C++ Exercises
Set 6

Author(s): Olivier Gelling, Leon Lan, Mohammad Habibi
–
13:28

October 20, 2025

44

```
Exercise 44:

Change Strings class to use double pointers and be more memory efficient.

We amend the class to use an array of raw memory for double pointers which in
turn get initialised with new string objects. We also added a setNull function
which sets all raw pointer locations to null for safety, albeit knowing that
is not exactly necessary as long as d_size correctly tracks the size of the
array.

Our class doubles in size every time it reaches its capacity, checking after
adding a new string, rather than before, as it should not matter when this is
done since either the capacity will be enlarged right after it is reached, or
it will be enlarged just before it is exceeded.

We are not sure whether swap is still needed but we have amended it so it can
be used with double pointers if necessary.
```

Listing 1: strings/strings.h

```
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <iosfwd>

class Strings
{
    size_t d_size;
    size_t d_capacity = 1;
    std::string **d_data = nullptr;

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

        ~Strings();                         // Added destructor.

        void swap(Strings &other);

        size_t size() const;
        std::string const *const *data() const;

        std::string const &at(size_t idx) const;
        std::string &at(size_t idx);

        void resize(size_t const newSize);
        void reserve(size_t const newCapacity);

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

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

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

        void enlarge(size_t const newSize);
        std::string **rawPointers(size_t const newSize);
        void copyPtrsInto(std::string **rawMemory);
        void setNull(size_t const from, size_t const to);
        void initialiseStrings(size_t const newSize);

        void destroy();
        void destroyStrings(size_t const cutoff);

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

inline size_t Strings::size() const         // potentially dangerous practice:
{                                           // inline accessors
    return d_size;
}

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

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

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

inline size_t Strings::capacity() const
{
    return d_capacity;
}

#endif
```

Listing 2: strings/strings.ih

```
#include "strings.h"

#include <istream>
#include <string>

using namespace std;
```

Listing 3: strings/strings1.cc

```
#include "strings.ih"

Strings::Strings()
:
    d_size(0),
    d_data(rawPointers(1))
{
    setNull(0, d_capacity); // Not really needed I think?
}
```

Listing 4: strings/strings2.cc

```
#include "strings.ih"

Strings::Strings(size_t argc, char **argv)
:
    d_size(argc),
    d_capacity(argc ? argc : 1), // argc is never gonna be 0 I think?
    d_data(rawPointers(d_capacity))
{
    setNull(0, d_capacity);
    fill(argv);
}
```

Listing 5: strings/strings3.cc

```
#include "strings.ih"

Strings::Strings(char **environLike)
:
    d_size(count(environLike)),
    d_data(rawPointers(d_size))
{
    fill(environLike);
}
```

Listing 6: strings/strings4.cc

```
#include "strings.ih"

Strings::Strings(istream &in)
:
    d_size(0),
    d_data(rawPointers(1))
{
    string line;
    while (getline(in, line))
        add(line);
}
```

Listing 7: strings/strings5.cc

```
#include "strings.ih"

Strings::~Strings()
{
    destroy();
}
```

Listing 8: strings/swap.cc

```
#include "strings.ih"

void Strings::swap(Strings &other)
{
    string **tmp = d_data;
    d_data = other.d_data;
    other.d_data = tmp;

    size_t size = d_size;
    d_size = other.d_size;
    other.d_size = size;

    size_t tmpCapacity = d_capacity;
    d_capacity = other.d_capacity;
    other.d_capacity = tmpCapacity;
}
```

Listing 9: strings/resize.cc

```
#include "strings.ih"

void Strings::resize(size_t newSize)
{
    if (newSize < d_size)
        destroyStrings(newSize);
    else if (newSize > d_size)
    {
        reserve(newSize);
        initialiseStrings(newSize);
    }
}
```

Listing 10: `strings/reserve.cc`

```
#include "strings.ih"

    // by

void Strings::reserve(size_t const newCapacity)
{
    if (newCapacity > d_capacity)
        enlarge(newCapacity);
}
```

Listing 11: `strings/add.cc`

```
#include "strings.ih"

    // by strings4.cc

void Strings::add(string const &next)
{
    if (d_size == d_capacity)
        enlarge(d_capacity * 2);

    d_data[d_size++] = new string{ next };
}
```

Listing 12: `strings/fill.cc`

```
#include "strings.ih"

void Strings::fill(char **ntbs)
{
    for (size_t index = 0; index != d_size; ++index)
        d_data[index] = new string(ntbs[index]);
}
```

Listing 13: `strings/safeat.cc`

```
#include "strings.ih"

namespace {
    string emptyString;
}

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

    return *d_data[idx];
}
```

Listing 14: strings/enlarge.cc

```
#include "strings.ih"

    // by add.cc & reserve.cc

void Strings::enlarge(size_t const newSize)
{
    string **stringPtrArray = rawPointers(newSize);
    copyPtrsInto(stringPtrArray);

    operator delete(d_data);
    d_data = stringPtrArray;

    setNull(d_capacity, newSize);       // set pointers to nullptr

    d_capacity = newSize;
}
```

Listing 15: strings/rawPointers.cc

```
#include "strings.ih"

    // by enlarge.cc & strings1.cc & strings2.cc & strings3.cc & strings4.cc

string **Strings::rawPointers(size_t const newCapacity)
{
    return static_cast<string **>(
                    operator new(newCapacity * sizeof(string *)));
}
```

Listing 16: strings/copyPtrsInto.cc

```
#include "strings.ih"

    // by enlarge.cc

void Strings::copyPtrsInto(string **rawMemory)
{
    for (size_t index = 0; index != d_size; ++index)
        rawMemory[index] = d_data[index];
}
```

Listing 17: strings/setNull.cc

```
#include "strings.ih"

    // by enlarge.cc

void Strings::setNull(size_t const from, size_t const to)
{
    for (size_t index = from; index != to; ++index)
        d_data[index] = nullptr;
}
```

Listing 18: strings/initialisestrings.cc

```
#include "strings.ih"

    // by

void Strings::initialiseStrings(size_t const newSize)
{
    for (size_t index = d_size; index != newSize; ++index)
        d_data[index] = new string;

    d_size = newSize;
}
```

Listing 19: strings/destroy.cc

```
#include "strings.ih"

    // by strings5.cc

void Strings::destroy()
{
    for (size_t index = 0; index != d_size; ++index)
        delete d_data[index];

    operator delete(d_data);
}
```

Listing 20: strings/destroystrings.cc

```
#include "strings.ih"

    // by resize.cc

void Strings::destroyStrings(size_t const cutoff)
{
    for (size_t index = d_size; index-- != cutoff; )
        delete d_data[index];

    setNull(cutoff, d_size);
    d_size = cutoff;
}
```

Listing 21: strings/count.cc

```
#include "strings.ih"

// static
size_t Strings::count(char **environLike)
{
    char **ptr = environLike;

    while (*ptr++)        // find the 0-pointer
        ;

                          // ptr just passed beyond the 0-ptr
    return  (ptr - 1) - environLike;
}
```

45

```
Exercise 45:

We re-amend the Strings class to use a block of raw memory (not double
pointers) that gets initialised by placement new when new data is needed. All
the data is copied to a new block once the capacity is reached.

This is similar in structure to the previous exercise, and also has overlap
with 41.
```

Listing 22: strings/strings.h

```
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <iosfwd>

class Strings
{
    size_t d_size;
    size_t d_capacity = 1;
    std::string *d_data = nullptr;
    public:
        Strings();
        Strings(size_t argc, char **argv);
        Strings(char **environLike);
        Strings(std::istream &in);

        ~Strings();                             // Added destructor.

        void swap(Strings &other);

        size_t size() const;
        std::string const *data() const;

        std::string const &at(size_t idx) const;
        std::string &at(size_t idx);

        void resize(size_t const newSize);
        void reserve(size_t const newCapacity);

        size_t capacity() const;

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

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

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

        void enlarge(size_t const newSize);
        std::string *rawStrings(size_t const newSize);
        void copyStringsInto(std::string *stringArray);

        void destroy();
        void destroyPart(size_t const newSize);

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

inline size_t Strings::size() const             // potentially dangerous practice:
{                                               // inline accessors
    return d_size;
}

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

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

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

inline size_t Strings::capacity() const
{
    return d_capacity;
}

#endif
```

Listing 23: strings/strings.ih

```
#include "strings.h"

#include <istream>
#include <string>
```

```
using namespace std;
```

Listing 24: strings/strings1.cc

```
#include "strings.ih"

Strings::Strings()
:
    d_size(0),
    d_data(rawStrings(1))
{}
```

Listing 25: strings/strings2.cc

```
#include "strings.ih"

Strings::Strings(size_t argc, char **argv)
:
    d_size(argc),
    d_capacity(argc),
    d_data(rawStrings(argc))
{
    fill(argv);
}
```

Listing 26: strings/strings3.cc

```
#include "strings.ih"

Strings::Strings(char **environLike)
:
    d_size(count(environLike)),
    d_data(rawStrings(d_size))
{
    fill(environLike);
}
```

Listing 27: strings/strings4.cc

```
#include "strings.ih"

Strings::Strings(istream &in)
:
    d_size(0),
    d_data(rawStrings(1))
{
    string line;
    while (getline(in, line))
        add(line);
}
```

Listing 28: strings/strings5.cc

```
#include "strings.ih"

Strings::~Strings()
{
    destroy();
}
```

Listing 29: strings/swap.cc

```
#include "strings.ih"

void Strings::swap(Strings &other)
{
    string *tmp = d_data;
    d_data = other.d_data;
    other.d_data = tmp;

    size_t size = d_size;
    d_size = other.d_size;
    other.d_size = size;

    size_t tmpCapacity = d_capacity;
    d_capacity = other.d_capacity;
    other.d_capacity = tmpCapacity;
}
```

Listing 30: strings/resize.cc

```
#include "strings.ih"

void Strings::resize(size_t newSize)
{
    if (newSize < d_size)
        destroyPart(newSize);
    else if (newSize > d_size)
        reserve(newSize);
    d_size = newSize;
}
```

Listing 31: strings/reserve.cc

```
#include "strings.ih"

    // by

void Strings::reserve(size_t const newCapacity)
{
    if (newCapacity > d_capacity)
        enlarge(newCapacity);
}
```

Listing 32: strings/add.cc

```
#include "strings.ih"

    // by strings4.cc

void Strings::add(string const &next)
{
    if (d_size == d_capacity)
        enlarge(d_capacity * 2);

    new (d_data + d_size) string{ next };
    d_size++;
}
```

Listing 33: strings/fill.cc

```
#include "strings.ih"

void Strings::fill(char **ntbs)
{
    for (size_t index = 0; index != d_size; ++index)
        new (d_data + index) string(ntbs[index]);
}
```

Listing 34: strings/safeat.cc

```
#include "strings.ih"

namespace {
    string emptyString;
}

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

    return d_data[idx];
}
```

Listing 35: `strings/enlarge.cc`

```
#include "strings.ih"

    // by add.cc & reserve.cc

void Strings::enlarge(size_t const newSize)
{
    string *stringArray = rawStrings(newSize);
    copyStringsInto(stringArray);

    destroy();

    d_data = stringArray;
    d_capacity = newSize;
}
```

Listing 36: `strings/rawStrings.cc`

```
#include "strings.ih"

    // by

string *Strings::rawStrings(size_t const nStrings)
{
    return static_cast<string *>(operator new(nStrings * sizeof(string)));
}
```

Listing 37: `strings/copyStringsInto.cc`

```
#include "strings.ih"

    // by enlarge.cc

void Strings::copyStringsInto(string *stringArray)
{
    for (size_t index = 0; index != d_size; ++index)
        new (stringArray + index) string(d_data[index]);
}
```

Listing 38: `strings/destroy.cc`

```
#include "strings.ih"

    // by strings5.cc

void Strings::destroy()
{
    for (size_t index = 0; index != d_size; ++index)
        d_data[index].~string();

    operator delete(d_data);
}
```

Listing 39: strings/destroypart.cc

```
#include "strings.ih"

    // by resize.cc

void Strings::destroyPart(size_t const newSize)
{
    for (size_t index = d_size; index-- != newSize; )
        d_data[index].~string();
}
```

Listing 40: strings/count.cc

```
#include "strings.ih"

// static
size_t Strings::count(char **environLike)
{
    char **ptr = environLike;

    while (*ptr++)       // find the 0-pointer
        ;

                         // ptr just passed beyond the 0-ptr
    return  (ptr - 1) - environLike;
}
```