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 ofstream class with our own buffer. The program should work correctly with the syntaxis in the question.

Code listings

Listing 1: main.ih

```
#include "bistream.h"
#include "bistreambuffer.h"

#include <iostream>
#include <fstream>
```

Listing 2: main.cc

```
#include "main.ih"

int main()
{
  std::ofstream one("one");
    std::ofstream two("two");

    BiStream ms(one, two);

    ms << "Hello world" << std::endl << std::flush;
}
```

Listing 3: bistream.h

```
#ifndef BISTREAM_H
#define BISTREAM_H

#include "main.ih"
class BiStream: public std::ostream
{
  public:
    BiStream(std::ofstream &one, std::ofstream &two);
    ~BiStream();
};

#endif
```

Listing 4: bistreambuffer.h

```
#ifndef BISTREAMBUFFER_H
#define BISTREAMBUFFER_H

#include "main.ih"

class BiStreamBuffer: public std::streambuf
{
  std::ostream *d_stream1, *d_stream2;

  public:
    BiStreamBuffer(std::ofstream &one,
      std::ofstream &two);
    std::streamsize xsputn(const char* s,
      std::streamsize n) override;
};

#endif
```

Listing 5: bufferconstructor.cc

```
#include "main.ih"

BiStreamBuffer::BiStreamBuffer(std::ofstream &one,
  std::ofstream &two)
:
  d_stream1(&one),
  d_stream2(&two)
{
}
```

Listing 6: streamconstructor.cc

```
#include "main.ih"

BiStream::BiStream(std::ofstream &one,
  std::ofstream &two)
:
  std::ostream(new BiStreamBuffer(one, two))
{
}
```

Listing 7: streamdestructor.cc

```
#include "main.ih"

BiStream::~BiStream()
{
  delete this->rdbuf();
}
```

Listing 8: xsputn.cc

```
#include "main.ih"

std::streamsize BiStreamBuffer::xsputn(const char* s,
  std::streamsize n)
{
  *d_stream1 << s;
  *d_stream2 << s;
  return 0;
}
```

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

```
#include "ifdstreambuf.h"
#include <unistd.h>
#include <memory.h>

using namespace std;
```

Listing 10: mode.h

```
#ifndef FDBUFFERMODE_H_
#define FDBUFFERMODE_H_

enum FDBufferMode
{
    KEEP_FD,
    CLOSE_FD
};

#endif
```

Listing 11: ifdstreambuf.h

```
#ifndef IFDSTREAMBUF_H
#define IFDSTREAMBUF_H

#include "fdbuffermode.h"
#include <streambuf>

class IFdStreambuf: public std::streambuf
{
  int d_FD;
  FDBufferMode d_mode;
  std::size_t d_bufferSize = 100;
  char *d_buffer = new char[100];

  protected:
    explicit IFdStreambuf(
      FDBufferMode mode = KEEP_FD);
        explicit IFdStreambuf(int FD,
      FDBufferMode mode = KEEP_FD);
    int underflow() override;
    int uflow() override;
    std::streamsize xsgetn(char* buffer,
      std::streamsize size) override;

    public:
        ~IFdStreambuf();
        int close(int FD);
        void open(int FD,
      FDBufferMode mode = KEEP_FD);
};

#endif
```

Listing 12: close.cc

```
#include "ifdstreambuf.ih"

int IFdStreambuf::close(int FD)
{
    return ::close(FD);
}
```

Listing 13: cnstr1.cc

```
#include "ifdstreambuf.ih"

IFdStreambuf::IFdStreambuf(FDBufferMode mode)
:
  d_mode(mode)
{
  setg(d_buffer, d_buffer + d_bufferSize,
    d_buffer + d_bufferSize);
}
```

Listing 14: cnstr2.cc

```
#include "ifdstreambuf.ih"

IFdStreambuf::IFdStreambuf(int FD, FDBufferMode mode)
:
  d_FD(FD),
  d_mode(mode)
{
  setg(d_buffer, d_buffer + d_bufferSize,
    d_buffer + d_bufferSize);
}
```

Listing 15: destructor.cc

```
#include "ifdstreambuf.ih"

IFdStreambuf::~IFdStreambuf()
{
  delete[] d_buffer;
    if (d_mode == CLOSE_FD)
    close(d_FD);
}
```

Listing 16: open.cc

```
#include "ifdstreambuf.ih"

void IFdStreambuf::open(int FD, FDBufferMode mode)
{
    d_FD = FD;
    d_mode = mode;
}
```

Listing 17: uflow.cc

```
#include "ifdstreambuf.ih"

int IFdStreambuf::uflow()
{
  int UFlowChar = underflow();
  setg(eback(), eback(), egptr());
  return UFlowChar;
}
```

Listing 18: underflow.cc

```
#include "ifdstreambuf.ih"

```

```
int IFdStreambuf::underflow()
{
  if (!read(d_FD, d_buffer,
    d_bufferSize * sizeof(char)))

    return EOF;

  return *eback();
}
```

Listing 19: xsgetn.cc

```
#include "ifdstreambuf.ih"

std::streamsize IFdStreambuf::xsgetn(char* buffer,
  std::streamsize size)
{
  size_t remaining = egptr() - gptr();
  if (size <= remaining)
  {
    memcpy(buffer, d_buffer,
      remaining * sizeof(char));
    read(d_FD, buffer + remaining,
      (size - remaining) * sizeof(char));
    read(d_FD, eback(), d_bufferSize * sizeof(char));
    setg(eback(), eback(), egptr());
  }
  else
  {
    memcpy(buffer, d_buffer, size * sizeof(char));
    gbump(size);
  }
  return size;
}
```

Exercise 17, design streambuf 2

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

Code listings

Listing 20: ofdstreambuf.ih

```
#include "ofdstreambuf.h"
#include <unistd.h>
#include <memory.h>

using namespace std;
```

Listing 21: ofdstreambuf.h

```
#ifndef OFDSTREAMBUF_H
#define OFDSTREAMBUF_H

#include "fdbuffermode.h"
#include <streambuf>

class OFdStreambuf: public std::streambuf
{
  int d_FD;
  FDBufferMode d_mode;
  size_t d_bufferSize = 100;
  char *d_buffer = new char[100];

  private:
    int sync() override;

  protected:
    explicit OFdStreambuf(
      FDBufferMode mode = KEEP_FD);
        explicit OFdStreambuf(int FD,
      FDBufferMode mode = KEEP_FD);
    int pSync();

    public:
        ~OFdStreambuf();
        int close(int FD);
        void open(int FD,
      FDBufferMode mode = KEEP_FD);
        std::streamsize xsputn(char const *buffer,
      std::streamsize size) override;
        int overflow(int character = EOF) override;
};

#endif
```

Listing 22: close.cc

```
#include "ofdstreambuf.ih"

int OFdStreambuf::close(int FD)
{
    return ::close(FD);
}
```

Listing 23: cnstr1.cc

```
#include "ofdstreambuf.ih"

OFdStreambuf::OFdStreambuf(FDBufferMode mode)
:
  d_mode(mode)
{
  setp(d_buffer, d_buffer + d_bufferSize);
}
```

Listing 24: cnstr2.cc

```
#include "ofdstreambuf.ih"

OFdStreambuf::OFdStreambuf(int FD, FDBufferMode mode)
:
  d_FD(FD),
  d_mode(mode)
{
  setp(d_buffer, d_buffer + d_bufferSize);
}
```

Listing 25: destructor.cc

```
#include "ofdstreambuf.ih"

OFdStreambuf::~OFdStreambuf()
{
  delete[] d_buffer;
```

```
    if (d_mode == CLOSE_FD)
    close(d_FD);
}
```

Listing 26: open.cc

```
#include "ofdstreambuf.ih"

void OFdStreambuf::open(int FD, FDBufferMode mode)
{
    d_FD = FD;
    d_mode = mode;
}
```

Listing 27: overflow.cc

```
#include "ofdstreambuf.ih"

int OFdStreambuf::overflow(int character)
{
    sync();
    char castChar = character;
  xsputn(&castChar, 1);
  return character;
}
```

Listing 28: psync.cc

```
#include "ofdstreambuf.ih"

int OFdStreambuf::pSync()
{
  return sync();
}
```

Listing 29: sync.cc

```
#include "ofdstreambuf.ih"

int OFdStreambuf::sync()
{
  size_t remaining = epptr() - pptr();
  if (!write(d_FD, d_buffer,
    remaining * sizeof(char)))

    return -1;

  setp(pbase(), epptr());

  return 0;
}
```

Listing 30: xsputn.cc

```
#include "ofdstreambuf.ih"

streamsize OFdStreambuf::xsputn(char const *buffer,
  streamsize size)
{
  int remaining = epptr() - pptr();
  if (size <= remaining)
  {
    memcpy(pptr(), buffer, size * sizeof(char));
    pbump(size);

    if (size == remaining)
      sync();

    return size;
  }
  sync();

  if (!write(d_FD, buffer, size))
    return 0;

  return size;
}
```

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 31: main.h

```
#include "ifdstream.h"
#include "ofdstream.h"
```

Listing 32: main.cc

```
#include "main.h"
#include <string>

int main(int argc, char **argv)
{
  IFdStream in(0);
  OFdStream out(1);

  std::string variable;
  in >> variable;
  out << variable << '\n' << std::flush;
}
```

iFdStreambuf

Listing 33: ifdstream.h

```
#ifndef IFDSTREAM_H
#define IFDSTREAM_H

#include <iostream>
#include "ifdstreambuf.h"

class IFdStream: public std::istream
{
  public:
    explicit IFdStream(int FD);
    ~IFdStream();
};

#endif
```

Listing 34: istreamconstr.cc

```
#include "main.h"
```

```

IFdStream::IFdStream(int FD)
:
  std::istream(new IFdStreambuf(FD))
{
}
```

Listing 35: ostreamconstr.cc

```
#include "main.h"

IFdStream::~IFdStream()
{
  delete this->rdbuf();
}
```

oFdStreambuf

Listing 36: ofdstream.h

```
#ifndef OFDSTREAM_H
#define OFDSTREAM_H

#include <iostream>
#include "ofdstreambuf.h"

class OFdStream: public std::ostream
{
  public:
    explicit OFdStream(int FD);
    ~OFdStream();
};

#endif
```

Listing 37: istreamdestr.cc

```
#include "main.h"

OFdStream::OFdStream(int FD)
:
  std::ostream(new OFdStreambuf(FD))
```

```
{
}
```

Listing 38: ostreamdestr.cc

```
#include "main.h"

OFdStream::˜OFdStream()
{
  delete this->rdbuf();
}
```

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

```
Parent process 26432 here!
BEEP
Child process 26433 here!
BOOP
```

Code listings

Listing 39: main.cc

```
#include "fork/fork.h"

int main(int argc, char **argv)
{
  Tester test;
  test.fork();
}
```

Listing 40: fork.ih

```
#include "fork.h"

#include <iostream>
```

Listing 41: fork.h

```
#ifndef FORK_H_
#define FORK_H_

#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

class Fork
{
  pid_t d_pid = 0;

  public:
    void fork();
    virtual ~Fork();
  protected:
    pid_t pid();
    int waitForChild() const;
  private:
    virtual void parentProcess() = 0;
    virtual void childProcess() = 0;
};

class Tester: public Fork
{
  public:
    Tester() = default;
    ~Tester() override;
    Tester(Tester const &other) = delete;
    void operator=(Tester const &other) = delete;
  private:
    void parentProcess() override;
    void childProcess() override;
};

#endif
```

Listing 42: childprocesstester.cc

```
#include "fork.ih"
```

```

void Tester::childProcess()
{
  std::cout << "Child process " << getpid()
    << " here!\nBOOP\n";
}
```

Listing 43: forkdestructor.cc

```
#include "fork.ih"

Fork::~Fork()
{
}
```

Listing 44: parentprocesstester.cc

```
#include "fork.ih"

void Tester::parentProcess()
{
  std::cout << "Parent process " << getpid()
    << " here!\nBEEP\n";
}
```

Listing 45: pid.cc

```
#include "fork.ih"

pid_t Fork::pid()
{
  return d_pid;
}
```

Listing 46: testdestructor.cc

```
#include "fork.ih"

Tester::~Tester()
{
}
```

Listing 47: waitforchild.cc

```
#include "fork.ih"

int Fork::waitForChild() const
{
  int status;

  waitpid(d_pid, &status, 0);

  return WEXITSTATUS(status);
}
```