

CH-230-A

Programming in C and C++

C/C++

Tutorial 13

Dr. Kinga Lipskoch

Fall 2020

Moving Within a Stream

- ▶ The next byte you will put/get to/from a stream has a position
- ▶ `tell` functions return the current absolute position in a stream
- ▶ `seek` functions move the specified positions
- ▶ Positions can be either absolute or relative:
 - ▶ `ios::beg` relative to beginning (absolute)
 - ▶ `ios::cur` relative to current position
 - ▶ `ios::end` relative to end of file
- ▶ `p` (put) suffix for `ostreams`
- ▶ `g` (get) suffix for `istreams`

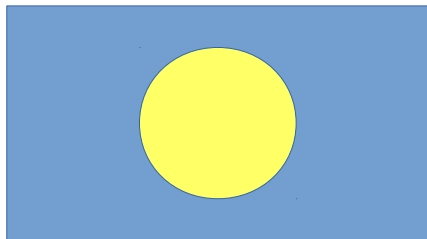
How to Move in a Stream?

- ▶ `ios::pos_type tellp()` returns current position
- ▶ `ios::pos_type tellg()` returns current position
- ▶ `seekp(pos_type)` moves to given absolute position
- ▶ `seekg(pos_type)` moves to given absolute position
- ▶ `seekp(pos_type, off_type)` moves to relative position, as given by `off_type`
- ▶ `seekg(pos_type, off_type)` moves to relative position, as given by `off_type`
- ▶ `pos_type` can be converted to `int` or `long`
- ▶ `off_type` is either `ios::beg`, `ios::cur` or `ios::end`

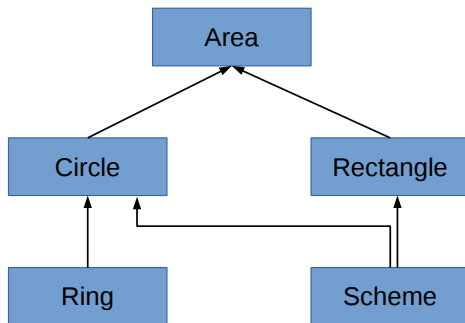
Some Examples

- ▶ `seek.cpp`
- ▶ The same stream can be used multiple times for reading from different files
- ▶ Just close it and reopen it to bind to a different file

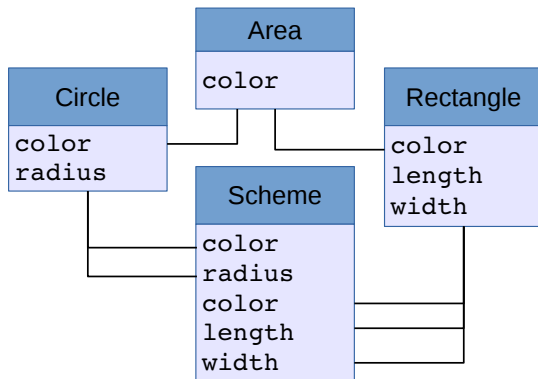
Multiple Inheritance: A Scheme



Class Hierarchy



Class Hierarchy (non-virtual Base Class)



Non-virtual Base Classes

- ▶ With polymorphism it is possible that a class can act as indirect base class multiple times
- ▶ Here Scheme will include members of the indirect base class Area multiple times
- ▶ One possibility to solve this problem is to use `static_cast<Circle *>` to create unambiguoussness

Virtual Base Class

- ▶ Create unambiguousness by declaring a base class as virtual (i.e., inherit with `: public virtual`)
- ▶ Then only one subobject will be created
- ▶ In our example we need to explicitly call the Area constructor (with parameters) since no default constructor exists
- ▶ `nonvirtualinheritance.cpp`
- ▶ `virtualinheritance.cpp`
- ▶ `nonvirtualinheritance2.cpp`
- ▶ `virtualinheritance2.cpp`

Exceptions

Errors happen because of:

- ▶ Hardware
- ▶ Changed environments
- ▶ Wrong usage or operation
- ▶ Bugs

Conventional Error Handling

- ▶ Already available in C
 - ▶ Check whether pointer is NULL
 - ▶ Check `errno`
- ▶ `conventional_error_handling.cpp`

New Keywords (1)

```
1 try
2 {
3     // code, where exception
4     // might occur
5 }
6 catch (char* text)
7 {
8     // statements to be executed if
9     // char* exception occurs
10 }
```

New Keywords (2)

- ▶ Statement that explicitly triggers a `char *` exception
 - ▶ `throw "No memory available";`
- ▶ Statement that explicitly triggers an `int` exception
 - ▶ `throw 12345;`

try and catch (1)

- ▶ No exception in try-block
 - ▶ No exception handler is called
 - ▶ Program continues after catch-block
- ▶ `throw` within try creates exception
 - ▶ No further code in try-block is executed
 - ▶ Destructor for locally defined objects is called, before code in exception handler is run

try and catch (2)

- ▶ Exception in try-block
 - ▶ First matching catch-block is executed
 - ▶ All other handlers are ignored
 - ▶ At most one handler is being called
- ▶ Exception in try-block, but no matching handler
 - ▶ Default action for uncaught exceptions
 - ▶ Usually it ends the program

Exception Handling

- ▶ Blocks of code are specially marked
- ▶ If error occurs than control goes to special error routines
- ▶ `exception_handler.cpp`

exception Class

- ▶ Class defines error class that receives objects via throw on exception
- ▶ Provides methods to give information about the error
- ▶ `class_exception.h`
- ▶ `class_test.h`
- ▶ `class_test.cpp`
- ▶ `test_exception.cpp`
- ▶ `test_exception2.cpp`

All-round Handler

`terminate.cpp`