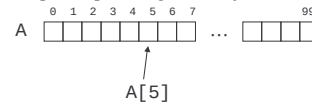


Pointers and Lists

CSE 326
Data Structures
Lecture 5

Basic Types and Arrays

- Basic Types
 - › integer, real (floating point), boolean (0,1), character
- Arrays
 - › $A[0..99]$: integer array

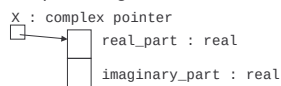


Pointers and Lists - Lecture 5

2

Records and Pointers

- Record (also called a struct or data object)
 - › Group data together that are related



- › To access the fields we use "dot" notation.

X.real_part
X.imaginary_part

Pointers and Lists - Lecture 5

3

Record Definition

- Record definition creates a new type

Definition

```
record complex : (  
  real_part : real,  
  imaginary_part : real  
)
```

Use in a declaration

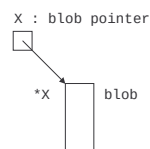
X : complex

Pointers and Lists - Lecture 5

4

Pointer

- A pointer is a reference to a variable or record (or object in Java world).



- In C, if X is of type pointer to Y then *X is of type Y

Pointers and Lists - Lecture 5

5

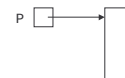
Creating a Record

- We use the "new" operator to create a record.

P : pointer to blob;

P  (null pointer)

P := new blob;

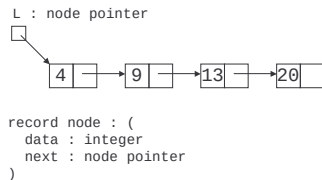


Pointers and Lists - Lecture 5

6

Simple Linked List

- A linked list
 - › Group data together in a flexible, dynamic way.
 - › We'll describe several list ADTs later.



Pointers and Lists - Lecture 5

7

Memory Management – Global Allocator

- Global Allocator's store – always get and return blocks to global allocator
 - + Necessary for dynamic memory.
 - + Blocks of various sizes can be merged if they reside in contiguous memory.
 - Allocator may not handle blocks of different sizes well.
 - Allocator may be slower than a private store.

Pointers and Lists - Lecture 5

8

Memory Management – Garbage Collection

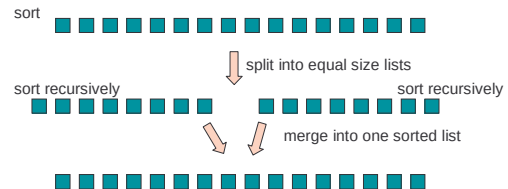
- Garbage collection – run time system recovers inaccessible blocks from time-to-time. Used in Lisp, Smalltalk, Java.
 - + No need to return blocks to an allocator or keep them in a private store.
 - Care must be taken to make unneeded blocks inaccessible.
 - When garbage collection kicks in there may be undesirable response time.

Pointers and Lists - Lecture 5

9

List Mergesort

- Overall sorting plan



Pointers and Lists - Lecture 5

10

Mergesort pseudocode

```

Mergesort(p : node pointer) : node pointer {
  Case {
    p = null : return p; //no elements
    p.next = null : return p; //one element
    else
      d : duo pointer; // duo has two fields first,second
      d := Split(p);
      return Merge(Mergesort(d.first),Mergesort(d.second));
  }
}
  
```

Note: Mergesort is destructive.



Pointers and Lists - Lecture 5

11

Split

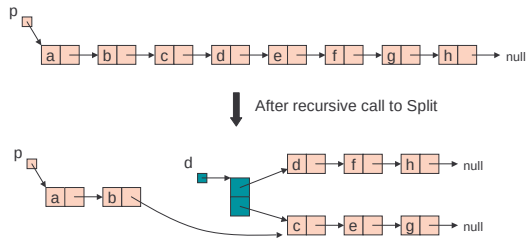
```

Split(p : node pointer) : duo pointer {
  d : duo pointer;
  Case {
    p = null : d := new duo; return d
    p.next = null : d := new duo; d.first := p ; return d
    else :
      d := Split(p.next.next);
      p.next.next := d.first;
      d.first := p.next;
      p.next := d.second;
      d.second := p;
      return d;
  }
}
  
```

Pointers and Lists - Lecture 5

12

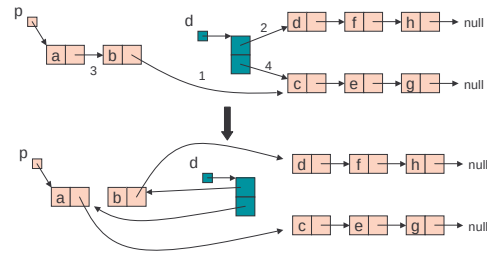
Split Example



Pointers and Lists - Lecture 5

13

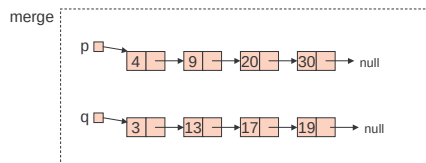
Split Example



Pointers and Lists - Lecture 5

14

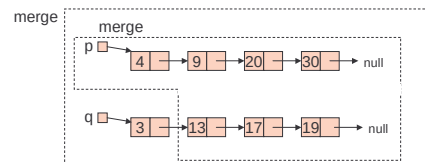
Merge Example



Pointers and Lists - Lecture 5

15

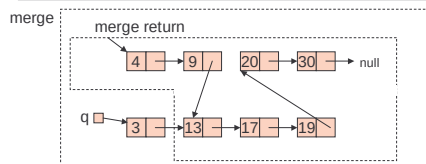
Merge Example



Pointers and Lists - Lecture 5

16

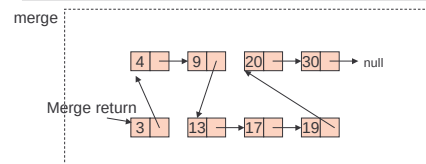
Merge Example



Pointers and Lists - Lecture 5

17

Merge Example



Pointers and Lists - Lecture 5

18

Merge Pseudocode

```
Merge(p,q : node pointer): node pointer{
  \precondition p and q point to sorted lists
```

```
}
```

Pointers and Lists - Lecture 5

19

Implementing Pointers in Arrays – “Cursor Implementation”

- This is needed in languages like Fortran, Basic, and assembly language
- Easiest when number of records is known ahead of time.
- Each record field of a basic type is associated with an array.
- A pointer field is an unsigned integer indicating an array index.

Pointers and Lists - Lecture 5

20

Idea

Pointer World

n nodes
data next

data : basic type
next : node pointer

Nonpointer World

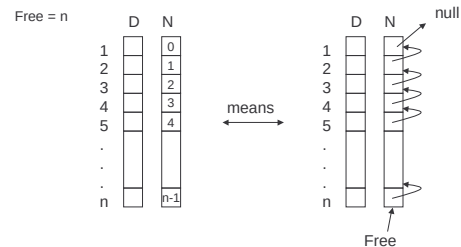
	D	N
1		
2		
3		
4		
5		
.		
.		
.		
n		

- D[] : basic type array
- N[] : integer array
- Pointer is an integer
- null is 0
- p.data is D[p]
- p.next is N[p]
- Free list needed for node allocation

Pointers and Lists - Lecture 5

21

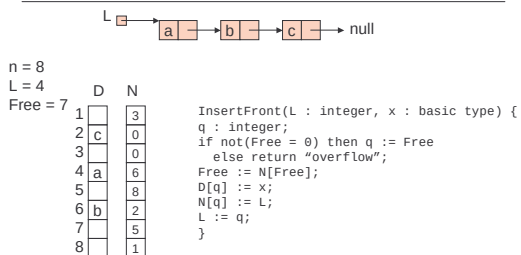
Initialization



Pointers and Lists - Lecture 5

22

Example of Use



Pointers and Lists - Lecture 5

23

Try DeleteFront

- Class Participation
- Define the cursor implementation of DeleteFront which removes the first member of the list when there is one.
 - › Remember to add garbage to free list.

```
DeleteFront(L : integer) {
  ???
}
```

Pointers and Lists - Lecture 5

24

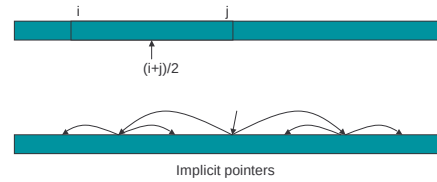
Pointer Summary

- Pointers can be implemented in several ways
 - › Explicit using “dot” notation in most high level languages
 - › Explicit using arrays of indices in all high level languages. (Cursor implementation)
 - › Implicit using calculation instead of storage. (Commonly used in nice structures like trees)

Pointers and Lists - Lecture 5

25

Implicit Pointers in Binary Search



Pointers and Lists - Lecture 5

26

DeleteFront Solution

```
DeleteFront(L : integer) {  
  q : integer;  
  if L = 0 then return "underflow"  
  else {  
    q := L;  
    L := N[L];  
    N[q] := Free;  
    Free := q;  
  }  
}
```

Pointers and Lists - Lecture 5

27