- **Strings and streams**
  - **Stroustrup Chs. 36, 38 - highlights**
- **string class**
  - **Stroustrup 36**
  - **can get c_str() char * pointer, but note limitations**
    - *use it just long enough to copy it out to somewhere else,*
    - *once string has been changed, or ceases to exist, then the pointer is undefined*
  - **positions in the string are given with index, like array subscript**
    - *first position is 0, last position is string::length() - 1*
      - “past the end/not there” is a special value, named string::npos
      - string::npos also means "all of the characters -
        npos is larger than the largest possible character position
      - length() < npos by definition
    - *positions, string::length() (the same as string::size()) , returns a type for the size:*
      - string::size_type
        - which is probably the same type as std::size_t
      - be careful! string::size_type is an unsigned type! watch out for unsigned arithmetic!
      - best policy - use string::size_type for all variables in which you store string positions.
  - **functions like substr, find_first_not_of, are handy for parsing**
    - *see examples in Stroustrup*
  - **Note also complete iterator interface for string**
    - *can use it like a container, apply the algorithms to it*
    - *sometimes handy to use both: e.g. turn a string to uppercase with (transform ... ) then take out a substring of it*
  - **note line oriented input function:**
    - *getline(istream&, string_var);*
      - not a member because iostream library doesn’t depend on string
      - consumes the newline character in the stream at the end of a line,
        but doesn't store it in the string.
        - So end of string corresponds to end of line.
- **File streams**
  - **Stroustrup 38**
  - **see handouts on basic streams, file streams**
  - **use just like cin/cout**
    - *#include <fstream>*
    - *ifstream infile;*
    - *ofstream outfile;*
    - *if give file name/path in ctor, will open*
      - otherwise, use open function
      - ifstream infile(“data_in.txt”);

        - ifstream infile;
          ...
          infile.open(“data_in.txt”);
        - infile.close();
        - good etiquette: close file streams when you are done with them.
        - can reuse the same stream object by closing and re-opening with same or different file
    - *in C++98, iostreams didn't know about std::string, open has to have a C-string!!*
    - ***in C++11, open will take a std::string argument directly, so do it that way!***
        - string filename;
          cout << “enter file name: “;
          cin >> filename;
          ifstream infile(filename);
    - *test for open success before proceeding*
        - if(infile.is_open())
        - if(infile)
        - if(infile.good())
    - *use idiomatic form for reading the file:*

        -
          ```
          while (infile >> datum) {
                  /* use datum */
                  }
          // here because input didn’t work - why?
                  if(infile.eof())
                          // hit the end of file - do what it appropriate
                  else if(infile.fail())
                          // couldn’t read the datum type because it didn’t look right
                  else if (infile.bad())
                          // some horrible I/O error
                  else
                          can’t get here ...
          ```
    - *Simple case often applies:*
        - if you can assume that the data is always readable, (or you can’t do anything useful if it isn’t), then usually you just read until the stream fails, which would be at end of file.
          If so, then this is all you need:

        -
          ```
          while(infile >> datum) {
                  // use it
                  }
          // finished with input - go on
          ```
    - *NOTE: if you get end-of-file, stream object is in the fail state, and you have to clear it with .clear() to use it again.*
        - e.g. if you want to recycle the stream object and open it on the same or different file
- **stringstreams**
    - **Stroustrup 38.2**
    - **a stream based on a string**
        - *most programming languages allow you do I/O operations on a string in addition to an I/O device*
    - **use together with strings to provide ways to read and write strings using standard i/o operators**
        - *#include <sstream>*

- *ostringstream // write to it with output operators*
- *istringstream // read from it with input operators*
  - test for fail/eof like for regular stream - eof is trying to read past the end of the string
- **E.g. suppose you need to prepare a string containing values from numbers?**
  - *C alternative is sprintf into a char array - have to be sure the array is big enough no matter what!*
  - *e.g. idiomatic way to put a number into a string - heavyweight but won't crash*

    -
      ```
      string int_to_string(int i)
      {
              ostringstream ss;
              ss << i;
              return ss.str();
      }
      ```
  - *can get fancy:*

    -
      ```
      ostringstream ss;
      ss << "The value of i is " << i << " in the case named " << setw(8) << label_string
      << " with parameter " << setprecision(3) << setw(8) << theta
      << "\nNote that these results have an error of +/- " << error;
      ```
- **example - you get a string and need to extract some numbers and a string from it**

  -
    ```
    string str = get_input();
    istringstream iss(str); // construct a streamstring from the string
    int i;
    double x;
    string s;
    if(iss >> i >> x >> s)
            cout << "we got " << i << ' ' << x << ' '  << s;
    ```
- **example - read a whole line, then decide how to parse it based on first character**
  - *file format is a series of lines, each starts with a code character.*
    - if code character is 'a' data consists of two integers
    - otherwise data consists of a double value followed by a string value.

    -
      ```
      string line;
      getline(in_file, line);
      istringstream iss(line); // construct a streamstring from the line
      iss >> code;
      if(code == ‘a’)
              iss >> int_var1 >> int_var2;
      else
              iss >> double_var1 >> string_var2;
      ```