aggregate data types

- class and struct
- struct comes from C
- class is new in C++
- both are aggregate types, meaning that they group together multiple fields of data
- for example:
  ```
  struct point {
      double x, y;
  };
  ```
- don’t forget to put a semi-colon at the end of the structure definition!
- in C, the tag (point) is optional and does not constitute a data type (you need to use typedef as well)
- but in C++, the tag is considered a data type, hence the above example is a data type definition
- which means that you can use point as a data type, e.g.:

```
point p;
``` 

member functions

- in C++, members of aggregate data types can be functions
- (C only allows data members)
- in object-oriented programming (OOP) lingo, the word “method” is often used instead of “function”
- the reason to define functions inside an aggregate data type is to follow the OOP principle of encapsulation—operations should be packaged with data
- for example:

```
#include <iostream>
using namespace std;
struct point {
    double x, y;
    void print() const {
        cout << "\n" << x << "," << y << 
    }
};
```
void set( double u, double v ) {
    x = u;
    y = v;
} // end of struct--don't forget semi-colon!

int main() {
    point w;
    w.set( 1.2, 3.4 );
    cout << "point = ";
    w.print();
}

• notes:
  – const keyword in definition of print method indicates that the data members will not
    be modified inside the method
  – notice that the set method changes the values of the data members—this is considered
    good OOP practise
  – defining the methods inside the struct definition is called "in-line declaration"; this is
    generally only okay for short, concise methods

• the class scope operator can be used when in-line declarations are inappropriate
• for example:

```c++
#include <iostream>
using namespace std;

struct point {
    double x, y;
    void print() const;
    void set( double u, double v );
} // end of struct--don't forget semi-colon!

void point::print() const {
    cout << "( \( x \), \( y \) )\n";
} // end of print()

void point::set( double u, double v ) {
    x = u;
    y = v;
} // end of set()
```

public and private access

• members of structures can be public or private
• public means that any code can access the members
• private means that only code inside the class or struct can access the members (or
  "friend" classes, to be discussed later in the term)
• typically, following good OOP practice, all data members are private and only function
  members are public (but not all—only those that need to be accessed outside of the
  struct or class)
• for example:

```c++
struct point {
    public:
        void print() const;
        void set( double u, double v );
    private:
        double x, y;
} // end of struct--don't forget semi-colon!
```
classes vs structs

- in C++, keyword `class` is introduced
- the difference between structs and classes is:
  - in a struct, the members are public by default
  - in a class, the members are private by default

- for example:

```cpp
#include <iostream>
using namespace std;

class point {
  double x, y;
  public:
    void print() const;
    void set( double u, double v );
}; // end of struct -- don't forget semi-colon!
```

- otherwise, class and struct are the same
- but by convention, C++ programmers tend to use `class`
++:count; // global count
     ) // end of how_many()

• this is only necessary since count is declared twice
• if you didn’t have the ::count, then the second time, it would also refer to the local
variable
• it is better practice not to use global variables; or at least if you do, give them unique
names to avoid confusion :-)

nested classes

• classes can be nested
• here’s another confusing example from the book:
  char c; // global scope
class X {
    public:
      char c; // local scope in class X
      class Y {
        public:
          void foo( char e ) { X t; ::c = t.c = c = e; }
        private:
          char c; // local scope in class Y
        }
    }

• the scope of the third (last) c is X::Y::c

"this" pointer

• the keyword this is used to refer to an instance of a class from within itself
• it is a pointer — something we will discuss at length in the next unit
• for example:
  point inverse() {
    x = -x;
y = -y;
    return (*this);
  }
• this function returns a pointer to itself, i.e., the address of the object in memory
• we’ll come back to this when we discuss pointers

"static" members

• the static keyword is used to refer to members that do not need to be instantiated
• in other words, it is independent of any class variable
• for example:
  class point {
    public:
      static int dimensions;
    ...
  }
  ...
  int main() {
    ...
  }
"const" members

- Members with the `const` keyword in their definition cannot be modified.
- This refers either to data members or to function members to indicate that the data members contained therein are not modified.
- For example:
  ```cpp
class point {
    double x, y;
    public:
      const int dimensions = 2;
      void print() const;
    }
  void point::print() const {
    cout << "(" << x << ", " << y << "\n";
  }  // end of print()
```
- Note that the `mutable` keyword can override this.
- For example:
  ```cpp
  mutable int delta;
  ```
  means that even if `delta` is referenced inside a `const` function, its value can be modified.

Special types of classes: "containers"

- There are several special types of classes in C++.
- The first we will discuss is called a container.
- It is a class designed to hold large numbers of objects.
- For example:
  ```cpp
  #include <iostream>
  using namespace std;

  class ch_stack {
    public:
      void reset() { top = EMPTY; }
      void push( char c ) { s[++top] = c; }
      char pop() { return s[top--]; }
      char top_of() const { return s[top]; }
      bool empty() const { return( top==EMPTY ); }
      bool full() const { return( top==FULL ); }
    private:
  }
  ```
enum { max_len = 100, EMPTY = -1, FULL = max_len - 1 };  
char s[max_len];  
int top;
}

int main() {
    ch_stack s;
    char str[40] = { "hello world!" };  
    int i = 0;
    cout << "str=" << str << endl;
    s.reset();
    while( str[i] && ! s.full() ) {
        s.push( str[i++];
    }
    cout << "reversed str=";
    while ( ! s.empty() ) {
        cout << s.pop();
    }
    cout << endl;
} // end of main()

class design

• data members should be private ("hidden")
• function members are often public (but not always—private function members can be used for computations internal to a class)
• functions that do not modify data members should be const
• pointers add indirection (we’ll talk about that later)
• a uniform set of functions should be included: set(), get(), print()
• UML (unified modeling language) provides a graphical method for representing classes

point

dimension
x
y
print()
set()
inverse()