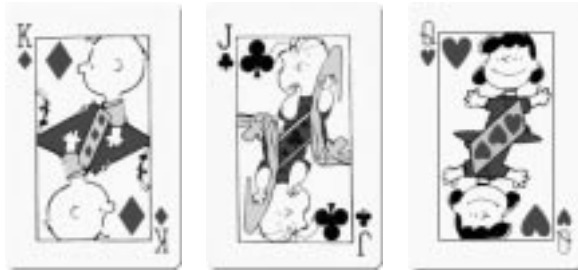
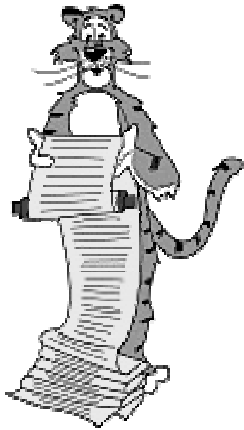


Lecture P9: WAR Card Game



Overview

Write a program to play the card game "War."

Goals.

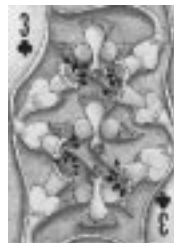
- Practice with linked lists and pointers.
- Appreciate the central role played by data structures.
- Learn how to design a "large" program.
- Learn how to read a "large" program.



WAR Demo

Rules of the game.

- Each player is dealt half of the cards.
- Each player plays top card.
 - whichever is higher captures both cards
 - in event of tie, WAR
- Repeat until one player has all the cards.



WAR demo. 

Before You Write Any Code

Determine a high-level view of the code you plan to write.

Break it up into manageable pieces.

- Create the deck of cards.
- Shuffle the cards.
- Deal the cards.
- Play the game.

Determine how you will represent the data.

- The cards.
- The deck.
- The hands.



Representing The Cards

Represent 52 cards using an integer between 0 and 51.

Clubs		Diamonds		Hearts		Spades	
Card	#	Card	#	Card	#	Card	#
2 ♣	0	2 ♦	13	2 ♥	26	2 ♠	39
3 ♣	1	3 ♦	14	3 ♥	27	3 ♠	40
4 ♣	2	4 ♦	15	4 ♥	28	4 ♠	41
...
K ♣	11	K ♦	24	K ♥	37	K ♠	50
A ♣	12	A ♦	25	A ♥	38	A ♠	51

6

Representing The Cards

Represent 52 cards using an integer between 0 and 51.

- War if (rank(c1) == rank(c2))



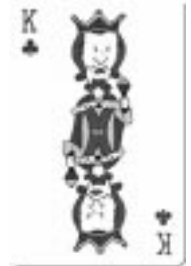
$$46 = 3 * 13 + 9$$

```

Card type
typedef int Card;

int rank(Card c) {
    return c % 13;
}

int suit(Card c) {
    return (c % 52) / 13;
}
    
```



12

c % 52 to allow for multiple deck war

7

Representing The Cards

```

Card type

void showcard(Card c) {
    switch (rank(c)) {
        case 0: printf("Deuce of "); break;
        case 1: printf("Three of "); break;
        . . .
        case 12: printf("Ace of " ); break;
    }

    switch (suit(c)) {
        case 0: printf("Clubs\n"); break;
        case 1: printf("Diamonds\n"); break;
        case 2: printf("Hearts\n"); break;
        case 3: printf("Spades\n"); break;
    }
}
    
```

8

Testing the Code

```

war.c (test code)

#include <stdio.h>
#define DECKSIZE 52

typedef int Card;

int rank(Card c) {...}
int suit(Card c) {...}
void showCard(Card c) {...}

int main(void) {
    Card c;
    for (c = 0; c < DECKSIZE; c++)
        showCard(c);
    return 0;
}
    
```

```

Unix

% gcc war.c
% a.out

Deuce of Clubs
Three of Clubs
Four of Clubs
Five of Clubs
Six of Clubs
Seven of Clubs

. . .

King of Spades
Ace of Spades
    
```

9

Representing the Deck and Hands

Use a linked list to represent the deck and hands.

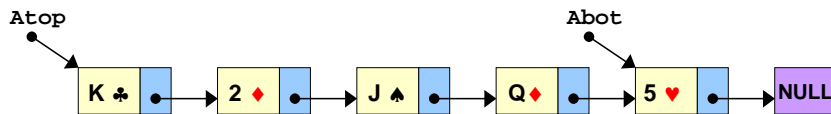
standard linked list structure

represent a pile of cards

```
typedef struct node* link;
struct node {
    Card card;
    link next;
};
```

maintain pointer to first and last card in A's pile

```
link Atop, Abot;
link Btop, Bbot;
```



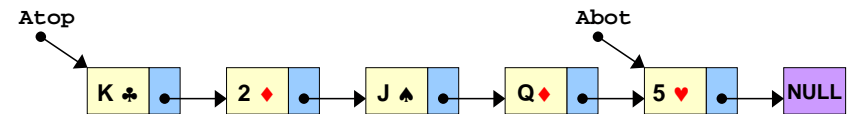
10

Representing the Deck and Hands

Use a linked list to represent the deck and hands.

Why use linked lists?

- Draw cards from the top, captured cards go to bottom.
 - need direct access to top and bottom cards
 - no need for direct access to middle cards
- Gain practice with linked lists.



11

Showing a Hand

Use printf() method for debugging.

- May need to build supplemental functions to print out contents of data structures.
- Print out contents of player's hand.

standard linked list traversal

showPile()

```
void showPile(link pile) {
    link x;
    for (x = pile; x != NULL; x = x->next)
        showCard(x->card);
}
```

12

Showing a Hand

Use printf() method for debugging.

- May need to build supplemental functions to print out contents of data structures.
- Print out contents of player's hand.
- Count number of cards in player's hand.

standard linked list traversal

countPile()

```
int countPile(link pile) {
    link x;
    int cnt = 0;
    for (x = pile; x != NULL; x = x->next)
        cnt++;
    return cnt;
}
```

13

Creating the Deck

Goal: create a 52 card deck.

- Need to dynamically allocate memory.
- Good programming practice to write helper function to allocate memory and initialize it.

```

needed for malloc() → #include <stdlib.h>
allocate memory → link NEWnode(Card card, link next) {
    link x;
    x = malloc(sizeof *x);
malloc() failed → if (x == NULL) {
    printf("Out of memory.\n");
    exit(EXIT_FAILURE);
    }
initialize node → x->next = next;
    x->card = card;
    return x;
    }
    
```

14

Creating the Deck

Goal: create a 52 card deck.

- Need to dynamically allocate memory.

```

x is link to top of pile → link makePile(int N) {
    link x = NULL;
    Card c;

add next card to top of pile → for (c = N - 1; c >= 0; c--)
    x = NEWnode(c, x);

    return x;
    }
    
```

15

Testing the Code

```

war.c
#include <stdio.h>
#include <stdlib.h>
#define DECKSIZE 52

typedef int Card;
[ rank(), suit(), showCard() ]

typedef struct node* link ...
link NEWnode(Card card, link next) {...}
link makePile(int N) {...}
link showPile(link pile) {...}

int main(void) {
    link deck;
    deck = makePile(DECKSIZE);
    showPile(deck);
    return 0;
}
    
```

```

Unix
% gcc war.c
% a.out

Deuce of Clubs
Three of Clubs
Four of Clubs
Five of Clubs
Six of Clubs
Seven of Clubs

. . .

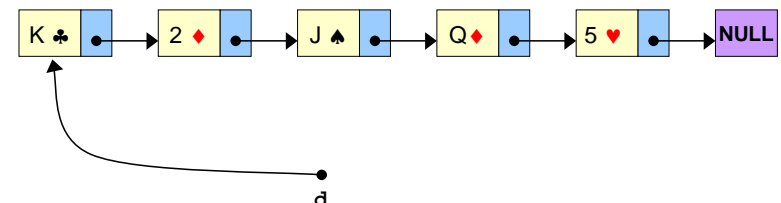
King of Spades
Ace of Spades
    
```

16

Dealing

Deal cards one at a time.

- Input: deck of cards (linked list).
- Creates: two new linked lists for players A and B.
 - global variable A_{top}, B_{top} point to first node
 - global variable A_{bot}, B_{bot} point to last node
- Does not create (malloc) new nodes.



17

Dealing Code

handle first card of each pile

assumes deck has even # cards

mark end of piles

```

deal()
void 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;
        Bbot->next = d; Bbot = d; d = d->next;
    }
    Abot->next = NULL; Bbot->next = NULL;
}
    
```

18

Testing the Code

```

war.c
... as before
link Atop, Abot, Btop, Bbot;

void deal(link d) { ... }

int main(void) {
    link deck;
    deck = makePile(DECKSIZE);
    deal(deck);
    printf("PLAYER A\n");
    showPile(Atop);
    printf("\nPLAYER B\n");
    showPile(Btop);
    return 0;
}
    
```

```

Unix
% gcc war.c
% a.out

PLAYER A
Deuce of Clubs
Four of Clubs
Six of Clubs
...
King of Spades

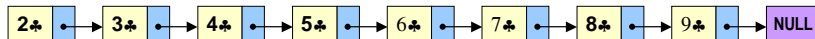
PLAYER B
Three of Clubs
Five of Clubs
Seven of Clubs
...
Ace of Spades
    
```

19

Shuffling the Deck

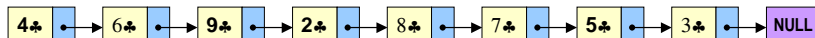
Shuffle the deck.

- Disassemble linked list elements and put into an array.
- Shuffle array elements (using algorithm from Lecture P3).
- Reassemble linked list from shuffled array.



Array index	0	1	2	3	4	5	6	7
Value	2♣	3♣	4♣	5♣	6♣	7♣	8♣	9♣

Array index	0	1	2	3	4	5	6	7
Value	4♣	6♣	9♣	2♣	8♣	7♣	5♣	3♣



20

Shuffling the Deck

```

shuffle pile of cards
link shufflePile(link pile) {
    int i, n;
    link x;
    link a[DECKSIZE];

    for (x = pile, n = 0; x != NULL; x = x->next, n++)
        a[n] = x;

    shuffle(a, n);
    for (i = 0; i < n - 1; i++)
        a[i]->next = a[i+1];
    a[n-1]->next = NULL;

    return a[0];
}
    
```

shuffle array elements

reassemble linked list

21

Testing the Code

```

war.c
... as before

int randomInteger(int n) { }
void shufflePile(link pile) { ...}

int main(void) {
    link deck;
    deck = makePile(DECKSIZE);
    deck = shufflePile(deck);
    deal(deck);
    printf("PLAYER A\n");
    showpile(Atop);
    printf("\nPLAYER B\n");
    showpile(Btop);
    return 0;
}
    
```

```

Unix
% gcc war.c
% a.out

PLAYER A
Eight of Diamonds
Ten of Hearts
Four of Clubs
...
Nine of Spades

PLAYER B
Jack of Hearts
Jack of Clubs
Four of Diamonds
...
Ten of Clubs
    
```

22

Peace Code

```

war.c
void play (void) {
    int Aval, Bval;
    link Ttop, Tbot;
    while ((Atop != NULL) && (Btop != NULL)) {
        Aval = rank(Atop->card);
        Bval = rank(Btop->card);
        Ttop = Atop; Tbot = Btop;
        Atop = Atop->next; Btop = Btop->next;
        Ttop->next = Tbot; Tbot->next = NULL;

        A wins
        B wins
        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;
        }
    }
}
    
```

Until a player loses

24

Game Never Ends

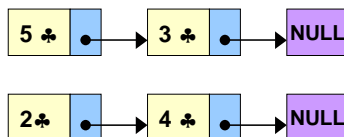
"Peace" (war with no wars).

- Starting point for implementation.
- Assume player B wins if a tie.

What should happen?



What actually happens?



25

One Bit of Uncertainty

What actually happens?

- Game "never" ends for many (almost all) deals.

Proper use of randomization is vital in simulation applications.

- Randomly exchange two cards in battle when picked up.

```

if (randomInteger(2) == 1) {
    Ttop = Atop;
    Tbot = Btop;
}
else {
    Ttop = Btop;
    Tbot = Atop;
}
    
```

exchange cards randomly

Ten Typical Games

```

B wins in 446 steps.
A wins in 404 steps.
B wins in 330 steps.
B wins in 1088 steps.
B wins in 566 steps.
B wins in 430 steps.
A wins in 208 steps.
B wins in 214 steps.
B wins in 630 steps.
B wins in 170 steps.
    
```

30

Add Code for War

Add code to handle ties.

- Insert in `play()` before `if (Aval > Bval)`

while not if to
handle multiple wars

A's war card

add WAR.SIZE cards
to temporary pile

B's war card

```
while (Aval == Bval) {
    for (i = 0; i < WAR.SIZE; i++) {
        if (A.top == NULL)
            return;
        Tbot->next = A.top; Tbot = A.top;
        A.top = A.top->next;
    }
    Aval = rank(Tbot->card);

    for (i = 0; i < WAR.SIZE; i++) {
        if (B.top == NULL)
            return;
        Tbot->next = B.top; Tbot = B.top;
        B.top = B.top->next;
    }
    Bval = rank(Tbot->card);
}
Tbot->next = NULL;
```

31

Answer

Q. "So how long does it take?"

A. "About 10 times through deck (254 battles)."

Q. "How do you know?"

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

Ten Typical Games

B wins in 60 steps.
A wins in 101 steps.
B wins in 268 steps.
A wins in 218 steps.
B wins in 253 steps.
A wins in 202 steps.
B wins in 229 steps.
A wins in 78 steps.
B wins in 84 steps.
A wins in 654 steps.

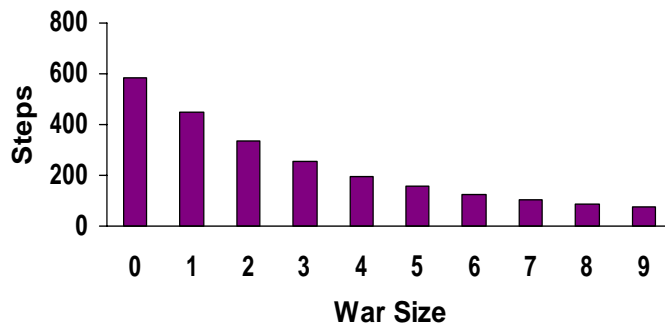
32

Answer

Q. "That sounds like fun."

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

Average # of Steps in War



33

Problems With Simulation

Doesn't precisely mirror game.

- Deal allocates piles in reversed order.
- People pick up cards differently.
- "Sort-of" shuffle prize pile after war?
- Separate hand and pile.
 - could have war as pile runs out
- Our shuffling produces perfectly random deck. (modulo "randomness" of `rand()`)

Tradeoffs.

- Convenience for implementation.
- Fidelity to real game.
 - Such tradeoffs are typical in simulation.
 - Try to identify which details matter.

34

War Using Queue ADT

Use first class queue ADT. Why queue?



```
deal()
Queue A, B;

void deal(Queue Deck) {
    A = QUEUEinit();
    B = QUEUEinit();

    while (!QUEUEisempty(Deck)) {
        QUEUEput(A, QUEUEget(Deck));
        QUEUEput(B, QUEUEget(Deck));
    }
}
```

35

War Using Queue ADT

Use first class queue ADT. Why queue?



```
peace.c

void play(Queue A, Queue B) {
    Card Acard, Bcard;
    Queue T = QUEUEinit();

    while (!QUEUEisempty(A) && !QUEUEisempty(B)) {
        Acard = QUEUEget(A); Bcard = QUEUEget(B);
        QUEUEput(T, Acard); QUEUEput(T, Bcard);
        if (rank(Acard) > rank(Bcard))
            while (!QUEUEisempty(T))
                QUEUEput(A, QUEUEget(T));
        else
            while (!QUEUEempty(T))
                QUEUEput(B, QUEUEget(T));
    }
}
```

36

War Using Queue ADT

Use first class queue ADT. Why queue?



Advantages:



Disadvantage:



37

Summary

How to build a "large" program?

- Use top-down design.
- Break into small, manageable pieces. Makes code:
 - easier to understand
 - easier to debug
 - easier to change later on
- Debug each piece as you write it.
- Good algorithmic design starts with judicious choice of data structures.

How to work with linked lists?

- Draw pictures to read and write pointer code.

38