SIMPLE CLASSES

Today

- This lecture looks at simple classes.
- Classes are the foundation of *object-oriented programming*
- FINAL EXAM: Tuesday 23rd December
- Review sessions.

Simple classes

- Classes are ways of organizing programs to provide structure
- A class is a special kind of *compound* data type
- Classes are compound because they have *members*
- There are two types of members in classes:
  - *data* members
  - *function* members
- The *dot operator* (.) is used to indicate the member of a class

- You have already used three classes this semester:
  - `string`
  - `ifstream`
  - `ofstream`
- Can you think of some of the member functions that belong to these classes?

- 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 (which we'll talk about later):
  - `string::npos`
  - `ios::in`, `ios::out` — these belong to the `ios` class (`ifstream` and `ofstream` are created based on the `ios` 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

Simple class example

- 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 `class1.cpp`.

- The idea is that it is annoying to have to keep track of so many parallel arrays
- So this is where we introduce a *class*
- A class will help us 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:

```
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;
};
```

- Things to notice:
  - Two new C++ keywords: `class` and `public`
  - There is a semi-colon at the END OF THE CLASS DEFINITION, after the last curly brace (`}`)
- Now `class2.cpp` is our example 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 of a class.
- Each element in the array is then an *object* of that class.
- Our example, with an array of `person` objects is in `class3.cpp`.
- The array definition is just:

```
person p[3];
```

Nested classes

- Finally, you can *nest* classes.
- This means you declare a data member in one class whose data type is that of another class.
- A modified version of the one-person address book, using two classes is given in `class4.cpp`.
- The class that gets nested is

```
class name {
public:
  string last;
  string first;
};
```

- The modified `person` class is then

```
class person {
public:
  name my_name;
  string cell_number;
  string email;
  string home_number;
  string work_number;
  int birth_day;
  int birth_month;
  int birth_year;
};
```

Summary

- This lecture introduced the ideas of simple classes.
- We discussed:
  - How to define classes.
  - How to use classes.
  - Arrays of classes.
  - Nested arrays.
- There is a lot more to classes — some of this is convered in CIS 15.