So you think you can count?

with apologies to
C. E. Linderholm, author of
Mathematics Made Difficult

Overview

- Counting is harder than it looks at first, especially if you want the right answer
- There are ways to count more accurately and easily
- These techniques suggest useful ways of writing programs that deal with arrays and other sequences

Counting can be hard

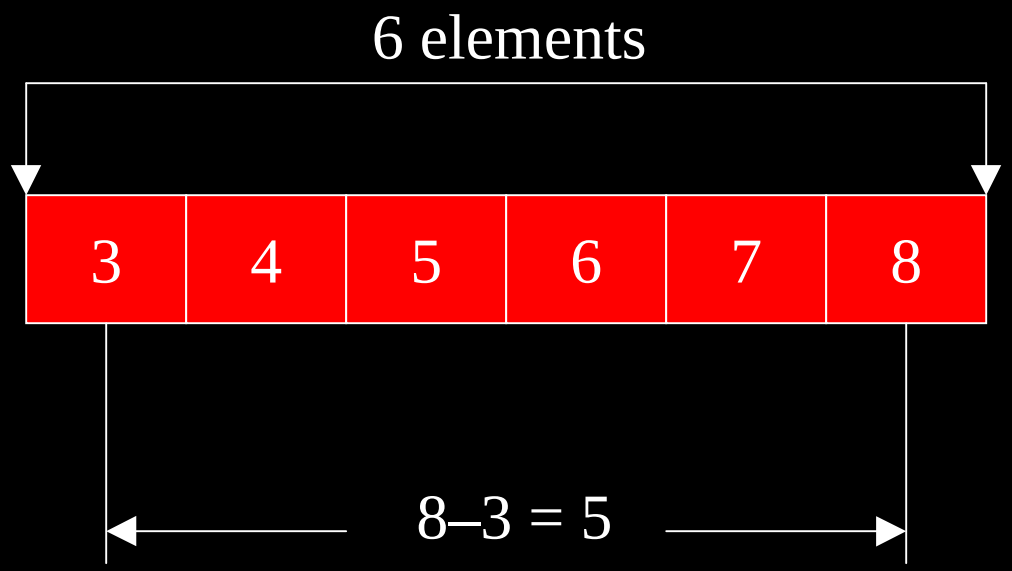
- You leave for vacation on the 6th and return on the 17th. How many days are you gone? How many nights?
- When is three weeks from January 19?
- How many fence posts 10 feet apart are needed to support 120 feet of fence?

Fencepost errors

- Errors in counting are often called fencepost errors, especially when they involve an uncertainty of ±1.
- Fencepost errors are among the most common—and hardest to find—of programming errors.

Understanding fenceposts

- (last–first) = #elements–1
- (last–first) = #(interior fenceposts)



Another viewpoint

- If first == last, then
  - there is one element
  - no fenceposts are crossed between the first and last elements
  - so there are no interior fenceposts

Yet another viewpoint

- The number of days you stay in a hotel is ambiguous; the number of nights (usually) is not
- Departure date – arrival date = number of nights, not number of days
- Nights behave like interior fenceposts

A useful convention

- Represent ranges by the index of the first element in the range and the first element *beyond* it
- Each index is the beginning of something
- A null range corresponds to two identical indices

If you use this convention...

- length = end – start
- end ≥ start, always
- A common way of expressing such ranges is as [*m*, *n*)

Another useful convention

- Check your computations on a range with no elements at all (or one element)
  - How many nights am I in a hotel if I arrive on day `m` and leave on day `n`?
  - Try it for `m==n`; the answer is obviously zero
  - `n-m` is zero, so that's our answer

Measuring your vacation

- If you leave on the 6th and return on the 17th, then
  - The first day you're not gone is the 18th
  - 18–6 = 12, so you're gone 12 days
  - The first night you're not gone is the 17th
  - 17–6 = 11, so you're gone 11 nights

Three weeks from 1/19

- One week from today is the first day that's not part of the 7-day period that started today. Therefore
  - One week from 1/19 is 1/26
  - Three weeks from 1/19 is 1/40 *[sic]*
- We must now figure out which day in February corresponds to January 40

Combining intervals, part 1

- How many days of our interval lie in January?
  - The first day is January 19
  - The first day that is outside the part in January is February 1, which we can think of as January 32
  - 32–19 = 13, so there are 13 days in January

Combining intervals, part 2

- We have a 21-day interval, 13 days of which are in January
- Therefore, there are 21–13, or 8 days in February
- The first of those days is 2/1, therefore
- The first day beyond the interval is 2/9
- Therefore, three weeks from 1/19 is 2/9

Fenceposts, for real

- If the fenceposts are 10 feet apart, they will divide 120 feet of fence into 12 segments, each 10 feet long
- If there are 12 segments, there will be 13 fenceposts, so that there will be one at each end
- We might number them [0, 12]

Fenceposts and buffers

- Suppose we have an N-element array that we wish to fill progressively and empty whenever it fills up
- Then we might use n to represent
  - the number of elements used, or
  - (equivalently) the index of the first element beyond the occupied area (if the initial element has index 0)

Filling a buffer

```
Elem buffer[N];
int n = 0;
…
   if (n == N) {
        empty_the_buffer();
        n = 0;
   }
   buffer[n++] = value;
```

Once more, with pointers

```
Elem buffer[N];
Elem *p = buffer;
…
    if (p == buffer + N) {
        empty_the_buffer();
        p = buffer;
    }
    *p++ = value;
```

A concrete example

- Suppose we have a program that generates integers
- We want to print those integers in pages that are divided into columns
- Assume that `rows` is the number of rows and `columns` is the number of columns on each page

How might the interface look?

- For the first try, we'll keep it simple:

```
void start();
void print(int);
void finish();
```

A note on interfaces

- Our “finish” operation presumably prints the last (partial) page. Why isn’t there an operation to print the other pages?
- The other pages can be printed “automatically” when we try to put a number into a buffer that is full already

Buffer definition

```
static const int rows = 50;
static const int columns = 5;
static int buffer[rows][columns];
static int row, col;
```

Buffering conventions

- We will fill the buffer a column at a time and print it a row at a time
- Whenever we are about to put a value into the buffer, we will put it at position `(row, col)`
- After putting something into the buffer, we will increment `(row, col)` and flush the buffer if needed

Initialization

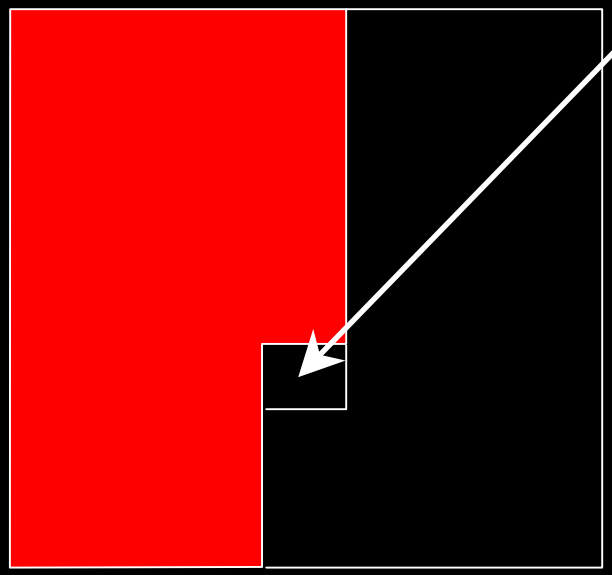
```
void start()
{
     row = 0;
     col = 0;
}
```

“Print” a number

```
void print(int n)
{
    buffer[row][col] = n;
    if (++row == rows) {
        row = 0;
        if (++col == columns) {
            flush_buffer();
            col = 0;
        }
    }
}
```

Empty the buffer

- Assume the buffer is full to (`row`, `col`)



Implications

- We would like the same code to print full pages as will print the partial page at the end of the output.
- The width of a row is either `col+1` (for rows less than `row`) or `col` (otherwise)
- If the first column is not full, we will print one or more empty rows (because `col == 0` and `row < rows`)

Emptying the buffer

```
static void flush_buffer()
{
    for (int r = 0; r < rows; ++r)
        print_row
            (r, r < row? col+1: col);
    if (row != 0 || col != 0)
        new_page();
}
static void new_page()
{
    printf("\f");
}
```

Printing a row

```
static void print_row(int r, int w)
{
    if (w != 0) {
        for (int j = 0; j < w; ++j)
            printf("%8d",buffer[r][j]);
        printf("\n");
    }
}
```

The last page

```
void finish()
{
    flush_buffer();
}
```

Policy decisions

- How shall we print an empty row?
- How shall we print an empty page?
- What is the format within a row?
- How large are the pages?
- All these decisions are well isolated

Summary

- There are (y–x) fenceposts in [x, y]
- There are (y–x) elements in [x, y)
- It is often easiest to represent a range by the first element in it and the first element beyond it
- It is useful to check for fencepost errors by using empty ranges (or ranges with only one element)