C++ Exercises
Demo submitinfo.txt

Author(s): Frank Brokken

–

12:05

November 13, 2023

1

```
Demo showing the organization of an exercise for the mailhandler.
Note that not all the implementations of members/constructors in the class
Strings are provided: that's not necessary if the exercise doesn't ask for it.

This demo also doesn't use an inline function implementation file: it could
have been used, in which case the order.txt file would start like this:

    descr.txt
    strings/strings.h
    strings/strings.i
    strings/strings.ih
    ....

The pdf constructed from this demo is strings.pdf and can be downloaded
from the same location as the current zip file.
```

Listing 1: strings/strings.h

```
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <iosfwd>

class Strings
{
    size_t d_size = 0;
    std::string *d_str = 0;

    std::string *(Strings::*d_ptr)();       // either enlargeByCopy
                                            // or enlargeByMove
    size_t d_nIterations;

    public:
        struct POD
        {
            size_t      size;
            std::string *str;
        };

        Strings();
        Strings(int argc, char **argv);
        Strings(char **environLike);
        Strings(std::istream &in);

        Strings(size_t nIterate, bool copy = true);

        void swap(Strings &other);

        size_t size() const;
        std::string const *data() const;
        POD release();
```

```
        std::string const &at(size_t idx) const;     // for const-objects
        std::string &at(size_t idx);                 // for non-const objects

        void add(std::string const &next);           // add another element

        void iterate(char **environLike);

    private:
        void fill(char **ntbs);                      // fill prepared d_str

        std::string &safeAt(size_t idx) const;       // private backdoor

        std::string *enlargeByCopy();
        std::string *enlargeByMove();

        void destroy();

        static size_t count(char **environLike);     // # elements in env.like

};

inline size_t Strings::size() const
{
    return d_size;
}

inline std::string const *Strings::data() const
{
    return d_str;
}

inline std::string const &Strings::at(size_t idx) const
{
    return safeAt(idx);
}

inline std::string &Strings::at(size_t idx)
{
    return safeAt(idx);
}


#endif
```

Listing 2: strings/strings.ih

```
#include "strings.h"

#include <istream>
#include <string>

using namespace std;
```

Listing 3: strings/strings5.cc

```
#include "strings.ih"

Strings::Strings(size_t nIterate, bool copy)
:
    d_ptr(copy ? &Strings::enlargeByCopy : &Strings::enlargeByMove),
    d_nIterations(nIterate)
{}
```

Listing 4: strings/add.cc

```
#include "strings.ih"

void Strings::add(string const &next)
{
```

```
    string *tmp = (this->*d_ptr)();      // make room for the next string,
                                         // tmp is the new string *
                                         // either use copying or moving

    tmp[d_size] = next;                  // store next

    destroy();                           // return old memory

    d_str = tmp;                         // update d_str and d_size
    ++d_size;
}
```

Listing 5: strings/enlargebymove.cc

```
#include "strings.ih"

string *Strings::enlargeByMove()
{
    string *ret = new string[d_size + 1];       // room for an extra string

    for (size_t idx = 0; idx != d_size; ++idx)  // move existing strings
        ret[idx] = move(d_str[idx]);

    return ret;
}
```

Listing 6: strings/safeat.cc

```
#include "strings.ih"

namespace {
    string n_empty;
}

std::string &Strings::safeAt(size_t idx) const
{
    if (idx >= d_size)
    {
        n_empty.clear();
        return n_empty;
    }

    return d_str[idx];
}
```

Listing 7: strings/destroy.cc

```
#include "strings.ih"

void Strings::destroy()
{
    delete[] d_str;
}
```

Listing 8: strings/enlargebycopy.cc

```
#include "strings.ih"

string *Strings::enlargeByCopy()
{
    string *ret = new string[d_size + 1];       // room for an extra string

    for (size_t idx = 0; idx != d_size; ++idx)  // copy existing strings
        ret[idx] = d_str[idx];

    return ret;
}
```

Listing 9: strings/iterate.cc

```
#include "strings.ih"

void Strings::iterate(char **environLike)
{
    for (; d_nIterations--; )
    {
        destroy();

        d_size = 0;
        d_str = 0;

        char **ptr = environLike;
        while (*ptr)
            add(*ptr++);
    }
}
```

Listing 10: main.ih

```
#include <iostream>
#include <string>

#include "strings/strings.h"

extern char **environ;

using namespace std;
```

Listing 11: main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
    if (argc == 1)
    {
        cout << "Use arg1: #iterations, arg2: any arg to move, else copy\n";
        return 0;
    }

    Strings str(stoul(argv[1]), argc == 2); // n iterations, copy if 2 args

    str.iterate(environ);

    if (str.size() != 0)                    // available strings...
        cout << "First element: " <<str.at(0) << "\n"
                "Last element:  " <<str.at(str.size() - 1) << '\n';
}
```