The `string` class

- an instantiation of a class template:

  ```
  typedef basic_string<char> string;
  ```

- declared in the header file `string`
- for "old" C-style strings, use the header file `cstring`
- a string can contain any character; `'\0'` is not a special character
- `>>` and `<<` can be used for input & output

  ```
  string  s;    // default ctor; empty string
  if (cin >> s) // read next word; skip leading whitespace
    cout << s;
  ```

- the `getline` function can also be used to input into a string

  ```
    string  s;
    while (getline(cin, s)) {
      // for each line read ('\n' is thrown away)
    }

    while (getline(cin, s, ':')) {
      // for each token read
      // tokens are separated by ':' (':' is thrown away)
      // token may contain '\n'
    }
  ```

- several constructors

  ```
  string s1,              // empty string
         s2(10, 'a'),     // string of ten a's
         s3("hello"),     // contains 'h','e','l','l','o'
         s4("hello!", 4),// 'h','e','l','l' (1st 4 chars)
         s5(s3, 3),       // 'l','o' (start from pos 3)
         s6(s3, 1, 2),    // 'e','l' (2 chars, from pos 1)
         s7(s3);          // copy of s3
  ```

  Note: some string functions take arguments that specify a starting position & the number of chars (e.g. `s6` above); these arguments have type `string::size_type` which is an unsigned integral type

  For "number of chars", the special value `string::npos` can usually be used to specify all remaining chars
  example: `string s8(s3,3,string::npos)` would give the same string as `s5` above

  `string::npos` is defined as:

  ```
  static const size_type npos = -1; /* ??  */
  ```

- automatic conversion from `const char *` into strings, but no automatic conversion in the other direction

- use the member function `c_str()` to return the content as a "constant" C-string (`const char *`)

  Note: the returned pointer is valid only until the next call to a nonconstant member function for the same string

  ```
  string  s("123");
  int     n = atoi(s.c_str());
  ```

- the length of a string is returned by the `length()` & `size()` member functions

- `max_size()` returns the max number of characters a string may hold

- `capacity()` returns the total number of characters a string can hold in the memory it has been allocated; `reserve(size_t)` may be used to change the capacity

- `resize()` can be used to resize a string

```
string  s("hello");
s.resize(4);
cout << s;      // prints: hell

string  s2("hello");
s2.resize(7);
cout << s2;     // prints hello & 2 null chars
                // (null char is not special)
string  s3("hello");
s3.resize(7, 'o');
cout << s3;     // prints: hellooo
```

- to check for an empty string, use the `empty()` member function

- can combine strings with C-strings in many situations (comparing, appending, inserting, etc)

- assignment operator (=): the new value can be given as a string, a C-string or a single char

- comparison operators: `==   !=   <   >   <=   >=`

- concatenation (+) & appending (+=)

```
string s, s1("hello"), s2("world");
s = s1 + " " + s2;
cout << s << endl;   // prints: hello world
s1 += '!';
cout << s1 << endl;  // prints: hello!
```

- element access: use [] operator or `at()` member function

```
const_reference  operator[](size_type pos) const;
reference        operator[](size_type pos);
const_reference  at(size_type pos) const;
reference        at(size_type pos);
```

  - [] does <u>not</u> check that position used is valid; `at()` does (it throws `out_of_range` exception if invalid)
  - for the <u>constant</u> version of []; the position after the last char is valid (it returns `'\0'`)
  - for other cases, the actual number of chars is an invalid index
  - the nonconstant versions of [] & `at()` return a character reference which becomes invalid on reallocation

```
string s("hello");
char &p = s[0];
char *q = &s[1];

p = 'H';            // s contains 'H','e','l','l','o'
*q = 'E';           // s contains 'H','E','l','l','o'
s += " world";      // may invalidate p & q
p = 'h';            // may lead to undefined behaviour
```

- use the `substr()` member function to extract a substring

```
string s("goodbye");
s.substr();        // returns a copy of s
s.substr(4);       // returns string("bye")
                   //   (from position 4)
s.substr(4,2);     // returns string("by")
                   //   (2 chars from position 4)
```

- can use the `append()`, `insert()`, `replace()` & `erase()` member functions to modify a string

```
string s, s1("hello"), s2("world");

s1.append(s2);                // s1:helloworld
s1.insert(5," cruel ");       // s1:hello cruel world
s1.replace(6,5,"wonderful");// s1:hello wonderful world
s1.erase(6,9);                // s1:hello  world
                              //   (9 chars from pos 6)
s1.erase(5);                  // s1:hello
s1.insert(1, 3, 'e');         // s1:heeeello
s2.replace(1,string::npos,s1,4,3);  // s2:well
```

these functions return the modified string

```
string s("hi");
cout << s.append("ll") << endl;  // prints hill
cout << s << endl;               // same
```

- there are a number of search functions:
  - `find()` & `rfind()`, `find_first_of()` & `find_last_of()`, `find_first_not_of()` & `find_last_not_of()`
  - each has several versions; e.g.

```
// start looking from position pos
size_type find(const string& str, size_type pos = 0) const;

// start from position pos, compare with first n chars in s
size_type find(const char* s, size_type pos, size_type n) const;

size_type find(const char* s, size_type pos = 0) const;
size_type find(char c, size_type pos = 0) const;
```

    Examples

```
 string  s("okeley-dokeley");
 cout << s.find("ley") << endl;    // prints 3
 cout << s.rfind("ley") << endl;   // prints 11
 cout << s.find_first_of("key") << endl; // prints 1
 cout << s.find_last_of("key") << endl;  // prints 13
```

  - they return `string::npos` (of type `string::size_type`) if the string is not found:

```
string::size_type idx;

idx = s.find("hello");   // assume we have a string s
if (idx == string::npos)
  cout << "'hello' not found!" << endl;
```

Example:

```
// Program name: replace
// Purpose: to replace all occurrences of a specified
//   string by another in a file
// - the old & new strings are specified on the commandline
// - use I/O redirection to read from & write to files
// Example: replace oldstring newstring < infile > outfile
#include <iostream>
#include <string>
using namespace std;

int main(int argc, char *argv[]) {
  if (argc != 3) {
    cerr << "usage: replace <old string> <new string>\n";
    return 0;
  }
  string             s, os(argv[1]), ns(argv[2]);
  string::size_type  idx, oslen = os.length(),
                     nslen = ns.length();
  while (getline(cin, s)) {
    idx = 0;
    // while string is found in the line, replace it
    while ((idx = s.find(os, idx)) != string::npos) {
      s.replace(idx, oslen, ns);
      idx += nslen;
    }
    cout << s << endl;
  }
  return 0;
}
```

Note: there are several versions of `replace`; e.g., a more general version is: `s.replace(pos1, n1, s2, pos2, n2)`