Be the compiler

```c
#include <stdio.h>
int main()
{
    int card = 1;
    if (card > 1)
        card = card - 1;
        if (card < 7)
            puts("Small card");
    else {
        puts("Ace!");
    }
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int card = 1;
    if (card > 1) {
        card = card - 1;
        if (card < 7)
            puts("Small card");
    else
        puts("Ace!");
    }
    return 0;
}
```

```
Small card
```

But may be missing { }

Misplaced }

Be the compiler

```c
#include <stdio.h>
int main()
{
    int card = 1;
    if (card > 1) {
        card = card - 1;
        if (card < 7)
            puts("Small card");
    } else
        puts("Ace!");
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int card = 1;
    if (card > 1) {
        card = card - 1;
        if (card < 7)
            puts("Small card");
    else
        puts("Ace!");
    return 0;
}
```

```
Ace!
```

```
F:\CS111\wrong.c: In function 'main':
F:\CS111\wrong.c:12:1: error: expected
declaration or statement at end of inp
 }
 ^
```

Missing a }!

Branches

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char card_name[3];
    puts("Enter the card_name: ");
    scanf("%2s", card_name);
    int val = 0;
    if (card_name[0] == 'A') {
        val = 1;
    } else if (card_name[0] == 'J') {
        val = 11;
    } else if (card_name[0] == 'Q') {
        val = 12;
    } else if (card_name[0] == 'K') {
        val = 13;
    } else {
        val = atoi(card_name);
    }
    printf("The card value is: %i\n", val);
    return 0;
}
```

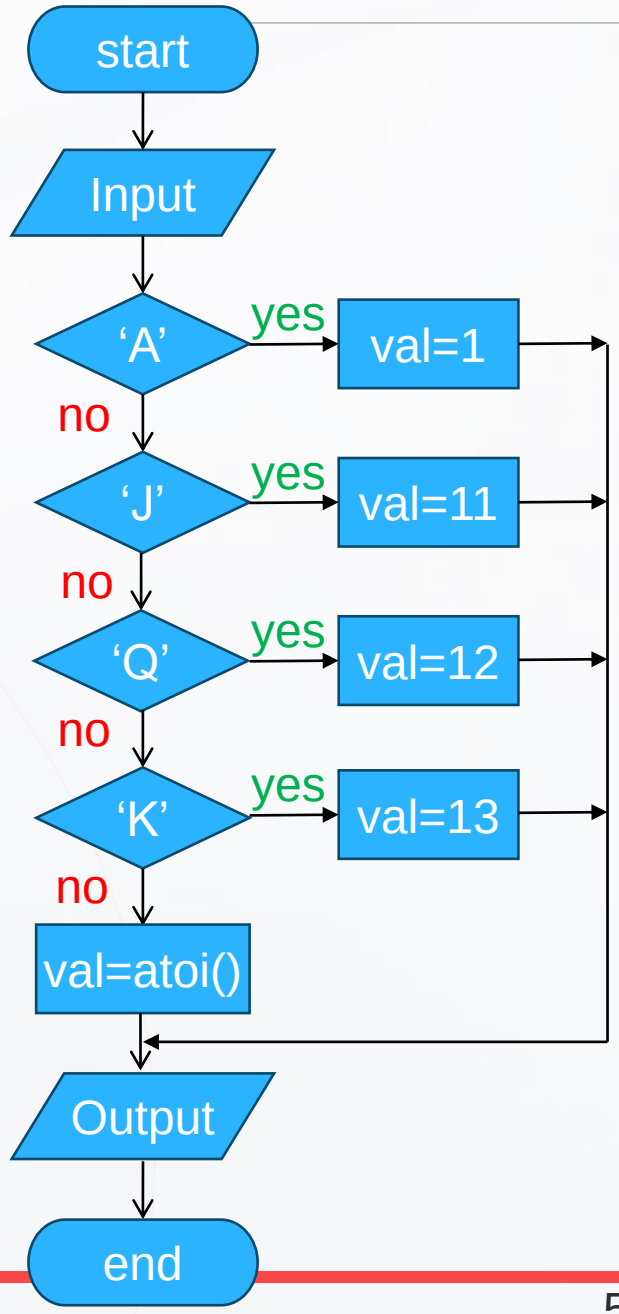
All the “if”s are checking the same variable val. Is there any alternative/more efficient way?

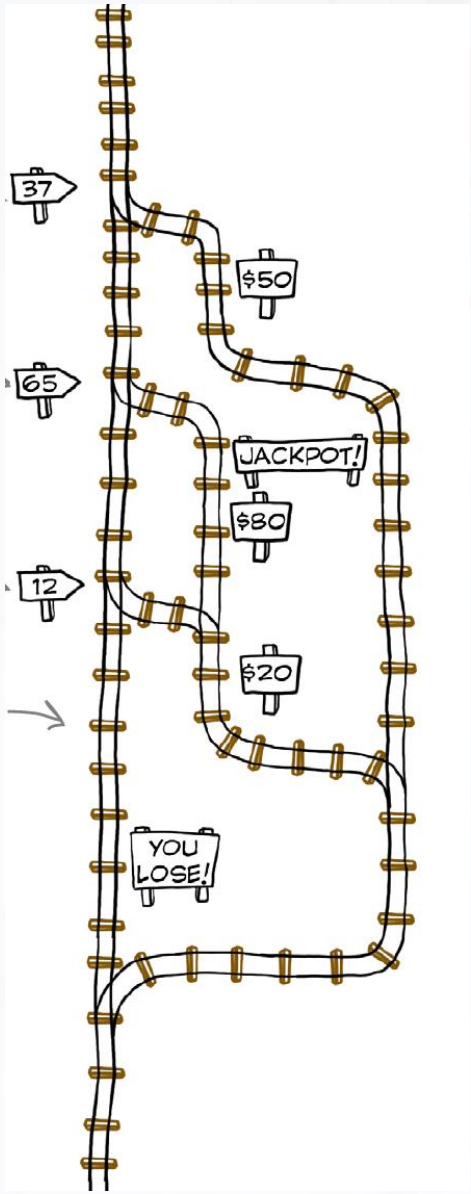
The switch statement

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char card_name[3];
    puts("Enter the card_name: ");
    scanf("%2s", card_name);
    int val = 0;
    if (card_name[0] == 'A') {
        val = 1;
    } else if (card_name[0] == 'J') {
        val = 11;
    } else if (card_name[0] == 'Q') {
        val = 12;
    } else if (card_name[0] == 'K') {
        val = 13;
    } else {
        val = atoi(card_name);
    }
    printf("The card value is: %i\n", val);
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char card_name[3];
    puts("Enter the card_name: ");
    scanf("%2s", card_name);
    int val = 0;
    switch (card_name[0]) {
    case 'A':
        val = 1;
        break;
    case 'J':
        val = 11;
        break;
    case 'Q':
        val = 12;
        break;
    case 'K':
        val = 13;
        break;
    default:
        val = atoi(card_name);
    }
    printf("The card value is: %i\n", val);
    return 0;
}
```

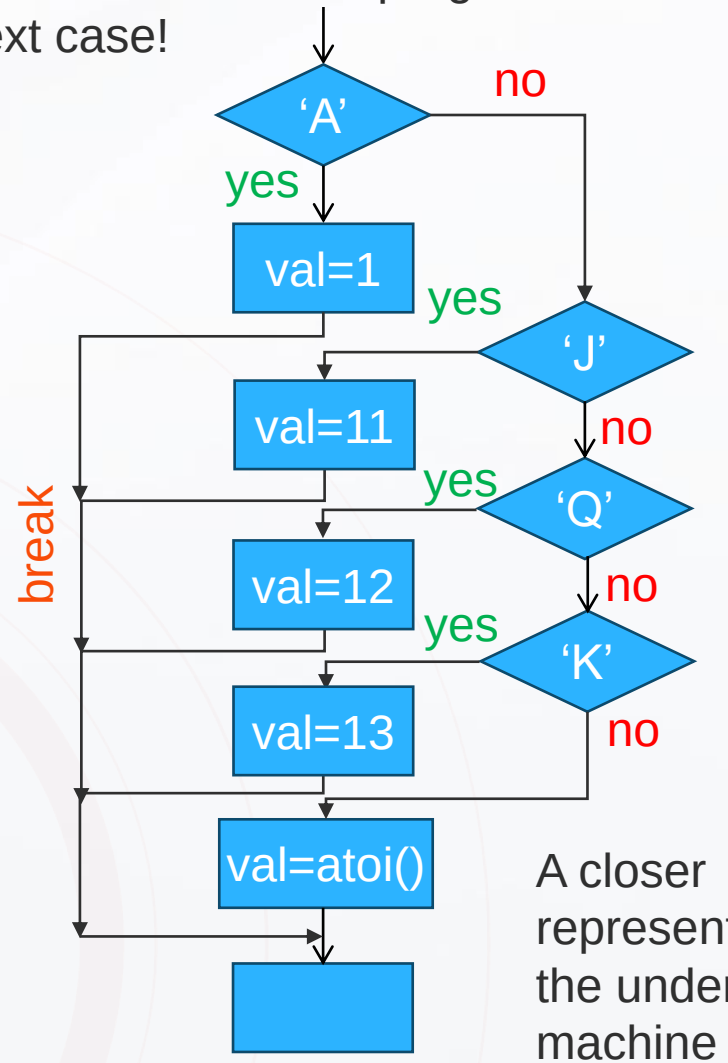
Flowchart


start
Input
'A'
yes
val=1
no
'J'
yes
val=11
no
'Q'
yes
val=12
no
'K'
yes
val=13
no
val=atoi()
Output
end


37
$50
65
JACKPOT!
$80
12
$20
YOU LOSE!

Breaks after each case!

Without the breaks, the program will continue into the next case!



A closer representation of the underlying machine code

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char card_name[3];
    puts("Enter the card_name: ");
    scanf("%2s", card_name);
    int val = 0;
    switch (card_name[0]) {
    case 'A':
        val = 1;
        break;
    case 'J':
        val = 11;
        break;
    case 'Q':
        val = 12;
        break;
    case 'K':
        val = 13;
        break;
    default:
        val = atoi(card_name);
    }
    printf("The card value is: %i\n", val);
    return 0;
}
```

Bugs cost much more than efficiency

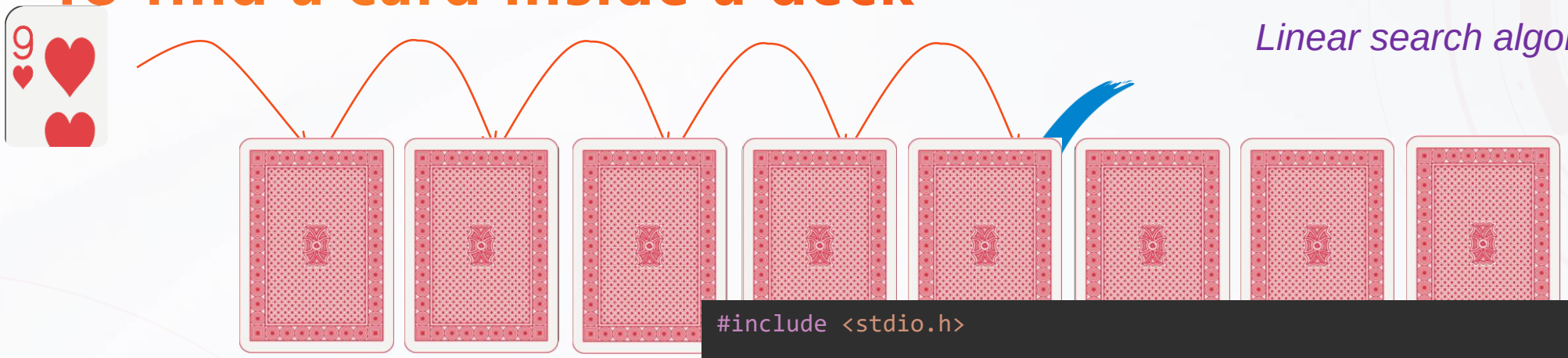
If vs. Switch

- *switch* replaces a sequence of *if*s (clearer and more readable)
- *switch* checks a single value only (no strings)
- starts to run from the first matching *case*
- continues to run after the matching case block, until a *break*.

Lecture 3 Loops

To find a card inside a deck

Linear search algorithm



```c
#include <stdio.h>

int main()
{
    char card[10] = {'5','6','3','A','8','K','Q','9','7','\0'};
    char target = '9';
    if (card[0]==target)
    {
        puts("Target found at location 1!");
        return 0;
    }
    if (card[1]==target)
    {
        puts("Target found at location 2!");
        return 0;
    }
    if (card[2]==target)
    {
        puts("Target found at location 2!");
        return 0;
    }
    ...
}
```

Machines are good at repeating!

```c
#include <stdio.h>

int main()
{
    char card[10] = {'5','6','3','A','8','K','Q','9','7','\0'};
    char target = '9';
    int index = 0;  /* Index for the card number */
    while (index<9) /* Loop over all 9 cards */
    {
        if (card[index]==target)
        {
            printf("Target found at location %d", index+1);
            break;  /* Card found, break the loop */
        }
        index++;
    }
    if (index>=9)
    {   /* Not found after looping over all cards */
        puts("Target not found!");
        return(1);
    }
    return(0);
}
```

The while loop

This block is to be repeated

Until this condition is not true

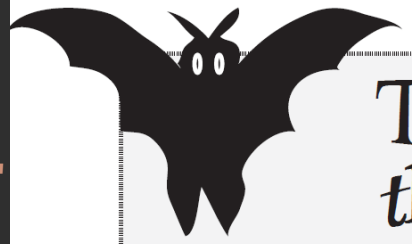
Or a break statement

A continue statement continues with the next loop

break terminates a loop or switch, but not if!

```c
#include <stdio.h>

int main()
{
    char card[10] = {'5','6','3','A','8','K','Q','9','7','\0'
    char target = '9';
    int index = 0;  /* Index for the card number */
    while (index<9) /* Loop over all 9 cards */
    {
        if (card[index]==target)
        {
            printf("Target found at location %d", index+1);
            break;  /* Card found, break the loop */
        }
        index++;
    }
    if (index>=9)
    {   /* Not found after looping over all cards */
        puts("Target not found!");
        return(1);
    }
    return(0);
}
```



Tales from the Crypt

breaks don't break if statements.

On January 15, 1990, AT&T's long-distance telephone system crashed, and 60,000 people lost their phone service. The cause? A developer working on the C code used in the exchanges tried to use a `break` *to break out of an* `if` *statement. But* `break`*s don't break out of* `if`*s. Instead, the program skipped an entire section of code and introduced a bug that interrupted 70 million phone calls over nine hours.*

break vs. continue

```
while(feeling_hungry)
{
    eat_cake();
    if (feeling_queasy)
    {
        break;
        /* Break out of the while loop */
    }
    drink_coffee();
}
```

```
while(feeling_hungry)
{
    if (not_lunch_yet)
    {
        /* Go back to the loop condition */
        continue;
    }
    eat_cake();
}
```

"break" skips out of the loop immediately

"continue" takes you back to the start of the loop

(skipping the remaining part)

The for loop

```
#include <stdio.h>

int main()
{
    char card[10] = {'5','6','3','A','8','K','Q','9','7','\0'};
    char target = '9';
    int index;  /* Index for the card number */
    for (index = 0; index<9; index++) /* Loop over all 9 cards */
    {
        if (card[index]==target)
        {
            printf("Target found at location %d", index+1);
            break;  /* Card found, break the loop */
        }
    }
    if (index>=9)
    {   /* Not found after looping over all cards */
        puts("Target not found!");
        return(1);
    }
    return(0);
}
```

Condition for continuation

Run after each loop

Initialization /prepare for the loop

The loop block

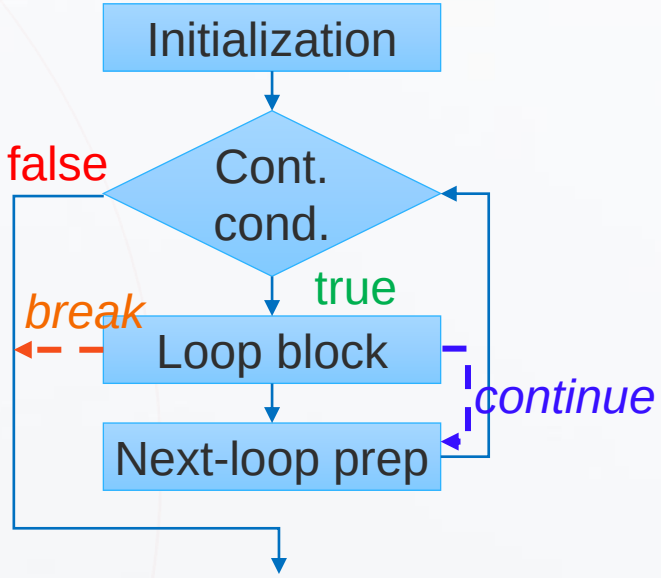
Four elements in the for loop

```
for (
{
    …
}
```

Initialization ; Continuation condition ; Next-loop preparation)

The loop block

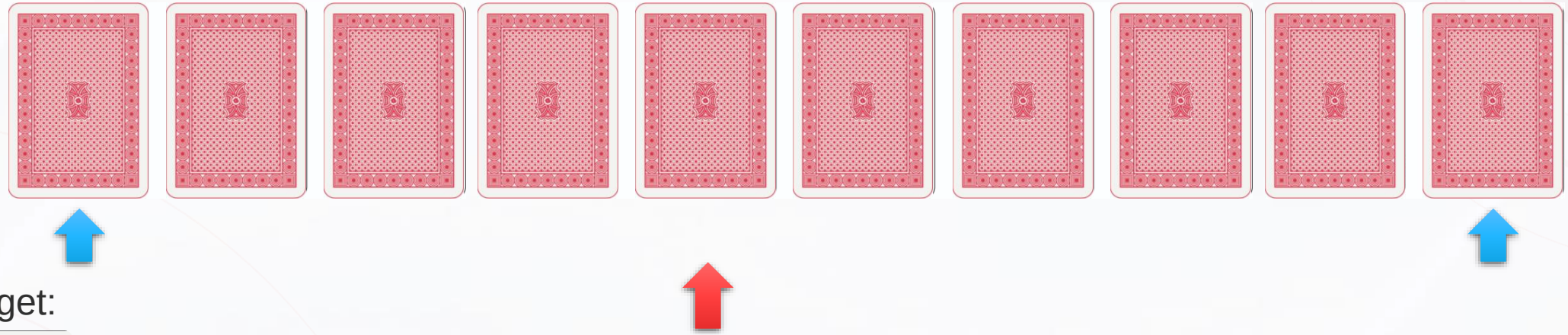
May contain various types of statement, or left blank



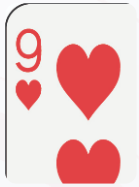
You may also use break or continue in a for loop

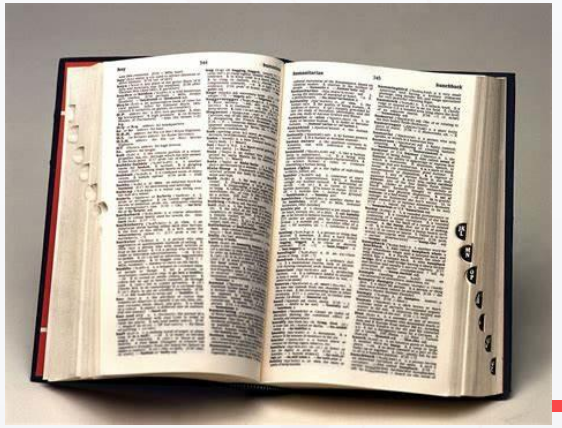
Can we do better?

What if the cards are pre-sorted?



Target:





Binary search

Binary search

```c
#include <stdio.h>

int main()
{
    int card[10] = {1,3,5,6,7,8,9,11,12,13}; /* Pre-sorted ascending */
    int target = 8;     /* Target card to search for */
    int low = 0, high = 9;  /* Lower and upper bounds of the interval */
    int index;          /* Index of the card to compare */
    while (low <= high) /* Loop while the interval is not empty */
    {
        index = (low + high)/2; /* Guess the middle of the interval */
        if (card[index]==target)
        {
            printf("Target found at location %d", index+1);
            return(0);  /* Card found. Terminate the search */
        }
        else if (card[index]>target)    /* Guess index too big */
            high = index - 1;
        else    /* Guess index too small */
            low = index + 1;
    }
    /* Not found after looping over all cards */
    puts("Target not found!");
    return(1);
}
```