J.B

50 ?
51 0
52 8

C++ Exercises
Set 6

Author(s): Tjalling Otter, Emiel Krol

–

October 22, 2018

50

Listing 1: main.ih

```
#include <iostream>

#include "charcount/charcount.h"

using namespace std;

void showChar(char ch);
```

Listing 2: main.cc

```
#include "main.ih"

int main()
{
    CharCount cc;
                                                 // process and count all chars
    cout << "processed " << cc.count(cin) << " characters\n";

    CharCount::CharInfo const &info = cc.info();

    for (size_t idx = 0; idx != info.nChar; ++idx)
    {
        showChar(info.ptr[idx].ch);
        cout << ": " << info.ptr[idx].count << " times\n";
    }
}
```

Listing 3: showChar.cc

```
#include "main.ih"

void showChar(char ch)
{
    cout << "char ";

    switch (ch)
    {
        case '\n':
            cout << "'\\n'";
        break;

        case '\t':
            cout << "'\\t'";
        break;

        case ' ':
            cout << "' '";
        break;

        default:
            if (isprint(ch))
                cout << '\'' << ch << '\'';
            else
                cout << static_cast<size_t>( static_cast<unsigned char>(ch) );
        break;
    }
}
```

Listing 4: charcount/charcount.h

```
#ifndef INCLUDED_CHARCOUNT_
#define INCLUDED_CHARCOUNT_

#include <iosfwd>

class CharCount
{
  enum Action
    {
        APPEND = 0,
        INSERT = 1,
        ADD = 2
    };

    public:
        struct Char
        {
            char ch;
            size_t count;
        };

        struct CharInfo
        {
            Char *ptr;
            size_t nChar;
        };
```

NMN Waarom?

```
    private:

        size_t d_cap = 255; //maximum size since no more ASCII values possible
        size_t d_size = 8;  //starting size
```

?

```
        CharInfo d_info =
        {
          static_cast<CharCount::Char *>(
          operator new(d_size * sizeof(CharCount::Char)))
          , 0
        };

        //allocating raw memory block for the array of Char Objects
```

capacity()?

```
    public:
        size_t count(std::istream &in);
        CharInfo const &info() const;
        ~CharCount(); //defining destructor so that the used memory is
                      //freed at the end of main.

    private:
        void process(char ch);
        //calls locate and then the appropriate action
        //(append, insert or add)
        Action locate(size_t *idx, char ch);
        //locates the index of the current char and returns the appropriate
        //action
        void append(char ch, size_t idx); // inserts char at nChar
        void insert(char ch, size_t idx); // inserts char at index
                                          // and calls transfer

        void add(char ch, size_t idx);
                                          //increases count by 1 of already
```

```
                                    //existing char object
        void transfer(Char *dest, size_t begin, size_t end);
        //moves all chars from begin to an index 1 higher, starting at the
        //highest index
        void enlarge(); //allocates a new raw block of memory, twice the size
                        //of the previous memory
        void destroy();
                        //frees the memory of the Char objects and afterwards
                        //the memory used by the pointer itself.

        CharCount::Char rawCapacity() const;

        //returns the current raw capacity

        static void (CharCount::*s_action[])(char ch, size_t idx);

        //declares the array of pointers so it can reach private member
        //functions (add, insert, append)
};

inline CharCount::CharInfo const &CharCount::info() const
{
    return d_info; //returns a reference to charinfo object
}

inline CharCount::Char CharCount::rawCapacity() const
{
    return *(d_info).ptr; //returns current raw capacity
}

inline CharCount::~CharCount()
{
  destroy();   //destructor
}


#endif
```

DATA, NOT A FUNCTION.

Waarom?

Destructors NIET inline => college!!

Listing 5: charcount/charcount.ih

```
#include "charcount.h"

#include <iostream>
using namespace std;
```

Listing 6: charcount/add.cc

```
#include "charcount.ih"

void CharCount::add(char ch, size_t idx)
{
    ++d_info.ptr[idx].count;
}
```

Listing 7: charcount/append.cc

```
#include "charcount.ih"

void CharCount::append(char ch, size_t idx)
{
    insert(ch, d_info.nChar);
}
//appendix character at the end, adding index so the array of pointers
//to members works.
```

Listing 8: charcount/count.cc

```
#include "charcount.ih"
```

```
size_t CharCount::count(istream &in)
{
    size_t nChars = 0;

    char ch;

    while (in.get(ch))
    {
        ++nChars;
        process(ch);                        // add ch to the set of characters
    }

    return nChars;
}
```

Listing 9: charcount/destroy.cc

```
#include "charcount.ih"

void CharCount::destroy()
{
  for (Char *end = d_info.ptr + d_size; end-- != d_info.ptr; )
      end->~Char();

  operator delete(d_info.ptr);
}
```

NSC

Listing 10: charcount/enlarge.cc

```
#include "charcount.ih"


void CharCount::enlarge()
{
  if  ((d_size <<= 1) > d_cap)
    d_size = d_cap;

  CharCount::Char *tmp = static_cast<CharCount::Char *>(
    operator new(d_size * sizeof(CharCount::Char)));

  for  (size_t index = d_size; index--; )
    new(tmp + index) CharCount::Char{ d_info.ptr[index] };

  destroy();
  d_info.ptr = tmp;
}
```

Listing 11: charcount/insert.cc

```
#include "charcount.ih"

void CharCount::insert(char ch, size_t idx)
{
    if  (d_size == d_info.nChar + 1)
      enlarge();

    Char *&ptr = d_info.ptr;
                                        // transfer the rest
    transfer(ptr + d_info.nChar + 1, idx, d_info.nChar);

    ptr[idx] = Char{ ch, 1 };           // insert the new element

    ++d_info.nChar;                     // added new element
    d_info.ptr = ptr;                   // point at the new Char array
}
```

Listing 12: charcount/locate.cc

```
#include "charcount.ih"
```

```
CharCount::Action CharCount::locate(size_t *destIdx, char ch)
{
    size_t uCh = static_cast<unsigned char>(ch);

    for (size_t idx = 0; idx != d_info.nChar; ++idx)
    {
        size_t value = static_cast<unsigned char>(
                                d_info.ptr[idx].ch
                            );

        if (uCh > value)
            continue;

        *destIdx = idx;

        return uCh == value ?
                    ADD
                :
                    INSERT;
    }

    return APPEND;                          // append at the end
}
```

NSC

Listing 13: charcount/process.cc

```
#include "charcount.ih"

void (CharCount::*CharCount::s_action[])(char ch, size_t idx) =
{
    &CharCount::append,
    &CharCount::insert,
    &CharCount::add,
};


void CharCount::process(char ch)
{
    size_t idx;
    Action loc_action = locate(&idx, ch);
    (this->*s_action[loc_action])(ch, idx);
}

//defines array of pointers to member functions
//locate finds the index of the current character and what action to take
//this action and index is then passed to the array of pointers
```

Q: put static data in a data.cc file

Listing 14: charcount/transfer.cc

```
#include "charcount.ih"

void CharCount::transfer(Char *dest, size_t begin, size_t end)
{
    for (; begin - 1 != end; --end)
        *dest-- = move(d_info.ptr[end]);
}
```

What's the gain?

51

51 0

Exercise 51

New / delete variants

– New variable or array

Allocates new memory sized appropriately to the type and dimensions specified. Thereafter it will attempt to construct and initialise these objects. Finally, it will return a pointer to the start of the memory allocated to these objects. The advantage and consideration of constructing an object (array) is that one has control over when it is deleted (if at all).

```
—— Example ——
int main()
{
  Class *pointer;
  if (boolAppropriate)
    pointer = new Class;

  if (boolNoLongerNecessary)
    delete pointer;
}
\end{lstlisting}
—— End ——
```

incomplete

Without the use of new (i.e. Class newClass), the scope of the newly constructed class would simply be limited to within the if statement, and thereafter destroyed. However, now the object persists until manually deleted. As such, the creation and destruction do not take place unless necessary. Another example would be in the creation of arrays with dimensions that are not previously established. \\

– Placement new

Allows for 'filling' an already allocated piece of memory, possibly with another object or objects than the one one / those that it was originally allocated for. In other words, it allows us to simply construct objects in memory previously allocated. This could be useful when imagining a shared or transient storage location in memory that can be easily located, whether it contains characters, integers, or whatever else.

```
—— Example ——
int main()
{
  char block[10 * sizeof(ExClass)]; // Block of memory necessary later

  if (boolNeedsTypeA)
    for (size_t idx = 0; index != 10; ++index)  // Create ten 'ExClass's in the
      ExClass *sEC = new(block + idx * sizeof(ExClass)) // space, using constructor
                          ExClass(typeA);                // A.

  if (boolNeedsTypeB)
    for (size_t idx = 0; index != 10; ++index)  // Create ten 'ExClass's in the
      ExClass *sEC = new(block + idx * sizeof(ExClass)) // space, using constructor
                          ExClass(typeB);                // B.

  sEC->~ExClass();          // Destruct the data
  delete[] block;           // Deallocate the memory
}
—— End ——
```

Nothing's allocated: delete breaks your program

incomplete

In this example, we know that a block of memory sized to fit ten 'ExClass's is required, but not yet which constructor will be used to fill it. Hence, it can already be allocated, but left empty until that decision is made. In the end, \\

– Operator new

This allocates raw memory sized to fit a specified number of specified objects.

```
—— Example ——
int main()
{
  string *block = static_cast<string *>(operator new(8 * sizeof(std::string)));

  string *pNames = new(block + 5) std::string("John");

  pNames->~string();

  operator delete(block);
}
—— End ——
```

Without actually filling this memory, it is rather useless on its own. Instead, it can be viewed as a preferable alternative to placement new discussed previously. Even without repeating/rewriting the loop, it is already obvious that the notation is simpler: there is no continuous evaluation of sizeof throughout the loop. Instead, just the offset index is used.

52

As can be seen, the new addition consists of a destructor ~Strings(); defined in the header and implemented in dtor.cc. In turn, this destructor calls the previously-defined destroy(); (see destoy.cc, included for convenience). The destructor is automatically called when the object goes out of scope, so this ensures that the memory allocated by the objects is deleted in such a situation (as it would be before enlarging it).

Listing 15: strings/strings.h

852

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

    ~Strings(); // New addition: destructor.

    void swap(Strings &other);

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

    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);
}

#endif
```

Listing 16: `strings/dtor.cc`

```
#include "strings.ih"

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

Listing 17: `strings/destroy.cc`

```
#include "strings.ih"

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