Pre-Lists: Getting the hang of linked data structures

CMPT 115/117 lecture slides

Notes written by Mark Eramian, Ian McQuillan, Michael Horsch, Lingling Jin, and Dmytro Dyachuk

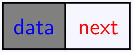
Objectives

After this topic, students are expected to

1. Explain the way records can link to records of the same type
2. Use linked records in simple expressions.
3. Draw diagrams of linked records.
4. Use linked records in simple algorithms.

Data Structures: Node record type to link things together

```
Node
    Element data;       // placeholder type
    refToNode next;  // points to another node
end Node
```



- A `node` record can link one node to another node.
- We'll need a node record for each element in a list.
- `Element` is a placeholder type ; we'll see examples that store integers or strings. . . Focus on the linking, not the data here!

Draw a diagram for the following pseudo-code sequence:

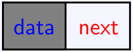
```
Node *x ← allocate new Node
x *⇒ data ← 5
x *⇒ next ← NULL

Node *y ← allocate new Node
y *⇒ data ← 1
y *⇒ next ← x

Node *z ← allocate new Node
z *⇒ data ← 8
z *⇒ next ← y

print (z *⇒ data)
print (z *⇒ next *⇒ data)
print (z *⇒ next *⇒ next *⇒ data)
```

Remember:


data next

- x *⇒ $data$ means $(*x).data$
- x *⇒ $next$ means $(*x).next$

Warning:

You cannot do this reliably in your head. Draw a diagram.

What's displayed?

```
Node *x ← allocate new Node
x *⇒ data ← "This"
x *⇒ next ← NULL

Node *y ← allocate new Node
y *⇒ data ← "is"
y *⇒ next ← x

Node *z ← allocate new Node
z *⇒ data ← "new"
z *⇒ next ← y

// Now do the two exercises i
// on the right.
```

```
Node *here ← z
print (here *⇒ data)
here ← here *⇒ next
print (here *⇒ data)
```

```
Node *there ← y
while (there != NULL)
  print (there *⇒ data)
  there ← there *⇒ next
done
```

Warning:

You cannot do this reliably in your head. Draw a diagram.

Practice expressions

```
Node *x ← allocate new Node
x *⇒ data ← 2
x *⇒ next ← NULL

Node *y ← allocate new Node
y *⇒ data ← 3
y *⇒ next ← x

Node *z ← allocate new Node
z *⇒ data ← 5
z *⇒ next ← y
```

Using the data on the left, display an expression that:

1. Evaluates to 1
2. Evaluates to 7
3. Evaluates to 15

Example solution:

1. Evaluates to 1: `(y*=>data) - (x*=>data)`

Practice expressions

```
Node *head
Node *n ← allocate new Node
head ← n

n *=> data ← 2
n *=> next ← allocate new Node

n ← n *=> next
n *=> data ← 3
n *=> next ← allocate new Node

n ← n *=> next
n *=> data ← 5
n *=> next ← NULL
```

Using the data on the left, display an expression that:

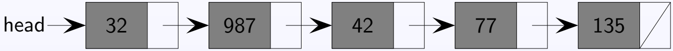
1. Evaluates to 1 without using the name n
2. Evaluates to 7 without using the name n
3. Evaluates to 15 without using the name n

Example solution:

1. Evaluates to 1: `(head*=>next*=>data) - (head*=>data)`

Simple algorithms on Node records

Suppose the variable `head` is a reference to the first node in the sequence:



Write simple pseudo-code to:

1. Display all numbers
2. Count the even numbers
3. Change the list so that 77 follows 987 (“delete 42”)
4. Drop 32 from the sequence (so `head` points to 987)
5. Add a new value 66 at the beginning of the sequence

NULL pointers

- NULL means “nothing”
- Precise meaning depends on context
  - No list
  - No nodes
  - No more nodes

You must always check if a pointer is NULL before you dereference it.