• more about simple classes
• where have we already seen classes?
• why are classes useful?
• how to define and use your own classes and objects
• arrays of objects
• object members in classes

where have we already seen classes?
• you have already used three classes this semester:
  – string
  – ifstream
  – ofstream
• here are some of the member functions that belong to these classes:
  – string
    ∗ length(), clear(), erase(), replace(), insert(), find(), substr()
  – ifstream:
    ∗ open(), close(), eof()
  – ofstream
    ∗ open(), close()

• we have also mentioned a few data members, though all of these are actually constants
  and so are treated somewhat different from data variables:
  – string::npos
  – ios::in, ios::out — these belong to the ios class (ifstream and ofstream are
    created based on the ios class)
• the syntax with the class name followed by two colons (::) is used to indicate which class
  the member after the two colons belongs to.
  for example:
  – string::npos — string is the name of the class and npos is the name of the
    constant data member belonging to that class
  – ios::in — ios is the name of the class and in is the name of the constant data
    member belonging to that class
  – ios::out — ios is the name of the class and out is the name of the constant data
    member belonging to that class

• we use these classes by declaring variables whose data type is one of these classes, e.g.:
  string x;
• we call x an object of type string
• then we can use the string member functions to operate on the object x, e.g.:
  string x;
  x.clear();
  x.insert( 0, "hello" );
  notice the x. ("x dot") notation
why are classes useful?

• suppose we wanted to create a program that contains the address book from your cell phone

• look at your cell phone address book:
  – what kind of information is listed for each entry?
  – for example:
    ∗ name (first name and last name)
    ∗ cell phone number
    ∗ email address
    ∗ home phone number
    ∗ work phone number

• these are called fields

• if we wanted to write a program that stored all this information for everyone in our cell phone address book, we could do something like example p1.cpp (we'll pretend we only have 3 friends...)

defining a simple class

• it is annoying to have to keep track of so many parallel arrays! so this is why the notion of a class is so useful. we can use a class to link together all the fields for each entry in the cell phone book

• here is a definition of a class that can hold such an entry:

```cpp
class person {
public:
  string last_name;
  string first_name;
  string cell_number;
  string email;
  string home_number;
  string work_number;
  int birth_day;
  int birth_month;
  int birth_year;
};
```

• example p2.cpp shows the previous example (p1.cpp) re-written using this simple class (but for only one person—next, we'll show how to do it with more than one person)
arrays of objects

- you can declare an array where the elements in the array are objects (e.g., instead of ints)
- each element in the array is an object of that class
- for example:
  ```cpp
  person p[3];
  ```
  shows how to declare an array of 3 elements where each element is an object of type person
- you address the elements of the class using a combination of the array [] notation and the dot notation, like this:
  ```cpp
  p[0].last_name = "sklar";
  ```
- example p3.cpp shows the same example as p1.cpp, but with an array of person objects

objects as class members

- finally, you can define classes that have data members which are objects
- suppose that we wanted to define a special class just for storing the name data:
  ```cpp
class name {
  public:
  string last;
  string first;
};
```  
- then we could use the name class when defining the person class:
  ```cpp
class person {
  public:
  name my_name;
  string cell_number;
};
```  
- you declare a variable of type person, as before:
  ```cpp
  person p;
  ```
- and you address the elements of a nested class using double dot notation, like this:
  ```cpp
  p.my_name.last = "sklar";
  ```
- example p4.cpp is a modified version of p2.cpp, using two classes