, it takes many more processing cycles to do a BubbleSort than a MergeSort.

[p2]

O(1)

The Big-O of the odd numbers is O(1) (we can go straight to the center spot), and the Big-O of the even numbers is also O(1) (for the same reason). Add these two together, and the result is O(1).

[p3]

O(Q*P*N)

If we are searching for a node in an unsorted array, then we have to search all of the nodes in the arrays, O(Q), search all of the linked lists, O(P), and search all of the nodes in the binary search tree, O(N). Similar to last week's p8 where we have to search all of the nodes in the BST since the node we are searching for can be in any one of the tree nodes.

[p4]

Concurrency: when multiple tasks (which may be duplicates or distinct) can be executed at the same time, reguardless of the number of processors, resulting in a reduction of overall time to preform the series of tasks.

Parallelism: when multiple processors are used to execute the same task

[p5]

Multiprogramming: Many tasks are done by one processor. May have time-sharing or priority algorithm to sort tasks. User notices no discontinuity because of the rapid swapping of tasks.

Multiprocessing: One or more tasks are completed by many processors. Processors may have timeshare between the tasks.

Multitasking: Many tasks are done by one or more processors.

Distributed: Many tasks are done by many processors, at long distances. Eliminates a central point of failure. Efficiency can be increased by localizing the processing.

[p6]

Mutual Exclusion: If one user is trying to access some data, it locks all other users from accessing that data. It makes only one data process at a time, insuring data integrity.

Fairness: Faster jobs can be done first, but if there are many fast jobs, then the longer jobs may never finish.

Deadlock: If resources are shared, then some jobs my last forever. For example, the first job is holding a resource the second job needs, and the second job holds a resource the first job needs. This process will last forever, so careful planning, using order and priorities, is required.

Synchronization: For one process to exchange data with another process, they get together to coordinate their communications.

[p7]

Each operation is done on a separate ~~computer~~ processor, and each piece of data is fed into the queue of computers one at a time, in a first-in, first-out basis.

– 3pts

CS1501 Lab Sept 24, 1998

- Projects due Thursday at noon
40% off if turned in at Fri by noon

~ cs1501 setup
pico .profile

workon cs1501 viewsection

getjob␣<jobname>

turnin <hw1> // turn in HW
getback // view HW turned in
viewgrades
workoff

EX: turnin␣hw1␣<hw1a.txt>

workon cs1501 survey high structure ✓

viewsection Jeff: tup1@cc
Kendrick: kenc@cc

mail: cs1501-prj@cc
Subj: LAB␣1

~cs1501/xsetup

git.cc.class.1501.announce
git.cc.class.150X

Project: getjob prj1

Projects TR at noon
BuzzB Sun evening

[p7]

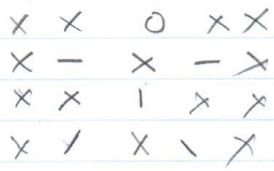
```
algorithm Get_Grade
// Purpose: to calculate and print a letter
// grade from 3 lab grades, three test grades, and
// a final exam grade.
grade_avg , final_grade
test_avg , lab_avg, exam is of type Num
test_or_lab is of type String

~~print ("Enter student's first test grade:")~~
~~read (test_one)~~

// Find out what the test average is.
test_or_lab <- "test"      // Used the the procedure's prompt so the
                              user will know if he's
                              entering a test or a lab grade
Grade_Average (test_or_lab)
test_avg <- grade_avg
// Find out what the lab average is.
test_or_lab <- "lab"
lab_avg <- grade_avg

print ("Enter the final exam grade")
read (exam)

final_grade <- Overall_Avg (avg_lab, avg_test, exam)
print ( Letter_Grade (final_grade))

end algorithm
```

[p8]

~~function~~ procedure Falling returns a Num (number is of type in Num)

// blah

```
if (number = 0) then
    ~~print (number)~~
else
    print (number)
    Falling (number - 1)
end if
end procedure // Falling
```



10
9
8
7
6
5
4
3
2
1
0

Falling (0) → 0

Falling (1) → Falling (0)

Falling (9) → Falling (8)

Falling (10) → Falling (9) → print 9 → print 10

[p1]

Numbers, Pointers, Booleans, and Pointers

EX:

Number: 42

Character: 'c'

Boolean: TRUE

Pointer:

[p2]

Constant - stores a value that remains the same throughout an algorithm

Variable - contains a value that can change throughout an algorithm

[p3]

A.) checking_account isoftype Num

B.) my_birthday isoftype String

C.) class_grade isoftype Char

D.) soc_sec isoftype String

E.) in_college isoftype Boolean

[p4]

```
algorithm Two Numbers
// Purpose: to prompt the user for two numbers then
// either average or add them.
  // get two numbers from user
  value_one isoftype Num
  print ("Enter first number:")
  read (value_one)

  value_two isoftype Num
  print ("Enter second number:")
  read (value_two)

  // ask user what shall be done with the numbers
  answer isoftype Num
  user_choice isoftype Char
  print ("Shall I average the two numbers? Y/N")
```

```
if (user_choice = 'Y') then
    answer <- (value_one + value_two) /2
    print ("The average of the two values is", answer)
else
    answer <- (value_one + value_two
    print ("the sum of the two values is", answer)
end if

endalgorithm // Two Numbers
```

[p5]

A.) formal parameters - where the type of each parameter

B.) Actual parameters -

C.) In parameters

D.) Out Parameters

[p6]

A.) function Average returns a Num (value_one isoftype in Num, value_two isoftype in Num)

B.) procedure Print_Average (value_one isoftype in Num, value_two isoftyp in Num)

C.) procedure Average_And_Sum (num_avg isoftype out Num, num_sum isoftype out Num, num_one isoftype in Num, num_two isoftype in Num, num_three isoftype in Num)

D.) procedure Divide_by_Two (num_one isoftype in Num, num_two isoftype in Num, half_one isoftype out Num, half_two isoftype out Num)

E.) procedure Sum_of_Three (num_one iot in Num, num_two iot in Num, num_three iot in Num, num_sum iot out Num)

[p7]

```
function Safe_to_Sail returns a Boolean (wind_speed iot Num)
    safe_speed iot Num
    safe_speed <- 50
    if (wind_speed > safe_speed) then
        Safe_to_Sail returns TRUE
    else
        Safe_To_Sail returns FALSE
    end if
end algorithm
```

Sept 29, 1998

```
algorithm compute area

PI is 3.14159265359
radius, num_2 iot Num
print (" circle radius")
 read (radius)
area ← PI * radius * radius
print ("the area of circle of " radius ", area)

end algorithm
```

CS1501

Sept 29, 1998

Lecture → Homework → Quiz

- Algorithm - "recipe for action"
  - transforms input to output
- abstraction - general characteristics of what has to be done.

HW Question

ATOMIC DATA TYPES

Numbers ← 1.5
Characters ← 'F'
Booleans ← True/False
(Pointers)

Different Types because: they each do different things

BUILT-IN TYPES

String ← "My Age" or "S"

Assignment
Age ← 27
"Age gets 27"

Age is of type Num
Name is of type String

Name ← "David Smith"

Age.next.year is of Num
Age.next.year ← Age + 1

BUILT-IN
print
read

```
num_1, num_2, num_3   int  Num
print("give me 2 numbers")
read (num_1, num_2)
num_3 <- num_1 + num_2
print ("The ~~answer~~ is ", num_3)
```

sum of, "num_1", and, "num_2"

```
if ( .... ) then ....
  else ....
  end if
```

could also do:
switch ⇐ boolean

could have another if
* be sure to indent

```
if (    ) then
    if ( ) then
else
end if
```

A > B → Result is Boolean

NOT (A > B) same as B >= A

0 < X < 10 → (0 < X) AND (X < 10)

// comments

PI : 3.14159265359

num_3 <- 3.14159265359 * num_1 * num
PI

Tue, 01 Dec 1998 11:06:48 git.cc.class.1501.announce Thread 46 of 49
Lines 31 Re: Finals time sign-up/information Respno 1 of 2
uni@unicom.resnet.gatech.edu The Little Unicorn at College of Computing

In git.cc.class.1501.announce, The Little Unicorn <uni@unicom.resnet.gatech.edu>
calls on the power of The Little Unicorn:
You will need to let us know which exam period you wish to attend.
You must show up to the exam period you committed to.

You may take the final EARLIER than or during your scheduled time, but not after your schedule time. (For example, if you are in the 12:00 class you may take the Monday, Tuesday or Thursday offering of the final but NOT the Friday offering of the final.)

To sign up for a different time, other than the one you are schedule to take, type survey at the workon prompt. You must complete this survey no later than friday, otherwise, you will be stuck with the final you are scehduled to take, no exceptions.

The survey has been updated to reflect the errors of the final times. The final times are:

ALTERNATE	Monday	6:00 pm - 8:50 pm
9:30 Class	Tuesday	8:00 am - 10:50 am
3:00 Class	Thursday	11:30 am - 2:20 pm
12:00 Class	Friday	8:00 am - 10:50 am

As of now, we are not sure where the alternate final will be held. An announcement will be made when we finalize this detail. If you signed up earlier, you will need to take the survey again. Sorry for the inconvience.
--
. *=* .
\,/| Susan H Chan CS1501 recitation TA unicorn@cc.gatech.edu |\,/
/@) "If you'll believe in me, I'll believe in you" Lewis Carroll (@\

<n>=set current to n, TAB=next unread, /=search pattern, ^K)ill/select, a)uthor search, B)ody search, c)atchup, f)ollowup, K=mark read, |=pipe, m)ail, o=print, q)uit, r)eply mail, s)ave, t)ag, w=post

--More--(96%) [2096/2170]

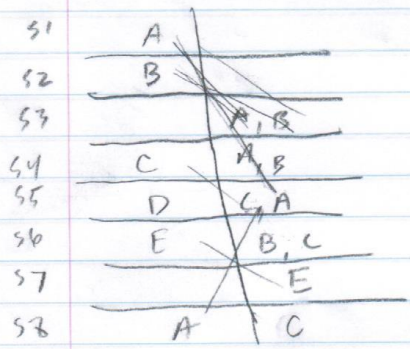
Mon, 07 Dec 1998 14:48:28 git.cc.class.1501.announce Thread 49 of 49
Lines 23 Lab Final Grades Online No responses
benji@cc.gatech.edu Ben Parrott at College of Computing, Georgia Tech

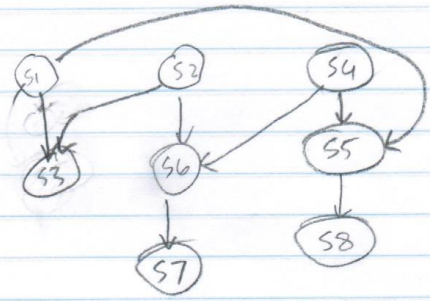
The lab final grades are now online. The grades were taken out of 240
point to account for the difficulty of the exam.

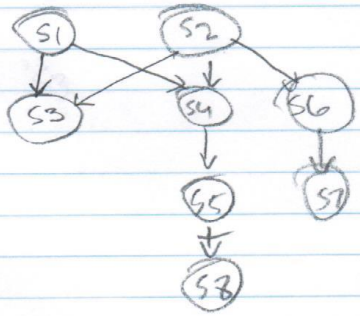
Info about the "3 big stats":
Avg: 73.85
Mode: 83.33
Median:75

Info about the grade distribution:
Grades > 100% = 41
A's = 136
B's = 105
C's = 125
D's = 76
F's = 147

Note these numbers do not include the scores of people who did not show
up to take the exam.


S1
S2
S3
S4
S5
S6
S7
S8
A
B
A, B
C
A, B
D
C, A
E
B, C
E
A
C

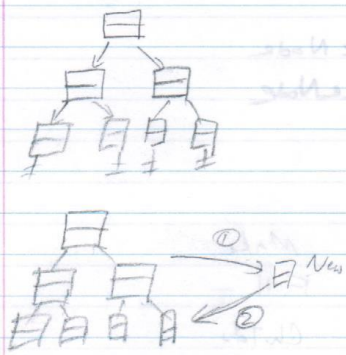

S1
S2
S4
S3
S6
S5
S7
S8


S1
S2
S3
S4
S6
S5
S7
S8

p1]

Not NIL

p2]



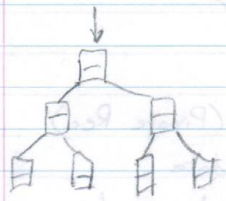
Equal, Left, or Right

E, L, or R

p3]

In Order?

p4]

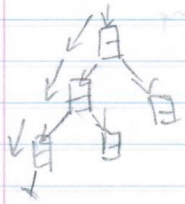


Not NIL

If Equal, Print

If Ptr = NIL? → Error?

p5]



p6]

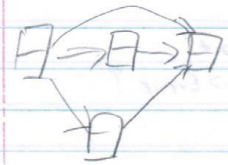
M
F
C

Make New
Fill it in
Children

Tree Node defines a Record
data int Num
left child int ptr to a Tree Node
right child int ptr to a TreeNode
end rec

if (currentptr = NIL) then
currentptr <- new (PirateRec) Make
currentptr^.data <- PirateRec^.data Fill
currentptr^.left_child <- NIL Childs
elseif currentptr^.data < PirateRec^.data then
Add Node to BST

temp <- new (Pirate Rec)
temp^.data <- PirateRec^.data
temp^.next <- nodeptr

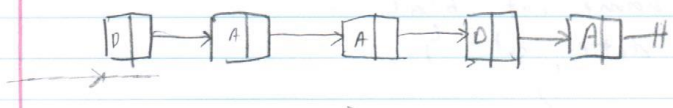


temp <- new (Pirate Rec)
temp^.data <- data
temp^.next <- node pointer
nodepointer <- temp

temp <- new (PirateRec)
temp^.data <- Data
temp^.next <- currentptr
currentptr <- temp

```
                                   in/out
procedure Two_Lists (oldpointer isoftype Ptr toa PirateRec,
                     newpointer  "   "  "   "   "   "   )

temp isoftype Ptr toa PirateRec
temp <- new (PirateRec)
```

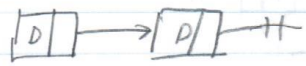
p↑ Old list



if (nodepointer = Dead) send record to new list
if

Result Old List

New List



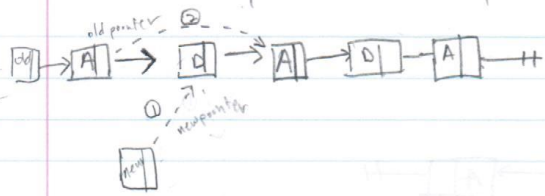
```
DeadPrivateRec   defines a   record
    name : of blah
    date : int
end record

procedure MakeTwoLists (oldpointer, newpointer)
    temp <- new (PrivateRec)
    if (oldpointer = dead_private) OR (NIL)
        newpointer <- oldpointer^.data

    oldpointer <- oldpointer^.next
```

→ D → A

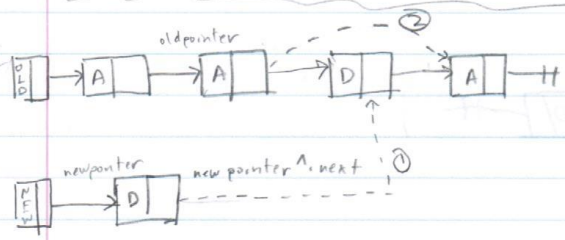
D → New list

temp → A

I



II



http://www.wg.omron.co.jp/cgi-bin/j-el

getjob → buzzwords
codes

404-894-4139

130.207.244.251

slirp

✓ Log on
✓ TCP/IP

HONOR STATEMENT
Fall 1998

Levi D. Smith
Post Office Box 326745
User: gte187k

I understand that all assignments associated with this course are to be completed by me without any unacknowledged help from other persons or documents. I understand that my first responsibility is to attempt to complete each assignment such that all the work is my own.

I understand that, if I find myself unable to successfully complete an assignment due to lack of knowledge or understanding, it is my responsibility to take the initiative to get help and guidance from the professor, from the teaching assistants, and/or from my fellow students.

To the extent that I have completed work with the assistance of people other than instructional staff (i.e., other students, friends, etc.), such help must be explicitly acknowledged in the work I submit, including explicit identification of each individual who has helped (if a student, via his/her "gt number"), along with a brief statement summarizing the nature of the collaboration. I understand that submitting work that is based on assistance from other students while failing to explicitly acknowledge, in writing, the source and nature of that assistance constitutes plagiarism.

I understand that I may provide limited assistance to other students to the extent that I am able to do so. I understand that I may discuss ideas and concepts with other students, and that I may offer hints or suggestions. Whenever I provide assistance to another student on any submitted assignment, I understand that it is my responsibility to explicitly acknowledge that assistance, in writing. However, I also understand that I am not to share my solutions, whether complete or partial, with other students.

Furthermore, I understand that it is not permissible to make copies of my work in any form available to any student for any purpose. I also understand that the homework assignments in CS 1501 are not intended to be collaborative efforts and are not to be worked on in groups, whether on paper, on the computer, or via some other medium. With the limited exceptions noted above, I understand that the homework assignments I submit for evaluation are to contain the results of my efforts alone.

Finally, I understand that if I fail to follow the guidelines stated above, I may be charged with academic misconduct, which carries severe penalties not only for myself, but for all other parties who might also be involved in the act of academic misconduct. I will take the time to become familiar with Georgia Tech's definition of academic misconduct and the policies and procedures pertaining to academic misconduct.

September 28, 1998
Date

[p1]

```
ARRAYMAX is 100
ARRAYMIN is 1

problem_one_array defines a array [ARRAYMIN.. ARRAYMAX] of Num
```

[p2]

```
function
~~procedure~~ Fill_Array ( array_to_fill isoftype in ~~out~~ problem_one_array)
  MAGIC NUMBER is 101   // Magic Number is the number that is subtracted from
  ~~LIMITBREAK is 100~~  // to get the desired cell data
  i isoftype Num        // ex, to fill in numbers in cells (1 - 100) with 100 - 1,
                        // you can subtract the cell number from the magic number (101)
  i <- 1                // to get the correct cell data: 101 - 1 = 100, 101 - 2 = 99, ...
  loop                  // or if MAGIC NUMBER is 201 then you will get the nums from 200 - 101
    exit if ( i > ARRAYMAX ~~LIMITBREAK~~ )
    array_to_fill [i] <- MAGIC NUMBER ~~101~~ - i
    i <- i + 1
    endloop
  Fill_Array returns array_to_fill
endfunction
```

[p3]

```
procedure Print_Less_Than_50 (array_to_print isoftype in problem_one_array)

  i isoftype Num
  i <- 1

  loop
    exit if (i > ARRAYMAX)
    if (array_to_print [i] < 50)
      print (array_to_print[i])
    endif
    i <- i + 1
  endloop
end procedure
```

[p4] ROWMIN is 1
ROWMAX is 70

SEATMIN is 1
SEATMAX is 50

Row_Array defines a array [ROWMIN .. ROWMAX] of String
Seat - Array defines a array [SEATMIN .. SEATMAX] of ~~Row~~ Row Array

[p5]

```
procedure Print_Out_Seat ( Search_Array of in Seat_Array, Name of in String)
    i, j of Num
    i <- 1
    loop
    exit if (( i > ROW ) or ( Search_Array [i] [j] = Name )
    j <- 1
    loop
    exit if ( Search_Array [i] [j] = Name )
    j <- j + 1
    end loop
    i <- i + 1
    end loop
    print ( Name, "'s seat is Row ", i, " Seat ", j " )
```

[p6] 1, 9, 4, 18, 5, 6, 33

[p7] Bubble Sort takes the first two numbers, and puts the larger Num (or leaves) to the right. Then it takes the 2nd and 3rd and puts (or leaves) the larger Num to the right. It ~~continues this process until~~ finishes when it gets to end of the array. Overall, the largest number ends up at the far right. This process must be repeated over and over if all the numbers in the array are to be in correct order.

Merge Sort - The array is divided in half (making two sub-arrays), and the two halves are divided into halves (making 4 sas), and this process continues until each sub-array only contains one cell. The

recursion - a function calls itself

Iteration vs. Recursion

Loops

not necessarily

1.) Exit condition
2.) Steps toward exit condition
3.) Calls itself

$3 \times 4 = 12$

$3 + 3 + 3 + 3$

$3 + (3 \times 3)$

$\hookrightarrow 3 + (3 \times 2)$

$\hookrightarrow 3 + (3 \times 1)$

```
function Multiply returns a Num (Num1, Num2 as in Num)
    if (Num2 = 1)
        Multiply returns Num1
    else
        Multiply returns Num1 + (Multiply (Num1, Num2 - 1))
    end if
end function
```

3

Multiply (3, 2)	3 + Multiply (3, 1)
Multiply (3, 3)	3 + Multiply (3, 2)
Multiply (3, 4)	3 + Multiply (3, 3)

algo —
~~x~~ int Num
x ← Multiply (3, 4)

[p 1]

a.) the name that is given to a variable once it enters a module

b.) the name of a variable that is used in the main algorithm

c.) An In parameter lets the data from a variable enter a module. Any changes to the var in the module will not be sent back to the main alg

d.) An Out parameter only allows changes to a variable in a procedure to be sent back to the main algorithm. Values for variables of this type cannot be read from the main alg.

e.) An In/Out parameters lets values for a variable to be read from the main alg to a mod and sent back to the main alg from a procedure once the variable has been manipulated.

[p 4]

```
procedure Grade_Average (grade_one, grade_two,
        grade_three isoftye in Num, grade_avg isoftype
out Num, test_or_lab is of type in String)
  // Purpose: to read in three grades and return
  // their average
   NUM_OF_GRAES is 3
   print ("Enter the first", test_or_lab, "grade.")
   read (grade_one)
                        second
   read (grade_two)
                        third
   read (grade_three)
   grade_avg <- (grade_one + grade_two + grade_three) / NUM_OF_GRADES
   print ("the Average of the", NUM_OF_GRADES, " is ", grade_avg
end procedure // Grade_Average
```

3 ping
> tracert

3 nslookup

Comment → Name, gt#
Sentence

winipcfg

[p5]

```
function Overall_Avg returns a Num (avg_lab,
avg_test, exam isoftype in Char)
  // Takes in three averages, weights them,
  // and returns an overall average.
  ~~// There is only one exam grade, but the~~
  ~~// name avg_exam keeps the names c~~
  WEIGHTLAB is 0.2
  WEIGHTTEST is 0.3
  WEIGHTEXAM is 0.5

  Overall_Avg returns (avg_lab * WEIGHTLAB) +
    (avg_test * WEIGHTTEST) + (exam * WEIGHTEXAM)
end function //Overall_Avg
```

[p6]

```
function Letter_Grade returns a Char (
                number_grade isoftype in Num)
// Takes in a number grade and returns
// a letter grade
  if (number_grade ≥ 90) then
      Letter_Grade returns 'A'
  end if
  if (number_grade < 90 AND number_grade ≥ 80) then
      Letter_Grade returns 'B'
  end if
```

↓

$(3x \cdot 4y) + (2x \cdot y) = (3x \cdot 2x) +$

$(A \cdot \overline{B}) + (\overline{A} \cdot B)$

$(A \cdot \overline{A}) + (A \cdot B) + (\overline{B} \cdot \overline{A}) + (\overline{B} \cdot B)$

$\emptyset \qquad \emptyset$

$(A \cdot B) + (\overline{B} \cdot \overline{A})$

$\overline{A + B} = \overline{A} \cdot \overline{B}$

$A \oplus B = A \cdot \overline{B} + \overline{A} + B$

$x + \emptyset = x$	$x \cdot \emptyset = \emptyset$
$x + 1 = 1$	$x \cdot 1 = x$
$x + x = x$	$x \cdot x = x$
$x + \overline{x} = 1$	$x \cdot \overline{x} = \emptyset$

$\overline{\overline{x}} = x$

$x + \emptyset = x$	$x \cdot \emptyset = \emptyset$
$x + 1 = 1$	$x \cdot 1 = x$
$x + x = x$	$x \cdot \overline{x} = x$
$x + \overline{x} = 1$	$x \cdot \overline{x} = \emptyset$

$A + (A \cdot B) = A$

$A \cdot (A + B) = A$

$x + 1 = 1$	$x \cdot \emptyset = \emptyset$
$x + 0 = x$	$x \cdot 1 = x$
$x + \overline{x} = 1$	$x \cdot \overline{x} = \emptyset$
$x + x = x$	$x \cdot x = x$

$A \oplus B = A \cdot \overline{B} + \overline{A} + B$

ゲエスト　レクテゥレ　プロフ　ゼエベルハウス

Thurs — Mon　　S/C
CmpE - Test
Connection
CD-R
Tech / West GA
Sout Cal - arcade - coming to DSO - Yosh

p1]

```
Ship Rec definesa Record
  shipname iot String
  captainsname iot String
  piratesonboard iot Num
  isseaworthy iot Boolean
end Record // Ship Rec

Pirate Rec definesa Record
  name iot String
  age iot Num
  fleettenure iot Num
  hasscurvy iot Boolean
```

p2]

```
public Add ( Rec iot in RecType)

private Add (rec
head iot ;  Add2 (head, rec)
            Add 2( head ~~ , rec
```

DO NOT PASS IN HEAD PTR

p4] a parameter that lets a class use piles of any data type. they are used when you want to use one pile for more than one purpose.

p5] allows the building on existing code without having to copy, rewrite, or modify it, use it when you want to add only ~~one or a~~ a few things to existing code.
A videogame store has code to calculate the price for buying and renting Playstation games. The store then begins selling N64 games, but does not want to write the account program over again

p6] it allows one to manipulate objects of various classes and invoke methods on an object without knowing that object type