

C++ Course Assignment 6

Contents

Exercise 43	2
strings.h	2
strings5.cc	3
Exercise 44	3
Exercise 45	3
Exercise 46	3
Exercise 47	3
cpu.h	4
dereference.cc	5
memoryreturn.cc	5
readoperand.cc	5
registerreturn.cc	5
valuereturn.cc	6
start.cc	6
errorwrap.cc	6
execute.cc	6
stp.cc	7
store.cc	7
storememory.cc	7
storeregister.cc	7
storevalue.cc	8
Exercise 48	8
Data Model	8
csv.h	8
main.cc	9
Exercise 49	9
csv.h	9
csv.ih	10
allocate.cc	10
clear.cc	11
csv1.cc	11
data.cc	11
doublesize.cc	11
lastline.cc	12
main.cc	12
read.cc	12
read1.cc	13
release.cc	13

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

```
1  #ifndef INCLUDED_STRINGS_
2  #define INCLUDED_STRINGS_
3
4  #include <iosfwd>
5
6  class Strings
7  {
8      size_t d_size;
9      std::string *d_str;
10
11     public:
12         struct POD
13         {
14             size_t      size;
15             std::string *str;
16         };
17
18         Strings();
19         ~Strings();
20         Strings(int argc, char *argv[]);
21         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.i.h"          // 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 by any 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
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

```
1  #include "cpu.ih"
2
3  int (CPU::*CPU::readOperand[])(Operand const &value) =    // order as in enums.h
4  {
5      nullptr,                // padding for syntax, will never be called
6      &CPU::valueReturn,
7      &CPU::registerReturn,
8      &CPU::memoryReturn
9  };
```

registerreturn.cc

```
1  #include "cpu.ih"
2
3  int CPU::registerReturn(Operand const &value)
4  {
5      return d_register[value.value];
6  }
```

valuereturn.cc

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

start.cc

```
1  #include "cpu.ih"
2
3  void CPU::start()
4  {
5      while (true)
6      {
7          (this->*execute[d_tokenizer.opcode()])( );
8          d_tokenizer.reset();
9      }
10 }
```

errorwrap.cc

```
1  #include "cpu.ih"
2
3  void CPU::errorwrap()
4  {
```

```
5     error();
6 }
```

execute.cc

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

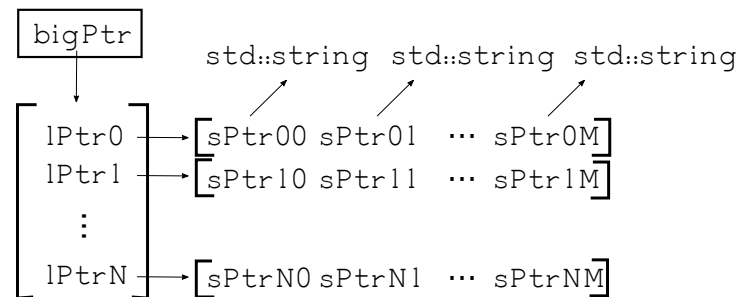


Figure 1: `bigPtr` is a triple pointer. It points to an array of 'line pointers', each of these point to an array of `std::string` pointers representing the comma-separated values. For example: using the notation above we have `bigPtr[1][1] = sPtr11` for the second value on the second line.

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 separator (default comma)
13
14     std::string ***bigPtr;                          // pointer to array of line pointers (see also big comment below)
15
16     public:
17         CSV(size_t field, char fieldSep = ',');
18
19         std::string const *const *const *data() const;    // return pointer to data
20         std::string const &lastline() const;             // ref last extraction
21
22         size_t nFields() const;                          // values per line, set in first read
23         size_t size() const;                             // number of currently stored lines
24
25         size_t read(std::istream &in, size_t nLines = 0); // read lines using read1, return number read
26
27         std::string ***release();                        // return pointer to data, move responsibility for data
28                                                         // to caller. Resets bigPtr but does not erase stored lines.
29         void clear(size_t nFields = 0);                 // erase everything
30     private:
31         bool read1(std::istream &in);                    // read 1 line, parse for CSV's, set nFields
32 };
33
34 #endif // CSV_HEADER_H
35
36 // Line pointers point to array of pointers
37 // to std::string. i.e. :
38 // bigPtr -> [Lptr0 Lptr1 ... LptrN]
39 // where Lptri -> [strPtri1 strPtri2 ... strPtriM] for i = 1,...,N
40 // where strPtrik -> std::string for k = 1,...,M/
41 // see also the figure in the report.

```

main.cc

```

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

```

Exercise 49

Problem statement. Implement the CSV class member functions.

Solution.

csv.h

```

1 #ifndef CSV_HEADER_H
2 #define CSV_HEADER_H
3
4 #include <string>                                // std::string
5 #include <istream>                               // std::istream
6 #include "../csvextractor/csvextractor.h"
7
8 class CSV
9 {
10     size_t d_size;                                // number of lines allocated
11     size_t d_nLines;                              // number of lines read
12     size_t d_nFields;                             // number of values per line
13     char d_fieldSep;                              // field separator (default comma)
14     std::string d_lastLine;

```



```

15
16     std::string ***bigPtr;                                // pointer to array of line pointers (see also big comment below)
17
18     public:
19         CSV(size_t field, char fieldSep = ',');
20
21         std::string const *const *const *data() const;      // return pointer to data
22         std::string const &lastline() const;                // ref last extraction
23
24         size_t nFields() const;                             // values per line, set in first read
25         size_t size() const;                                // number of currently stored lines
26
27         size_t read(std::istream &in, size_t nLines = 0);   // read lines using read1, return number read
28
29         std::string ***release();                           // return pointer to data, move responsibility for data
30                                                         // to caller. Resets bigPtr but does not erase stored lines.
31         void clear(size_t nFields = 0);                     // erase everything
32     private:
33         bool read1(std::istream &in);                       // read 1 line, parse for CSV's, set nFields
34         void allocate();
35         void doubleSize();
36 };
37
38 inline size_t CSV::nFields() const
39 {
40     return d_nFields;
41 }
42
43 inline size_t CSV::size() const
44 {
45     return d_size;
46 }
47 #endif // CSV_HEADER_H
48
49 // Line pointers point to array of pointers
50 // to std::string. i.e. :
51 // bigPtr -> [Lptr0 Lptr1 ... LptrN]
52 // where Lptri -> [strPtri1 strPtri2 ... strPtriM] for i = 1,...,N
53 // where strPtrik -> std::string for k = 1,...,M/
54 // see also the figure in the report.
55
56 // - memcpy copies raw bytes

```

csv.ih

```

1  #include "csv.h"
2  #include <string>
3  #include <iostream>
4  using std::string;

```

allocate.cc

```

1  #include "csv.ih"
2
3  void CSV::allocate()
4  {
5      bigPtr = new std::string **[d_size];                // line array
6      for(string **line = bigPtr[0]; line != bigPtr[d_size - 1]; ++line) // strings in lines
7          line = new std::string *[d_nFields];
8  }
9  // allocate me some memory

```

clear.cc

```
1  #include "csv.ih"
2
3  void CSV::clear(size_t nFields)                // nFields defaults to 0
4  {
5                                                  // de-allocate all every line array
6      for (string **line = bigPtr[0]; line != bigPtr[d_nLines - 1]; ++line)
7          delete[] line;
8
9          delete[] bigPtr;                        // de-allocate array of lines
10
11          // reset parameters
12      d_size = 1;
13      d_nLines = 0;
14      d_nFields = nFields;
15
16      // re-allocate (should be private helper)
17      allocate();
18 }
```

csv1.cc

```
1  #include "csv.ih"
2
3  CSV::CSV(size_t field, char fieldSep)
4  :
5      d_size(1),                // to allocate: 1 line
6      d_nLines(0),              // 0 lines read so far
7      d_nFields(field),         // to allocate: 'field' fields
8      d_fieldSep(fieldSep),     // set field separator, default ','
9      d_lastLine()
10 {
11     allocate();
12 }
```

data.cc

```
1  #include "csv.ih"
2
3  std::string const *const *const CSV::data() const
4  {
5      return bigPtr;
6  }
```

doublesize.cc

```
1  #include "csv.h"
2
3  void CSV::doubleSize()
4  {
5      d_size = d_size << 1;
6      allocate();
7  }
```

lastline.cc

```
1  #include "csv.h"
2
3  std::string const &CSV::lastline() const
4  {
```

```
5     return d_lastLine;
6 }
```

main.cc

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

read.cc

```
1 #include "csv.h"
2
3 size_t CSV::read(std::istream &in, size_t nLines) // nLines defaults to 0
4 {
5     size_t lines = 0;
6
7     if (nLines == 0)
8         while (in.good()) // read all lines
9             {
10                 read1(in);
11                 ++lines;
12             }
13     else
14         while (lines != nLines && in.good()) // read 'nLines' lines
15             {
16                 read1(in);
17                 ++lines;
18             }
19     return lines;
20 }
21
22 // By default, all lines of in are read and are processed by the read1 member.
23 // By specifying a non-zero value for the nLines parameter the specified number of
24 // lines is read from in. Reading stops once in's status is not good. When nLines
25 // is specified as zero, then reading continues until all CSV lines have been processed.
26 // The number of successfully processed lines is returned.
```

read1.cc

```
1 #include "csv.h"
2
3 bool CSV::read1(std::istream &in)
4 {
5     CSVextractor csvFile(in); // CSVextractor takes 1 line from stream 'in'
6
7     if (d_nFields == 0) // field count
8         d_nFields = csvFile.nFields();
9
10    if (d_size - d_nLines == 0) // increase capacity
11        doubleSize();
12
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..
```
