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

```
#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[]);
        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

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

        std::string &safeAt(size_t idx) const;    // private backdoor
        std::string *enlarge();
        void destroy();

        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_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);
}

```

```
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. *gi* **Solution.** go

Exercise 45

Problem statement. *gi* **Solution.** go

Exercise 46

Problem statement. *gi* **Solution.** go

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

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

dereference.cc

```
1   #include "cpu.ih"
2   int CPU::valueReturn(Operand const &value)
3   {
4       return value.value;
5   }
6
7   int CPU::registerReturn(Operand const &value)
8   {
9       return d_register[value.value];
10  }
11
12  int CPU::memoryReturn(Operand const &value)
```

```
{
    return d_memory.load(value.value);
}

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
};

int CPU::dereference(Operand const &value)
{
     return (this->*readOperand[value.type])(value);
}
```

start.cc

```
#include "cpu.ih"

void (CPU::*CPU::execute[])() =                 // order as in enums.h
{                                               // seperate file, add to header
    &CPU::errorwrap,            // bool
    &CPU::mov,
    &CPU::add,
    &CPU::sub,
    &CPU::mul,
    &CPU::div,
    &CPU::neg,
    &CPU::dsp,
    &CPU::stp
};

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

void CPU::stp() // seperate file, add to header
{
}

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

```

store.cc

```
#include "cpu.ih"

void CPU::store(Operand const &lhs, int value)
{
    (this->*storeValue[lhs.type])(lhs.value, value);
}
void CPU::storeRegister(int place, int value)
{
```

```
    d_register[place] = value;
}

void CPU::storeMemory(int place, int value)
{
    d_memory.store(place, value);
}

void (CPU::*CPU::storeValue[])(int place, int value)
{
    nullptr,
    nullptr,                                    // these should never be called
    &CPU::storeRegister,
    &CPU::storeMemory
};
```

Exercise 48

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

Data Model

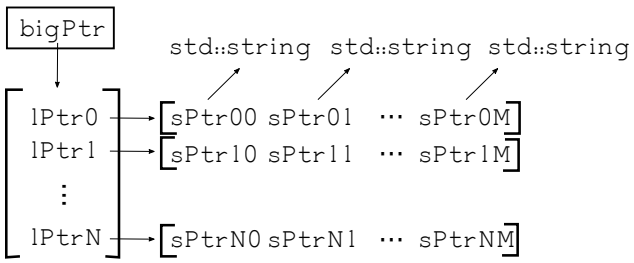

Figure 1: `bigPtr` is a triple pointer. It points to an array of 'line pointers', each of these point to an array of `std::string` pointers representing the comma-seperated values. For example: using the notation above we have `bigPtr[1][1] = sPtr11` for the second value on the second line.

csv.h

```
#ifndef CSV_HEADER_H
#define CSV_HEADER_H

#include <string>                                   // std::string
#include <istream>                                  // std::istream

class CSV
{
    size_t d_size = 1;                              // number of lines allocated
    size_t d_nLines = 0;                            // number of lines read
    size_t d_nFields = 1;                           // number of values per line
    char d_fieldSep;                                // field seperator (default comma)

    std::string ***bigPtr;                          // pointer to array of line pointers (see also big comment below)

    public:
        CSV(size_t field, char fieldSep = ',');

        std::string const *const *const *data() const;     // return pointer to data
        std::string const &lastline()           const;     // ref last extraction

```

```
        size_t nFields()                        const;     // values per line, set in first read
        size_t size()                           const;     // number of currently stored lines

        size_t read(std::istream &in, size_t nLines = 0);  // read lines using read1, return number read

        std::string ***release();                          // return pointer to data, move responsibility for data
                                                           // to called. Resets bigPtr but does not erase stored lines.
        void clear(size_t nFields = 0);                    // erase everything
    private:
        bool read1(std::istream &in);                      // read 1 line, parse for CSV's, set nFields
};

#endif // CSV_HEADER_H

// Line pointers point to array of pointers
// to std::string. i.e. :
// bigPtr -> [Lptr0 Lptr1 ... LptrN]
// where Lptri -> [strPtri1 strPtri2 ... strPtriM] for i = 1,...,N
// where strPtrik -> std::string                   for k = 1,...,M/
// see also the figure in the report.
```

csh.ih

```
#include "csv.h"
```

csh1.cc

```
#include "csv.ih"

CSV::CSV(size_t field, char fieldSep)
    :
        d_fieldSep(fieldSep)
{
    bigPtr = new std::string **[1];       // allocate array of line pointers
    bigPtr[0] = new std::string *[field]; // allocate array of string pointers (line)
}
```

main.cc

```
#include "csv.h"

int main()
{
    CSV file1(5, ',');
}
```

Exercise 49

Problem statement. *Implement the CSV class member functions.*
Solution.

csv.h

```
#ifndef CSV_HEADER_H
#define CSV_HEADER_H

#include <string>                                  // std::string
#include <istream>                                 // std::istream
```

```
#include "../csvextractor/csvextractor.h"

class CSV
{
    size_t d_size;                                  // number of lines allocated
    size_t d_nLines;                                // number of lines read
    size_t d_nFields;                               // number of values per line
    char d_fieldSep;                               // field seperator (default comma)
    std::string d_lastLine;

    std::string ***bigPtr;                                  // pointer to array of line pointers (see also big comment below)

    public:
        CSV(size_t field, char fieldSep = ',');

        std::string const *const *const *data() const;      // return pointer to data
        std::string const &lastline()           const;      // ref last extraction

        size_t nFields()                        const;      // values per line, set in first read
        size_t size()                           const;      // number of currently stored lines

        size_t read(std::istream &in, size_t nLines = 0);   // read lines using read1, return number read

        std::string ***release();                           // return pointer to data, move responsibility for data
                                                            // to called. Resets bigPtr but does not erase stored lines.
        void clear(size_t nFields = 0);                     // erase everything
    private:
        bool read1(std::istream &in);                       // read 1 line, parse for CSV's, set nFields
        void allocate();
        void doubleSize();
};

inline size_t CSV::nFields() const
{
    return d_nFields;
}

inline size_t CSV::size() const
{
    return d_size;
}
#endif // CSV_HEADER_H

// Line pointers point to array of pointers
// to std::string. i.e. :
// bigPtr -> [Lptr0 Lptr1 ... LptrN]
// where Lptri -> [strPtri1 strPtri2 ... strPtriM] for i = 1,...,N
// where strPtrik -> std::string                   for k = 1,...,M/
// see also the figure in the report.

// - memcpy copies raw bytes
```

csv.ih

```
#include "csv.h"
#include <string>
#include <iostream>
using std::string;
```

allocate.cc

```
#include "csv.ih"

void CSV::allocate()
```

```
{
    bigPtr = new std::string **[d_size];                                // line array
    for(string **line = bigPtr[0]; line != bigPtr[d_size - 1]; ++line) // strings in lines
        line = new std::string *[d_nFields];
}
// allocate me some memory
```

clear.cc

```
#include "csv.ih"

void CSV::clear(size_t nFields)                                         // nFields defaults to 0
{
                                                                        // de-allocate all every line array
    for (string **line = bigPtr[0]; line != bigPtr[d_nLines - 1]; ++line)
        delete[] line;
                                                                        // de-allocate array of lines
    delete[] bigPtr;
                                                                        // reset parameters
    d_size = 1;
    d_nLines = 0;
    d_nFields = nFields;
                                                                        // re-allocate (should be private helper)
    allocate();
}
```

csv1.cc

```
#include "csv.ih"

CSV::CSV(size_t field, char fieldSep)
    :
        d_size(1),                  // to allocate: 1 line
        d_nLines(0),                 // 0 lines read so far
        d_nFields(field),            // to allocate: 'field' fields
        d_fieldSep(fieldSep),         // set field seperator, default ','
        d_lastLine()
{
    allocate();
}
```

data.cc

```
#include "csv.ih"

std::string const *const *const *CSV::data() const
{
    return bigPtr;
}
```

doublesize.cc

```
#include "csv.h"

void CSV::doubleSize()
{
    d_size = d_size << 1;
    allocate();
}
```

lastline.cc

```
#include "csv.h"

std::string const &CSV::lastline() const
{
    return d_lastLine;
}
```

main.cc

```
#include "csv.h"

int main()
{
    CSV file1(5, ',');
}
```

read.cc

```
#include "csv.h"

size_t CSV::read(std::istream &in, size_t nLines) // nLines defaults to 0
{
    size_t lines = 0;

    if (nLines == 0)
        while (in.good())                       // read all lines
            {
                    read1(in);
                    ++lines;
            }
    else
         while (lines != nLines && in.good())     // read 'nLines' lines
            {
                    read1(in);
                    ++lines;
            }
    return lines;
}

// By default, all lines of in are read and are processed by the read1 member.
// By specifying a non-zero value for the nLines parameter the specified number of
// lines is read from in. Reading stops once in's status is not good. When nLines
// is specified as zero, then reading continues until all CSV lines have been processed.
// The number of successfully processed lines is returned.
```

read1.cc

```
#include "csv.h"

bool CSV::read1(std::istream &in)
{
    CSVextractor csvFile(in);              // CSVextractor takes 1 line from stream 'in'

    if (d_nFields == 0)                    // field count
        d_nFields = csvFile.nFields();

    if (d_size - d_nLines  == 0)           // increase capacity
        doubleSize();

```

```
13      return csvFile.parse(bigPtr);
14  }
15
16  // One line is read from in and is parsed for its CSVs. If parsing fails, false is returned.
17  // After successfully calling read1 for the first time all subsequent lines read by read1 must
18  // have the same number of comma separated values as encountered when calling read1 for the first time..
```

release.cc

```
1
```