FIRST-ORDER OBJECTS

- Ideally, all the data types in a programming language should be first-order objects.
  - I.e., all the data types should be manipulated in the "usual ways."
  - They should be comparable using the normal operators, passed by value (unless explicitly stated otherwise) to functions, etc. etc.

- C++ has gone a step further than Java in this respect.
  - Indeed, even the "primitive" types can be considered classes; there is only one class of objects in the C++ discourse.

- But then take (yeas, please take) arrays (and thus strings).
  - They cannot be manipulated in the usual way.
  - Indeed, they are in fact pointers to the actual content, so they cannot be meaningfully compared using usual operators, are always passed by reference to functions, etc.
  - Tired of that `strcmp` yet?

VECTORS

- A vector is a relocating, expandable, polymorphic array.
  - They are polymorphic in the usual sense, not Java or Lisp sense.
  - I.e., you can declare vectors that hold any data type, but a given vector instance can hold data of a single type.
- Quick random access but slow copying and expansion.
- Before you begin:

```
#include <vector>
```

- Declaring a vector:

```
vector<int> a(3);  // a vector holding ints, of (initial) size 3
vector<char> b;    // a vector holding chars, of default initial size 0
```

- Accessing values in a vector:

```
a[1] = a[1] + 5;
```

  - `operator[]` does not check for array bounds.

VECTORS (CONT'D)

- Other goodies:
  - You can obtain the size of a vector by using the member function `size()`.
  - You can resize a vector using the member function `resize(int)`.
    - Expensive operation (if size increases)!
  - You cannot initialize a vector using a literal array or for that matter any array.
    - You have to use a loop to initialize the values in a vector, or be happy with the default.

```
void get_ints (vector<int> array) {
  int read_so_far = 0, input;
  while ( cin >> input ) {
    if ( read_so_far == array.size() )
      array.resize(array.size() * 2 + 1);
    array[read_so_far++] = input;
  }
  array.resize(read_so_far);
}
```

VECTORS (CONT'D)

- Other goodies:
  - You can obtain the size of a vector by using the member function `size()`.
  - You can resize a vector using the member function `resize(int)`.
    - Expensive operation (if size increases)!
  - You cannot initialize a vector using a literal array or for that matter any array.
    - You have to use a loop to initialize the values in a vector, or be happy with the default.

```
void get_ints (vector<int>& array) {
  int read_so_far = 0, input;
  while ( cin >> input ) {
    if ( read_so_far == array.size() )
      array.resize(array.size() * 2 + 1);
    array[read_so_far++] = input;
  }
  array.resize(read_so_far);
}
```

SIZE, CAPACITY, PUSHING

- There are two “sizes”: how many elements are stored in the vector (`size()`) and how many elements can be held (`capacity()`).
  - But don’t get excited, when you declare `vector<int> a(3)` the size is set to 3, even if you did not put anything in there explicitly.
  - In most of the cases, you should forget the existennce of `capacity()`.
- Another version of `get_ints`:

```
void get_ints (vector<int>& array) {
  array.resize(0);
  while ( cin >> input ) {
    array.push_back(input);
  }
}
```

  - The member function `push_back` increases the size by 1, and adds the argument as the last element in the vector.
    - ∗ Capacity is also increased if needed.

STRINGS AS FIRST-ORDER OBJECTS

- Before you begin: `#include <string>`
- Declaring strings:

```
string s;            // an (initially empty) string
string s1("hello");  // a string initialized by means of a string literal
```

Operation on string s	Result
`s.length()`	returns the length of s
`s[2]`	accesses the third character in s
`s = "hi";`	assignment operator
`s == "hi"`	true!; special functions no longer needed
`s >= "hello"`	true!
`s = s + " there"`	s becomes `"hi there"`
`s += " there"`	same as above
`s.c_str()`	returns a pointer to the C string held by s (`const char*`, null-terminated)