Problem Solving 문제해결기법

2015 Spring Semester

Jinkyu Lee

Dept. of Computer Science and Engineering,
Sungkyunkwan University (SKKU), Republic of Korea

Contents

- **Chapter 1 – Getting started**
- Homework 2
- Chapter 2 – Date structure

Tips for programming

- Write the comments first
- Document each variable
- Use symbolic constants
- Use enumerated types for a reason
- Use subroutines to avoid redundant code
- Make your debugging statements meaningful

Tips for programming

- Write the comments first

Tips for programming

- Document each variable

Tips for programming

- Use symbolic constants
  - Math constant, e.g., PI
  - Length of data structure
  - Size of input

  - static int num_of_words = 10;

Tips for programming

- Use enumerated types for a reason

```
[Ex]  enum day { sun, mon, tue, wed, thu, fri, sat };

      typedef enum day  day;

      day find_next_day(day d) {
            day next_day;
            next_day =  (day) ( ( (int) d + 1 ) % 7 ) ;
            return  next_day;
      }
```

Tips for programming

- Use subroutines to avoid redundant code

```
while (c != '0') {
   scanf("%c", &c);
   if (c == 'A') {
       if (row-1 >= 0) {
           temp = b[row-1][col];
           b[row-1][col] = ' ';
           b[row][col] = temp;
           row = row-1;
       }
   }
   else if (c == 'B') {
       if (row+1 <= BOARDSIZE-1) {
           temp = b[row+1][col];
           b[row+1][col] = ' ';
           b[row][col] = temp;
           row = row+1;
       }
   }
```

Move(b,-1,row,col)

Move(b,1,row,col)

```
void Move(int b[][], int shift, int row, int col)
{
    int temp=b[row+shift][col];
    b[row+shift][col] = ' ';
    b[row][col] = temp;
    row = row+shift;
}
```

Tips for programming

- Make your debugging statements meaningful

Tips for programming

- Write the comments first
- Document each variable
- Use symbolic constants
- Use enumerated types for a reason
- Use subroutines to avoid redundant code
- Make your debugging statements meaningful

Problems in Chapter 1

- The 3n+1 problem (homework 1)
- Minesweeper (printed out)
- The trip
- LCD display
- Graphical Editor
- Interpreter
- Check the check
- Australian voting

Minesweeper

- Sample input

```
4 4
*...
....
.*..
....
3 5
**...
.....
.*...
0 0
```

- Sample output

```
Field #1:
*100
2210
1*10
1110

Field #2:
**100
33200
1*100
```

Contents

- Chapter 1 – Getting started
- **Homework 2**
- Chapter 2 – Date structure

Homework 2 - Boker hands

- A boker deck contains 52 cards. Each card has a suit of either clubs, diamonds, hearts, or spades (denoted C, D, H, S in the input data). Each card also has a value of either 2 through 10, jack, queen, king, or ace (denoted 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A). For scoring purposes card values are ordered as above, with 2 having the lowest and ace the highest value. The suit has no impact on value.

- A boker hand consists of three cards dealt from the deck. Boker hands are ranked by the following partial order from lowest to highest.

Homework 2 - Boker hands

- Boker hands
  - *High Card.* Hands which do not fit any higher category are ranked by the value of their highest card. If the highest cards have the same value, the hands are ranked by the next highest, and so on.
  - *Pair.* Two of the three cards in the hand have the same value. Hands which both contain a pair are ranked by the value of the cards forming the pair. If these values are the same, the hands are ranked by the values of the cards not forming the pair, in decreasing order.
  - Flush. Hand contains three cards of the same suit. Hands which are both flushes are ranked using the rules for High Card.

Homework 2 - Boker hands

- Boker hands
  - *Straight.* Hand contains three cards with consecutive values. Hands which both contain a straight are ranked by their highest card.
  - *Three of a Kind.* All the three cards in the hand have the same value. Hands which both contain three of a kind are ranked by the value of the three cards.
  - *Straight Flush.* Three cards of the same suit with consecutive values. Ranked by the highest card in the hand.
- Your job is to compare several pairs of boker hands and to indicate which, if either, has a higher rank. Each player has three cards, but each player can change the suit of one card so as to make the player's boker hand higher.

Homework 2 - Boker hands

- Input
  - The input file contains several lines, each containing the designation of six cards: the first three cards are the hand for the player named "Black" and the next three cards are the hand for the player named "White".
- Output
  - For each line of input, print a line containing one of the following:

    Black wins.

    White wins.

    Tie.

Homework 2 - Boker hands

- Sample input

  2C 4D 4H 2H 4S 4C

  5H 6H 7D AH KC 3S

  9D 5C 9H AH 9D AC

- Sample output

  Tie.

  Black wins.

  White wins.

Why such output?

Tie. (2C 4D 4H = 2H 4S 4C)

*Black wins. (5H 6H **7H** > AH KC 3S)*

White wins. (3D 5C 9H < 4H 9D AC)

Homework 2 - Multiplication

- For given positive integer values a and n, write an algorithm that calculates $b=a^n$ with the minimum number of multiplications.
  - For example, if n=13,

```
x1 := a;
x2 := x1*x1;
x3 := x2*x1;
x4 := x3*x1;
x5 := x4*x1;
x6 := x5*x1;
…
x13:=x12*x1;
b := x13;
```

Is this the minimum numbers?

  - The number of multiplications is 12.

Homework 2 - Multiplication

- For given positive integer values a and n, write an algorithm that calculates $b=a^n$ with the minimum number of multiplications.
  - For example, if n=13,

```
x1 := a;
x2 := x1*x1;
x3 := x2*x2;
x4 := x3*x1;
x5 := x3*x3;
x6 := x5*x4;
b := x6;
```

Is this the minimum numbers?

  - The number of multiplications is 5.

Homework 2 – Finding the Kth element

- We have two arrays X[1..n] and Y[1..n]. All elements in each array is sorted in an ascending order.
- For given K (1 <= K <= 2n), can you efficiently find the Kth largest elements among all 2n elements?
- Sample input:

  K=8

  2 5 6 7 10 13 14 19

  1 4 9 11 12 15 18 20

- Sample output:

  10

Homework 2 – in your C code

- Submit a source file whose name is Yourid_HW2.c For example, if your id is 2000123456, the file name should be 2000123456_HW2.c
  - Implement a C program for Problem 1 (Boker hand).
  - Add comments for you codes.

Homework 2 – in your report

- Submit a report whose file name is Yourid_HW2.yy For example, if your id is 2000123456 and your report file type is MS word, the file name should be 2000123456_HW2.docx
  - Explain your algorithm for Problems 2 and 3.
  - For Problem 2, show the number of multiplications with n=15, 20 and 25.
  - For Problem 3, show the worst-case number of items to be investigated, with n=8. Also, express the worst-case number of items to be investigated as big-O notation with general n.

Contents

- Chapter 1 – Getting started
- Homework 2
- **Chapter 2 – Date structure**

Tips for testing

- Test the given input
- Test incorrect input
- Test boundary conditions
- Test instances where you know the correct answer
- Test big examples where you don't know the correct answer

Sample Input	Sample Output
1 10	1 10 20
100 200	100 200 125
201 210	201 210 89
900 1000	900 1000 174

Tips for debugging

- Get to know your debugger
- Display your data structures
- Test invariant rigorously
- Inspect your code thoroughly
- Make your print statements mean something
- Make your arrays a little larger than necessary
- Make sure your bugs are really bugs

Program design example: going to war

- Pages 34 – 39
- In the children's card game War, a standard 52-card deck is dealt to two players (1 and 2) such that each player has 26 cards. Players do not look at their cards, but keep them in a packet face down. The object of the game is to win all the cards.
- Both players play by turning their top cards face up and putting them on the table. Whoever turned the higher card takes both cards and adds them (face down) to the bottom of their packet. Cards rank as usual from high to low: *A, K, Q, J, T, 9, 8, 7, 6, 5, 4, 3, 2*. Suits are ignored. Both players then turn up their next card and repeat. The game continues until one player wins by taking all the cards.

Program design example: going to war

- When the face up cards are of equal rank there is a war. These cards stay on the table as both players play the next card of their pile face down and then another card face up. Whoever has the higher of the new face up cards wins the war, and adds all six cards to the bottom of his or her packet. If the new face up cards are equal as well, the war continues: each player puts another card face down and one face up. The war goes on like this as long as the face up cards continue to be equal. As soon as they are different, the player with the higher face up card wins all the cards on the table.

Program design example: going to war

- If someone runs out of cards in the middle of a war, the other player automatically wins. The cards are added to the back of the winner's hand in the exact order they were dealt, specifically 1's first card, 2's first card, 1's second card, etc.
- As anyone with a five year-old nephew knows, a game of War can take a long time to finish. But how long? Your job is to write a program to simulate the game and report the number of moves.

Program design example: going to war

- Think about how to implement this program, focusing on data structure.

1. Example : Going to War – Intro. <1>

❖ Game Description

- Two players have 26 cards, respectively.
- Players keep them in a packet face down.
- The object of the game is to WIN all the cards!

❖ Game Rule

① Two players play by turning top cards face up and putting them on the table

② Whoever turned the higher card takes both cards and adds them (face down) to the bottom of their packet

③ Cards rank from high to low A,K,Q,J,T,9,8,7,6,5,4,3,2 (Suits are ignored)

④ Steps ①~③ continue until one player wins by taking all the cards

⑤ When the face up cards are equal rank in Step ②, there is a **WAR**!!

⑥ Putting the next card of their packet face down on the table

⑦ Turning up another card face up

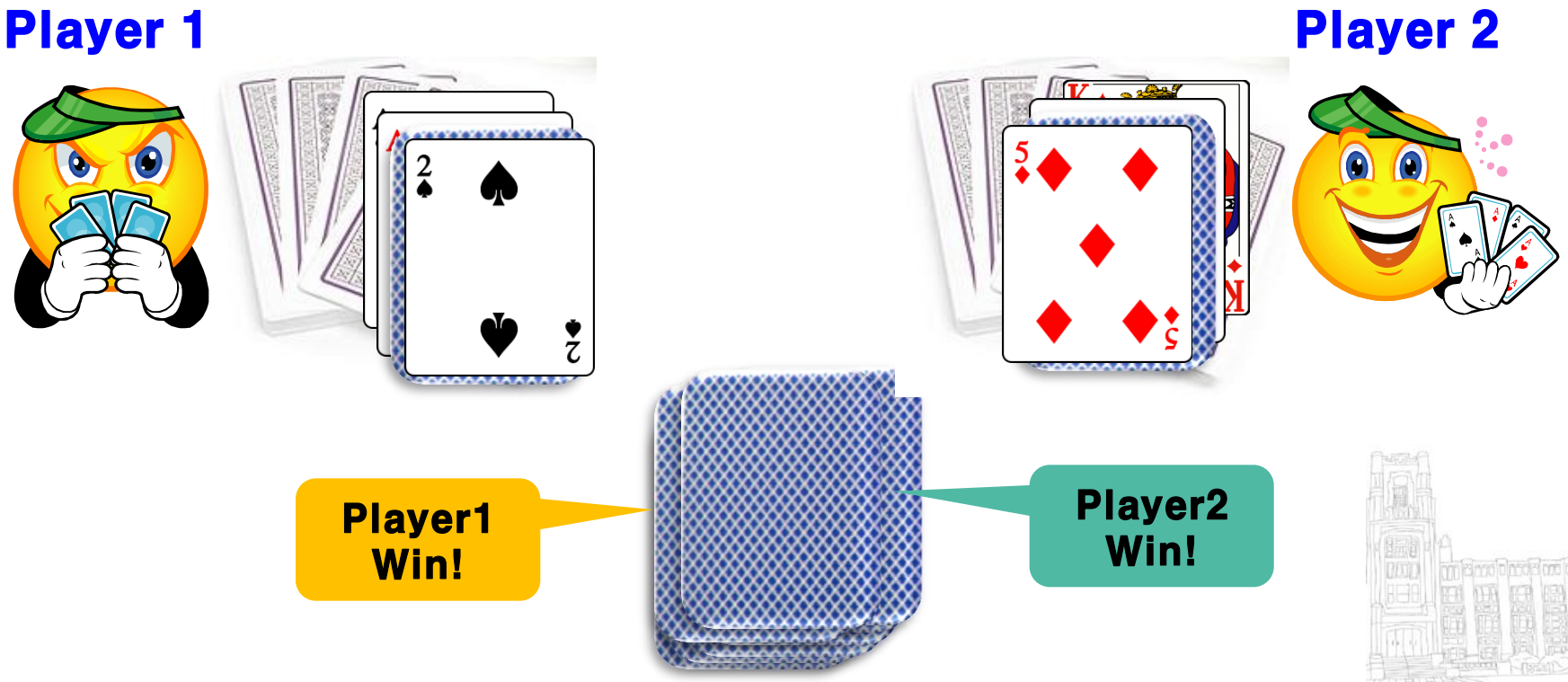
⑧ Whoever has the higher of the new face up cards win the war, and adds all six cards to the bottom of his packet.

⑨ If the new face up cards are equal as well, the war continues

1. Example : Going to War – Intro. <2>

❖ The Order of Added Cards

- The cards are added to the **back** of the winner's hand
- They are piled in the **exact order** they were dealt…
  - 1's first card, 2's first card,
    1's second card, 2's second card …



2. Example : Going to War – Solving <1>

❖ Which Data Structure to Represent a deck of cards?

- The answer depends on what you are going to do with them; the primary action defines the operation of the data structure!
- From our deck, we are dealing cards out from the top and adding them to the rear of our deck.

➔ It is natural to represent each player's hand using First-In First-Out (FIFO)!

Data Structure = Specification + Operation

Specification: Necessary elements for managing data

Operation: Operators performed on the elements for accessing data

3. Background – Queue <1>

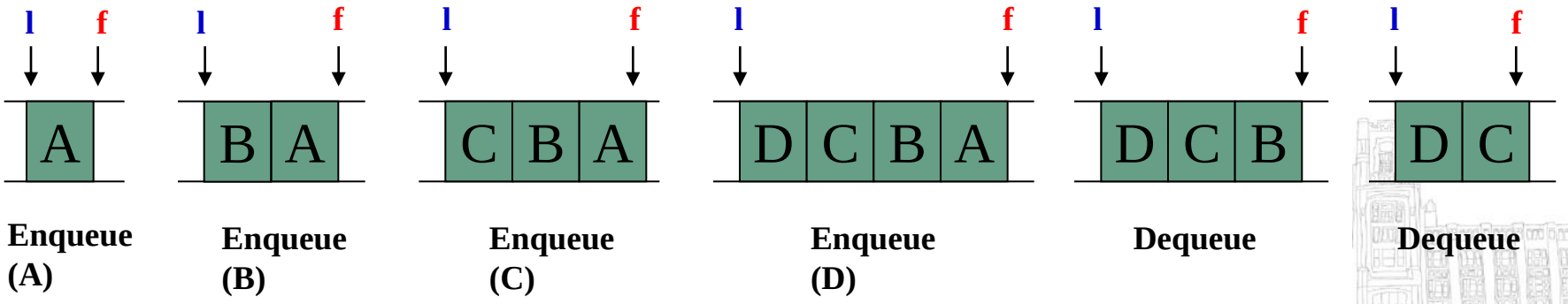
What is Queue??

- It represents FIFO (First-In-First-Out) structure.
- The element first put is the element first served!
- The input/output are performed independently

Operation

- Enqueue(x, Q) – Insert x item into the end of Q
- Dequeue(Q) – Return/delete the first item of Q
- Initialize(Q), Full(Q), Empty(Q)

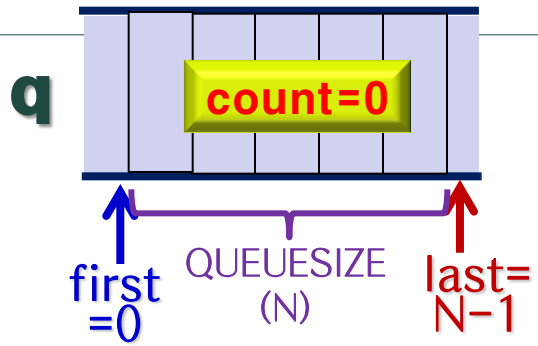
f: first, l: last



3. Background – Queue <2>

❖ Type Definition

```
typedef struct {
        int q[QUEUESIZE+1];          /* body of queue */
        int first;                   /* position of first element */
        int last;                    /* position of last element */
        int count;                   /* number of queue elements */
} queue;
```



❖ Initialization & Empty Check

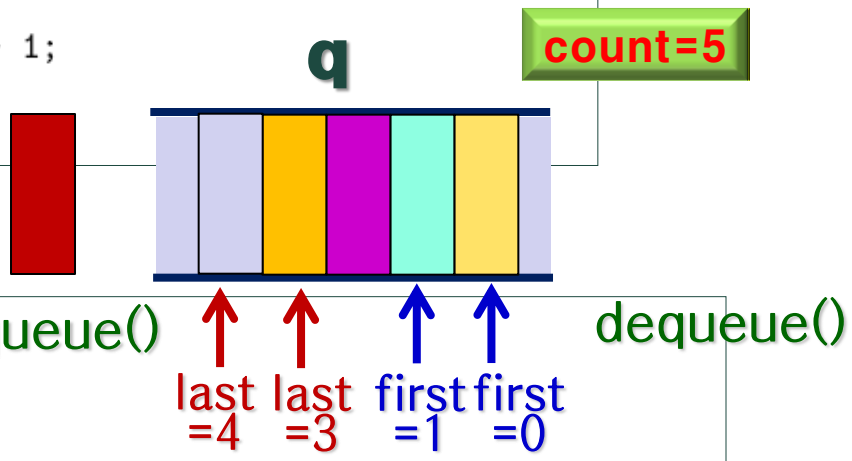
```
init_queue(queue *q)
{
        q->first = 0;
        q->last = QUEUESIZE-1;
        q->count = 0;
}
```

```
int empty(queue *q)
{
        if (q->count <= 0) return (TRUE);
        else return (FALSE);
}
```

3. Background – Queue <3>

Enqueue

```
enqueue(queue *q, int x)
{
        if (q->count >= QUEUESIZE)
                printf("Warning: queue overflow enqueue x=%d\n",x);
        else {
                q->last = (q->last+1) % QUEUESIZE;
                q->q[ q->last ] = x;
                q->count = q->count + 1;
        }
}
```



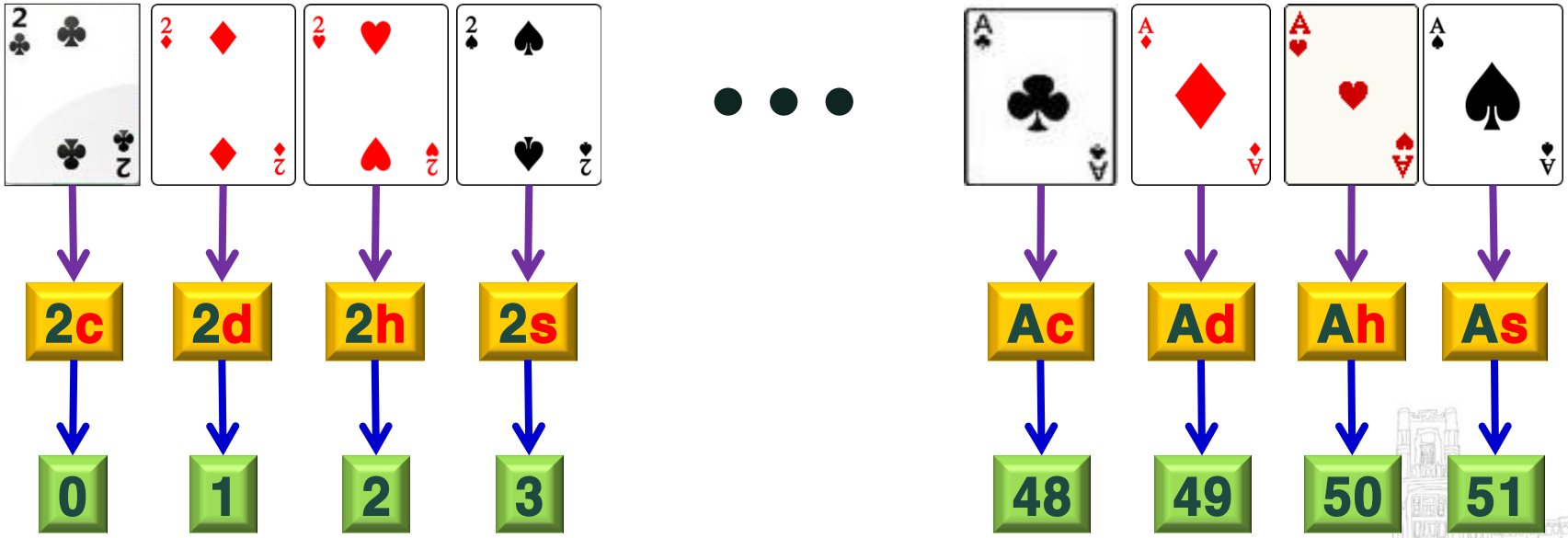
Dequene

```
int dequeue(queue *q)
{
        int x;

        if (q->count <= 0) printf("Warning: empty queue dequeue.\n");
        else {
                x = q->q[ q->first ];
                q->first = (q->first+1) % QUEUESIZE;
                q->count = q->count - 1;
        }
        return(x);
}
```

2. Example : Going to War – Solving <2>

How do we represent each card?

- Cards have both suits (clubs, diamonds, hearts, spades) and values (2-10, jack, queen, king, ace)
- A possible approach: each card [suit, value] is mapped into a distinct integers, 0 ~ 51
- How to map in back and forth between numbers and cards as needed?

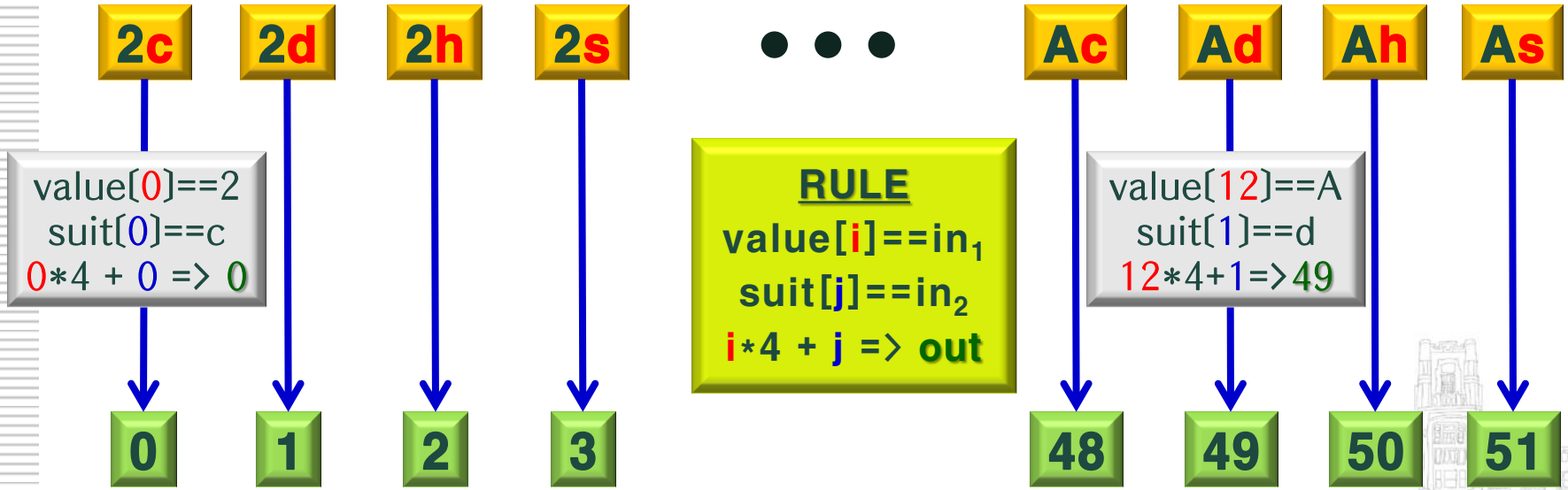


2. Example : Going to War – Solving <3>

How de we represent each card?

- Card ranks are 13 in total, and each rank has 4 suits
  - Thus, we can map each card into an integer by the following strings!

```
values = "23456789TJQKA"
suits  = "cdhs"
```



4. Example : Going to War – Solving <4>

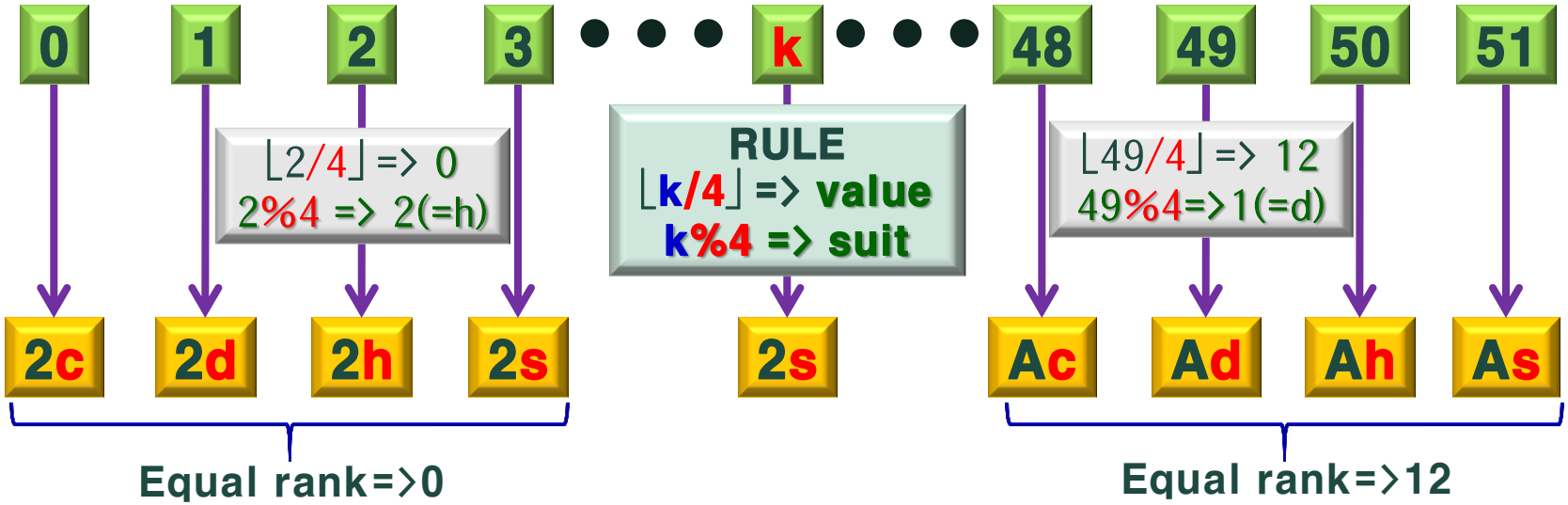
❖ Codes for Mapping Cards from 0 to 51

```
#define NCARDS  52      /* number of cards */
#define NSUITS  4       /* number of suits */

char values[] = "23456789TJQKA";
char suits[] = "cdhs";

int rank_card(char value, char suit)
{
        int i,j;        /* counters */

        for (i=0; i<(NCARDS/NSUITS); i++)
                if (values[i]==value)
                        for (j=0; j<NSUITS; j++)
                                if (suits[j]==suit)
                                        return( i*NSUITS + j );

        printf("Warning: bad input value=%d, suit=%d\n",value,suit);
}
```

2. Example : Going to War – Solving <5>

How to recover the card's information from the integer?

- We need to extract the values and suits of cards from the mapped integers!
  - Cf) In this example, only the values of cards were used for playing game.



Codes for Extracting Values & Suits from Integers

```
char value(int card)
{
        return( values[card/NSUITS] );
}
```

```
char suit(int card)
{
        return( suits[card % NSUITS] );
}
```

4. Example : Going to War – Solving <6>

Player 1

Player 2

```
4d Ks As 4h Jh 6h Jd Qs Qh 6s 6c 2c Kc 4s Ah 3h Qd 2h 7s 9s 3c 8h Kd 7h Th Td
8d 8c 9c 7c 5d 4c Js Qc 5s Ts Jc Ad 7d Kh Tc 3s 8s 2d 2s 5h 6d Ac 5c 9h 3d 9d
d 9d 8c 4s Kc 7c 4d Tc Kd 3s 5h 2h Ks 5c 2s Qh 8d 7d 3d Ah Js Jd 4c Jh 6c Qc
d Qs 9s Ac 8h Td Jc 7s 2d 6s As 4h Ts 6h 2c Kh Th 7h 5s 9c 5d Ad 3h 8s 3c
```

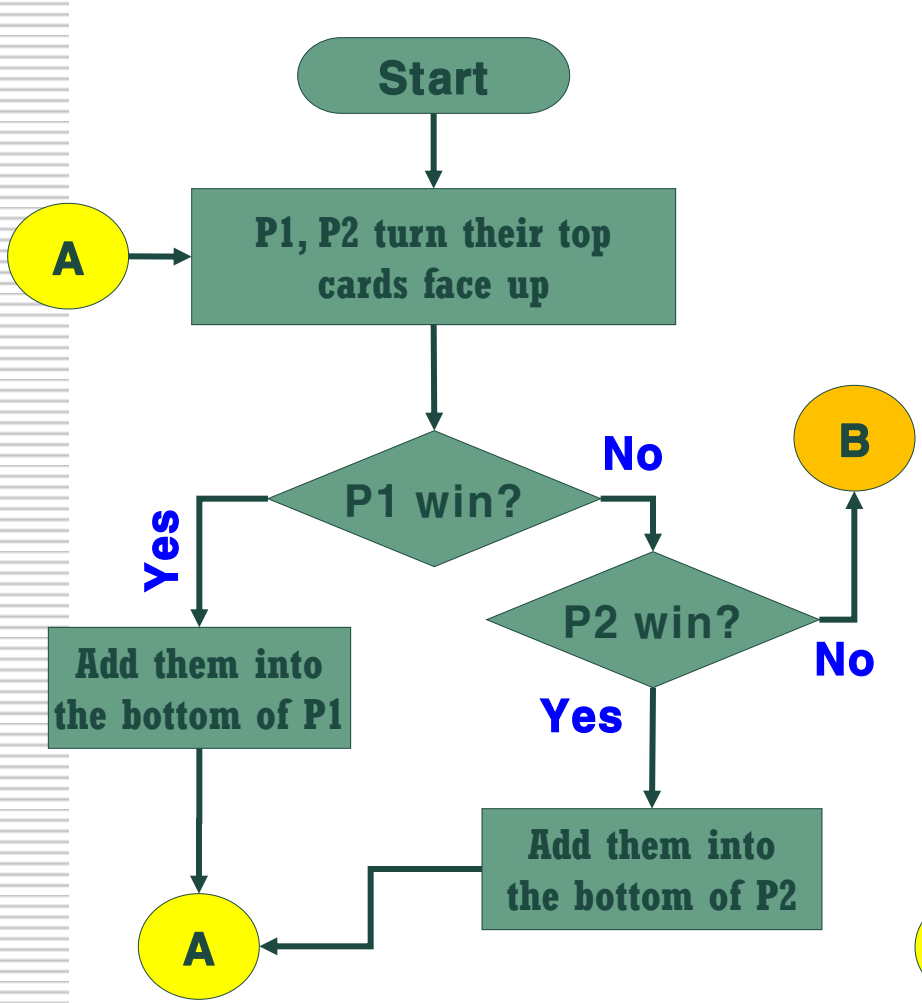
```
        queue decks[2];                 /* player's decks */
        char value,suit,c;              /* input characters */
        int i;                          /* deck counter */

        while (TRUE) {
            for (i=0; i<=1; i++) {
                init_queue(&decks[i]);

                while ((c = getchar()) != '\n') {
                    if (c == EOF) return;
                    if (c != ' ') {
                        value = c;
                        suit = getchar();
                        enqueue(&decks[i],rank_card(value,suit));
                    }
                }
            }

            war(&decks[0],&decks[1]);
        }
}
```

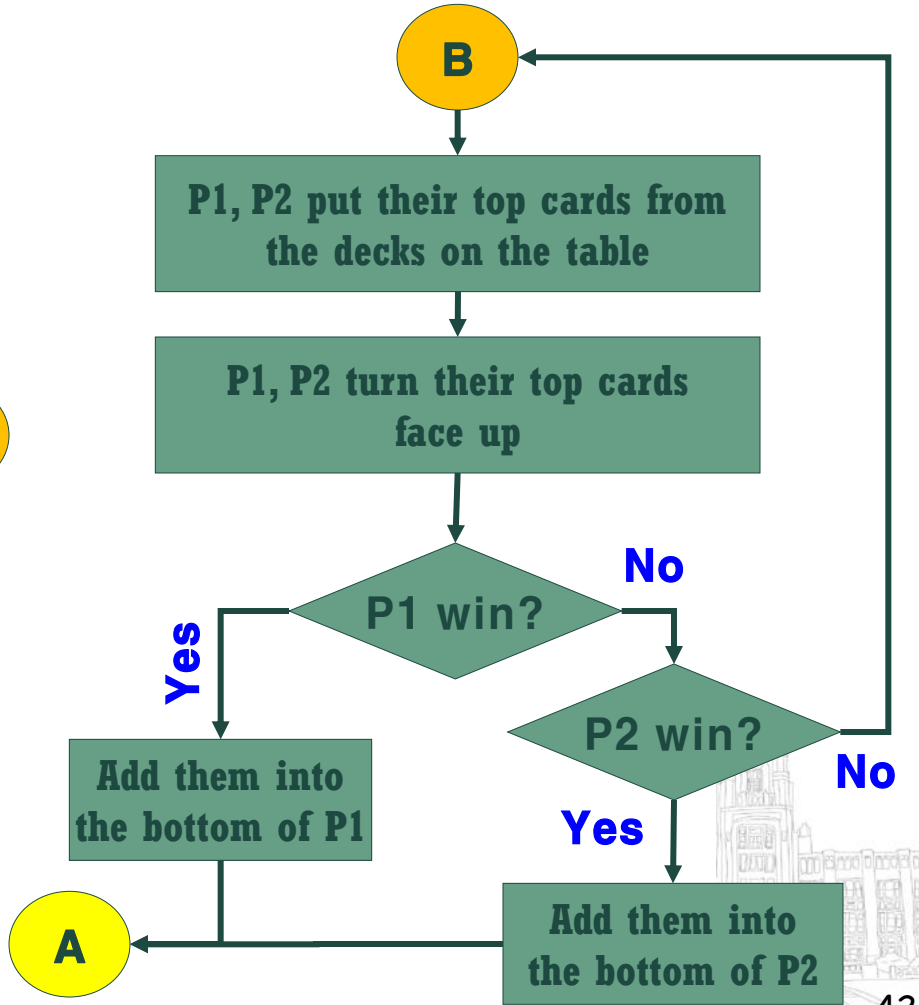
In this example, getchar() is used!

2. Example : Going to War – Solving <7>

❖ The Rule for Winning the War in Card Game?

- After comparing their top cards face up, consider WIN/LOSE/DRAW case!





4. Example : Going to War – Solving <8>

❖ **Codes for Winning the Wa**

```
war(queue *a, queue *b)
{
        int steps=0;                    /* step counter */
        int x,y;                        /* top cards */
        queue c;                        /* cards involved in the war */
        bool inwar;                     /* are we involved in a war? */

        inwar = FALSE;
        init_queue(&c);

        while ((!empty(a)) && (!empty(b) && (steps < MAXSTEPS))) {
                steps = steps + 1;
                x = dequeue(a);
                y = dequeue(b);
                enqueue(&c,x);
                enqueue(&c,y);
                if (inwar) {
                        inwar = FALSE;
                } else {
                        if (value(x) > value(y))
                                clear_queue(&c,a);
                        else if  (value(x) < value(y))
                                clear_queue(&c,b);
                        else if (value(y) == value(x))
                                inwar = TRUE;
                }
        }

        if (!empty(a) && empty(b))
                printf("a wins in %d steps \n",steps);
        else if (empty(a) && !empty(b))
                printf("b wins in %d steps \n",steps);
        else if (!empty(a) && !empty(b))
                printf("game tied after %d steps, |a|=%d |b|=%d \n",
                        steps,a->count,b->count);
        else
                printf("a and b tie in %d steps \n",steps);
}
```

```
clear_queue(queue *a, queue *b)
{
        while (!empty(a)) enqueue(b,dequeue(a));
}
```

Problems in Chapter 2

- Jolly jumpers
- Poker hands
- Hartals
- Crypt kicker
- Stack'em up
- Erdos numbers
- Contest scoreboard
- Yahtzee

Binary search