Objectives

- This topic should allow students to use arrays to handle problems whose size is unknown in advance.

- This topic is not covered in the textbook.
Problem

- Up till now we’ve used arrays to deal with problems of known size
  - 5 grades
  - Reverse 10 words
  - 12 months of rainfall
- Often we want to use arrays but we don’t know in advance how many students, employees, words,…. there will be.

Solution

- Declare an array big enough to handle the maximum possible problem size.
  - E.g. DECLARED_SIZE
- Keep track of how much of the array you fill up.
  - E.g. call it array_size, or num_grades
- Use the latter number in for loops that process the array.
Example

- Write a program to square a sequence of numbers, terminated by a 0.
  - Use a function `readtosentinel` to read in the sequence.
    - `readtosentinel` takes in the array, the declared size of the array and a parameter count.
    - `readtosentinel` updates the count variable so it must be a call-by-reference parameter to `readtosentinel`
  - Use count to control subsequent loops.
- See `squarearbseq.cpp`

readtosentinel function

```cpp
void readtosentinel (double a[], int capacity, int& size, double &sentinel)
{
    size = 0;
    double x;
    cin >> x;
    while (size < capacity && x != sentinel) {
        a[size] = x;
        size = size + 1;
        cin >> x;
    }
}
```
readtosentinel

- Called like
  
  \[ \text{readtosentinel (numbers, MAX\_SIZE, count, 0);} \]
  
  - The 0 is the sentinel value – our function will handle any sentinel of type double.
  
  - readtosentinel must watch for too many values being entered.
  
  - Note the pattern
    
    \[ \text{read x; while (x ok)} \]
    \[ \quad \{ \text{process x; ... read next x; ...} \} \]

  - We’ve seen this pattern many times before.

Summary

- To handle arrays of unknown size
  
  - Declare a large enough array for all cases
  
  - Use a function like readtosentinel to read in values and set the actual number of values read in.
    
    - You would need a variant to read strings or ints
    
    - Don’t read too many values in!
    
    - Use the actual size of the array in subsequent loops / array functions.
Exercise: indexing text

- Read text, terminated by “END”, into an array of words.
  - Assume no punctuation, just words.
- Then count the occurrences of specified terms in the text.
  - We’ll count each one in turn, since we have permanent access to all the text.
- Identify the most common of those terms.

Advanced indexing exercise

- Extend the previous idea
- Instead of having a specified list of terms to count, extract every unique word from the input and count all of them.
- Write a function to pass through the text array and add newly found words to the terms array.
  - This function will generate an array of terms whose size will be set by the function.
- Then the previous program will work with that terms array and it’s size.
Another exercise

- Write a function `between` to extract from an array of numbers, those which are between two values. The extracted values should be put into an array.

```c
void between(double n[], int asize,
             double b[], int& bsize);
```

- Embed your function in a main program that tests it.