Software Design
Abstract Data Types

1. Software Design
   - top down, bottom up, object-oriented
   - abstract data types
2. Specifying a Class `Clock`
   - date and time in a C++ program
   - encapsulating C code
   - public and private parts, a friend operator
3. Implementing the Class `Clock`
   - the `.cpp` file
   - the `this` pointer

MCS 360 Lecture 4
Introduction to Data Structures
Jan Verschelde, 22 January 2020

Abstract Data Types

1. Software Design
   - top down, bottom up, object-oriented
   - abstract data types
2. Specifying a Class `Clock`
   - date and time in a C++ program
   - encapsulating C code
   - public and private parts, a friend operator
3. Implementing the Class `Clock`
   - the `.cpp` file
   - the `this` pointer

Software Design

C supports top down and bottom up design:

- top down: main program divided in tasks,
  the tasks are implemented by functions;
- bottom up: libraries of functions on data types
  e.g.: mathematical libraries for linear algebra
  (LINPACK, EISPACK, LAPACK).

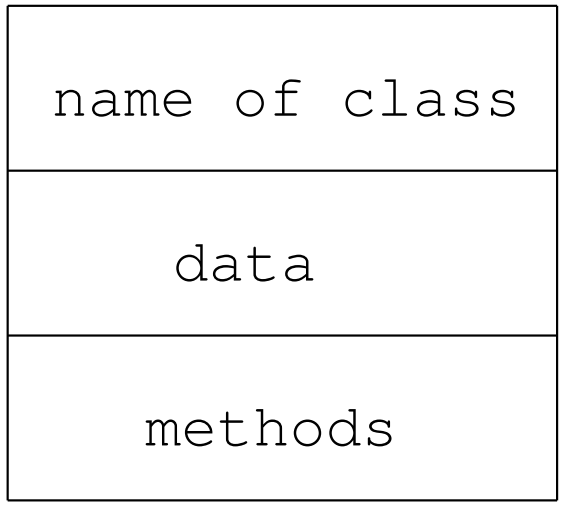
C++ is an object-oriented language:

1. model real-world entities by objects,
2. every object belongs to a class,
3. a class has data and functional attributes.

Unified Modeling Language (UML)

UML is a graphical language to model, design, and construct object-oriented software.

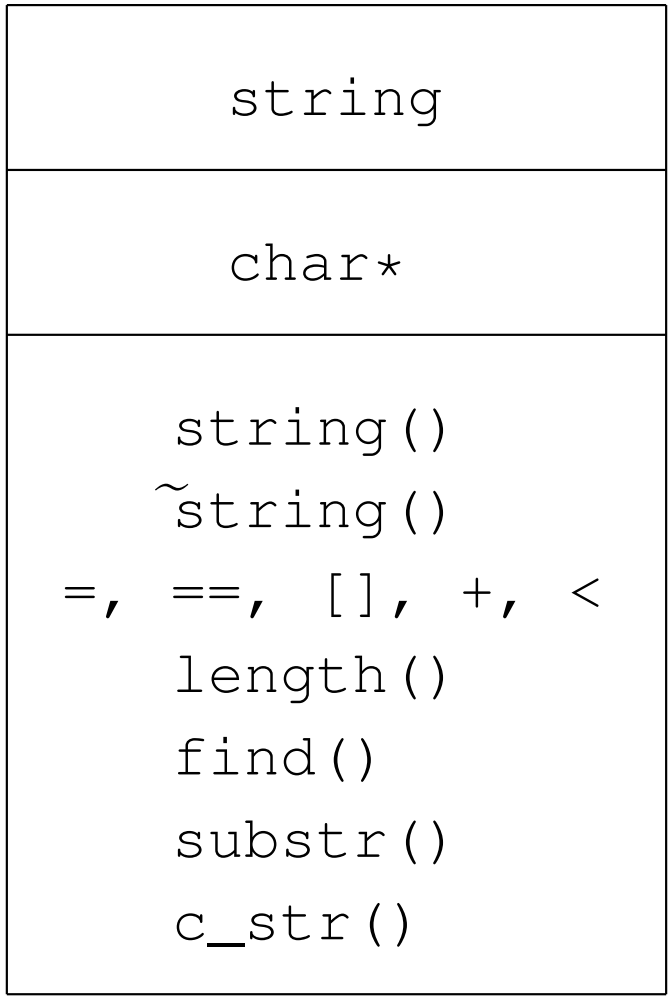
A class diagram:



Three principles of good design:

1. information hiding
2. low coupling and high cohesion
3. design for change

class diagram for strings



Abstract Data Types

1. Software Design
   - top down, bottom up, object-oriented
   - abstract data types
2. Specifying a Class `Clock`
   - date and time in a C++ program
   - encapsulating C code
   - public and private parts, a friend operator
3. Implementing the Class `Clock`
   - the `.cpp` file
   - the `this` pointer

Abstract Data Types

A class implements an *abstract data type* (ADT).
An ADT defines the *interface* of a class,
what is public of the data and methods.

An ADT is a ***contract*** between software designer
and the programmer who writes the code.

For every operation we specify

- a *precondition*: requirements on the input parameters,
- a *postcondition*: properties of the values on output.

If preconditions hold on input
and the implementation of the function is correct,
then the postconditions hold on output.

specifying `find`

The method `find()` on any string
could be specified as follows:

```
size_t find ( const string& t, size_t p ) const;
/*
   Preconditions:
     t is target we search for in a string;
     p is the start position for the search,
     if omitted, then the search starts at 0.
   Postcondition:
     if r == s.find(t,p) then either
       s[r] == t if t occurs in s,
     or r == string::npos otherwise. */
```

Note: `-1` $\notin$ `size_t`.
`s.find()` does not change `s`: so `const` at end

specifying `substr`

The method `substr()` on any string could be specified as follows:

```
string substr ( size_t p,
                size_t n = string::nops ) const;
/*
   Preconditions:
     p is a start position in the string;
     n is the number of characters of the
       substring on return, may be omitted.
   Postcondition:
     if ss == s.substr(p) then
       ss[k] == s[p+k], k in 0..s.length()-p-1
     if ss == s.substr(p,n) then
       ss[k] == s[p+k], k in 0..n-1. */
```

modeling a bubble gum dispenser

Exercise 1:

1. Consider a bubble gum dispenser.

   The dispenser releases one bubble gum at a time until empty.

   Filling of the dispenser adds a positive number of bubble gums.

   Write an Abstract Data Type for a bubble gum dispenser.

   Draw the UML class diagram.

Abstract Data Types

1. Software Design
   - top down, bottom up, object-oriented
   - abstract data types

2. Specifying a Class `Clock`
   - date and time in a C++ program
   - encapsulating C code
   - public and private parts, a friend operator

3. Implementing the Class `Clock`
   - the `.cpp` file
   - the `this` pointer

date and time

Typing `date` at the command prompt of a Unix(-like) OS shows the date and the current time.

We can get this information in a C++ program, in the PowerShell:

```
Lec04> ./current_time
Fri Jan 24 09:29:32 2020
the raw time is 1579879772
The current date is 2020/01/24.
The current time is 09:29:32.
Lec04>
```

date as a string

```
#include <ctime>
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
   time_t now = time(0); // current time

   string s = asctime(localtime(&now));
   cout << s;
```

the tm structure

```
struct tm *tm_ptr;
// tm_ptr = gmtime(&now); // Greenwich Mean time
tm_ptr = localtime(&now);

cout << "the raw time is " << now << endl;

cout << setfill('0');

cout << "The current date is "
     << 1900 + tm_ptr -> tm_year << "/"
     << setw(2) << 1 + tm_ptr -> tm_mon << "/"
     << setw(2) << tm_ptr -> tm_mday << "." << endl;

cout << "The current time is "
     << setw(2) << tm_ptr -> tm_hour << ":"
     << setw(2) << tm_ptr -> tm_min << ":"
     << setw(2) << tm_ptr -> tm_sec << "." << endl;
```

Abstract Data Types

1. Software Design
   - top down, bottom up, object-oriented
   - abstract data types

2. Specifying a Class `Clock`
   - date and time in a C++ program
   - encapsulating C code
   - public and private parts, a friend operator

3. Implementing the Class `Clock`
   - the `.cpp` file
   - the `this` pointer

the main C++ program

```
$ time_with_clock
The current time is 17:35:32.
$
```

```
#include "a_class_clock.h"
#include <iostream>
using namespace std;

int main()
{
   Clock c;

   cout << "The current time is "
        << c << "." << endl;

   return 0;
}
```

defining a class

```
#ifndef A_CLASS_CLOCK_H
#define A_CLASS_CLOCK_H

class Clock
{
   public:
      // exported data and members

   private:
      // hidden class attributes
};

#endif
```

.h and .cpp files

The header file a_class_clock.h contains
the prototypes of all member functions.

In the file a_class_clock.cpp:

```
#include "a_class_clock.h"
```

If we save the main program in time_with_clock.cpp
then we compile at the command prompt $ as

```
$ g++ -c a_class_clock.cpp
$ g++ -o time_with_clock \
  a_class_clock.o time_with_clock.cpp
```

a makefile defines the compilation instructions

For our example we define a `makefile`:

```
gpp=C:\MinGW\bin\g++

time_with_clock:
        $(gpp) -c a_class_clock.cpp
        $(gpp) -c time_with_clock.cpp
        $(gpp) a_class_clock.o time_with_clock.o \
               -o time_with_clock
```

Then typing `make time_with_clock` at the command line will execute the compilation and linking commands.

- With `gpp = C:\MinGW\bin\g++` we define a specific compiler.
- Instead of `g++` we write `$(gpp)` to compile.
- Observe the TABs after before the compilation statements.

Abstract Data Types

1. Software Design
   - top down, bottom up, object-oriented
   - abstract data types

2. Specifying a Class `Clock`
   - date and time in a C++ program
   - encapsulating C code
   - public and private parts, a friend operator

3. Implementing the Class `Clock`
   - the `.cpp` file
   - the `this` pointer

public part of `Clock`

```
public:

   Clock();
   /* sets the clock to the current time */

   void set_clock(int h, int m, int s);
   /* sets clock to hour given in h,
      minutes to m, and seconds to s. */

   int get_hours() const;
   /* returns the value of hours */

   int get_minutes() const;
   /* returns the value of minutes */

   int get_seconds() const;
   /* returns the value of seconds */
```

a friend method

To define the insertion operator:

```
friend std::ostream& operator<<
   (std::ostream& os, const Clock& c);
/* defines output of an object clock */
```

A `friend` method does not apply to an object to the class, but has access to all attributes of the object.

private parts

```
private:

   int hours;    /* hours in 24 hour format
                    integer in the range 0..23 */
   int minutes; /* minutes in range 0..59 */
   int seconds; /* integers in range 0..61
                    allowing for leap second */
```

Principle of information hiding:

- only through `get_hours()`
  do we get the value of `hours`
- changing of `hours` is done only
  via methods defined in the class.

a bubble gum dispenser as a C++ class

Exercise 2:

2. Consider the bubble gum dispenser of the previous exercise.

   Define a C++ class for a bubble gum dispenser object.

   The number of bubble gums in the dispenser is private.

Abstract Data Types

1. Software Design
   - top down, bottom up, object-oriented
   - abstract data types
2. Specifying a Class `Clock`
   - date and time in a C++ program
   - encapsulating C code
   - public and private parts, a friend operator
3. Implementing the Class `Clock`
   - the `.cpp` file
   - the `this` pointer

a_class_clock.cpp

```
#include "a_class_clock.h"
#include <ctime>
#include <iomanip>

Clock::Clock()
{
   time_t now = time(0);
   struct tm *tm_ptr;

   tm_ptr = localtime(&now);
   hours = tm_ptr -> tm_hour;
   minutes = tm_ptr -> tm_min;
   seconds = tm_ptr -> tm_sec;
}
```

the other methods

```
void Clock::set_clock(int h, int m, int s)
{
   hours = h;
   minutes = m;
   seconds = s;
}
int Clock::get_hours() const
{
   return hours;
}
int Clock::get_minutes() const
{
   return minutes;
}
int Clock::get_seconds() const
{
   return seconds;
}
```

defining the output

```
std::ostream& operator<<
   (std::ostream& os, const Clock& c)
{
    os << std::setfill('0')
       << std::setw(2) << c.get_hours() << ":"
       << std::setw(2) << c.get_minutes() << ":"
       << std::setw(2) << c.get_seconds();
    return os;
}
```

Abstract Data Types

1. Software Design
   - top down, bottom up, object-oriented
   - abstract data types
2. Specifying a Class `Clock`
   - date and time in a C++ program
   - encapsulating C code
   - public and private parts, a friend operator
3. Implementing the Class `Clock`
   - the `.cpp` file
   - the `this` pointer

defining equality

Add to `public:` of the `class Clock` in the file `a_class_clock.h`:

```
bool Clock::operator==(const Clock& c) const;
/* returns true if the time stored at the clock
   equals the time stored at c */
```

Testing in `time_with_clock.cpp`:

```
Clock d;
d.set_clock(16,52,3);
bool test_eq = (c == d);
cout << c << " == " << d << " is "
     << test_eq << endl;
```

the `this` parameter

`this` is a pointer to the object to which we send the member function:

```
bool Clock::operator==(const Clock& c) const
{
   if((this->hours == c.hours)
     && (this->minutes == c.minutes)
     && (this->seconds == c.seconds))
      return true;
   else
      return false;
}
```

Notation: `this->minutes == (*this).minutes`.
Note that `*this == c` can be interpreted as the comparison of a pointer with a reference.

implementing the bubble gum dispenser

Exercise 3:

3. Consider again the bubble gum displenser of the previous two exercises.

   Write an implementation for the class.

   Write a simple test program to demonstrate that your class is implemented correctly.

Summary and Additional Exercises

We started Chapter 1: *Introduction to Software Design*.
Classes in C++ are a way to implement ADTs.

Additional Exercises:

4. Extend the class `Clock` with a method `update()` that sets the data attributes to the hours, minutes, and seconds of the current time.
5. Write a C++ program that prompts the user to hit the enter key twice. Show the elapsed (wall clock) time between the two hits.
6. Extend the class `Clock` with the `<` operator.
7. Describe how to extend the class `Clock` so it serves as an alarm clock.