CS 126 Lecture P4:
An Example Program

Outline

- **Introduction**
- Program
  - Data structures
  - Code
- Conclusions

Goals

- Gain insight of how to put together a “large” program
- Learn how to read a “large” program
- Appreciate the central role played by data structures
- Master the manipulation of linked lists (pointers)

Central Role of Data Structures

- How to choose data structure
  - Ease of programming
  - Time efficient
  - Space efficient
- Design of algorithms is largely design of data structures
  - Data structures largely determine the algorithms

Outline

- ~~Introduction~~
- **Program**
  - **Data structures**
  - Code
- Conclusions

Represent A Single Card

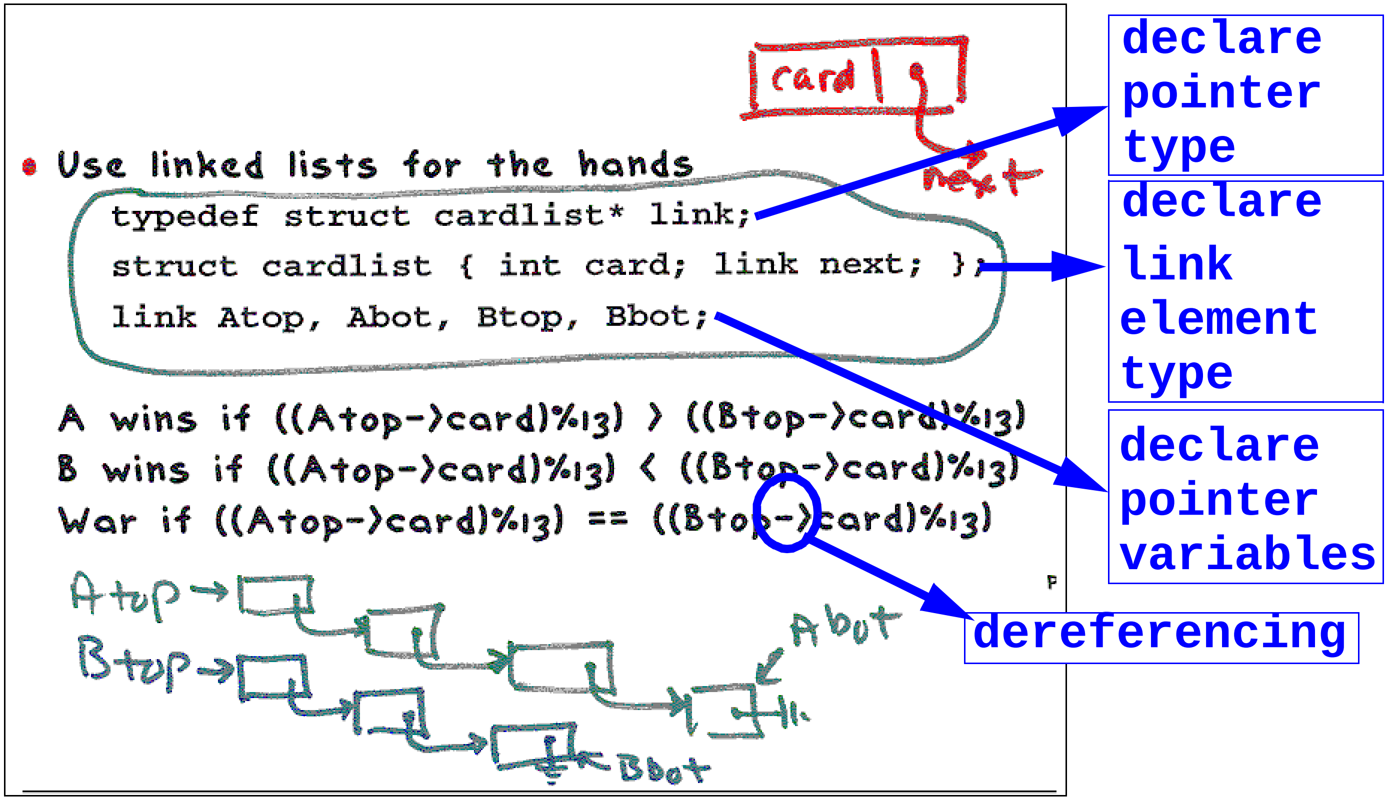
Use integers 0-51 for the cards

C	D	H	S
0	13	26	39
1	14	27	40
2	15	28	41
3	16	29	42
4	17	30	43
5	18	31	44
6	19	32	45
7	20	33	46
8	21	34	47
9	22	35	48
10	23	36	49
11	24	37	50
12	25	38	51

`card % 13`: face value

`card / 13`: kind

Represent the Decks

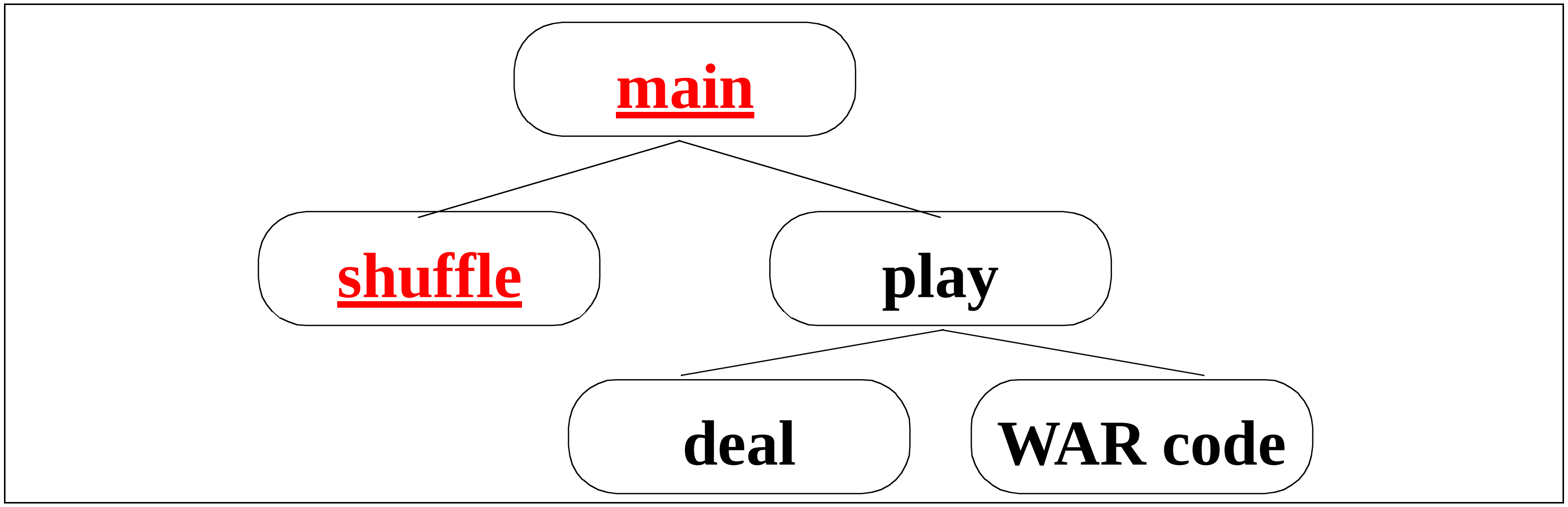


- Why linked lists?
  - We want you to learn linked lists :)
  - Little need for fast random access of the deck, mostly at the top and bottom of the stack

Sample game of War

```
A: 10  1 13  0 24 ... 11  7 29 26 41
B: 51 21 43 38 44 ... 45  2 50 48  9

A:  1 13  0 24 27 ...  7 29 26 41
B: 21 43 38 44  6 ...  2 50 48  9 51 10

A: 13  0 24 27 13 ... 29 26 41
B: 43 38 44  6 39 ... 50 48  9 51 10  1 21

A:  0 24 27 13 36 ... 26 41
B: 38 44  6 39  4 ... 48  9 51 10  1 21 16 43

A: 24 27 13 36 40 ... 41
B: 44  6 39  4  5 ...  9 51 10  1 21 16 43 38  0

A: 27 13 36 40 14 ... 44 24
B:  6 39  4  5 47 ... 51 10  1 21 16 43 38  0

A: 13 36 40 14 35 ... 24
B: 39  4  5 47 12 ... 10  1 21 16 43 38  0 27  6

A: ... 24
B: ... 10  1 21 16 43 38  0 27  6 13 36 40 14 35 ...
                                  39  4  5 47 12
```

Outline



- ~~Introduction~~
- **Program**
  - ~~Data structures~~
  - **Code**
- Conclusions

main()

```
main()
  { int cnt;
    cnt = play(shuffle());
    if (Btop == NULL)
      printf("A wins in %d steps ", cnt);
    if (Atop == NULL)
      printf("B wins in %d steps ", cnt);
  }
```

returns a stack of cards

- Revisiting the concept of top-down design
- Revisit how to read code
- All your functions should be this short and readable (although the lecture notes don’t always practice this)

Create and shuffle the deck (algorithm)

Hard to do efficiently without an array (!)

Goal: create a linked list of random cards

- Fill an array with integers in order ← "create" cards
- Make a pass through to shuffle
  - * pick up a new card
  - * pick a random position among cards in hand
  - * exchange new card with card at that position

Ex: 10 cards

start with sorted cards

.	0	1	2	3	4	5	6	7	8	9	
.	0										
.	0	1									1
.	2	1	0								0
.	2	1	3	0							2
.	2	4	3	0	1						1
.	5	4	3	0	1	2					0
.	5	4	3	0	6	2	1				4
.	5	4	7	0	6	2	1	3			2
.	5	4	7	0	6	2	8	3	1		6
.	5	4	7	0	6	2	8	3	9	1	8

ime ↓

exchange

index of the card to exchange with

- Pass through array to build list

Create and shuffle the deck (code)

```
int randI(int i)
   { return rand() / (RAND_MAX/i + 1); }
link shuffle(int N)
   { int j, k, t;
     int a[N];
     link x, deck = malloc(sizeof *deck);
     for (k = 0; k < N; k++) a[k] = k;
     for (k = 1; k < N; k++)
       {
         j = randI(k);
         t = a[k]; a[k] = a[j]; a[j] = t;
       }
     x = deck; x->card = a[0];
     for (k = 1; k < N; k++)
       {
         x->next = malloc(sizeof *x);
         x = x->next; x->card = a[k];
       }
     x->next = NULL;
     return deck;
   }
```

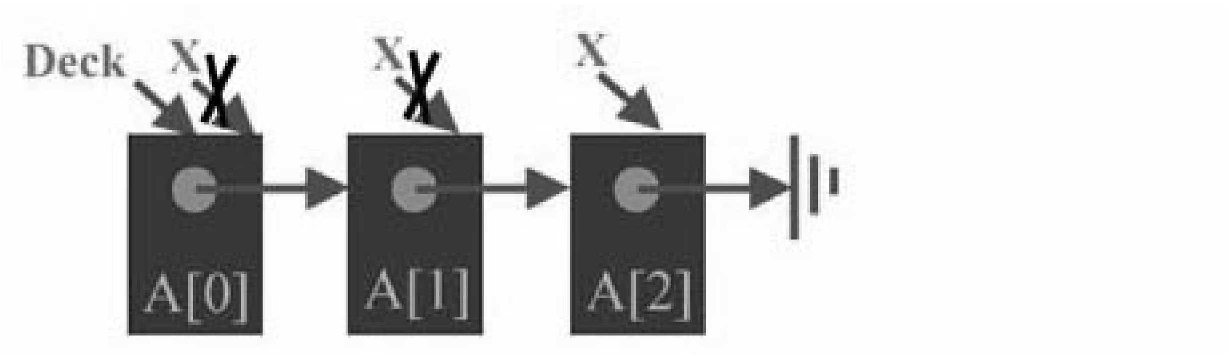
- randI: random int less than i
- **shuffle array**
  - fill array with sorted cards
  - for each card in the array
  - pick a random card in front of it
  - swap this and the random card
- **build linked list**
  - start the deck with the first card
  - for each card in the array
  - add this card to the bottom of deck
  - mark the end of the deck

Shuffle a linked list directly??

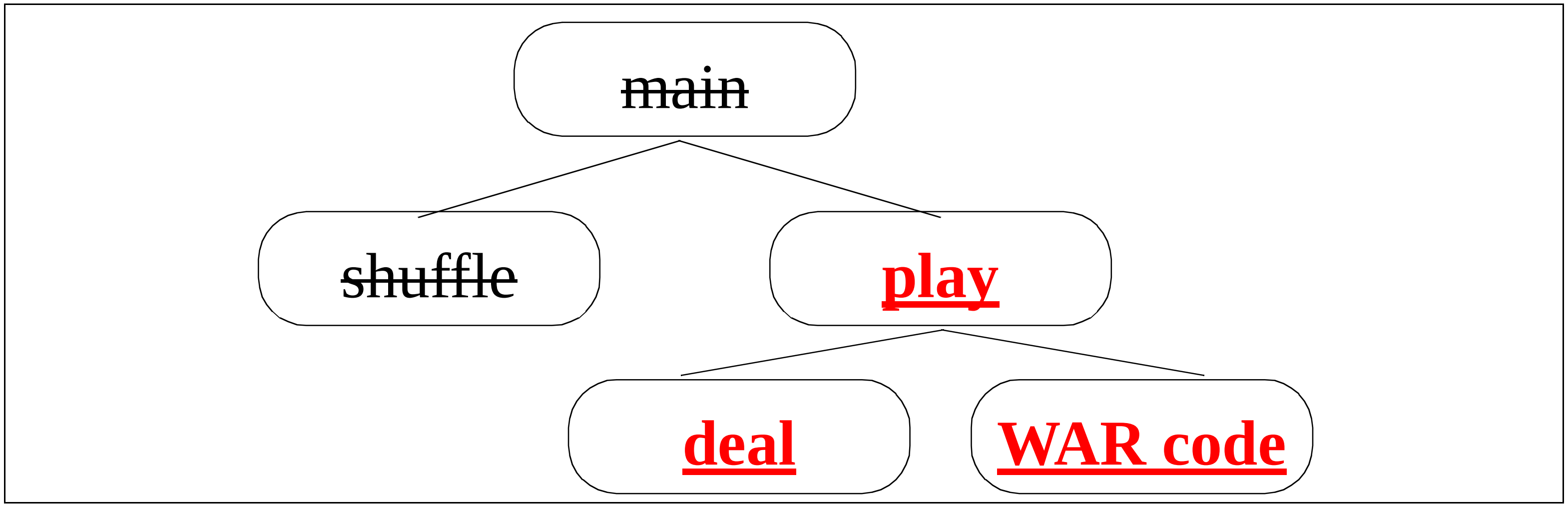
put ith card in random position?

works, but too slow for huge lists

Demo Part of `shuffle()`



Outline



- ~~Introduction~~
- **Program**
  - ~~Data structures~~
  - **Code**
- Conclusions

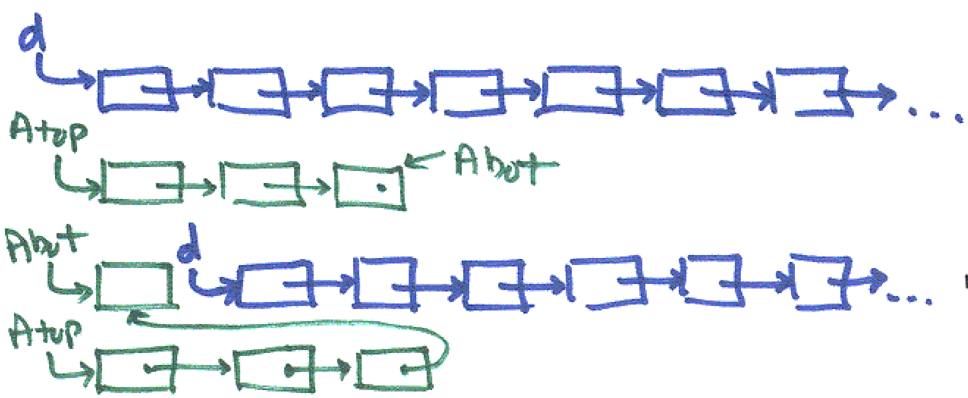
Deal the cards

- Function with a linked list as argument
- Makes two new linked lists for players A and B
- Sets global variables
  - Atop, Abot: links to first, last nodes of A
  - Btop, Bbot: links to first, last nodes of B

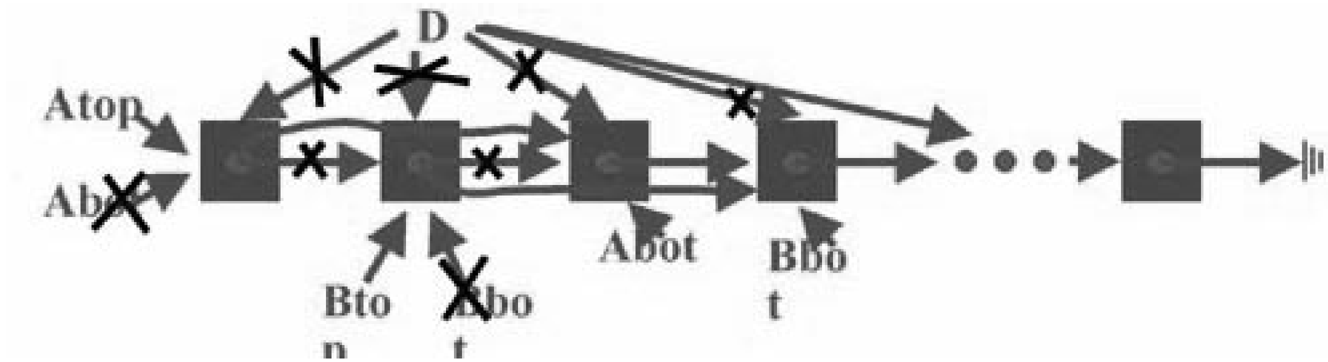
Does *not* create any new nodes

```
deal(link d)
  {
    Atop  = d; Abot = d; d = d->next;
    Btop  = d; Bbot = d; d = d->next;
    while (d != NULL)
      {
        Abot->next = d; Abot = d; d = d->next;
        if (d == NULL) break;
        Bbot->next = d; Bbot = d; d = d->next;
      }
    Abot->next = NULL; Bbot->next = NULL;
  }
```

“move” one card from deck to A pile
“move” one card from deck to B pile
As long as the deck is not empty
move one more from deck to A
stop if the deck is empty
move one more from deck to B
end of piles are marked



Demo deal()



Peace (war with no wars)

Starting point for implementation
("Why do we have wars, anyway?")


```
int play(link deck)
  { int Aval, Bval, cnt = 0; link Ttop, Tbot;
    deal(deck);
    while  ((Atop != NULL) && (Btop != NULL))
      { cnt++;
        Aval = Atop->card % 13;
        Bval = Btop->card % 13;
        Ttop = Atop; Tbot = Btop;
        Atop = Atop->next; Btop = Btop->next;
        Ttop->next = Tbot; Tbot->next = NULL;
        if (Aval > Bval)
          {
            if (Atop == NULL) Atop = Ttop;
              else Abot->next = Ttop;
            Abot = Tbot;
          }
        else
          {
            if (Btop == NULL) Btop = Ttop;
              else Bbot->next = Ttop;
            Bbot = Tbot;
          }
      }
    return cnt;
  }
```

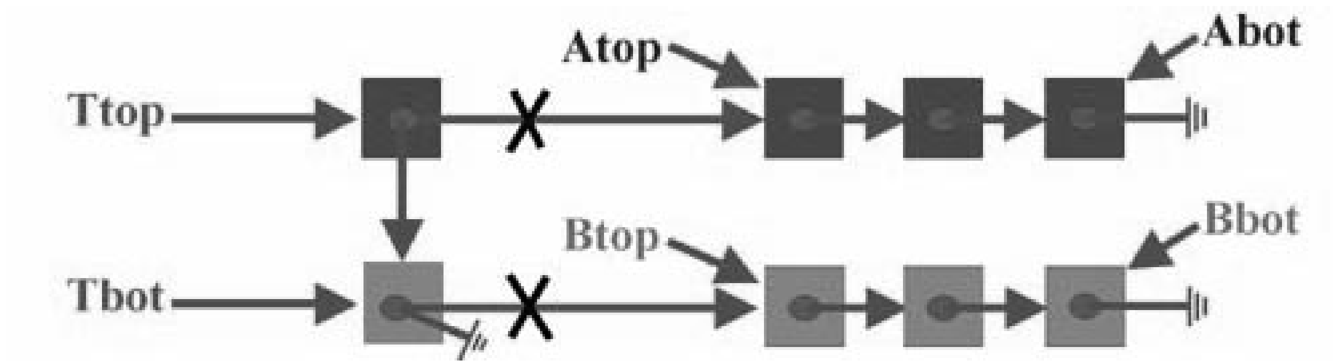
Take one card from each of the A, B piles and form a 2-card stack (Ttop, Tbot).

Put the 2-card stack at the bottom of the A pile

- lcc peace.c; a.out

Game "never" ends, for many (almost all?) deals
("Maybe *that's* why we have wars")

Demo `play()`



Add the code for war

Add the following code before the

```
        if (Aval > Bval)
```

test in "peace" code

put 8 cards in T pile

```
    while (Aval == Bval)
      {
         for (i = 0; i <= WAR; i++)
           {
             if (Atop == NULL) return cnt;
             Tbot->next = Atop; Tbot = Atop;
             Atop = Atop->next;
           }
         Aval = Tbot->card % 13;
         for (i = 0; i <= WAR; i++)
           {
             if (Btop == NULL) return cnt;
             Tbot->next = Btop; Tbot = Btop;
             Btop = Btop->next;
           }
         Bval = Tbot->card % 13;
      }
    Tbot->next = NULL;
```

move a number of cards from A pile to T pile

peek at top of A pile

"while" not "if", to handle multiple wars
BUG (?) A wins even if both empty on same war

- Game STILL *never* ends:
  thousands of moves, or more

Why?

- Assume two cards in battles are randomly exchanged when picked up

```
if (randI(2))
      { Ttop = Atop; Tbot = Btop; }
else { Ttop = Btop; Tbot = Atop; }
```

- Typical of simulation applications: proper use of randomness is vital!

- Ten typical games

B wins in 60 steps
A wins in 101 steps
B wins in 268 steps
B wins in 218 steps
B wins in 253 steps
A wins in 202 steps
A wins in 229 steps
B wins in 78 steps
B wins in 84 steps
B wins in 656 steps

Outline

- ~~Introduction~~
- ~~Program~~
  - ~~Data structures~~
  - ~~Code~~
- **Conclusions**

Answer

Q: "So, how long does it take?"

A: "About 10 times through the deck (254 battles)

Q: "How do you know?"

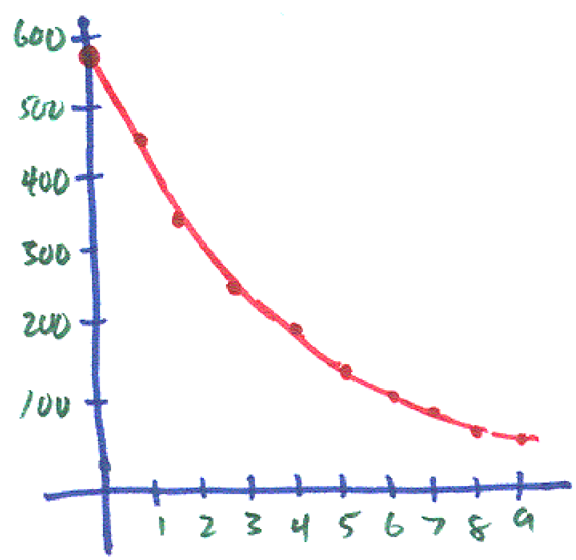
A: "I played a million games..."

Q: "That sounds like fun!"

A: "Let's try having bigger battles..."

[change value of WAR]

100000	trials
0	583
1	448
2	337
3	254
4	197
5	155
6	126
7	103
8	87
9	75



Problems with simulation

- Doesn't precisely mirror real game

- People pick up cards differently

- Separate hand, pile
  - requires much more code to handle
  - example: could have war as pile runs out
  - no real reason to simulate that part (?)
  - sort-of-shuffle pile after war?

- Tradeoff
  - convenience for implementation
  - fidelity to real game

Such tradeoffs typical in simulation
- try to identify which details matter

Stuff We Have Learned in This Lecture

- The process of constructing a “complex” program in a top-down fashion
- Reading a “complex” program to trace its top-down structure
- Judicious algorithm design starts with judicious choice of **data structures**
- Good examples of linked list (and pointer) manipulation
  - Draw pictures to read and write pointer codes