§12.8-12.11: Linked Lists

28 Nov 2005
CMPT14x
Dr. Sean Ho
Trinity Western University

Reminders:

- *journals in folder*
- *Quiz ch11 today*

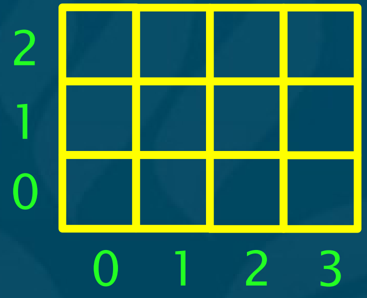
TRINITY WESTERN UNIVERSITY

http://cmpt14x.seanho.com/

Quiz ch11: 2 questions, 20 marks, 10 minutes

- [10] Translate into a CASE statement:
  - Hint: “No selector constant may be used twice in the list of selectors”

```
ReadChar (ans);
IF CAP (ans) = 'Y' THEN
    quit := TRUE
ELSIF (CAP (ans) >= 'A') AND (CAP (ans) <= 'Z') THEN
    quit := FALSE
ELSE
    error := TRUE
END;
```



- [10] Find a knight's tour of a 3x4 board starting from (0,0). Hint: next move: (row1,col2).
  - (Partial credit for showing backtracking work)

Quiz ch11 answers

- CASE statement:

```
    ReadChar (ans);
    CASE [1] CAP (ans) OF [1]
        'Y' [1] : [1]
            quit := TRUE | [1]
        'A' .. [1] 'X' [2], 'Z' :
            quit := FALSE |
        ELSE [2]
            error := TRUE
        END;
```

- Two knight's tours:

2	3	6	11	8
1	12	9	2	5
0	1	4	7	10
	0	1	2	3

2	3	6	9	12
1	8	11	2	5
0	1	4	7	10
	0	1	2	3

Review of last time (12.6-12.7)

- Endianness
- Pointer applications
  - Sorting using pointers
  - Resize-able dynamic array ADT
    - Type definition
    - Indexing the array
    - Creating a new array, resizing an existing one

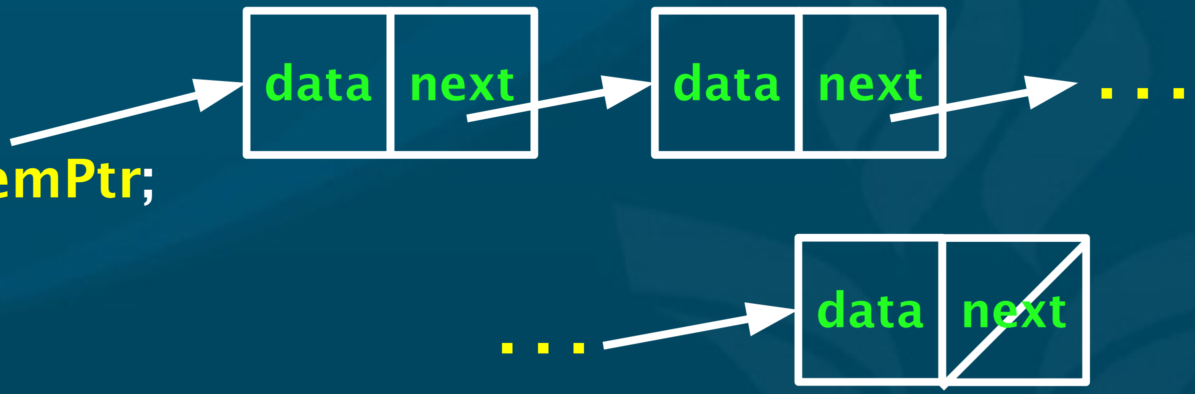
What's on for today (12.8-12.12)

- Linked lists
  - Type definition, creating a new list
    - Inserting in nth position
    - Insert at head, append to tail
    - Deleting
  - Algorithmic efficiency
  - Circularly linked lists
  - Bidirectional lists
- Trees
  - Binary search trees

Linked lists: creating

- A linked list is a dynamic ADT where each item in the list contains a pointer to the next item:

```
TYPE
    ListItemPtr = POINTER TO ListItem;
    ListItem = RECORD
        data : DataType;
        next : ListItemPtr;
    END;
VAR
    listHead : ListItemPtr;
BEGIN
    NEW (listHead);
    listHead^.next := NIL;
```



Operations on linked lists

- Insert an item in the nth position:

```
PROCEDURE Insert (head: ListItemPtr, data: DataType,
    pos: CARDINAL);
VAR
    cur, newitem : ListItemPtr;
    count : CARDINAL;
BEGIN
    cur := head;
    FOR count := 1 TO pos DO cur := cur^.next END;
    NEW (newitem);
    newitem^.data := data;
    newitem^.next := cur^.next;
    cur^.next := newitem;
```

Special cases of insert

- Insert() didn't work for head or tail of list
  - Also check if pos is beyond end of list
- Insert at head:

```
NEW (newitem);
newitem^.data := data;
newitem^.next := head;
head := newitem;              (* the new head of the list *)
```

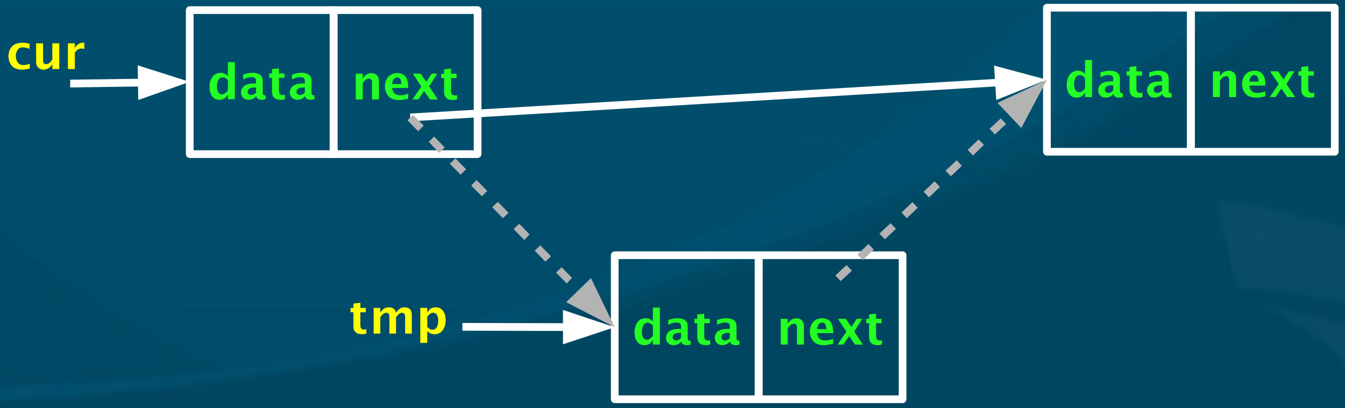
- Append to tail:

```
NEW (newitem);
newitem^.data := data;
newitem^.next := NIL;         (* mark end of list *)
cur^.next := newitem;         (* already set cur to tail *)
```

Deleting from a linked list

- Follow pointers to find the item we want to delete
  - Sew up links to skip over the item
  - Deallocate the item from memory

```
tmp := cur^.next;
cur^.next := tmp^.next;
DISPOSE (tmp);
```



Linked lists: algorithmic efficiency

- Big-O notation: O(n) means # operations varies linearly with n
- For a linked list with n items:
  - Insert at head: don't have to traverse list: O(1)
  - Append to tail: must walk list: O(n)
  - General insert:
    - Worst-case: O(n)
    - Average-case: O(n/2), which is also O(n)
  - Deleting: also O(n)
- Double-headed list (keep a tail pointer):
  - Speeds up append-to-tail to O(1)

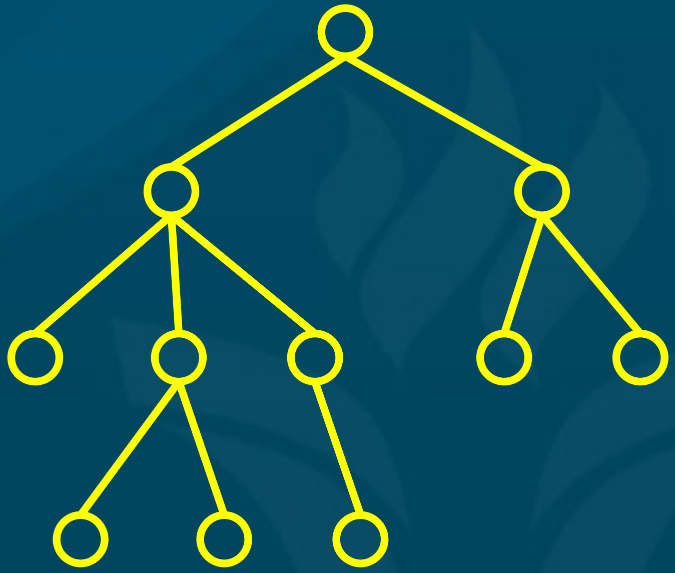
Variants of linked lists

- Circularly linked list:
    - tail^.next = head
- Bidirectional linked list:

```
TYPE
    ListItemPtr = POINTER TO ListItem;
    ListItem = RECORD
        data : DataType;
        prev : ListItemPtr;
        next : ListItemPtr;
    END;
```

Trees

- Another kind of dynamic ADT is the tree:
  - Root: starting node (one per tree)
    - Could also have a forest of several trees
  - Each node has at most one parent, and zero or more children
  - Leaves: no children
  - Depth: length of longest path from root
  - Degree: max # of children per node

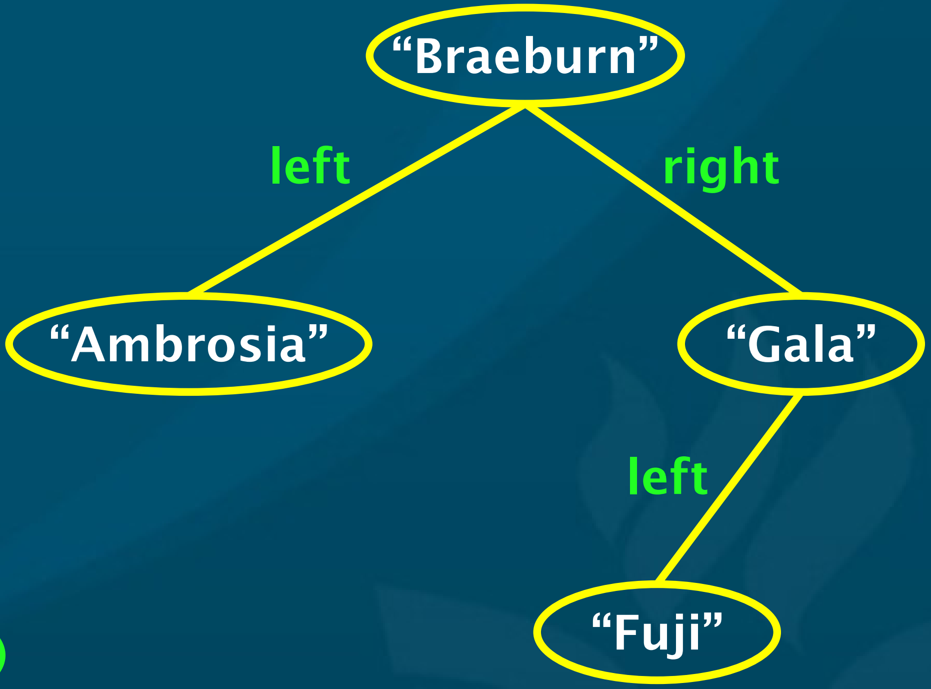


Binary search trees

- Binary trees (degree=2) are handy for keeping things in sorted order:

```
TYPE
    BinaryTree = POINTER TO
        BinaryTreeNode;
    BinaryTreeNode =
        RECORD
            data : String;
            left, right : BinaryTree;

            (* could also have root *)
        END;
```



BSTs and algorithmic efficiency

- Searching in a balanced binary search tree takes worst-case $O(\log n)$ running time:
  - Depth of balanced tree is $\log_2 n$

Review of today (12.8-12.11)

- Linked lists
  - Type definition, creating a new list
    - Inserting in nth position
    - Insert at head, append to tail
    - Deleting
  - Algorithmic efficiency
  - Circularly linked lists
  - Bidirectional lists
- Trees
  - Binary search trees

TODO items

- HW due Wed: 12.14 #43
- Reading: finish the book (yay!)
- No lab this week
- Paper due 1wk from Wed