Problem Solving Techniques 문제해결

Jinkyu Lee

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

Contents

- **C-programming: overview**
- Chapter 1 – Getting started
- Chapter 2 – Date structure
  - Program design example

Some slides are adapted from Prof. Chang Wook Ahn's slides.

Q & A

- What is the return type of conditional operation?
  - For example, if(a>1)
  - printf(“%d”,1<2);
  - printf(“%c”, (1<2)+64);
  - printf(“%hd”,1<2);
- The values other than 0 and 1 are allowed?
  - if (2)
  - if (-1)
  - if (1.1)
  - if (0.0)
  - if (0.1)

```
if () {
    printf(“A”);
} else {
    printf(“B”);
}
```

1. A
2. B
3. Error

Array

- Declaration

data_type variable_name[number][number];

Array dimensions
Declaration & Initialization of Arrays
int a[4] = {2, 4, 3, 0};
int b[2][3] = { {1, 6, 4}, {5, 3, 2} };
int c[2][2][3] = { { {1,2,0}, {3,5,4} }, { {9,8,7}, {14,15,16} } };

b[0][1]

c[1][1][1]

c[1][0][0]

Array

- Array access using pointer
  - a[i]: the ith column of a
  - a is equivalent to &a[0]

Equivalent to a[i]
*(a + i)
*(&a[0]+i)

For a[3][5], equivalent to a[i][j]
*(a[i] + j)
(*(a + i)) [j]
*((*(a + i)) + j)
*(&a[0][0] + 5 * i + j)

Array

- Passing arrays to functions
  - When an array is passed to a function, its address is passed by "call by value."
  - The values of an array is passed by "call by reference."

```
[Ex]
int sum( int a[], int n)
{
  int i, s = 0;

  for ( i = 0; i < n; ++i)
     s += a[ i ];
  return s;
}
```

int a[ ] is equivalent to int *a.

Various ways that sum() might be called	
Invocation	What gets computes and returned
sum(v, 100)	v[0] + v[1] + … + v[99]
sum(v, 88)	v[0] + v[1] + … + v[87]
sum(&v[7], k – 7)	v[7] + v[8] + … + v[k -1]
sum(v + 7, 2* k)	v[7] + v[8] + … + v[2 * k + 6]

Array

- Dynamic memory allocation

size of each object

```
void malloc ( object_size );
```

number of objects — size of each object

```
void calloc ( n, object_size );
```

```
void free(void *ptr);
```

De-allocate a memory block that ptr points

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int *a, i, n, sum = 0;
    scanf("%d" , &n );
    a = calloc(n, sizeof(int) );          /* get space for n ints */
    for ( i = 0; i < n; ++i )     scanf("%d", &a[ i ] );
    free(a);                              /*free the space */
    return 0;
}
```

calloc(), malloc(), free() belong to stdlib.h

a = malloc(n * sizeof(int));

De-allocate a memory block allocated by calloc()

Recursion

- Recursive problem solving: computing factorial

```
[Ex] /* Recursive version  */
int fact( int n)
{
    if (n <= 1)
        return 1;
    else
        return n * fact(n-1);
}
```

```
[Ex]  /* Iterative version */
int fact( int n )
{
   int result = 1;

   for ( ; n > 1; --n )
         result *= n;
   return result;
}
```

What i = fact(3) returns	
fact (3)	3 * fact (2) 3 * (2 * fact (1)) 3 * (2 * (1)) = 6

Structure

- The difference between array and structure
  - Array
    - All elements in an array should be the same type.
    - We can access individual elements of an array using their index.
  - Structure
    - A structure can consists of elements with different types.
    - Each element has its own name.
    - We can access individual elements of a structure using their name.
- Declaration of structure: collection of members

```
[Ex] struct part{ /* 3-element structure */
        int  number;
        char  name [20] ;
        int  on_hand;
    }  part1;
```

⇔

```
struct part {
......
};
struct part  part1;
```


Jinkyu Lee
Dept. of Computer Science and Engineering

Structure

- Accessing members
  - Struct member operator: “.”

```
[Ex] struct part {
            int number;
            char name[20];
            int on_hand;
      } part1;

      part1.number = 258;          /* assignment */
      scanf (“%d”, &part1.on_hand); /* reading using scanf() */
      scanf(“%s”, part1.name);
      part1.on_hand++;             /* increment */
```

Structure

- Accessing members
  - Struct pointer operator: “->”

```
[Ex] typedef struct complex {
          double re;
          double im;
     } complex;

    complex c1, c2, *a=&c1, *b=&c2;
    /* a refers structure c1, b refers c2*/

    a->re = b->re + 2 ;   /* c1.re = c2.re + 2*/
    b->im = a->im - 3;    /* c2.im = c1.im - 3*/

    printf ("value ; %f\n ", a->im);
    scanf("%f", &b->im);
```

Contents

- C-programming: overview
- **Chapter 1 – Getting started**
- Chapter 2 – Date structure
  - Program design example

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

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

  - const int num_of_words = 10;

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;
      }
   }     Move(b,-1,row,col)
   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)
```

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

Contents

- C-programming: overview
- Chapter 1 – Getting started
- **Chapter 2 – Date structure**
  - Program design example

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

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

Contents

- C-programming: overview
- Chapter 1 – Getting started
- Chapter 2 – Date structure
  - **Program design example**

From now on, slides are adapted from Prof. Chang Wook Ahn's slides.

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

❖ Game Description

- Two players have 26 cards, respectively.
- Players keep them in a packet face down.
- The objective 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

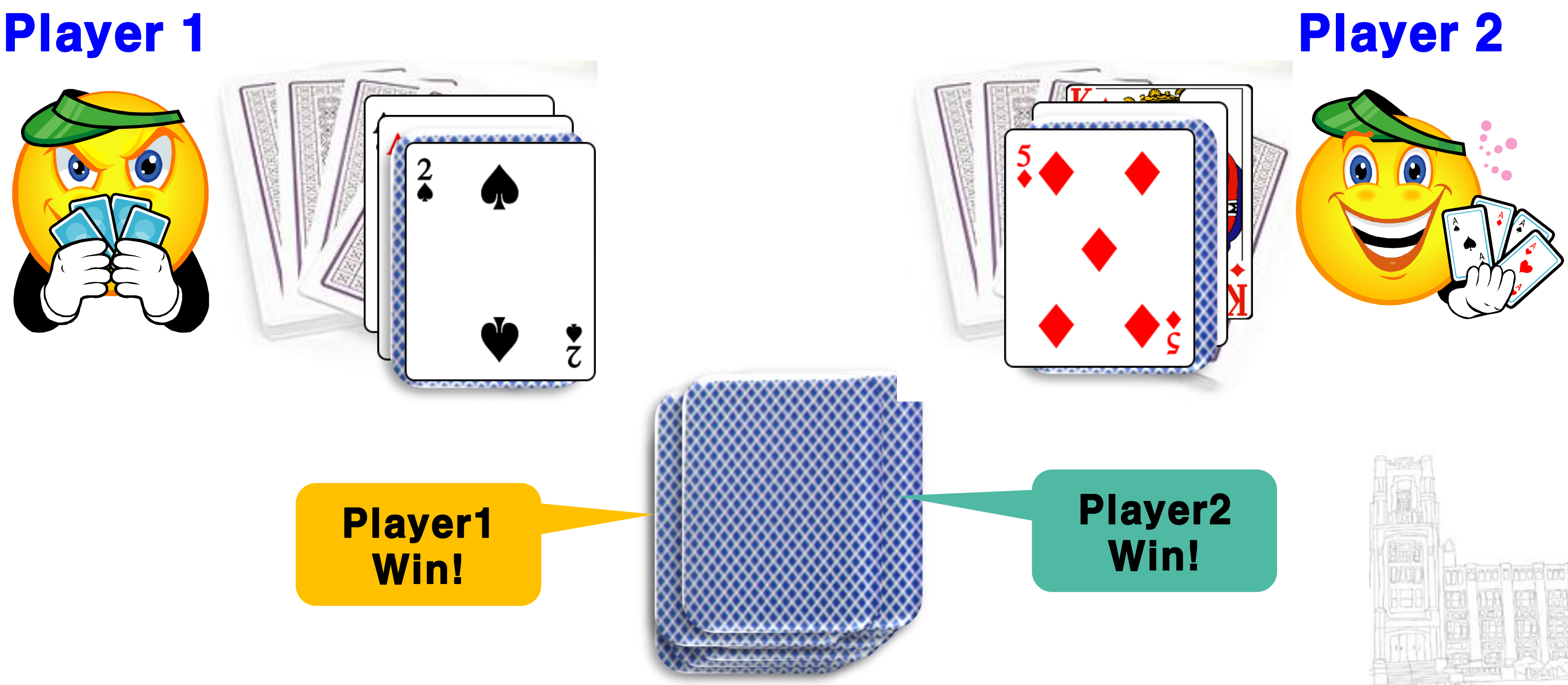
⑤ When the face up cards are equal rank in Step ②, there is a **WAR**!!

- 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. Background – Queue <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

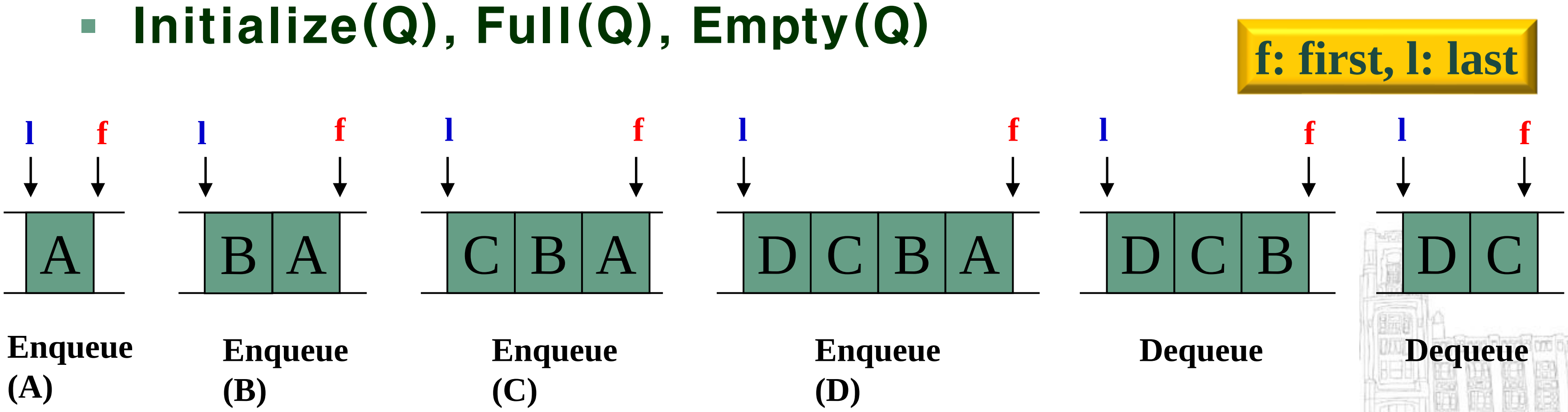
2. Background – Queue <2>

❖ 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

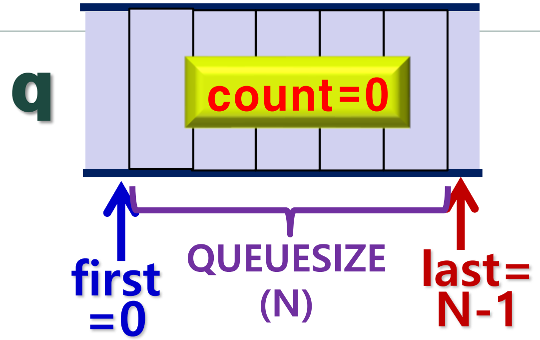
- Enqueue(x, Q) – Insert x item into the end of Q
- Dequeue(Q) – Return/delete the first item of Q
- Initialize(Q), Full(Q), Empty(Q)



2. Background – Queue <3>

❖ 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

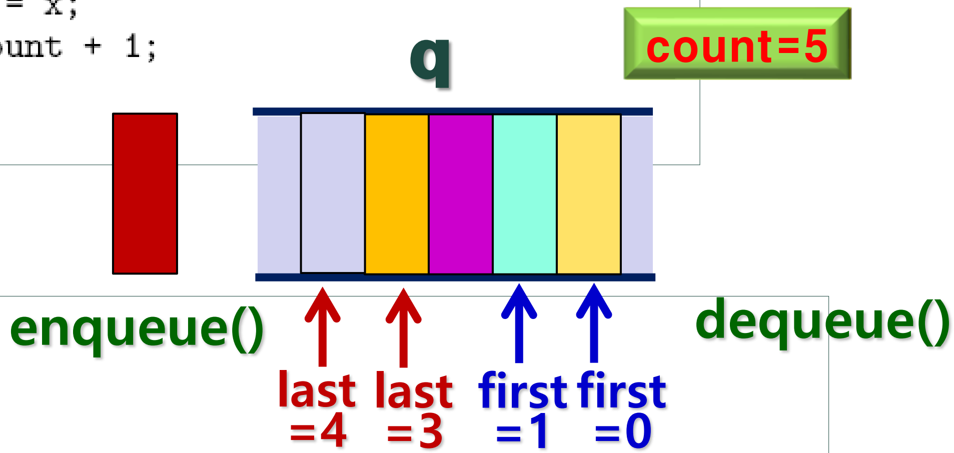
```
init_queue(queue *q)
{
        q->first = 0;
        q->last = QUEUESIZE-1;
        q->count = 0;
}
```

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

2. Background – Queue <4>

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;
        }
}
```



Dequeue

```
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);
}
```

3. 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

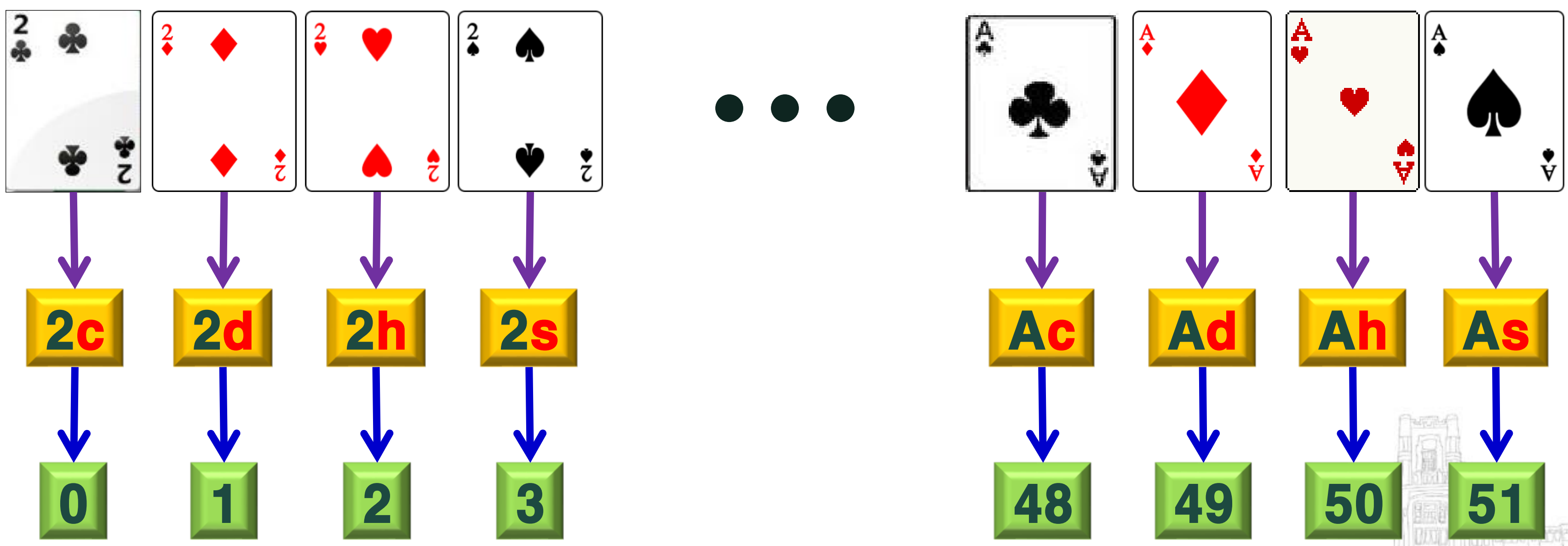
Specification: Necessary elements for managing data

Operation: Operators performed on the elements for accessing data

3. 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?

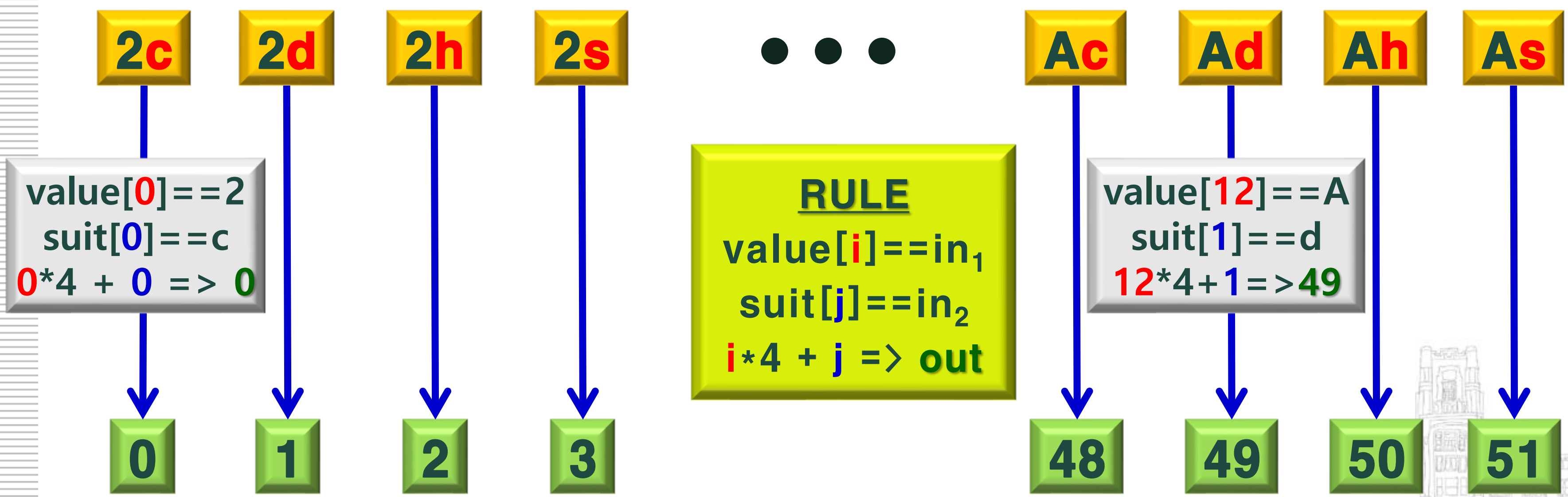


3. Example : Going to War – Solving <3>

❖ 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"
```



3. Example : Going to War - Solving <4>

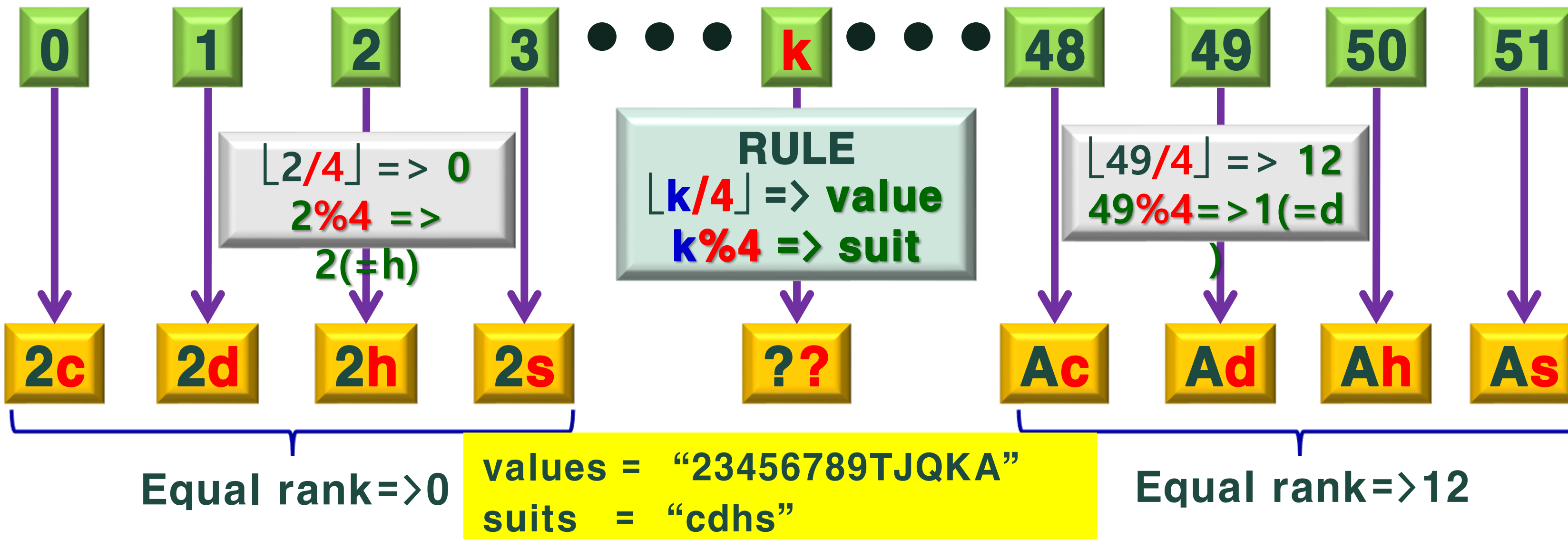
❖ Codes for Mapping Cards from 0 to 51

```
#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);
}
```

3. 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] );
}
```

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

Player 1

- 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

Player 2

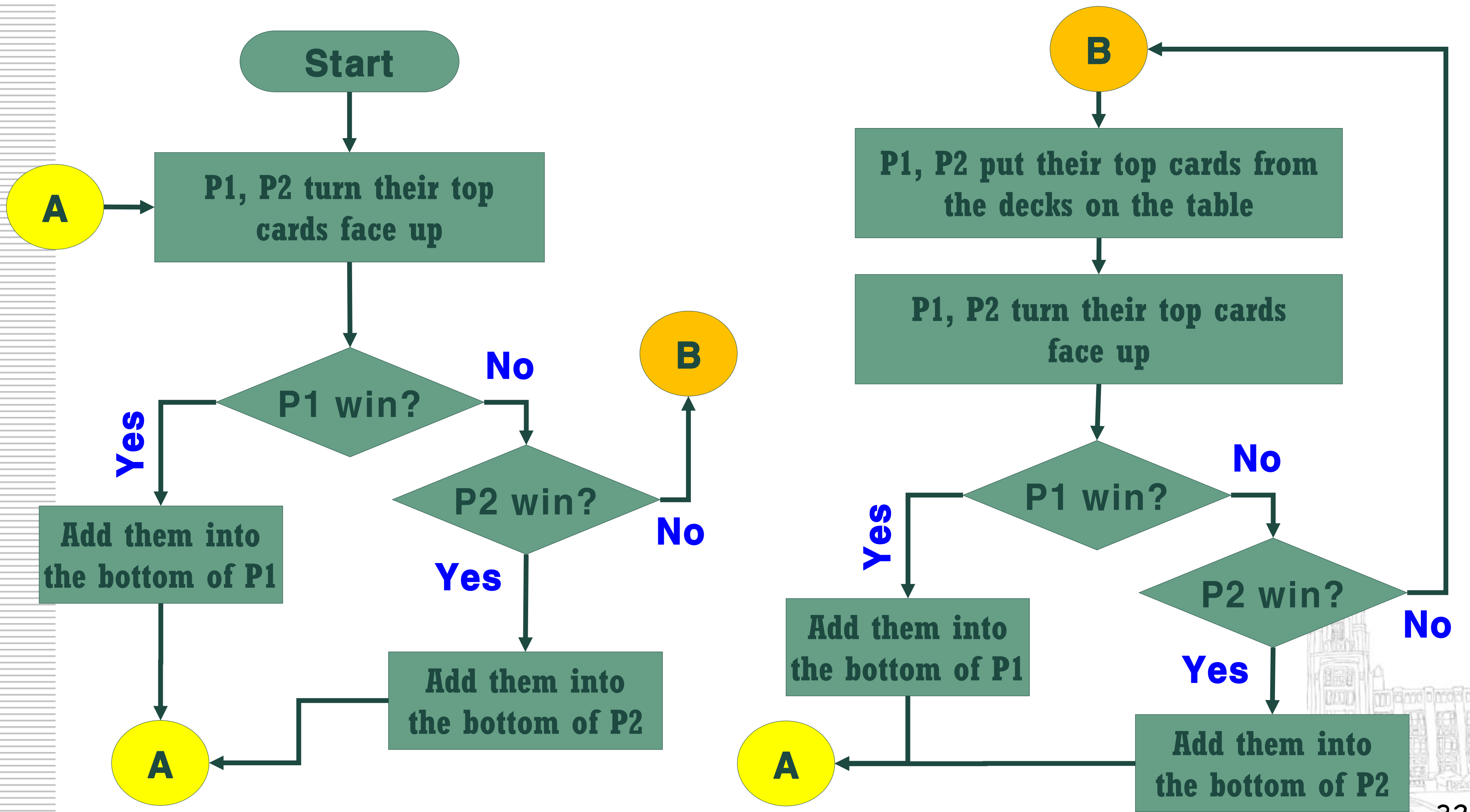
```
main()
{
        queue decks[2];                    /* player's decks */
        char value,suit,c;                 /* input characters */
        int i;                             /* deck counter */

        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!

3. Example : Going to War - Solving <7>

❖ The Rule for Winning the War in Card Game?

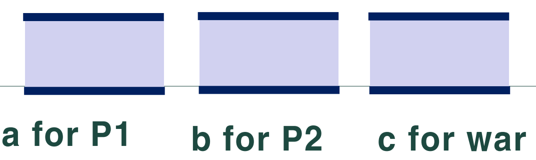
- **After comparing their top cards face up, consider WIN/LOSE/DRAW case!**



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

- **Codes for Winning the War**

```
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));
}
```