Jaap van der Leest
Timon van der Berg
October 25, 2017

C++ Course
Assignment 6

Contents

Exercise 43

Problem statement. *Fix the memory leak in the 'Strings' class.*
Solution. Because our own implementation of 'Strings' was not perfect, we instead modified the official solution provided in the answers of set 5.

strings.h

```cpp
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <iosfwd>

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

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

        Strings();
        ~Strings();
        Strings(int argc, char *argv[]);
        Strings(char *environLike[]);
```

```
22          Strings(std::istream &in);
23
24          void swap(Strings &other);
25
26          size_t size() const;
27          std::string const *data() const;
28          POD release();
29
30          std::string const &at(size_t idx) const;  // for const-objects
31          std::string &at(size_t idx);               // for non-const objects
32
33          void add(std::string const &next);        // add another element
34
35      private:
36          void fill(char *ntbs[]);                   // fill prepared d_str
37
38          std::string &safeAt(size_t idx) const;     // private backdoor
39          std::string *enlarge();
40          void destroy();
41
42          static size_t count(char *environLike[]);  // # elements in env.like
43
44  };
45
46  inline size_t Strings::size() const               // potentially dangerous practice:
47  {                                                 // inline accessors
48      return d_size;
49  }
50
51  inline std::string const *Strings::data() const
52  {
53      return d_str;
54  }
55
56  inline std::string const &Strings::at(size_t idx) const
57  {
58      return safeAt(idx);
59  }
60
61  inline std::string &Strings::at(size_t idx)
62  {
63      return safeAt(idx);
64  }
65
66
67  #endif
```

strings5.cc

```
1  #include "strings.ih"            // using namespace std;
2
3  Strings::~Strings()
4  {
5      delete[] d_str;             // 0 pointer allowed
6  }
```

Exercise 44

Problem statement. *Use double pointers in Strings class.*

Solution. Because our own implementation of 'Strings' was not perfect, we instead modified the official solution provided in the answers of set 5.

strings.h

```cpp
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <iosfwd>

class Strings
{
    size_t              d_size;       // number of stored strings
    size_t      d_capacity = 1;       // number of strings that can be stored
    std::string    **d_arrayStr = nullptr;    // pointer to pointers of string data

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

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

        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

        size_t const capacity() const;                  // return allocated memory in nr of strings

        void reserve(size_t const newCapacity);         // reserves memory to new size

        void resize(size_t const newCapacity);          // resizes and initializes


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

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

        void destroy();

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



        std::string** rawPointers(size_t nrPointers); // creates initialized array of
                                                        // pointers to strings
};

inline size_t Strings::size() const                    // potentially dangerous practice:
{                                                       // inline accessors
```

4

```
60      return d_size;
61  }
62
63
64  inline std::string **const Strings::data() const
65  {
66      return d_arrayStr;
67  }
68
69  inline std::string const &Strings::at(size_t idx) const
70  {
71      return safeAt(idx);
72  }
73
74  inline std::string &Strings::at(size_t idx)
75  {
76      return safeAt(idx);
77  }
78
79
80  #endif
```

strings.ih

```
1  #include "strings.h"
2
3  #include <istream>
4  #include <string>
5
6  using namespace std;
```

add.cc

```
1  #include "strings.ih"
2
3  void Strings::add(string const &next)
4
5  {
6      string *strPointer = new string(next);  // store new string address in pointer
7
8      if (d_size >= d_capacity)
9      {
10          d_capacity <<= 1;                   // multiply d_capacity by 2
11
12          reserve(d_capacity);                // reserve memory for strings
13      }
14
15      d_arrayStr[d_size] = strPointer;        // store new pointer in array
16      ++d_size;
17  }
18
```

capacity.cc

```
1  #include "strings.ih"
2
3
4  size_t const Strings::capacity() const
5  {
6      return d_capacity;
7  }
```

count.cc

```
1   #include "strings.ih"
2
3   // static
4   size_t Strings::count(char *environLike[])
5   {
6       size_t nElements = 0;
7
8       while (*environLike++ != 0)      // visit all defined elements
9           ++nElements;                 // inc. counter if one's found
10
11      return nElements;
12  }
```

destroy.cc

```
1   #include "strings.ih"
2
3   void Strings::destroy()
4   {
5       delete d_arrayStr;  // delete the array of pointers but not what they point to
6   }
```

fill.cc

```
1   #include "strings.ih"
2
3   void Strings::fill(char *ntbs[])
4   {
5       for (size_t index = 0; index != d_size; ++index)
6           *d_arrayStr[index] = string(ntbs[index]);
7   }
```

rawpointers.cc

```
1   #include "strings.ih"
2
3   string** Strings::rawPointers(size_t nrPointers)
4   {
5       string **newArray = new string*[nrPointers];   // create new pointer
6       for (size_t idx = 0; idx != nrPointers; ++idx)  // initialize array with pointers to initialized strings
7           newArray[idx] = new string;
8
9       return newArray;
10  }
```

release.cc

```
1   #include "strings.ih"
2
3   Strings::POD Strings::release()
4   {
5       POD ret{d_size, d_arrayStr};       // initialize the POD for the caller
6
7       d_size = 0;
8       d_capacity = 1;
9       d_arrayStr = nullptr;
10
```

```
11        return ret;
12    }
```

reserve.cc

```
1    #include "strings.ih"
2
3
4    void Strings::reserve(size_t const newCapacity)
5    {
6        string **newArray = nullptr;
7
8        if (newCapacity >= d_capacity)
9        {
10            d_capacity = newCapacity;
11            newArray = new string*[d_capacity];                // not initialized
12
13            for (size_t idx = 0; idx != d_size; ++idx)         // copy old pointers
14                newArray[idx] = d_arrayStr[idx];
15        }
16
17        else if (newCapacity < d_size)                         // if new array is too small
18        {
19            newArray = new string*[d_capacity];                // not initialized
20
21            for (size_t idx = 0; idx != newCapacity; ++idx)    // copy old pointers
22                newArray[idx] = d_arrayStr[idx];
23            for (size_t idx = newCapacity; idx != d_size; ++idx) // delete pointers and
24                delete d_arrayStr[idx];                        // strings outside new array
25        }
26
27        else                        // if newCapacity is between d_size and d_capacity
28        {
29            newArray = new string*[d_capacity];                // not initialized
30
31            for (size_t idx = 0; idx != d_size; ++idx)         // copy old pointers
32                newArray[idx] = d_arrayStr[idx];
33        }
34
35        destroy();                        // delete old array of pointers, not the string data
36
37        d_arrayStr = newArray;
38    }
```

resize.cc

```
1    #include "strings.ih"
2
3
4    void Strings::resize(size_t const newCapacity)
5    {
6        string **newArray = nullptr;
7
8        if (newCapacity >= d_capacity)
9        {
10            d_capacity = newCapacity;
11            newArray = new string*[d_capacity];                // not initialized
12
13            for (size_t idx = 0; idx != d_size; ++idx)         // copy old pointers
14                newArray[idx] = d_arrayStr[idx];
15            for (size_t idx = d_size; idx != d_capacity; ++idx) // initialize the rest
16                newArray[idx] = new string;
17
```

```
18        }
19
20        else if (newCapacity < d_size)                          // if new array is too small
21        {
22            newArray = new string*[d_capacity];                 // not initialized
23
24            for (size_t idx = 0; idx != newCapacity; ++idx)      // copy old pointers
25                newArray[idx] = d_arrayStr[idx];
26            for (size_t idx = newCapacity; idx != d_size; ++idx) // delete pointers and
27                delete d_arrayStr[idx];                          // strings outside new array
28        }
29
30        else                        // if newCapacity is between d_size and d_capacity
31        {
32            newArray = new string*[d_capacity];                 // not initialized
33
34            for (size_t idx = 0; idx != d_size; ++idx)           // copy old pointers
35                newArray[idx] = d_arrayStr[idx];
36            for (size_t idx = d_size; idx != d_capacity; ++idx)  // initialize the rest
37                newArray[idx] = new string;
38        }
39
40        destroy();                          // delete old array of pointers, not the string data
41
42        d_arrayStr = newArray;
43    }
44
45
46    // the function seems a lot like the reserve function
47    // the difference is the initialisation of the string objects when enlarging
48    // this doesn't seem to make much sense but is according the exercise.
```

safeat.cc

```
1    #include "strings.ih"
2
3    namespace {
4        string empty;
5    }
6
7    std::string &Strings::safeAt(size_t idx) const
8    {
9        if (idx >= d_size)
10        {
11            empty.clear();
12            return empty;
13        }
14
15        return *(d_arrayStr[idx]);
16    }
```

strings1.cc

```
1    #include "strings.ih"
2
3    Strings::Strings()
4    :
5        d_size(0),
6        d_arrayStr(rawPointers(d_capacity))
7    {}
```

strings2.cc

```
1   #include "strings.ih"
2
3   Strings::Strings(int argc, char *argv[])
4   :
5       d_size(argc),
6       d_capacity(argc),
7       d_arrayStr(rawPointers(d_capacity))
8   {
9       fill(argv);          // fill the newly created array
10  }
```

strings3.cc

```
1   #include "strings.ih"
2
3   Strings::Strings(char *environLike[])
4   :
5       d_size(count(environLike)),
6       d_capacity(d_size),
7       d_arrayStr(rawPointers(d_capacity))
8
9   {
10      fill(environLike);        // fill the newly created array
11  }
```

strings4.cc

```
1   #include "strings.ih"
2
3   Strings::Strings(istream &in)
4   :
5       d_size(0),
6       d_arrayStr(new string *[d_capacity])
7
8   {
9       string line;
10
11      while (getline(in, line))
12          add(line);
13  }
```

strings5.cc

```
1   #include "strings.ih"                        // using namespace std;
2
3   Strings::~Strings()
4   {
5       for (size_t idx = 0; idx != d_size; ++idx) // delete all strings by calling
6           delete d_arrayStr[idx];                // their destructor and free their memory
7
8       delete(d_arrayStr);                        // delete pointer to array of pointers
9   }
```

swap.cc

```
1   #include "strings.ih"
2
3   void Strings::swap(Strings &other)
```

```
4    {
5        string **tmp = d_arrayStr;
6        d_arrayStr = other.d_arrayStr;
7        other.d_arrayStr = tmp;
8
9        size_t size = d_size;
10       d_size = other.d_size;
11       other.d_size = size;
12   }
```

filter.h

```
1    #ifndef INCLUDED_FILTER_
2    #define INCLUDED_FILTER_
3
4    #include <iosfwd>
5    #include "../strings/strings.h"
6
7    class Filter
8    {
9        Strings d_lines;
10
11       public:
12           Filter(std::istream &in);
13
14           void display() const;
15
16       private:
17
18           static bool empty(std::string const &str);
19
20           static size_t firstNonEmpty(size_t size, std::string **const str);
21           static size_t beyondLastNonEmpty(size_t size, std::string **const str);
22
23
24   };
25
26   #endif
```

filter.ih

```
1    #include "filter.h"
2
3    #include <iostream>
4
5    using namespace std;
```

display.cc

```
1    #include "filter.ih"
2
3    void Filter::display() const
4    {
5        size_t size = d_lines.size();           // get number and contents
6        string **const str = d_lines.data();
7
8                                                // print fm first non empty through
9                                                // last non empty
10       for (size_t index = firstNonEmpty(size, str),
11               end = beyondLastNonEmpty(size, str);
12               index != end;
```

```
13              ++index
14      )
15          cout << *str[index] << '\n';
16
17  }
```

empty.cc

```
1   #include "filter.ih"
2
3   // static
4   bool Filter::empty(string const &str)
5   {
6       // find_first_not_of(" \t") returns index -> not empty,
7       //  so:
8       // find_first_not_of(" \t") != npos       -> not empty
9       //  so:
10      // find_first_not_of(" \t") == npos       -> empty
11
12      return str.find_first_not_of(" \t") == string::npos;
13  }
```

filter1.cc

```
1   #include "filter.ih"
2
3   Filter::Filter(istream &in)
4   :
5       d_lines(in)
6   {}
```

firstnonempty.cc

```
1   #include "filter.ih"
2
3   // static
4   size_t Filter::firstNonEmpty(size_t size, string **const str)
5   {
6       size_t idx = 0;
7                                       // skip initial empty lines
8       while (idx != size && empty(*str[idx]))
9           ++idx;
10
11      return idx;
12  }
```

lastnonempty.cc

```
1   #include "filter.ih"
2
3   // static
4   size_t Filter::beyondLastNonEmpty(size_t size, string **const str)
5   {
6       size_t idx = size;
7                                       // skip all empty lines at the end
8       while (idx-- && empty(*str[idx]))
9           ;
10
11      return idx + 1;                 // idx at the last non-empty line,
12  }                                   // but we must be beyond
```

Exercise 45

Problem statement. *Something something something*
Solution.

strings.h

```
1   #ifndef INCLUDED_STRINGS_
2   #define INCLUDED_STRINGS_
3
4   #include <iosfwd>
5
6   class Strings
7   {
8       size_t d_size;
9       size_t d_capacity = 1;
10      std::string *d_str;
11
12      public:
13          struct POD
14          {
15              size_t      size;
16              std::string *str;
17          };
18
19          Strings();
20          ~Strings();
21          Strings(int argc, char *argv[]);
22          Strings(char *environLike[]);
23          Strings(std::istream &in);
24
25          void swap(Strings &other);
26
27          size_t size() const;
28          std::string const *data() const;
29          POD release();
30
31          std::string const &at(size_t idx) const;    // for const-objects
32          std::string &at(size_t idx);                // for non-const objects
33
34          void add(std::string const &next);          // add another element
35
36          size_t const capacity() const;              // return allocated memory in nr of strings
37
38      private:
39          void fill(char *ntbs[]);                    // fill prepared d_str
40
41          std::string &safeAt(size_t idx) const;      // private backdoor
42
43          void destroy();                             // frees memory
44
45          static size_t count(char *environLike[]);   // # elements in env.like
46
47          std::string* rawStrings(size_t nrPointers); // allocates memory for strings
48
49          void reserve(size_t const newCapacity);     // reserve memory
50
51          void resize(size_t const newCapacity);      // resize capacity
52
53  };
54
55  inline size_t Strings::size() const        // potentially dangerous practice:
56  {                                          // inline accessors
57      return d_size;
```

```
58  }
59
60  inline std::string const *Strings::data() const
61  {
62      return d_str;
63  }
64
65  inline std::string const &Strings::at(size_t idx) const
66  {
67      return safeAt(idx);
68  }
69
70  inline std::string &Strings::at(size_t idx)
71  {
72      return safeAt(idx);
73  }
74
75
76  #endif
```

strings.ih

```
1  #include "strings.h"
2  #include <iostream>
3  #include <string>
4  #include <memory>  // placement new
5  using namespace std;
```

add.cc

```
1   #include "strings.ih"
2
3   void Strings::add(string const &next)
4   {
5       if (d_size == d_capacity)
6       {
7           d_capacity <<= 1;          // multiply d_capacity by 2
8
9           reserve(d_capacity);       // reserve memory for strings
10      }
11
12      d_str[d_size] = string(next);  // store new pointer in array
13
14      ++d_size;
15  }
```

capacity.cc

```
1  #include "strings.ih"
2
3
4  size_t const Strings::capacity() const
5  {
6      return d_capacity;
7  }
```

count.cc

```
1  #include "strings.ih"
2
```

```
3    // static
4    size_t Strings::count(char *environLike[])
5    {
6        size_t nElements = 0;
7
8        while (*environLike++ != 0)      // visit all defined elements
9            ++nElements;                 // inc. counter if one's found
10
11       return nElements;
12   }
```

destroy.cc

```
1    #include "strings.ih"
2
3    void Strings::destroy()
4    {
5        operator delete[](d_str);   // frees the allocated memory but doesn't
6                                    // delete the strings
7    }
```

fill.cc

```
1    #include "strings.ih"
2
3    void Strings::fill(char *ntbs[])
4    {
5        for (size_t index = 0; index != d_size; ++index)
6            d_str[index] = ntbs[index];
7    }
```

rawpointers.cc

```
1    #include "strings.ih"
2
3    string* Strings::rawStrings(size_t nrPointers)
4    {
5        string *newArray = static_cast<string *>(operator new[](nrPointers * sizeof(string)));
6
7        return newArray;
8    }
```

release.cc

```
1    #include "strings.ih"
2
3    Strings::POD Strings::release()
4    {
5        POD ret{ d_size, d_str };       // initialize the POD for the caller
6
7        d_size = 0;                     // reinitialize our data members
8        d_str = 0;
9
10       return ret;
11   }
```

reserve.cc

```
1   #include "strings.ih"
2
3
4   void Strings::reserve(size_t const newCapacity)
5   {
6       string *newArray = nullptr;
7
8       if (newCapacity >= d_capacity)
9       {
10
11          d_capacity = newCapacity;
12
13          newArray = rawStrings(d_capacity);                // not initialized
14
15          for (size_t idx = 0; idx != d_size; ++idx)        // copy old pointers
16              newArray[idx] = d_str[idx];
17
18      }
19
20      else if (newCapacity < d_size)                        // if new array is too small
21      {
22          newArray = rawStrings(d_capacity);                // not initialized
23
24          for (size_t idx = 0; idx != newCapacity; ++idx)   // copy old pointers
25              newArray[idx] = d_str[idx];
26          for (size_t idx = newCapacity; idx != d_size; ++idx) // delete pointers and
27              d_str[idx].~string();                         // strings outside new array
28      }
29
30      else                        // if newCapacity is between d_size and d_capacity
31      {
32          newArray = rawStrings(d_capacity);                // not initialized
33
34          for (size_t idx = 0; idx != d_size; ++idx)        // copy old pointers
35              newArray[idx] = d_str[idx];
36      }
37
38      destroy();                      // delete old array of pointers, not the string data
39
40      d_str = newArray;
41  }
```

resize.cc

```
1   #include "strings.ih"
2
3
4   void Strings::resize(size_t const newCapacity)
5   {
6       string *newArray = nullptr;
7
8       if (newCapacity >= d_capacity)
9       {
10          d_capacity = newCapacity;
11          newArray = rawStrings(d_capacity);                // not initialized
12
13          for (size_t idx = 0; idx != d_size; ++idx)        // copy old pointers
14              newArray[idx] = d_str[idx];
15          for (size_t idx = d_size; idx != d_capacity; ++idx) // initialize the rest
16              newArray[idx] = string();
17
18      }
```

```
19
20      else if (newCapacity < d_size)                      // if new array is too small
21      {
22          newArray = rawStrings(d_capacity);              // not initialized
23
24          for (size_t idx = 0; idx != newCapacity; ++idx)     // copy old pointers
25              newArray[idx] = d_str[idx];
26          for (size_t idx = newCapacity; idx != d_size; ++idx) // delete pointers and
27              d_str[idx].~string();                       // strings outside new array
28      }
29
30      else                        // if newCapacity is between d_size and d_capacity
31      {
32          newArray = rawStrings(d_capacity);              // not initialized
33
34          for (size_t idx = 0; idx != d_size; ++idx)          // copy old pointers
35              newArray[idx] = d_str[idx];
36          for (size_t idx = d_size; idx != d_capacity; ++idx)  // initialize the rest
37              newArray[idx] = string();
38      }
39
40      destroy();                          // delete old array of pointers, not the string data
41
42      d_str = newArray;
43  }
44
45
46  // the function seems a lot like the reserve function
47  // the difference is the initialisation of the string objects when enlarging
48  // this doesn't seem to make much sense but is according the exercise.
```

safeat.cc

```
1   #include "strings.ih"
2
3   namespace {
4       string empty;
5   }
6
7   std::string &Strings::safeAt(size_t idx) const
8   {
9       if (idx >= d_size)
10      {
11          empty.clear();
12          return empty;
13      }
14
15      return d_str[idx];
16  }
```

strings1.cc

```
1   #include "strings.ih"
2
3   Strings::Strings()
4   :
5       d_size(0),
6       d_str(rawStrings(d_capacity))
7   {}
```

strings2.cc

```
1   #include "strings.ih"
2
3   Strings::Strings(int argc, char *argv[])
4   :
5       d_size(argc),
6       d_capacity(d_size),
7       d_str(rawStrings(d_capacity))
8   {
9       fill(argv);
10  }
```

strings3.cc

```
1   #include "strings.ih"
2
3   Strings::Strings(char *environLike[])
4   :
5       d_size(count(environLike)),
6       d_capacity(d_size),
7       d_str(rawStrings(d_capacity))
8   {
9       fill(environLike);
10  }
```

strings4.cc

```
1   #include "strings.ih"
2
3   Strings::Strings(istream &in)
4   :
5       d_size(0),
6       d_str(rawStrings(d_capacity))
7   {
8       string line;
9       while (getline(in, line))
10          add(line);
11  }
```

strings5.cc

```
1   #include "strings.ih"                          // using namespace std;
2
3   Strings::~Strings()
4   {
5       for (size_t idx = 0; idx != d_size; ++idx) // call all destructors of strings
6           d_str[idx].~string();
7
8       operator delete[](d_str);                  // delete allocated memory
9   }
```

swap.cc

```
1   #include "strings.ih"
2
3   void Strings::swap(Strings &other)
4   {
5       string *tmp = d_str;
6       d_str = other.d_str;
```

```
7          other.d_str = tmp;
8
9          size_t size = d_size;
10         d_size = other.d_size;
11         other.d_size = size;
12     }
```

filter.h

```
1    #ifndef INCLUDED_FILTER_
2    #define INCLUDED_FILTER_
3
4    #include <iosfwd>
5    #include "../strings/strings.h"
6
7    class Filter
8    {
9        Strings d_lines;
10
11       public:
12           Filter(std::istream &in);
13
14           void display() const;
15
16       private:
17
18           static bool empty(std::string const &str);
19
20           static size_t firstNonEmpty(size_t size, std::string const *str);
21           static size_t beyondLastNonEmpty(size_t size, std::string const *str);
22
23
24   };
25
26   #endif
```

filter.ih

```
1    #include "filter.h"
2
3    #include <iostream>
4
5    using namespace std;
```

display.cc

```
1    #include "filter.ih"
2
3    void Filter::display() const
4    {
5        size_t size = d_lines.size();          // get number and contents
6        string const *str = d_lines.data();
7
8                                                // print fm first non empty through
9                                                // last non empty
10       for (size_t index = firstNonEmpty(size, str),
11                end = beyondLastNonEmpty(size, str);
12            index != end;
13                ++index
14       )
15           cout << str[index] << '\n';
```

```
16
17    }
```

empty.cc

```
1     #include "filter.ih"
2
3     // static
4     bool Filter::empty(string const &str)
5     {
6         // find_first_not_of(" \t") returns index -> not empty,
7         //  so:
8         //    // find_first_not_of(" \t") != npos        -> not empty
9         //     // so:
10        //        // find_first_not_of(" \t") == npos        -> empty
11
12        return str.find_first_not_of(" \t") == string::npos;
13    }
```

filter1.cc

```
1     #include "filter.ih"
2
3     Filter::Filter(istream &in)
4     :
5         d_lines(in)
6     {}
```

firstnonempty.cc

```
1     #include "filter.ih"
2
3     // static
4     size_t Filter::firstNonEmpty(size_t size, string const *str)
5     {
6         size_t idx = 0;
7                                         // skip initial empty lines
8         while (idx != size && empty(str[idx]))
9             ++idx;
10
11        return idx;
12    }
```

lastnonempty.cc

```
1     #include "filter.ih"
2
3     // static
4     size_t Filter::beyondLastNonEmpty(size_t size, string const *str)
5     {
6         size_t idx = size;
7                                         // skip all empty lines at the end
8         while (idx-- && empty(str[idx]))
9             ;
10
11        return idx + 1;                 // idx at the last non-empty line,
12    }                                   // but we must be beyond
```

Exercise 46

Problem statement. *Write a speed test.*

Solution. In 46 there is a main program provided. Sadly it uses a pointer as argument to the add member function. This will not work. Add expects a string as argument, not a pointer.

So we have altered the stresstest to the following code:

main.cc

```cpp
#include "strings/strings.h"
#include <string>

using namespace std;

extern char **environ;

int main()
{
    for (size_t iter = 0; iter != 1000; ++iter)
    {
        Strings env(environ);

        for (size_t rept = 0; rept != 100; ++rept)
        {
            for (char *ptr = *environ; *ptr; ++ptr)
                env.add(to_string(*ptr));
        }
    }
}
```

Given the short time remaining there was no way to come up with a better solution.

Here are the results we got:

Original class as provided: Output: none Takes quite some time to process, about 20 sec.

Strings class according to exercise 44: output: none pretty fast. Done in less than a sec.

Strings class according to exercise 45: Output: file:

ouput.txt

```
*** Error in `./main': double free or corruption (out): 0x0000000000d2d990 ***
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x777e5)[0x7f028bcfb7e5]
/lib/x86_64-linux-gnu/libc.so.6(+0x8037a)[0x7f028bd0437a]
/lib/x86_64-linux-gnu/libc.so.6(cfree+0x4c)[0x7f028bd0853c]
/usr/lib/x86_64-linux-gnu/libstdc++.so.6(_ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEE9_M_mutateEmmPKcm+0xfd)[0x7f028c5a052d]
/usr/lib/x86_64-linux-gnu/libstdc++.so.6(_ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEE10_M_replaceEmmPKcm+0x19b)[0x7f028c5a100b]
./main[0x4021e5]
./main[0x401be0]
./main[0x40158a]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xf0)[0x7f028bca4830]
./main[0x401479]
======= Memory map: ========
00400000-00403000 r-xp 00000000 08:03 526492                             /home/fastjack/C++/GIT/set 6/46/45/main
00602000-00603000 r--p 00002000 08:03 526492                             /home/fastjack/C++/GIT/set 6/46/45/main
00603000-00604000 rw-p 00003000 08:03 526492                             /home/fastjack/C++/GIT/set 6/46/45/main
00d18000-00d4a000 rw-p 00000000 00:00 0                                  [heap]
7f0284000000-7f0284021000 rw-p 00000000 00:00 0
7f0284021000-7f0288000000 ---p 00000000 00:00 0
7f028b97b000-7f028ba83000 r-xp 00000000 08:03 2364182                    /lib/x86_64-linux-gnu/libm-2.23.so
7f028ba83000-7f028bc82000 ---p 00108000 08:03 2364182                    /lib/x86_64-linux-gnu/libm-2.23.so
7f028bc82000-7f028bc83000 r--p 00107000 08:03 2364182                    /lib/x86_64-linux-gnu/libm-2.23.so
7f028bc83000-7f028bc84000 rw-p 00108000 08:03 2364182                    /lib/x86_64-linux-gnu/libm-2.23.so
7f028bc84000-7f028be44000 r-xp 00000000 08:03 2364112                    /lib/x86_64-linux-gnu/libc-2.23.so
7f028be44000-7f028c044000 ---p 001c0000 08:03 2364112                    /lib/x86_64-linux-gnu/libc-2.23.so
7f028c044000-7f028c048000 r--p 001c0000 08:03 2364112                    /lib/x86_64-linux-gnu/libc-2.23.so
7f028c048000-7f028c04a000 rw-p 001c4000 08:03 2364112                    /lib/x86_64-linux-gnu/libc-2.23.so
7f028c04a000-7f028c04e000 rw-p 00000000 00:00 0
7f028c04e000-7f028c066000 r-xp 00000000 08:03 2364258                    /lib/x86_64-linux-gnu/libpthread-2.23.so
```

```
30    7f028c066000-7f028c265000 ---p 00018000 08:03 2364258              /lib/x86_64-linux-gnu/libpthread-2.23.so
31    7f028c265000-7f028c266000 r--p 00017000 08:03 2364258              /lib/x86_64-linux-gnu/libpthread-2.23.so
32    7f028c266000-7f028c267000 rw-p 00018000 08:03 2364258              /lib/x86_64-linux-gnu/libpthread-2.23.so
33    7f028c267000-7f028c26b000 rw-p 00000000 00:00 0
34    7f028c26b000-7f028c281000 r-xp 00000000 08:03 2364150              /lib/x86_64-linux-gnu/libgcc_s.so.1
35    7f028c281000-7f028c480000 ---p 00016000 08:03 2364150              /lib/x86_64-linux-gnu/libgcc_s.so.1
36    7f028c480000-7f028c481000 rw-p 00015000 08:03 2364150              /lib/x86_64-linux-gnu/libgcc_s.so.1
37    7f028c481000-7f028c5f3000 r-xp 00000000 08:03 1050185              /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21
38    7f028c5f3000-7f028c7f3000 ---p 00172000 08:03 1050185              /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21
39    7f028c7f3000-7f028c7fd000 r--p 00172000 08:03 1050185              /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21
40    7f028c7fd000-7f028c7ff000 rw-p 0017c000 08:03 1050185              /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21
41    7f028c7ff000-7f028c803000 rw-p 00000000 00:00 0
42    7f028c803000-7f028c829000 r-xp 00000000 08:03 2364084              /lib/x86_64-linux-gnu/ld-2.23.so
43    7f028ca04000-7f028ca0a000 rw-p 00000000 00:00 0
44    7f028ca25000-7f028ca28000 rw-p 00000000 00:00 0
45    7f028ca28000-7f028ca29000 r--p 00025000 08:03 2364084              /lib/x86_64-linux-gnu/ld-2.23.so
46    7f028ca29000-7f028ca2a000 rw-p 00026000 08:03 2364084              /lib/x86_64-linux-gnu/ld-2.23.so
47    7f028ca2a000-7f028ca2b000 rw-p 00000000 00:00 0
48    7ffffb0b6000-7ffffb0d7000 rw-p 00000000 00:00 0                    [stack]
49    7ffffb13d000-7ffffb13f000 r--p 00000000 00:00 0                    [vvar]
50    7ffffb13f000-7ffffb141000 r-xp 00000000 00:00 0                    [vdso]
51    ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0            [vsyscall]
52    Aborted (core dumped)
```

No oportunity to speed test.

Best result we had with 44. This implementation worked and was a lot faster than the original. Not sure what happened with 45 though. Maybe by doubling the amount of memory the program wants to claim more than is available at that time. We could see it doubles a few times the memory before giving the error as stated.

Exercise 47

Problem statement.   *Replace the switches in the 'CPU' class using function pointers.*
Solution.   Because our own implementation of CPU was imperfect, we used the official solutions for Exercise 31. Our modified header is found below, followed byany new or modified helper functions. Everything not shown is assumed to be the unchanged.

cpu.h

```cpp
1    #ifndef INCLUDED_CPU_
2    #define INCLUDED_CPU_
3
4    #include "../tokenizer/tokenizer.h"
5    #include "../memory/memory.h"
6    #include "../enums/enums.h"
7
8    class Memory;  //    Jaap: why this?
9
10   class CPU
11   {
12       enum
13       {
14           NREGISTERS = 5,                                // a..e at indices 0..4, respectively
15           LAST_REGISTER = NREGISTERS - 1
16       };
17
18       struct Operand
19       {
20           OperandType type;
21           int value;
22       };
23
24       Memory &d_memory;
25       Tokenizer d_tokenizer;
26
27       int d_register[NREGISTERS];
28
```

```
29      public:
30          CPU(Memory &memory);
31          void start();
32
33      private:
34          bool error();                                       // show 'syntax error', and prepare for the
35                                                              // next input line
36                                                              // return a value or a register's or
37                                                              // memory location's value
38
39          void stp();                                         // helpers for start
40          static void (CPU::*execute[])();
41          void errorwrap();
42
43          int dereference(Operand const &value);
44          static int (CPU::*readOperand[])(Operand const &value);
45          int valueReturn(Operand const &value);
46          int memoryReturn(Operand const &value);
47          int registerReturn(Operand const &value);
48
49          bool rvalue(Operand &lhs);                           // retrieve an rvalue operand
50          bool lvalue(Operand &lhs);                           // retrieve an lvalue operand
51
52                                                              // determine 2 operands, lhs must be an lvalue
53          bool operands(Operand &lhs, Operand &rhs);
54
55          bool twoOperands(Operand &lhs, int &lhsValue, int &rhsValue);
56
57                                                              // store a value in register or memory
58          void store(Operand const &lhs, int value);
59          void storeRegister(int place, int value);
60          void storeMemory(int place, int value);
61          static void (CPU::*storeValue[])(int place, int value);
62
63          void mov();                                         // assign a value
64          void add();                                         // add values
65          void sub();                                         // subtract values
66          void mul();                                         // multiply values
67          void div();                                         // divide values (remainder: last reg.)
68                                                              // div a b computes a /= b, last reg: %
69          void neg();                                         // negate a value
70          void dsp();                                         // display a value
71  };
72
73  #endif
```

dereference.cc

```
1   #include "cpu.ih"
2
3   int CPU::dereference(Operand const &value)
4   {
5       return (this->*readOperand[value.type])(value);
6   }
```

memoryreturn.cc

```
1   #include "cpu.ih"
2
3   int CPU::memoryReturn(Operand const &value)
4   {
5       return d_memory.load(value.value);
6   }
```

readoperand.cc

```cpp
#include "cpu.ih"

int (CPU::*CPU::readOperand[])(Operand const &value) =       // order as in enums.h
{
    nullptr,                                     // padding for syntax, will never be called
    &CPU::valueReturn,
    &CPU::registerReturn,
    &CPU::memoryReturn
};
```

registerreturn.cc

```cpp
#include "cpu.ih"

int CPU::registerReturn(Operand const &value)
{
    return d_register[value.value];
}
```

valuereturn.cc

```cpp
#include "cpu.ih"

int CPU::valueReturn(Operand const &value)
{
    return value.value;
}
```

start.cc

```cpp
#include "cpu.ih"

void CPU::start()
{
    while (true)
    {
        (this->*execute[d_tokenizer.opcode()])();
        d_tokenizer.reset();
    }
}
```

errorwrap.cc

```cpp
#include "cpu.ih"

void CPU::errorwrap()
{
    error();
}
```

execute.cc

```cpp
#include "cpu.ih"

```

```
3   void (CPU::*CPU::execute[])() =                      // order as in enums.h
4   {
5       &CPU::errorwrap,
6       &CPU::mov,
7       &CPU::add,
8       &CPU::sub,
9       &CPU::mul,
10      &CPU::div,
11      &CPU::neg,
12      &CPU::dsp,
13      &CPU::stp
14  };
```

stp.cc

```
1   #include "cpu.ih"
2
3   void CPU::stp() // seperate file, add to header
4   {
5   }
```

store.cc

```
1   #include "cpu.ih"
2
3   void CPU::store(Operand const &lhs, int value)
4   {
5       (this->*storeValue[lhs.type])(lhs.value, value);
6   }
7   void CPU::storeRegister(int place, int value)
8   {
9       d_register[place] = value;
10  }
11
12  void CPU::storeMemory(int place, int value)
13  {
14      d_memory.store(place, value);
15  }
16
17  void (CPU::*CPU::storeValue[])(int place, int value)
18  {
19      nullptr,
20      nullptr,                                // these should never be called
21      &CPU::storeRegister,
22      &CPU::storeMemory
23  };
```

storememory.cc

```
1   #include "cpu.ih"
2
3   void CPU::storeMemory(int place, int value)
4   {
5       d_memory.store(place, value);
6   }
```

storeregister.cc

```
1   #include "cpu.ih"
2
```

```
3   void CPU::storeRegister(int place, int value)
4   {
5       d_register[place] = value;
6   }
```

storevalue.cc

```
1   #include "cpu.ih"
2
3   void (CPU::*CPU::storeValue[])(int place, int value)
4   {
5       nullptr,
6       nullptr,                                    // these should never be called
7       &CPU::storeRegister,
8       &CPU::storeMemory
9   };
```

Exercise 48

Problem statement. *Design the CSV class header.*
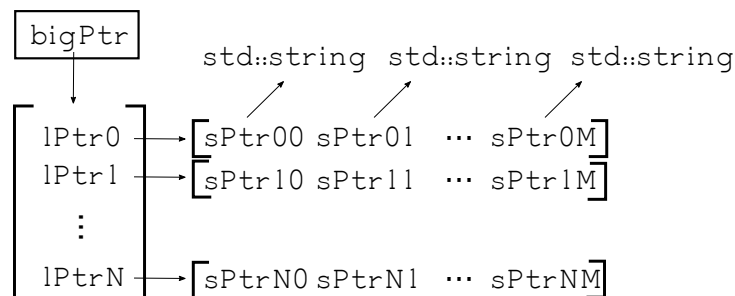**Solution.**

Data Model

Figure 1: `bigPtr` is a triple pointer. It points to an array of 'line pointers', each of these point to an array of `std::string` pointers representing the comma-seperated values. For example: using the notation above we have `bigPtr[1][1] = sPtr11` for the second value on the second line.

csv.h

```
1   #ifndef CSV_HEADER_H
2   #define CSV_HEADER_H
3
4   #include <string>                               // std::string
5   #include <istream>                              // std::istream
6
7   class CSV
8   {
9       size_t d_size = 1;                          // number of lines allocated
10      size_t d_nLines = 0;                        // number of lines read
11      size_t d_nFields = 1;                       // number of values per line
12      char d_fieldSep;                            // field seperator (default comma)
13
14      std::string ***bigPtr;                      // pointer to array of line pointers (see also big comment belo
15
16      public:
17          ~CSV();
18          CSV(size_t field, char fieldSep = ',');
19
```

```
20        std::string const *const *const *data() const;        // return pointer to data
21        std::string const &lastline()          const;         // ref last extraction
22
23        size_t nFields()                       const;         // values per line, set in first read
24        size_t size()                          const;         // number of currently stored lines
25
26        size_t read(std::istream &in, size_t nLines = 0);     // read lines using read1, return number read
27
28        std::string ***release();                             // return pointer to data, move responsibility for data
29                                                              // to called. Resets bigPtr but does not erase stored lines.
30        void clear(size_t nFields = 0);                       // erase everything
31    private:
32        bool read1(std::istream &in);                         // read 1 line, parse for CSV's, set nFields
33 };
34
35 #endif // CSV_HEADER_H
36
37 // Line pointers point to array of pointers
38 // to std::string. i.e. :
39 // bigPtr -> [Lptr0 Lptr1 ... LptrN]
40 // where Lptri -> [strPtri1 strPtri2 ... strPtriM] for i = 1,...,N
41 // where strPtrik -> std::string                   for k = 1,...,M/
42 // see also the figure in the report.
```

main.cc

```
1 #include "csv.h"
2
3 int main()
4 {
5     CSV file1(5, ',');
6 }
```