13

Pencil, please?

Previous attempt graded by JB.

Exercises week 4: Polymorphism - Revision

Klaas Isaac Bijlsma s2394480

David Vroom s2309939

December 20, 2017

Exercise 25 1

Learn to construct an `ostream` *class*

Before we passed the wrong type to the BiStream constructor. Now we passed two ostream-objects to its constructor. Furthermore the data members of BiStreamBuffer were implemented as pointers, this is now changed to reference to ostream-objects. By value isn't possible, because the copy constructor of ostream isn't available.

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

bistream/bistream.h

```
#ifndef INCLUDED_BISTREAM_
#define INCLUDED_BISTREAM_

#include <fstream>
#include "../bistreambuffer/bistreambuffer.h"

class BiStream: private BiStreamBuffer, public std::ostream
{
    public:
        BiStream(std::ostream &one, std::ostream &two);
};
```

```

#endif
```

bistream/bistream.ih

```
#include "bistream.h"

using namespace std;
```

bistream/bistream1.cc

```
#include "bistream.ih"

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

bistreambuffer/bistreambuffer.h

```
#ifndef INCLUDED_BISTREAMBUFFER_
#define INCLUDED_BISTREAMBUFFER_

#include <streambuf>

class BiStreamBuffer: public std::streambuf
{
    std::ostream &d_one;
    std::ostream &d_two;

    public:
       BiStreamBuffer(std::ostream &one, std::ostream &two);

    private:
        int overflow(int c) override;
};

```

```
#endif
```

bistreambuffer/bistreambuffer.ih

```
#include "bistreambuffer.h"
#include <ostream>

using namespace std;
```

bistreambuffer/bistreambuffer1.cc

```
#include "bistreambuffer.ih"

BiStreamBuffer::BiStreamBuffer(std::ostream &one, std::ostream &two)
:
    d_one(one),
    d_two(two)
{}
```

bistreambuffer/overflow.cc

```
#include "bistreambuffer.ih"

int BiStreamBuffer::overflow(int c)
{
    if (c == EOF)
    {
        d_one.flush();
        d_two.flush();
    }
    else
    {
        d_one.put(c);
        d_two.put(c);
    }
    return c;
}
```

Exercise 26 ?

Learn to design a streambuf reading from file descriptors

We solved the magic number 100 and prevented DRY in the constructor. Also we improved the member function `open()`.

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

```
#ifndef EX26_IFDSTREAMBUF_H
#define EX26_IFDSTREAMBUF_H

#include <streambuf>

class IFdStreambuf: public std::streambuf
{
    public:
        enum
        {
            BUFSIZE = 100
        };

        enum Mode
        {
            KEEP_FD,
            CLOSE_FD
        };

    protected:
        int d_fd;
        Mode d_mode;
        char *d_buffer = nullptr;

    public:
        explicit IFdStreambuf(Mode mode = KEEP_FD);           // 1
        explicit IFdStreambuf(int fd, Mode mode = KEEP_FD); // 2
        virtual ~IFdStreambuf();
```

Needs not be public.

Never deallocated

Copy ctor left in

```
        int close();
        void open(int fd, Mode mode = KEEP_FD);

    private:
        int underflow() override;
        std::streamsize xsgetn(char *dest, std::streamsize n) override;
};

#endif
```

ifdstreambuf.ih

```
#include "ifdstreambuf.h"
#include <unistd.h>         // read(), close()
#include <string.h>         // memcpy()

using namespace std;
```

close.cc

```
#include "ifdstreambuf.ih"

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

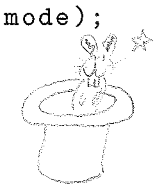
destructor.cc

```
#include "ifdstreambuf.ih"

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

is a boolean?

ifdstreambuf1.cc

```
#include "ifdstreambuf.ih"

IFdStreambuf::IFdStreambuf(Mode mode)
{
    open(-1, mode);
}
```



ifdstreambuf2.cc

```
#include "ifdstreambuf.ih"

IFdStreambuf::IFdStreambuf(int fd, Mode mode)
{
    open(fd, mode);  — Some checks?
    setg(0, 0, 0);           // buffer is initially empty
}
```

open.cc

```
#include "ifdstreambuf.ih"

void IFdStreambuf::open(int fd, Mode mode)
{
    if (d_buffer and d_mode == CLOSE_FD)
        close();

    d_fd = fd;
    d_mode = mode;

    if (d_fd != -1)
    {
        delete[] d_buffer;
        d_buffer = new char[BUFSIZE];
    }
}
```

underflow.cc

```
#include "ifdstreambuf.ih"

int IFdStreambuf::underflow()
{
    if (gptr() < egptr())
        return *gptr();

    int nRead = read(d_fd, d_buffer, BUFSIZE);

    if (nRead <= 0)
        return EOF;

    setg(d_buffer, d_buffer, d_buffer + nRead);
    return static_cast<unsigned char>(*gptr());
}
```

xsgetn.cc

```
#include "ifdstreambuf.ih"

streamsize IFdStreambuf::xsgetn(char *dest, streamsize n)
{
    if (n == 0)
        return 0;

    int nBuffer = in_avail(); // number of retrievable chars in buffer

    if (nBuffer > n)          // more chars in buffer than requested
        nBuffer = n;
                              // copy what's available in own buffer
    memcpy(dest, gptr(), nBuffer);
    gbump(nBuffer);           // update pointer
                              // try to read some more from FD
    int nFile = read(d_fd, dest + nBuffer, n - nBuffer);

    return nBuffer + nFile;
}
```

Exercise 27

?

Learn to design a streambuf writing to file descriptors

We solved the magic number 100 and prevented DRY in the constructor. Also we improved the member function `open()`.

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

```
#ifndef EX27_OFDSTREAMBUF_H
#define EX27_OFDSTREAMBUF_H

#include <streambuf>

class OFdStreambuf: public std::streambuf
{
    public:
        enum                  need not be public
        {
            BUFSIZE = 100
        };

        enum Mode
        {
            KEEP_FD,
            CLOSE_FD
        };

    protected:
        int d_fd;
        Mode d_mode;
        char *d_buffer = nullptr;

    public:
        explicit OFdStreambuf(Mode mode = KEEP_FD);           // 1
        explicit OFdStreambuf(int fd, Mode mode = KEEP_FD); // 2
        virtual ~OFdStreambuf();
        int close();
```

Copy ctor still there

```
        void open(int fd, Mode mode = KEEP_FD);

    private:
        int sync() override;
        int overflow(int c) override;
};

#endif
```

ofdstreambuf.ih

```
#include "ofdstreambuf.h"
#include <unistd.h>         // read(), close()

using namespace std;
```

close.cc

```
#include "ofdstreambuf.ih"

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

destructor.cc

```
#include "ofdstreambuf.ih"

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

    if (d_mode)
        close();
}
```

ofdstreambuf1.cc

```
#include "ofdstreambuf.ih"

OFdStreambuf::OFdStreambuf(Mode mode)
{
    open(-1, mode);
}
```

[illegible]

ofdstreambuf2.cc

```
#include "ofdstreambuf.ih"

OFdStreambuf::OFdStreambuf(int fd, Mode mode)
{
    open(fd, mode);
    setp(d_buffer, d_buffer + BUFSIZE);
}
```

open.cc

```
#include "ofdstreambuf.ih"

void OFdStreambuf::open(int fd, Mode mode)
{
    if (d_buffer)
    {
        sync();
        if (d_mode == CLOSE_FD)
            close();
    }

    d_fd = fd;
    d_mode = mode;

    if (d_fd != -1)
    {
        delete[] d_buffer;
        d_buffer = new char[BUFSIZE];
    }
}
```

overflow.cc

```
#include "ofdstreambuf.ih"

int OFdStreambuf::overflow(int c)
{
    sync();
    if (c != EOF)
    {
        *pptr() = c; // or static_cast<char>(c);
        pbump(1);
    }
    return c;
}
```

sync.cc

```
#include "ofdstreambuf.ih"

int OFdStreambuf::sync()
{
    if (pptr() > pbase())
    {
        write(d_fd, d_buffer, pptr() - pbase());
        setp(d_buffer, d_buffer + BUFSIZE);
    }
    return 0;
}
```

Exercise 28

7

Learn to design streams

We have changed the while loop in the main into an eternal loop with a break if EOF is reached. Our solution is based on the solution of exercise 13 of part I.

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

```
#ifndef EX28_IFDSTREAM_H
#define EX28_IFDSTREAM_H

#include <istream>
#include "../ifdstreambuf/ifdstreambuf.h"

class IFdStream: private IFdStreambuf, public std::istream
{
    public:
        explicit IFdStream(int fd);
};

#endif
```

ifdstream/ifdstream.ih

```
#include "ifdstream.h"

using namespace std;
```

ifdstream/ifdstream.cc

```
#include "ifdstream.ih"

IFdStream::IFdStream(int fd)
:
    IFdStreambuf(fd),
    istream(this)
{}
```

ofdstream/ofdstream.h

```
#ifndef EX28_OFDSTREAM_H
#define EX28_OFDSTREAM_H

#include <ostream>
#include "../ofdstreambuf/ofdstreambuf.h"

class OFdStream: private OFdStreambuf,  public std::ostream
{
    public:
        explicit OFdStream(int fd);
};

#endif
```

ofdstream/ofdstream.ih

```
#include "ofdstream.h"

using namespace std;
```

ofdstream/ofdstream.cc

```
#include "ofdstream.ih"

OFdStream::OFdStream(int fd)
:
    OFdStreambuf(fd),
    ostream(this)
{}
```

main.cc

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

using namespace std;

int main()
{
    IFdStream in{ 0 };        // keyboard
    OFdStream out{ 1 };       // screen

    string str;
    while (true)
    {
        getline(in, str);
        if (not in)
            break;
        out << str << endl;
    }
}
```

TC