# SIMPLE CLASSES

## Today

- This lecture looks at simple classes.
- Classes are the foundation of *object-oriented programming*
- FINAL EXAM: Wednesday 16th December from 10.30am until 12.30.
  - The room will be announced later.
- Review sessions.
- Sample code can be found on the class website in the files `class.cpp` and `simple-class.cpp`.



## 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

- The most important thing about classes for now is their ability to group bits of information together.
- Here is an example:

```
class simple{
public:

  int x;
  double d;
};
```

- This defines a class `simple` with one integer member `x` and one double member `d`.

- 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 (})



- We use the dot notation to access the members:

```
simple s1, s3;

s1.x = 3;
s1.d = 2.7;
```

- `s1` and `s3` are *objects*, *instances* of the new kind of data we defined.



- Like any kind of variable, we can pass instances of classes to functions.

```
void printSimple(simple s2){
  cout << "The int is: " << s2.x << endl;
  cout << "The double is: " << s2.d << endl;
}
```

- We can also return an object from a function:

```
simple doubleSimple(simple s1){
  s1.x *= 2;
  s1.d *= 2;

  return s1;
}
```



- Here's another example of a class.

```
class pose{
public:

  double px;
  double py;
  double pa;
};
```

- This comes from a robot simulator, and holds the three values that identify the location of the robot.
- More examples can be found in `simple-class.cpp` on the class website.

## Another 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*.



- 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;
};
```



## Arrays of objects

- You can declare an array of a class.
- Each element in the array is then an *object* of that class.
- An array of `person` objects is just:

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

- Each element of the array is then a `person` object, and we can do everything to it that we can do to a `person`:

```
p[0].first_name = "Simon";
p[2].birth_month = 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.
- As an example, we have:

```
class lessSimple{
public:

  int    v;
  pose   p;
  simple s;
};
```

which uses both the `simple` and `pose` classes.

- We use an extension of the dot notation to access the members of the members.
- For example

```
lessSimple l1;

l1.p.pa = 2.7;

l1.s.x = 15;
l1.s.d = 27.3;

printSimple(doubleSimple(l1.s));
```



- We can also use this idea in our address book example.
- We can create a class to hold name information

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



- We can then use this class in the address book.
- The modified `person` class is

```
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.