

Exercises week 4: Polymorphism

Klaas Isaac Bijlsma
s2394480

David Vroom
s2309939

December 20, 2017

Exercise 25

Learn to construct an `ostream` class

We constructed the class `BiStream`, which offers the same facilities as `ostream`, but inserts its information into two files, whose `ofstream`-objects are passed to this class's constructor. A second class `BiStreamBuffer` is made and used. We used the following code,

```
                                bistream/bistream.h
1 | #ifndef INCLUDED_BISTREAM_
2 | #define INCLUDED_BISTREAM_
3 |
4 | #include <fstream>
5 | #include "../bistreambuffer/bistreambuffer.h"
6 |
7 | class BiStream: private BiStreamBuffer, public std::ostream
8 | {
9 |     public:
10 |         BiStream(std::ofstream &one, std::ofstream &two);
11 | };
12 |
13 | #endif
```

```
                                bistream/bistream.ih
1 | #include "bistream.h"
```

```

2
3 using namespace std;

```

bistream/bistream1.cc

```

1 #include "bistream.i.h"
2
3 BiStream::BiStream(std::ofstream &one, std::ofstream &two)
4 :
5     BiStreamBuffer(one, two),
6     ostream(this)
7 {}

```

bistreambuffer/bistreambuffer.h

```

1 #ifndef INCLUDED_BISTREAMBUFFER_
2 #define INCLUDED_BISTREAMBUFFER_
3
4 #include <streambuf>
5
6 class BiStreamBuffer: public std::streambuf
7 {
8     std::ostream *d_one;
9     std::ostream *d_two;
10
11     public:
12         BiStreamBuffer(std::ostream &one, std::ostream &two);
13
14     private:
15         int overflow(int c) override;
16 };
17
18 #endif

```

bistreambuffer/bistreambuffer.i.h

```

1 #include "bistreambuffer.h"
2 #include <ostream>

```

```
3 |
4 | using namespace std;
```

bistreambuffer/bistreambuffer1.cc

```
1 | #include "bistreambuffer.ih"
2 |
3 | BiStreamBuffer::BiStreamBuffer(std::ostream &one, std::ostream &two)
4 | :
5 |     d_one(&one),
6 |     d_two(&two)
7 | {}
```

bistreambuffer/overflow.cc

```
1 | #include "bistreambuffer.ih"
2 |
3 | int BiStreamBuffer::overflow(int c)
4 | {
5 |     if (c == EOF)
6 |     {
7 |         d_one->flush();
8 |         d_two->flush();
9 |     }
10 |    else
11 |    {
12 |        d_one->put(c);
13 |        d_two->put(c);
14 |    }
15 |    return c;
16 | }
```

Exercise 26

Learn to design a streambuf reading from file descriptors

We designed the class `IFdStreambuf`, whose objects may be used as a `streambuf` of `istream` objects to allow extractions from an already open file descriptor. We used the following code,

```
                                ifdstreambuf.h
1  #ifndef EX26_IFDSTREAMBUF_H
2  #define EX26_IFDSTREAMBUF_H
3
4  #include <streambuf>
5
6  class IFdStreambuf: public std::streambuf
7  {
8      public:
9          enum Mode
10         {
11             KEEP_FD,
12             CLOSE_FD
13         };
14
15     protected:
16         int d_fd;
17         Mode d_mode;
18         size_t const d_bufsize = 100;
19         char *d_buffer;
20
21
22     public:
23         explicit IFdStreambuf(Mode mode = KEEP_FD);           // 1
24         explicit IFdStreambuf(int fd, Mode mode = KEEP_FD);  // 2
25         virtual ~IFdStreambuf();
26         int close();
27         void open(int fd, Mode mode = KEEP_FD);
28
29     private:
30         int underflow() override;
31         std::streamsize xsgetn(char *dest, std::streamsize n) override;
```

```

32 };
33
34 #endif

```

ifdstreambuf.ih

```

1 #include "ifdstreambuf.h"
2 #include <unistd.h>           // read(), close()
3 #include <string.h>           // memcpy()
4
5 using namespace std;

```

close.cc

```

1 #include "ifdstreambuf.ih"
2
3 int IFdStreambuf::close()
4 {
5     return ::close(d_fd);
6 }

```

destructor.cc

```

1 #include "ifdstreambuf.ih"
2
3 IFdStreambuf::~~IFdStreambuf()
4 {
5     delete[] d_buffer;
6     if (d_mode)
7         close();
8 }

```

ifdstreambuf1.cc

```

1 #include "ifdstreambuf.ih"
2
3 IFdStreambuf::IFdStreambuf(Mode mode)

```

```

4 | :
5 |     d_fd(-1),          // set later by open
6 |     d_mode(mode),
7 |     d_buffer(new char[d_bufsize])
8 | {}

```

ifdstreambuf2.cc

```

1 | #include "ifdstreambuf.ih"
2 |
3 | IFdStreambuf::IFdStreambuf(int fd, Mode mode)
4 | :
5 |     d_fd(fd),
6 |     d_mode(mode),
7 |     d_buffer(new char[d_bufsize])
8 | {
9 |     setg(0, 0, 0);          // buffer is initially empty
10 | }

```

open.cc

```

1 | #include "ifdstreambuf.ih"
2 |
3 | void IFdStreambuf::open(int fd, Mode mode)
4 | {
5 |     d_fd = fd;
6 |     d_mode = mode;
7 | }

```

underflow.cc

```

1 | #include "ifdstreambuf.ih"
2 |
3 | int IFdStreambuf::underflow()
4 | {
5 |     if (gptr() < egptr())
6 |         return *gptr();
7 |

```

```

8      int nRead = read(d_fd, d_buffer, d_bufsize);
9
10     if (nRead <= 0)
11         return EOF;
12
13     setg(d_buffer, d_buffer, d_buffer + nRead);
14     return static_cast<unsigned char>(*gptr());
15 }

```

xsgetn.cc

```

1  #include "ifdstreambuf.ih"
2
3  streamsize IFdStreambuf::xsgetn(char *dest, streamsize n)
4  {
5      if (n == 0)
6          return 0;
7
8      int nBuffer = in_avail(); // number of retrievable chars in buffer
9
10     if (nBuffer > n)           // more chars in buffer than requested
11         nBuffer = n;
12
13         // copy what's available in own buffer
14     memcpy(dest, gptr(), nBuffer);
15     gbump(nBuffer);           // update pointer
16
17     // try to read some more from FD
18     int nFile = read(d_fd, dest + nBuffer, n - nBuffer);
19
20     return nBuffer + nFile;
21 }

```

Exercise 27

Learn to design a streambuf writing to file descriptors

We designed the class `OFdStreambuf`, whose objects may be used as a `streambuf` of `ostream` objects to allow insertions into an file descriptor. We used the following code,

```
                                ofdstreambuf.h
1  #ifndef EX27_OFDSTREAMBUF_H
2  #define EX27_OFDSTREAMBUF_H
3
4  #include <streambuf>
5
6  class OFdStreambuf: public std::streambuf
7  {
8      public:
9          enum Mode
10         {
11             KEEP_FD,
12             CLOSE_FD
13         };
14
15     protected:
16         int d_fd;
17         Mode d_mode;
18         size_t const d_bufsize = 100;
19         char *d_buffer;
20
21     public:
22         explicit OFdStreambuf(Mode mode = KEEP_FD);           // 1
23         explicit OFdStreambuf(int fd, Mode mode = KEEP_FD);  // 2
24         virtual ~OFdStreambuf();
25         int close();
26         void open(int fd, Mode mode = KEEP_FD);
27
28     private:
29         int sync() override;
30         int overflow(int c) override;
31 };
32
```



```
33 | #endif
```

ofdstreambuf.ih

```
1 | #include "ofdstreambuf.h"
2 | #include <unistd.h>           // read(), close()
3 |
4 | using namespace std;
```

close.cc

```
1 | #include "ofdstreambuf.ih"
2 |
3 | int OFdStreambuf::close()
4 | {
5 |     return ::close(d_fd);
6 | }
```

destructor.cc

```
1 | #include "ofdstreambuf.ih"
2 |
3 | OFdStreambuf::~~OFdStreambuf()
4 | {
5 |     sync();
6 |     delete[] d_buffer;
7 |
8 |     if (d_mode)
9 |         close();
10 | }
```

ofdstreambuf1.cc

```
1 | #include "ofdstreambuf.ih"
2 |
3 | OFdStreambuf::OFdStreambuf(Mode mode)
4 | :
```

```

5 |     d_fd(-1),          // set later by open
6 |     d_mode(mode),
7 |     d_buffer(new char[d_bufsize])
8 | {}

```

ofdstreambuf2.cc

```

1 | #include "ofdstreambuf.ih"
2 |
3 | OFdStreambuf::OFdStreambuf(int fd, Mode mode)
4 | :
5 |     d_fd(fd),
6 |     d_mode(mode),
7 |     d_buffer(new char[d_bufsize])
8 | {
9 |     setp(d_buffer, d_buffer + d_bufsize);
10 | }

```

open.cc

```

1 | #include "ofdstreambuf.ih"
2 |
3 | void OFdStreambuf::open(int fd, Mode mode)
4 | {
5 |     d_fd = fd;
6 |     d_mode = mode;
7 | }

```

overflow.cc

```

1 | #include "ofdstreambuf.ih"
2 |
3 | int OFdStreambuf::overflow(int c)
4 | {
5 |     sync();
6 |     if (c != EOF)
7 |     {
8 |         *pptr() = c; // or static_cast<char>(c);

```

```
9      pbump(1);
10     }
11     return c;
12 }
```

sync.cc

```
1 #include "ofdstreambuf.ih"
2
3 int OFdStreambuf::sync()
4 {
5     if (pptr() > pbase())
6     {
7         write(d_fd, d_buffer, pptr() - pbase());
8         setp(d_buffer, d_buffer + d_bufsize);
9     }
10    return 0;
11 }
```

Exercise 28

Learn to design streams

We designed `IFdStream` and `OFdStream`, which are `istream` and `ostream` objects, respectively, reading from and writing to streams. We also made a main function that copies information entered at the keyboard to the screen. We used the following code,

ifdstream/ifdstream.h

```
1 | #ifndef EX28_IFDSTREAM_H
2 | #define EX28_IFDSTREAM_H
3 |
4 | #include <istream>
5 | #include "../ifdstreambuf/ifdstreambuf.h"
6 |
7 | class IFdStream: private IFdStreambuf, public std::istream
8 | {
9 |     public:
10 |         explicit IFdStream(int fd);
11 | };
12 |
13 | #endif
```

ifdstream/ifdstream.ih

```
1 | #include "ifdstream.h"
2 |
3 | using namespace std;
```

ifdstream/ifdstream.cc

```
1 | #include "ifdstream.ih"
2 |
3 | IFdStream::IFdStream(int fd)
4 | :
5 |     IFdStreambuf(fd),
6 |     istream(this)
7 | {}
```

ofdstream/ofdstream.h

```
1 | #ifndef EX28_OFDSTREAM_H
2 | #define EX28_OFDSTREAM_H
3 |
4 | #include <ostream>
5 | #include "../ofdstreambuf/ofdstreambuf.h"
6 |
7 | class OFdStream: private OFdStreambuf, public std::ostream
8 | {
9 |     public:
10 |         explicit OFdStream(int fd);
11 | };
12 |
13 | #endif
```

ofdstream/ofdstream.ih

```
1 | #include "ofdstream.h"
2 |
3 | using namespace std;
```

ofdstream/ofdstream.cc

```
1 | #include "ofdstream.ih"
2 |
3 | OFdStream::OFdStream(int fd)
4 | :
5 |     OFdStreambuf(fd),
6 |     ostream(this)
7 | {}
```

main.cc

```
1 | #include "ofdstream/ofdstream.h"
2 | #include "ifdstream/ifdstream.h"
3 |
4 | int main()
5 | {
```

```
6 |   IFdStream in{ 0 };      // keyboard
7 |   OFdStream out{ 1 };    // screen
8 |
9 |   std::string str;
10 |   while (getline(in, str))
11 |       out << str << std::endl;
12 | }
```

Exercise 29

Learn to implement a base class that can be used when applying the Template Method Design Pattern

We used the following code,

fork.h

```
1  #ifndef INCLUDED_FORK_
2  #define INCLUDED_FORK_
3
4  #include <unistd.h>
5
6  class Fork
7  {
8      pid_t d_pid;
9
10     public:
11         Fork() = default;
12         Fork(Fork const &other) = delete;
13         Fork(Fork &&tmp) = delete;
14         virtual ~Fork();
15
16         void fork();
17
18     protected:
19         pid_t pid() const;
20         int waitForChild() const;
21
22     private:
23         virtual void parentProcess() = 0;
24         virtual void childProcess() = 0;
25 };
26
27 inline pid_t Fork::pid() const
28 {
29     return d_pid;
30 }
31
32 #endif
```

fork.ih

```
1 | #include "fork.h"
2 |
3 | #include <sys/types.h>
4 | #include <sys/wait.h>
5 | #include <stdlib.h>
```

destructor.cc

```
1 | #include "fork.ih"
2 |
3 | Fork::~~Fork()
4 | {}
```

fork.cc

```
1 | #include "fork.ih"
2 |
3 | void Fork::fork()
4 | {
5 |     if ((d_pid = ::fork()) < 0)
6 |         throw "fork failed";
7 |     else if (d_pid == 0 )
8 |     {
9 |         childProcess();
10 |        exit(1);
11 |    }
12 |    else
13 |        parentProcess();
14 | }
```

waitforchild.cc

```
1 | #include "fork.ih"
2 |
3 | int Fork::waitForChild() const
4 | {
5 |     int status;
```



```
6 |  
7 |     waitpid(d_pid, &status, 0);  
8 |  
9 |     return WEXITSTATUS(status);  
10 | }
```