

Programming in C/C++

Exercises set three: polymorphism

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

November 30, 2016

Exercise 15, construct ostream class

We were tasked to construct an ostream class with our own buffer. The program should work correctly with the syntax in the question.

Code listings

Listing 1: main.ih

```
1 #include "bistream.h"
2 #include "bistreambuffer.h"
3
4 #include <iostream>
5 #include <fstream>
```

Listing 2: main.cc

```
1 #include "main.ih"
2
3 int main()
4 {
5     std::ofstream one("one");
6     std::ofstream two("two");
7
8     BiStream ms(one, two);
9
10    ms << "Hello world" << std::endl << std::flush;
```

```
11 }
```

Listing 3: bistream.h

```
1 #ifndef BISTREAM_H
2 #define BISTREAM_H
3
4 #include "main.ih"
5 class BiStream: public std::ostream
6 {
7     public:
8         BiStream(std::ofstream &one, std::ofstream &two);
9         ~BiStream();
10 };
11
12 #endif
```

Listing 4: bistreambuffer.h

```
1 #ifndef BISTREAMBUFFER_H
2 #define BISTREAMBUFFER_H
3
4 #include "main.ih"
5
6 class BiStreamBuffer: public std::streambuf
7 {
8     std::ostream *d_stream1, *d_stream2;
9
10     public:
11         BiStreamBuffer(std::ofstream &one,
12             std::ofstream &two);
13         std::streamsize xspn(const char* s,
14             std::streamsize n) override;
15 };
16
17 #endif
```

Listing 5: bufferconstructor.cc

```
1 #include "main.ih"
2
3 BiStreamBuffer::BiStreamBuffer(std::ofstream &one,
```

```

4     std::ofstream &two)
5 :
6     d_stream1 (&one),
7     d_stream2 (&two)
8 {
9 }

```

Listing 6: streamconstructor.cc

```

1  #include "main.ih"
2
3  BiStream::BiStream(std::ofstream &one,
4      std::ofstream &two)
5  :
6      std::ostream(new BiStreamBuffer(one, two))
7  {
8  }

```

Listing 7: streamdestructor.cc

```

1  #include "main.ih"
2
3  BiStream::~~BiStream()
4  {
5      delete this->rdbuf();
6  }

```

Listing 8: xspn.cc

```

1  #include "main.ih"
2
3  std::streamsize BiStreamBuffer::xspn(const char* s,
4      std::streamsize n)
5  {
6      *d_stream1 << s;
7      *d_stream2 << s;
8      return 0;
9  }

```

Exercise 16, design streambuf

We were tasked to design a streambuf class that is called IFdStreamBuff that allows extractions from a FD.

Code listings

Listing 9: ifdstreambuf.ih

```
1 #include "ifdstreambuf.h"
2 #include <unistd.h>
3 #include <memory.h>
4
5 using namespace std;
```

Listing 10: mode.h

```
1 #ifndef MODE_H_
2 #define MODE_H_
3
4 enum Mode
5 {
6     KEEP_FD,
7     CLOSE_FD
8 };
9
10 #endif
```

Listing 11: ifdstreambuf.h

```
1 #ifndef IFDSTREAMBUF_H
2 #define IFDSTREAMBUF_H
3
4 #include "mode.h"
5 #include <streambuf>
6
7 class IFdStreambuf: public std::streambuf
8 {
9     int d_FD;
10    Mode d_mode;
11    std::size_t bufferSize = 100;
12    char buffer[100] {0};
13    size_t place = 0;
14
15    protected:
16        int underflow() override;
17        int uflow() override;
```

```

18     std::streamsize xsgetn(char* s, std::streamsize n)
        override;
19
20     public:
21         explicit IFdStreambuf(Mode mode = KEEP_FD);
22         explicit IFdStreambuf(int FD, Mode mode =
KEEP_FD);
23         ~IFdStreambuf();
24         void close(int FD);
25         void open(int FD, Mode mode = KEEP_FD);
26 };
27
28 #endif

```

Listing 12: close.cc

```

1  #include "ifdstreambuf.ih"
2
3  void IFdStreambuf::close(int FD)
4  {
5      ::close(FD);
6      // code for setting mode to CLOSE_FD here
7  }

```

Listing 13: cnstr1.cc

```

1  #include "ifdstreambuf.ih"
2
3  IFdStreambuf::IFdStreambuf(Mode mode)
4  :
5      d_mode(mode)
6  {
7  }

```

Listing 14: cnstr2.cc

```

1  #include "ifdstreambuf.ih"
2
3  IFdStreambuf::IFdStreambuf(int FD, Mode mode)
4  :
5      d_FD(FD),
6      d_mode(mode)

```

```

7 {
8     read(FD, buffer, bufferSize * sizeof(char));
9 }

```

Listing 15: destructor.cc

```

1 #include "ifdstreambuf.ih"
2
3 IFdStreambuf::~~IFdStreambuf()
4 {
5     if (d_mode == CLOSE_FD)
6         close(d_FD);
7 }

```

Listing 16: 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     read(FD, buffer, bufferSize * sizeof(char));
8 }

```

Listing 17: uflow.cc

```

1 #include "ifdstreambuf.ih"
2
3 int IFdStreambuf::uflow()
4 {
5     char output[1] = {0};
6     if (place < bufferSize)
7     {
8         *output = *(buffer + place);
9         ++place;
10    }
11    else
12    {
13        read(d_FD, output, 1 * sizeof(char));
14    }
15    return *output;
16 }

```

Listing 18: undflow.cc

```
1 #include "ifdstreambuf.ih"
2
3 int IFdStreambuf::underflow()
4 {
5     if (place < bufferSize)
6     {
7         return *(buffer + place);
8     }
9     return EOF;
10 }
```

Listing 19: xsgetn.cc

```
1 #include "ifdstreambuf.ih"
2
3 std::streamsize IFdStreambuf::xsgetn(char* s, std::
    streamsize n)
4 {
5     int size = bufferSize;
6     if (n <= size)
7         memcpy(s, buffer, n * sizeof(char));
8     else
9     {
10         memcpy(s, buffer, bufferSize * sizeof(char));
11         read(d_FD, s + bufferSize, (n - bufferSize) *
            sizeof(char));
12     }
13     return n;
14 }
```

Exercise 17, design streambuf 2

We were tasked to design the OFdStreamBuff that allows insertions to a FD.

Code listings

Listing 20: ofdstreambuf.ih

```
1 #include "ofdstreambuf.h"
2 #include <unistd.h>
```

```

3 #include <memory.h>
4
5 using namespace std;

```

Listing 21: ofdstreambuf.h

```

1 #ifndef OFDSTREAMBUF_H
2 #define OFDSTREAMBUF_H
3
4 #include "mode.h"
5 #include <streambuf>
6
7 class OFdStreambuf: public std::streambuf
8 {
9     int d_FD;
10    Mode d_mode;
11    size_t bufferSize = 200;
12    char buffer[200];
13    size_t place = 0;
14
15    private:
16        int sync() override;
17
18    protected:
19        int pSync();
20
21    public:
22        explicit OFdStreambuf(Mode mode = KEEP_FD);
23        explicit OFdStreambuf(int FD,
24                               Mode mode = KEEP_FD);
25        ~OFdStreambuf();
26        void close(int FD);
27        void open(int FD, Mode mode = KEEP_FD);
28        std::streamsize xsputn(char const *s,
29                                std::streamsize n) override;
30 };
31
32 #endif

```

Listing 22: close.cc

```

1 #include "ofdstreambuf.ih"

```



```

2
3 void OFdStreambuf::close(int FD)
4 {
5     ::close(FD);
6     // code for setting mode to CLOSE_FD here
7 }

```

Listing 23: cnstr1.cc

```

1 #include "ofdstreambuf.ih"
2
3 OFdStreambuf::OFdStreambuf(Mode mode)
4 :
5     d_mode(mode)
6 {
7 }

```

Listing 24: cnstr2.cc

```

1 #include "ofdstreambuf.ih"
2
3 OFdStreambuf::OFdStreambuf(int FD, Mode mode)
4 :
5     d_FD(FD),
6     d_mode(mode)
7 {
8 }

```

Listing 25: destructor.cc

```

1 #include "ofdstreambuf.ih"
2
3 OFdStreambuf::~~OFdStreambuf()
4 {
5     if (d_mode == CLOSE_FD)
6         close(d_FD);
7 }

```

Listing 26: 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     //read(FD, buffer, 100);
8 }

```

Listing 27: psync.cc

```

1 #include "ofdstreambuf.ih"
2
3 int OFdStreambuf::pSync()
4 {
5     return sync();
6 }

```

Listing 28: sync.cc

```

1 #include "ofdstreambuf.ih"
2
3 int OFdStreambuf::sync()
4 {
5     write(d_FD, buffer, place * sizeof(char));
6     place = 0;
7
8     return 0;
9 }

```

Listing 29: xsputn.cc

```

1 #include "ofdstreambuf.ih"
2
3 std::streamsize OFdStreambuf::xsputn(char const *s,
4     std::streamsize n)
5 {
6     int bound = bufferSize - place;
7     if (n <= bound)
8     {
9         memcpy(buffer + place, s, n);
10        place += n;
11        return n;
12    }
13    else

```

```

14     sync();
15
16     size_t remaining = n;
17     while (remaining > bufferSize)
18     {
19         write(d_FD, s, bufferSize * sizeof(char));
20         s += bufferSize;
21         remaining -= bufferSize;
22     }
23     memcpy(buffer, s, remaining * sizeof(char));
24     place = remaining;
25     return n;
26 }

```

Exercise 18: FD streams

Here is the code for the streams of the corresponding FD buffers from exercises 16 and 17. The code in main echoes back whatever is typed into console.

Code listings

Listing 30: main.h

```

1 #include "ifdstream.h"
2 #include "ofdstream.h"

```

Listing 31: main.cc

```

1 #include "main.h"
2 #include <string>
3
4 int main(int argc, char **argv)
5 {
6     IFdStream in(0);
7     OFdStream out(1);
8
9     std::string variable;
10    in >> variable;
11    out << variable << '\n' << std::flush;
12 }

```

iFdStreambuf

Listing 32: ifdstream.h

```
1 #ifndef IFDSTREAM_H
2 #define IFDSTREAM_H
3
4 #include <iostream>
5 #include "ifdstreambuf.h"
6
7 class IFdStream: public std::istream
8 {
9     public:
10         explicit IFdStream(int FD);
11         ~IFdStream();
12 };
13
14 #endif
```

Listing 33: istreamconstr.cc

```
1 #include "main.h"
2
3 IFdStream::IFdStream(int FD)
4 :
5     std::istream(new IFdStreambuf(FD))
6 {
7 }
```

Listing 34: ostreamconstr.cc

```
1 #include "main.h"
2
3 IFdStream::~~IFdStream()
4 {
5     delete this->rdbuf();
6 }
```

oFdStreambuf

Listing 35: ofdstream.h

```
1 #ifndef OFDSTREAM_H
2 #define OFDSTREAM_H
3
```

```

4 #include <iostream>
5 #include "ofdstreambuf.h"
6
7 class OFdStream: public std::ostream
8 {
9     public:
10         explicit OFdStream(int FD);
11         ~OFdStream();
12 };
13
14 #endif

```

Listing 36: istreamdestr.cc

```

1 #include "main.h"
2
3 OFdStream::OFdStream(int FD)
4 :
5     std::ostream(new OFdStreambuf(FD))
6 {
7 }

```

Listing 37: ostreamdestr.cc

```

1 #include "main.h"
2
3 OFdStream::~~OFdStream()
4 {
5     delete this->rdbuf();
6 }

```

Exercise 19: Forks

We were tasked with making an abstract `Fork`. Its derived classes are able to fork themselves by calling the member `fork()`. A tester class was made to check if the forking works.

Sample output

```

1 Parent process 26432 here!
2 BEEP

```

```
3 Child process 26433 here!
4 BOOP
```

Code listings

Listing 38: main.cc

```
1 #include "fork/fork.h"
2
3 int main(int argc, char **argv)
4 {
5     Tester test;
6     test.fork();
7 }
```

Listing 39: fork.ih

```
1 #include "fork.h"
2
3 #include <iostream>
```

Listing 40: fork.h

```
1 #ifndef FORK_H_
2 #define FORK_H_
3
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/wait.h>
7
8 class Fork
9 {
10     pid_t d_pid = 0;
11
12     public:
13         void fork();
14         virtual ~Fork();
15     protected:
16         pid_t pid();
17         int waitForChild() const;
18     private:
19         virtual void parentProcess() = 0;
```

```

20     virtual void childProcess() = 0;
21 };
22
23 class Tester: public Fork
24 {
25     public:
26         Tester() = default;
27         ~Tester() override;
28         Tester(Tester const &other) = delete;
29         void operator=(Tester const &other) = delete;
30     private:
31         void parentProcess() override;
32         void childProcess() override;
33 };
34
35 #endif

```

Listing 41: childprocesstester.cc

```

1  #include "fork.ih"
2
3  void Tester::childProcess()
4  {
5      std::cout << "Child process " << getpid()
6          << " here!\nBOOP\n";
7  }

```

Listing 42: forkdestructor.cc

```

1  #include "fork.ih"
2
3  Fork::~Fork()
4  {
5  }

```

Listing 43: parentprocesstester.cc

```

1  #include "fork.ih"
2
3  void Tester::parentProcess()
4  {
5      std::cout << "Parent process " << getpid()

```

```
6     << " here!\nBEEP\n";
7 }
```

Listing 44: pid.cc

```
1 #include "fork.ih"
2
3 pid_t Fork::pid()
4 {
5     return d_pid;
6 }
```

Listing 45: testdestructor.cc

```
1 #include "fork.ih"
2
3 Tester::~Tester()
4 {
5 }
```

Listing 46: 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 }
```