LIX, École Polytechnique

Introduction to C++: Exercises

Leo Liberti

Last update: 16th November 2006

Contents

Chapter 1

Generalities

1.1 Definition of *program*

(a) Your clothes are dirty and you want to put them in the washing machine. You put the temperature at 50 C° and set the cycle at "coloured cottons"; you satisfy yourself that you don't have a full load, so you press the "half-load" button; finally, you start the washing cycle. Is this a program? (b) Now suppose your girlfriend tells you to "do the washing, please". Is this a program? Justify your answers.

1.2 Definition of *programming language*

Are the rules of chess a programming language? Justify your answer.

1.3 Indentation

Give a precise definition of the concept "well-indented C++ program". Check whether your definition correctly classifies the following snippets of code. In the case of programs that are not well-indented, find the reason why they are not.

```
// 1. this is a well-indented C++ program
#include<iostream>
int main(int argc, char** argv) {
  int ret = 0;
  std::cout << "Hello world" << std::endl;
  return ret;
}
```

```
// 2. this is a well-indented C++ program
#include<iostream>
const int theGlobalValue = 0;
int main(int argc, char** argv) {
  theGlobalValue = 1;
  std::cout << theGlobalValue << std::endl;
  return 0;
}

// 3. this is NOT a well-indented C++ program (why?)
#include<iostream>
int main(int argc, char** argv) {
  if (argc < 2) {
    std::cerr << "error" << std::endl;
    } else {
      std::cout << argv[1] << std::endl;
  }
}

// 3. this is NOT a well-indented C++ program (why?)
#include<iostream>
  const int theGlobalValue = 0;
  bool isArgcGood(int argc) {
    if (argc < 2) {
      std::cout << "error" << std::endl;
      return false; }
    return true;
  }
  int main(int argc, char** argv) {
  if (isArgcGood(argc)) {
    std::cout << argv[1] << std::endl;
  }
  return 0;
}

// 4. this is NOT a well-indented C++ program (why?)
#include<iostream>
int main(int argc, char** argv) {
  int i = 0;
  while(argv[i]) i++;
  std::cout << i << std::endl;
}
```

Chapter 2

Basic syntax

2.1 A minimal C++ program

Edit, save and build the following program.

```
/***********************************************
* Name:       minimal.cxx
* Author:     Leo Liberti
* Source:     GNU C++
* Purpose:    minimal C++ program
* Build:      c++ -o minimal minimal.cxx
* History:    060818 work started
***********************************************/

#include<iostream>

int main(int argc, char** argv) {
  using namespace std;
  int ret = 0;

  // your code goes here

  return ret;
}
```

Now modify it to print the array of characters `argv[0]`.

2.2 Variables and pointers

Consider the following program.

```
#include<iostream>

const char endOfCharArray = '\0';
```

```
int main(int argc, char** argv) {
  using namespace std;
  int ret = 0;
  if (argc < 2) {
    cerr << "error: no arguments on cmd line" << endl;
    ret = 1;
  } else {
    for(int i = 0; i < argc; i++) {
      while(*(argv[i]) != endOfCharArray) {
        cout << *(argv[i]);
        argv[i]++;
      }
      cout << endl;
    }
  }
  return ret;
}
```

(a) List the variables in the program and their types. (b) List the pointers in the program and their types. (c) How many pointers are there in the program when it is run with the command line `./variables_pointers 123 456 789`? (d) Edit, save, build and run the program with the command line above. What is the output?

2.3 The second word

(a) Write a program that writes the second word (and its length) in an array of characters. Words are separated by spaces, punctuation and tabs. Test your code on the sentence "Actually, couldn't you come with me?" The correct answer should be `couldn't(8)`. If you're stuck, you can follow the template below.

```
// test if c is in the array containing all the valid separators
bool isSeparator(char c, char* separators);
// return the length of the char array a
int charArrayLength(char *a);
int main(int argc, char** argv) {
  // find the beginning of the second word
  // find the end of the second word
}
```

2.4 Random numbers

The following code can be used to generate pseudo-random floating point numbers in $[0,1]$.

```
#include<iostream>
#include<sys/time.h>

int main(int argc, char** argv) {
  using namespace std;

  // initialize randomizer
  struct timeval theTV;
  struct timezone theTZ;
  gettimeofday(&theTV, &theTZ);
  srandom(theTV.tv_usec);

  // pick a random number between 0 and 1
  double normalizedRndCoeff = (double) random() / (double) RAND_MAX;
  cout << normalizedRndCoeff << endl;
  return 0;
}
```

Write a small program that generates a random sample of given size N (input from command line) and that computes the mean and the variance. Use your program to show empirically that

$$\alpha = \lim_{N\to\infty} \mathrm{Var}(X) \cong 0.083,$$

where X is a random variable. Prove that $\alpha = \frac{1}{12}$.

Optional: update your program to generate a random sample whose elements are within given numerical bounds.

Chapter 3

Classes

3.1 Complex numbers

The following code is the class header describing a complex number class.

```
/*
** Name:     complex.h
** Author:   Leo Liberti
** Purpose:  header file for a complex numbers class
** Source:   GNU C++
** History:  061019 work started
*/

#ifndef _COMPLEXH
#define _COMPLEXH

#include<string>
#include<iostream>

class Complex {

 public:

  Complex();
  Complex(double re, double im);
  ~Complex();

  double getReal(void);
  double getImaginary(void);
  void setReal(double re);
  void setImaginary(double im);
  void fromString(const std::string& complexString);

  Complex operator+(Complex& theComplex);
  Complex operator-(Complex& theComplex);
  Complex operator*(Complex& theComplex);
  Complex operator/(Complex& theComplex);

 private:
  double real;
  double imag;
};

std::ostream& operator<<(std::ostream& out, Complex& theComplex);

#endif
```

Write the corresponding implementation file `complex.cxx`. Test it on the following `main()` function definition, using the command `./test 2.1+0.2i / 0.2-0.3i`. The output should be `2.76923 + 5.15385i`.

```
/*
```

```
** Name:     test.cxx
** Author:   Leo Liberti
** Purpose:  testing the complex numbers class
** Source:   GNU C++
** History:  061019 work started
*/

#include <iostream>
#include <string>
#include "complex.h"

int main(int argc, char** argv) {
  using namespace std;
  if (argc < 4) {
    cerr << "need an operation on command line" << endl;
    cerr << "   e.g. ./test 4.3+3i - 2+3.1i" << endl;
    cerr << "   (no spaces within each complex number, use spaces to\n";
    cerr << "    separate the operands and the operator - use arithmetic\n";
    cerr << "    operators only)" << endl;
    return 1;
  }
  string complexString1 = argv[1];
  string complexString2 = argv[3];
  Complex complex1;
  complex1.fromString(complexString1);

  Complex complex2;
  complex2.fromString(complexString2);

  Complex complex3;
  if (argv[2][0] == '+') {
    complex3 = complex1 + complex2;
  } else if (argv[2][0] == '-') {
    complex3 = complex1 - complex2;
  } else if (argv[2][0] == '*' || argv[2][0] == '.') {
    argv[2][0] = '*';
    complex3 = complex1 * complex2;
  } else if (argv[2][0] == '/') {
    complex3 = complex1 / complex2;
  }
  cout << complex1 << " " << argv[2][0] << " (" << complex2 << ") = "
       << complex3 << endl;
  return 0;
}
```

Use the following `Makefile` to compile (use tabs, not spaces after each label — for that you need to use the Emacs editor):

```
# Name:     complex.Makefile
# Author:   Leo Liberti
# Purpose:  makefile for complex class project
# Source:   GNU C++
# History:  061019 work started

CXXFLAGS = -g

all: test

test: complex.o test.cxx
	c++ $(CXXFLAGS) -o test test.cxx complex.o

complex.o: complex.cxx complex.h
	c++ $(CXXFLAGS) -c -o complex.o complex.cxx

clean:
	rm -f *~ complex.o test
```

3.2 Virtual inheritance

The code below defines a virtual base class `VirtualBase` and a derived class `Derived` which implements it. The `VirtualBase` interface is simply to set and get an integer value. The `Derived` adds a method for printing the value. We then have two functions in the global namespace, `printTheValue1()` and printTheValue2(), which print out the values in different ways: the first simply uses the get method of the interface to retrieve the value; the second tries to transform the VirtualBase interface pointer passed as argument to a pointer to the Derived class, and then calls the Derived class' print method.

```
// this program does not compile!
#include<iostream>

class VirtualBase {
public:
  virtual ~VirtualBase() { }
  virtual void setValue(int i) = 0;
  virtual int getValue(void) = 0;
};

class Derived : public virtual VirtualBase {
public:
  ~Derived() { }
  void setValue(int i) {
    theValue = i;
  }
  int getValue(void) {
    return theValue;
  }
  void printValue(void) {
    std::cout << "Derived::printValue(): value is " << theValue << std::endl;
  }
private:
  int theValue;
};

void printTheValue1(VirtualBase* v) {
  std::cout << "printTheValue1(): value is " << v->getValue() << std::endl;
}

void printTheValue2(VirtualBase* v) {
  Derived* d = v;
  d->printValue();
}

int main(int argc, char** argv) {
  int ret = 0;
  Derived d;
  VirtualBase* v = &d;
  v->setValue(1);
  printTheValue1(v);
  printTheValue2(v);
  return ret;
}
```

The desired output is:

```
printTheValue1(): value is 1
Derived::printValue(): value is 1
```

However, the program fails to compile with the error:

```
virtual2.cxx: In function 'void printTheValue2(VirtualBase*)':
virtual2.cxx:31: error: invalid conversion from 'VirtualBase*' to 'Derived*'
virtual2.cxx:31: error: cannot convert from base 'VirtualBase'
                 to derived type 'Derived' via virtual base 'VirtualBase'
```

What has gone wrong? How can you fix this program? [**Hint**: look at C++ casting operators]

3.3 Virtual and nonvirtual inheritance

What is the output of the following code?

```
#include<iostream>

class A {
public:
  void f() {
    std::cout << "A::f" << std::endl;
  }
  virtual void g() {
    std::cout << "A::g" << std::endl;
  }
};

class B : public A {
public:
  void f() {
    std::cout << "B::f" << std::endl;
  }
  virtual void g() {
    std::cout << "B::g" << std::endl;
  }
};

int main(int argc, char** argv) {
  A a;
  B b;
  A* aPtr = &a;
  A* bPtr = &b;
  aPtr->f();
  aPtr->g();
  bPtr->f();
  bPtr->g();
  return 0;
}
```

Is there anything surprising? Can you explain it?

Since this is produced by this code:

```
aPtr->f();
```

```
aPtr->g();
bPtr->f();
bPtr->g();
```

It is surprising that the pointer to the `B` object does not print `B::f` but `A::f`. The reason is that the derived class `B` hides the method `A::f()`, which was *not* declared virtual. This is known as *compile-time polymorphism*: at compile-time, the compiler only knows that `bPtr` is of type `A*`, and therefore binds to it the `f()` method found in `A`. The `g()` method, however, was declared `virtual` in the base class, which means that the compiler will implement *run-time polymorphism* (almost always the desired kind), and will delegate binding decisions (i.e. , deciding what method to call) to run-time. At run-time, even though `bPtr` is of type `A*`, it is possible to find out that in fact it points to an object of type `B`, so the correct binding takes place.

3.4 Nonvirtual base class destructor

The code below shows the ill effects of hiding a non-virtual constructor of a base class, and then using the derived class as a base one.

```
// this program is buggy!
#include<iostream>

class Base {
public:
  Base() {
    std::cerr << "constructing Base " << this << std::endl;
    i = new int;
  }
  ~Base() {
    std::cerr << "destroying Base " << this << std::endl;
    delete i;
  }
private:
  int* i;
};

class Derived : public Base {
public:
  Derived() {
    std::cerr << "constructing Derived " << this << std::endl;
    d = new double;
  }
  ~Derived() {
    std::cerr << "destroying Derived " << this << std::endl;
    delete d;
  }
private:
  double* d;
};

int main(int argc, char** argv) {
  using namespace std;
  int ret = 1;
  Base* thePtr = new Derived;
  delete thePtr;
  return ret;
}
```

The output of this program is

```
constructing Base 0x804a008
constructing Derived 0x804a008
destroying Base 0x804a008
```

This is a bug, because the `Derived` object was never deallocated. At best, this is a memory leak. In some other cases it may become the source of more serious problems. Can you explain what happened and how to fix the program?

Chapter 4

Debugging

4.1 Segmentation fault

The following code contains three serious memory bugs, usually ending in a segmentation fault. Find the bugs, solve them and explain them.

```
// this program is buggy
#include<iostream>
int main(int argc, char** argv) {
  using namespace std;
  double* d = new double;
  for(unsigned int i = 0; i < 3; i++) {
    d[i] = 1.5 + i;
  }
  for(unsigned int i = 2; i >= 0; i--) {
    cout << d[i] << endl;
  }
}
```

4.2 Pointer bug

The following snippet of code behaves unpredictably: it will often print a value for `testInt` which is different from 1, although in the variable `testInt` is never explicitly changed. Debug the program using `ddd` and `valgrind`. What is the bug? How can you fix it? Change the order of the statements marked (a), (b), (c) and test the resulting programs: what is the behaviour? Can you explain it?

```
// this program is buggy!
#include<iostream>
#include<cstring>

const int bufSize = 20;

int main(int argc, char** argv) {
  using namespace std;
  if (argc < 3) {
    cerr << "need a char and a word as arguments on cmd line" << endl;
    return 1;
  }

  // a test integer to which we assign the value 1
  int testInt = 1;                                  // (a)

  // get the first char of the first argument
  char theChar = argv[1][0];                        // (b)

  // get the second argument as word
  char buffer[bufSize];                             // (c)
  strncpy(buffer, argv[2], bufSize);
  char* wordPtr = buffer;

  // skip characters in the word and delete them, until we find theChar
  while(*wordPtr != theChar) {
    *wordPtr = ' ';
    wordPtr++;
    cout << (int) (wordPtr - buffer) << endl;
  }

  // now see what value has the test integer
  cout << testInt << endl;
  return 0;
}
```

4.3 Scope of local variables

Inspect the following code. Convince yourself that the expected output should be `0 1 2 3 4`. Compile and test the code. The output is unpredictable (may be similar to `134524936 134524968 134524952 134524984 134525024` depending on the machine and circumstances). How do you explain this bug? How can you fix it?

```
#include<iostream>
#include<vector>

class ValueContainer {
public:
  ValueContainer() : theInt(0) { }
  ~ValueContainer() { }
  void set(int* t) {
    theInt = t;
  }
  int get(void) {
    return *theInt;
  }
private:
  int* theInt;
};

ValueContainer* createContainer(int i) {
  ValueContainer* ret = new ValueContainer;
  ret->set(&i);
  return ret;
}

const int vecSize = 5;
int main(int argc, char** argv) {
  using namespace std;
  vector<ValueContainer*> value;
  for(int i = 0; i < vecSize; i++) {
    value.push_back(createContainer(i));
  }
  for(int i = 0; i < vecSize; i++) {
    cout << value[i]->get() << " ";
  }
  cout << endl;
  for(int i = 0; i < vecSize; i++) {
    delete value[i];
  }
  return 0;
}
```

4.4 Erasing elements from a vector

The following code initializes an STL vector of integers to $(2, 1)$ then iterates to erase all elements with value 1. The code compiles but on running it we get a segmentation fault abort. Find and fix the bug.

```
// this program is buggy!
#include<vector>
int main(int argc, char** argv) {
  using namespace std;
  int ret = 0;
  vector<int> theVector;
  theVector.push_back(2);
  theVector.push_back(1);
  vector<int>::iterator vi;
  for(vi = theVector.begin(); vi != theVector.end(); vi++) {
    if (*vi == 1) {
      theVector.erase(vi);
    }
  }
  return ret;
}
```

Chapter 5

A full-blown application

In this chapter we will describe in a step-by-step fashion how to build a full-blown application in C++. The application is called WET (WWW Exploring Topologizer) and its purpose is to explore a neighbourhood of a given URL in the World Wide Web and to output it in the shape of a graph. We delegate graph visualization to the GraphViz library `www.graphviz.org`, and in particular to the `dot` and `neato` UNIX utilities. These accept the input graph in a particular format

```
digraph graphName {
# list of nodes with special properties
  0 [ label = "thenode", color = red ];
# list of arcs
   0 -> 1;
   1 -> 2;
}
```

So the task performed by WET is to download a given URL and explore its links up to the n-th recursion level, then to put the obtained graph in the above format.

This chapter starts with a section on the software architecture; each class is then described and its implementation mostly delegated as exercise. The software architecture section does not contain any exercises, but it is a prerequisite for understanding what follows. Moreover, it is a (crude) example showing how to formalize the architecture of a software.

5.1 The software architecture

5.1.1 Description

Given an URL, this tool explores a local neighbourhood of the URL. Each web page is represented by a vertex in a graph, and given two vertices u, v there is an arc between them if there is a hyperlink citing the web page v in the web page u. The graph is then printed on the screen.

5.1.2 Formalization of requirements

- Input:
  - the URL we must start the retrieval from

  - the maximum hyperlink depth
- Output:
  - graph and time when the exploration was undertaken shown on screen

5.1.3 Modularization of tasks

- web page retrieval: downloads the HTML page for a given URL
- web page parser: finds all the hyperlinks in a given HTML page
- watch: returns the current time
- graph: stores vertices, arcs and a timestamp; outputs the graph to screen

5.1.4 Main algorithm

1. Input options and necessary data (from command line)
2. Start recursive URL retrieval algorithm

```
retrieveData(URL, maxDepth, Graph) {
  get current timestamp;
  store timestamp in graph
  retrieveData(URL, maxDepth, Graph, 0);
}
store graph on disk

retrieveData(URL, maxDepth, Graph, currentDepth) {
  if (currentDepth < maxDepth) {
    retrieve URL;
    parse HTML, get list of sub-URLs;
    for each sub-URL in list {
      store arc (URL, sub-URL) in graph
      retrieveData(sub-URL, maxDepth, graph, currentDepth + 1);
    }
  }
}
```

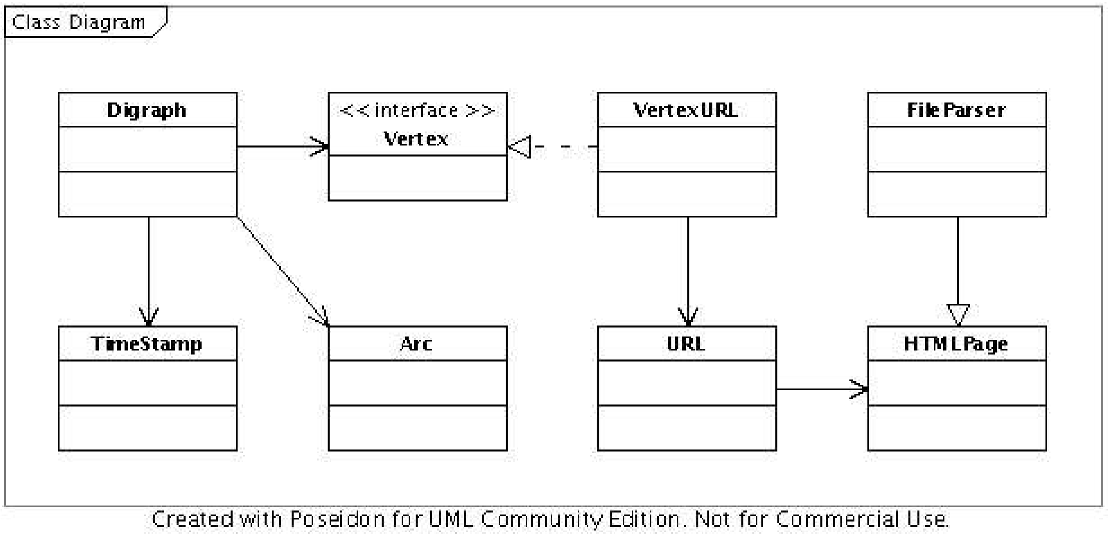
5.1.5 Fundamental data structures

- URL
- timestamp
- vertex
- arc
- graph

5.1.6 Classes

```
class TimeStamp; // stores a time stamp
class fileParser; // parses text files for tag=value (or tag="value") cases
class HTMLPage : public fileParser; // stores an HTML page
class URL; [contains HTMLPage object] // retrieves an HTML page
class Vertex; // public virtual class for a vertex
class VertexURL : public virtual Vertex; [contains URL] // implements a vertex
class Arc; [contains two Vertex objects] // arc
class Digraph; [contains a list of arcs and a timestamp] // graph
```

The diagram below shows the class interactions.



5.2 The TimeStamp class

Design and implement a class called `TimeStamp` managing the number of seconds elapsed since 1/1/1970. The class should have three methods: `long get(void)` to get the number of seconds since 1/1/1970, `void set(long t)` to set it, `void update(void)` to read it off the operating system. Furthermore, it must be possible to pass a `TimeStamp` object (call it `timeStamp`) on to a stream, as follows: `cout << timeStamp;` to obtain the date pointed at by `timeStamp`. Failures should be signalled by throwing an exception called `TimeStampException`.

You can use the following code to find the number of seconds elapsed since 1970 from the operating system:

```
#include<sys/time.h>
[...]
struct timeval tv;
struct timezone tz;
int retVal = gettimeofday(&tv, &tz);
long numberOfSecondsSince1970 = tv.tv_sec;
```

The following code transforms the number of seconds since 1970 into a meaningful date into a string:

```
#include<string>
#include<ctime>
#include<cstring>
[...]
time_t numberOfSecondsSince1970;
char* buffer = ctime(&numberOfSecondsSince1970);
buffer[strlen(buffer) - 1] = '\0';
std::string theDateString(buffer);
```

Write the class declaration in a file called `timestamp.h` and the method implementation in `timestamp.cxx`. Test your code with the following program `tsdriver.cxx`:

```
#include<iostream>
#include "timestamp.h"

int main(int argc, char** argv) {
  using namespace std;
  TimeStamp theTimeStamp;
  theTimeStamp.set(999999999);
  cout << "seconds since 1/1/1970: " << theTimeStamp.get()
       << " corresponding to " << theTimeStamp << endl;
  theTimeStamp.update();
  cout << theTimeStamp << " corresponds to " << theTimeStamp.get()
       << " number of seconds since 1970" << endl;
  return 0;
}
```

and verify that the output is:

```
seconds since 1/1/1970: 999999999 corresponding to Sun Sep  9 03:46:39 2001
Mon Sep 11 11:33:42 2006 corresponds to 1157967222 number of seconds since 1970
```

The solution to this exercise is included as an example of coding style. The rest of the exercises in this chapter should be solved by the student.

5.2.1 Solution

Here follows the header file.

```
/*******************************************************
** Name:        timestamp.h
** Author:      Leo Liberti
** Source:      GNU C++
** Purpose:     www exploring topologizer: TimeStamp class header
** History:     060820 work started
*******************************************************/

#ifndef _WETTIMESTAMPH
```

```
#define _WETTIMESTAMPH

#include<iostream>
#include<sys/time.h>

class TimeStampException {
 public:
  TimeStampException();
  ~TimeStampException();
};

class TimeStamp {

 public:

  TimeStamp() : timestamp(0) { } ;
  ~TimeStamp() { }

  long get(void) const;
  void set(long theTimeStamp);

  // read the system clock and update the timestamp
  void update(void) throw (TimeStampException);

 private:

  long timestamp;

};

// print the timestamp in humanly readable form
std::ostream& operator<<(std::ostream& s, TimeStamp& t)
     throw (TimeStampException);

#endif
```

The following points are worth emphasizing.

1. All header file code is structured as follows:
   ```
   #ifndef _INCLUDEFILENAMEH
   #define _INCLUDEFILENAMEH
   [...]
   #endif
   ```
   The purpose of this is to avoid that the contents of a header file should be compiled more than once even though the header file itself is referenced more than once (this may happen should a `.cxx` file reference two header files A, B where A also references B as its own header — this situation occurs quite often in practice).

2. All exception classes referred to in the main class being declared in the header should be declared first.

3. A function can be declared *const* (i.e. `const` is appended at the end of the declaration) as a promises not to change its owner object. This is a good way to catch some bad bugs, as the compiler signals an error if a `const` function has some code that might potentially change the `*this` object.

4. Notice that class and namespace closing braces are followed by a colon (';').
5. The overloaded `operator<<` function is declared outside the class (it takes an object of the class as an argument)

Here follows the implementation file.

```
/*******************************************************
** Name:        timestamp.cxx
** Author:      Leo Liberti
** Source:      GNU C++
** Purpose:     www exploring topologizer: TimeStamp class
** History:     060820 work started
*******************************************************/

#include <iostream>
#include <ctime>
#include "timestamp.h"

TimeStampException::TimeStampException() { }
TimeStampException::~TimeStampException() { }

long TimeStamp::get(void) const {
  return timestamp;
}

void TimeStamp::set(long theTimeStamp) {
  timestamp = theTimeStamp;
}

void TimeStamp::update(void) throw (TimeStampException) {
  struct timeval tv;
  struct timezone tz;
  using namespace std;
  try {
    int retVal = gettimeofday(&tv, &tz);
    if (retVal == -1) {
      cerr << "TimeStamp::updateTimeStamp(): couldn't get system time" << endl;
      throw TimeStampException();
    }
  } catch (...) {
    cerr << "TimeStamp::updateTimeStamp(): couldn't get system time" << endl;
    throw TimeStampException();
  }
  timestamp = tv.tv_sec;
}

std::ostream& operator<<(std::ostream& s, TimeStamp& t)
  throw (TimeStampException) {
  using namespace std;
  time_t theTime = (time_t) t.get();
  char* buffer;
  try {
    buffer = ctime(&theTime);
  } catch (...) {
    cerr << "TimeStamp::updateTimeStamp(): couldn't print system time" << endl;
    throw TimeStampException();
  }
  buffer[strlen(buffer) - 1] = '\0';
  s << buffer;
  return s;
}
```

1. Include files are usually included as follows: first the standard includes, then the user-defined ones. It helps to differentiate them by employing the two (equivalent) forms: `#include<file.h>` and `#include "file.h"` for standard and user-defined respectively.
2. Methods in an implementation file must be defined using the *scope* operator `::`
   $\boxed{\textit{returnType ClassName}\texttt{::}\textit{MethodName}\texttt{(}\textit{args}\texttt{)}\ \{\textit{definition}\}}$
3. The overloaded operator `<<` is outside the class definition, so there is no prepended scoping operator.

5.3 The FileParser class

Design and implement a class called `FileParser` that scans a file for a given string tag followed by a character `'='` and then reads and stores the string after the `'='` either up to the first space character or, if an opening quote character `'"'` is present after `'='`, up to the closing quote character. The class functionality is to be able to detect all contexts such as `tag = string` or `tag = "string"` in a text file, store the `string`s and allow subsequent access to them.

The class should be declared in `fileparser.h` and implemented in `fileparser.cxx`. You can use the following header code for class declaration.

```
/***************************************************************
** Name:        fileparser.h
** Author:      Leo Liberti
** Source:      GNU C++
** Purpose:     www exploring topologizer - file parser (header)
** History:     060820 work started
****************************************************************/

#ifndef _WETFILEPARSERH
#define _WETFILEPARSERH

#include<string>
#include<vector>

class FileParserException {
 public:
  FileParserException();
  ~FileParserException();
};

class FileParser {
 public:
  FileParser();
  FileParser(std::string theFileName, std::string theParseTag);
  ~FileParser();

  void setFileName(std::string theFileName);
  std::string getFileName(void) const;
  void setParseTag(std::string theParseTag);
  std::string getParseTag(void) const;

  // parse the file and build the list of parsed strings
  void parse(void) throw(FileParserException);

  // get the number of parsed strings after parsing
  int getNumberOfParsedStrings(void) const;

  // get the i-th parsed string
  std::string getParsedString(int i) const throw(FileParserException);

 protected:
  std::string fileName;
  std::string parseTag;

  // compare two strings (case insensitive), return 0 if equal
  int compareCaseInsensitive(const std::string& s1,
                             const std::string& s2) const;

  // return true if s2 is the tail (case insensitive) of string s1
  bool isTailCaseInsensitive(const std::string& s1,
                             const std::string& s2) const;

 private:
  std::vector<std::string> parsedString;

};

#endif
```

Pay attention to the following details:

- It is good coding practice to provide all classes with a set of get/set methods to access/modify internal (private) data.
- The `compareCaseInsensitive` and `isTailCaseInsensitive` methods are declared `protected` because they may not be accessed externally (i.e. they are for the class' private usage) but it may be useful to make them accessible to derived classes. Here is the definition of these two (technical) methods:

```
int FileParser::compareCaseInsensitive(const std::string& s1,
                                       const std::string& s2) const {
  using namespace std;
  string::const_iterator p1 = s1.begin();
  string::const_iterator p2 = s2.begin();
  while(p1 != s1.end() && p2 != s2.end()) {
    if (toupper(*p1) < toupper(*p2)) {
      return -1;
    } else if (toupper(*p1) > toupper(*p2)) {
      return 1;
    }
    p1++;
    p2++;
  }
  if (s1.size() < s2.size()) {
    return -1;
  } else if (s1.size() > s2.size()) {
    return 1;
  }
  return 0;
}

bool FileParser::isTailCaseInsensitive(const std::string& s1,
                                       const std::string& s2) const {
  using namespace std;
  int s2len = s2.size();
  if (s1.size() >= s2.size() &&
      compareCaseInsensitive(s1.substr(s1.size() - s2len, s2len), s2) == 0) {
    return true;
  }
  return false;
}
```

- In order to write the `parse` method, you have to scan the file one character at a time. The parser can be in any of the following three states: `outState`, `inTagState`, `inValueState`. Normally, the status is `outState`. When the parse tag is detected, a transition is made to `inTagState`. After that, if an 'equal' character (`'='`) is found, another transition is made to `inValueState`. The parser has three different behaviours, one in each state. When in `outState`, it simply looks for the parse tag `parseTag`. When in `inTagState`, it looks for the character `'='`, when in `inValueState`, it stores the value up to the first separating space or between quotes.

- The following code can be used to open a file and scan it one character at a time:

```
#include<iostream>
#include<fstream>
#include<istream>
#include<sstream>
#include<iterator>
[...]
// opens the file for input
string fileName("myfilename.ext");
ifstream is(fileName.c_str());
if (!is) {
  // can't open file, throw exception
}
// save all characters to a string buffer
char nextChar;
stringstream buffer;
while(!is.eof()) {
  is.get(nextChar);
  buffer << nextChar;
}
// output the buffer
cout << buffer.str() << endl;
// erase the buffer
buffer.str("");
```

- You can test your code with the following program `fpdriver.cxx`:

  ```
  #include<iostream>
  #include "fileparser.h"

  int main(int argc, char** argv) {
    using namespace std;
    if (argc < 3) {
      cerr << "missing 2 args on cmd line" << endl;
      return 1;
    }
    FileParser fp(argv[1], argv[2]);
    fp.parse();
    for(int i = 0; i < fp.getNumberOfParsedStrings(); i++) {
      cout << fp.getParsedString(i) << endl;
    }
    return 0;
  }
  ```

  Make a small HTML file as follows, and call it `myfile.html`:

  ```
  <html>
    <head>
      <title>MyTitle</title>
    </head>
    <body>
      My <a href="http://mywebsite.com/mywebpage.html">absolute link</a>
       and my <a href="otherfile.html">relative link</a>.
    </body>
  </html>
  ```

  Now run the `fpdriver` code as follows: `./fpdriver myfile.html href`, and verify that it produces the following output:

  ```
  http://mywebsite.com/mywebpage.html
  otherfile.html
  ```

5.4 The `HTMLPage` class

Design and implement a class called `HTMLPage` which inherits from `FileParser`. It should have the following functionality: opens an HTML file (of which the URL is known), reads and stores its links (transforming relative links into absolute links), allows external access to the stored links. This class is derived from `FileParser` because it uses `FileParser`'s functionality in a specific manner, performing some specific task (relative links are transformed into absolute links) of which `FileParser` knows nothing about.

- A link in an HTML page is the value in the tag `href="value"` (or simply `href=value`). HTML is a case insensitive language, so you may also find `HREF` instead of `href`.

- A WWW link is *absolute* if it starts with `http://` and a DNS name and is relative otherwise. Given a relative link (i.e. not starting with a slash '/' character, e.g. `liberti/index.html`) and an absolute link (e.g. `http://www.lix.polytechnique.fr/users`), the relative link is transformed into an absolute link by concatenating the two URLs (absolute and relative) with any missing slashes in between (e.g. `http://www.lix.polytechnique.fr/users/liberti/index.html`. On the other hand, if the relative link starts with a slash '/' (e.g. `/ liberti/index.html`) then the transformation of the relative link to absolute occurs by concatenating the first part of the absolute link (up to and including the DNS name) and the relative one (e.g. `http://www.lix.polytechnique.fr/~liberti/index.html`).

- Remember that not all links found next to HREF tags are valid HTTP links, some may concern different protocols. You should make sure that the link introduced by HREF is either relative (in which case you should make it absolute as explained above) or specifically an `http` link.

- Test your class implementation using the following driver `hpdriver.cxx`:

```
#include<iostream>
#include "htmlpage.h"

int main(int argc, char** argv) {
  using namespace std;
  if (argc < 3) {
    cerr << "missing 2 args on cmd line" << endl;
    return 1;
  }
  HTMLPage hp(argv[2], argv[1]);
  for(int i = 0; i < hp.getNumberOfLinks(); i++) {
    cout << hp.getLink(i) << endl;
  }
  return 0;
}
```

  and check that the output of the command `./hpdriver http://127.0.0.1 myfile.html` is

```
http://mywebsite.com/mywebpage.html
http://127.0.0.1/otherfile.html
```

5.5 The `URL` class

Design and implement a class called `URL` containing a pointer to an `HTMLPage` object. Its functionality is to download an HTML file from the internet given its URL, and to allow access to the links contained therein.

- We shall make use of the `wget` external utility to download the HTML file. This can be used from the shell: `wget http://my.web.site/mypage.html` will download the specified URL to a local file called `mypage.html`. We can specify the output file with the option `-O output_file_name`. In order to call an external program from within C++, use the following code.

```
#include<cstdlib>
[...]
char charQuote = '\"';
string theURL = "http://127.0.0.1/index.html";
string outputFile = ".wet.html";
string cmd = "wget --quiet -O " + outputFile + charQuote + theURL + charQuote;
int status = system(cmd.c_str());
```

- You will need to delete the temporary file `.wet.html` after having parsed it — use the following code.

```
#include<unistd.h>
[...]
unlink(outputFile.c_str());
```

- The `URL` class will expose a method `std::string getNextLink(void)` to read the links in the downloaded HTML page in the order in which they appear. Each time the method is called, the next link is returned. When no more links are present, the empty string should be returned. Subsequent calls to `getNextLink()` should return the sequence of links again as in a cycle.
- The `URL` class will contain a pointer to an `HTMLPage` object which parses the HTML code and finds the relevant links. Note that this means that a user memory allocation/deallocation will need to be performed.
- Test your `URL` implementation using the driver program `urldriver.cxx`:

```
#include<iostream>
#include "url.h"

int main(int argc, char** argv) {
  using namespace std;
  if (argc < 2) {
    cerr << "missing arg on cmd line" << endl;
    return 1;
  }
  URL url(argv[1]);
  url.download();
  string theLink = url.getNextLink();
  while(theLink.size() > 0) {
    cout << theLink << endl;
    theLink = url.getNextLink();
  }
  return 0;
}
```

Run the program

```
./urldriver http://www.enseignement.polytechnique.fr/profs/informatique/Leo.
Liberti/test.html
```

and verify that the output is

```
http://www.enseignement.polytechnique.fr/profs/informatique/Leo.Liberti/test3.html
http://www.enseignement.polytechnique.fr/profs/informatique/Leo.Liberti/test2.html
```

5.6 The `Vertex` interface and `VertexURL` implementation

Design and implement a class `VertexURL` inheriting from the pure virtual class `Vertex`:

```
/*******************************************************
** Name:        vertex.h
** Author:      Leo Liberti
** Source:      GNU C++
** Purpose:     www exploring topologizer -
**              vertex abstract class (header)
** History:     060820 work started
*******************************************************/

#ifndef _WETVERTEXH
#define _WETVERTEXH

#include<iostream>

class Vertex {
 public:
  virtual ~Vertex() {
#ifdef DEBUG
    std::cerr << "** destroying Vertex " << this << std::endl;
#endif
  }
  virtual int getID(void) const = 0;
  virtual int getNumberOfAdjacentVertices(void) const = 0;
  virtual int getAdjacentVertexID(int i) const = 0;
  virtual void addAdjacentVertexID(int ID) = 0;
};

#endif
```

`VertexURL` is an encapsulation of a `URL` object which conforms to the `Vertex` interface. This will be used later in the `Digraph` class to store the vertices of the graph. This separation between the `Vertex` interface and its particular implementation (for example the `VertexURL` class) allows the `Digraph` class to be re-used with other types of vertices, and therefore makes it very useful. An object conforming to the `Vertex` interface must know its own integer ID and IDs of the vertices adjacent to it. Note that `Vertex` has a virtual destructor for the following reason: should any agent attempt a direct deallocation of a `Vertex` object, the destructor actually called will be the destructor of the interface implementation (`VertexURL` in this case, but might be different depending on run-time conditions) rather than the interface destructor.

Test your implementation on the following code:

```
#include<iostream>
#include "vertexurl.h"

int main(int argc, char** argv) {
  using namespace std;
  if (argc < 2) {
    cerr << "missing arg on cmd line" << endl;
    return 1;
  }
  URL* urlPtr = new URL(argv[1]);
  urlPtr->download();
  VertexURL* vtxURLPtr = new VertexURL(0, urlPtr);
  Vertex* vtxPtr = vtxURLPtr;
  vtxPtr->addAdjacentVertexID(1);
  vtxPtr->addAdjacentVertexID(2);
  int starsize = vtxPtr->getNumberOfAdjacentVertices();
  cout << "vertex " << vtxPtr->getID() << ": star";
  for(int i = 0; i < starsize; i++) {
    cout << " " << vtxPtr->getAdjacentVertexID(i);
  }
  cout << endl;
  delete vtxPtr;
  return 0;
}
```

Run the program `./vurldriver http://kelen.polytechnique.fr` and verify that the output is vertex 0: star 1 2.

5.7 The `Arc` and `Digraph` classes

1. Design and implement an `Arc` class to go with the `Vertex` class above. In practice, an `Arc` object just needs to know its starting and ending vertex IDs.

2. Design and implement a `Digraph` class to go with the `Vertex` and `Arc` classes defined above. This needs to store a `TimeStamp` object, a vector of pointers to `Vertex` objects, a vector of pointers to `Arc` objects. We need methods both for constructing (`addVertex`, `addArc`) as well as accessing the graph information (`getNumberOfVertices`, `getNumberOfArcs`, `getVertex`, `getArc`). We also want to be able to send a `Digraph` object to the `cout` stream to have a GraphViz compatible text output which we shall display using the GraphViz utilities.

Test your implementation on the following program `dgdriver.cxx`:

```
#include<iostream>
#include "digraph.h"

int main(int argc, char** argv) {
  using namespace std;
  string urlName1 = "name1";
  string urlName2 = "name2";
  string urlName3 = "name3";
  URL* urlPtr1 = new URL(urlName1);
  URL* urlPtr2 = new URL(urlName2);
  URL* urlPtr3 = new URL(urlName3);
  VertexURL* vtxURLPtr1 = new VertexURL(1, urlPtr1);
  VertexURL* vtxURLPtr2 = new VertexURL(2, urlPtr2);
  VertexURL* vtxURLPtr3 = new VertexURL(3, urlPtr3);
  Vertex* vtxPtr1 = vtxURLPtr1;
  Vertex* vtxPtr2 = vtxURLPtr2;
  Vertex* vtxPtr3 = vtxURLPtr3;
  vtxPtr1->addAdjacentVertexID(2);
  vtxPtr1->addAdjacentVertexID(3);
  vtxPtr2->addAdjacentVertexID(1);
  vtxPtr3->addAdjacentVertexID(3);
  Arc* arcPtr1 = new Arc(1,2);
  Arc* arcPtr2 = new Arc(1,3);
  Arc* arcPtr3 = new Arc(2,1);
  Arc* arcPtr4 = new Arc(3,3);
  TimeStamp t;
  t.update();
  Digraph G;
  G.setTimeStamp(t);
  G.addVertex(*vtxPtr1);
  G.addVertex(*vtxPtr2);
  G.addVertex(*vtxPtr3);
  G.addArc(*arcPtr1);
  G.addArc(*arcPtr2);
  G.addArc(*arcPtr3);
  G.addArc(*arcPtr4);
  cout << G;
  return 0;
}
```

and verify that the output is:

```
# graphviz output by WET (L. Liberti 2006)
digraph www_1158015497 {
  0 [ label = "Tue Sep 12 00:58:17 2006", color = red ];
   1 -> 2;
   1 -> 3;
   2 -> 1;
   3 -> 3;
}
```

5.8 Putting it all together: `main()`

Code the `main` function and all auxiliary functions into the files `wet.h` (declarations) and `wet.cxx` (definitions). The program should: read the given URL and the desired downloading recursion depth limit from the command line, then recursively download the given URL and all meaningful sub-links up to the given depth limit, whilst storing them as vertices and arcs of a digraph, which will then be printed out in GraphViz format.

- The recursive part of the algorithm can be coded into a pair of functions (one is the "driver" function, the other is the truly recursive one) declared as follows:

```
    // retrieve the data, driver
    void retrieveData(std::string URL, int maxDepth, Digraph& G,
                      bool localOnly, bool verbose);
    // retrieve the data, recursive
    void retrieveData(std::string URL, int maxDepth, Digraph& G,
                      bool localOnly, bool verbose,
                      int currentDepth, VertexURL* vParent);
```

(also see Section 5.1.4).

- Endow your WET application with a command line option `-v` (stands for "verbose") that prints out the URLs as they are downloaded

- Endow your WET application with a command line option `-l` (stands for "local") that ignores all URLs referring to webservers other than localhost (127.0.0.1). **Optional**: also try to devise an option `-L` that ignores all URLs referring to webservers different from the one given on command line.

- The GraphViz system can read a text description of a given graph and output a graphical representation of the graph in various formats. The WET application needs to output the text description of the `Digraph` object. One self-explanatory example of the required format has been given at the beginning of this chapter. You can type `man dot` at the shell to get more information. Implement WET so that it outputs the graph description in the correct format on `cout`. When your WET application is complete, run it so that it saves the output to a file with the `.dot` extension: `./wet http://kelen.polytechnique.fr/ 3 > mygraph.dot`. The command `dot -Tgif -o mygraph.gif mygraph.dot` builds a GIF file which you can display using the command `xv mygraph.gif`.

Test your WET application by running
[`./wet http://www.enseignement.polytechnique.fr/profs/informatique/Leo.Liberti/test.html 4`] and verifying that the output is

```
# graphviz output by WET (L. Liberti 2006)
digraph www_1158043952 {
  0 [ label = "Tue Sep 12 08:52:32 2006", color = red ];
   0 -> 1;
   1 -> 0;
   1 -> 2;
   2 -> 0;
   2 -> 1;
   0 -> 2;
}
```

By saving `wet`'s output as `wetout.dot` and running `dot -Tgif -o wetout.gif wetout.dot` we get Fig. 5.1, left. Reducing the depth level to 2 yields Fig. 5.1, right.

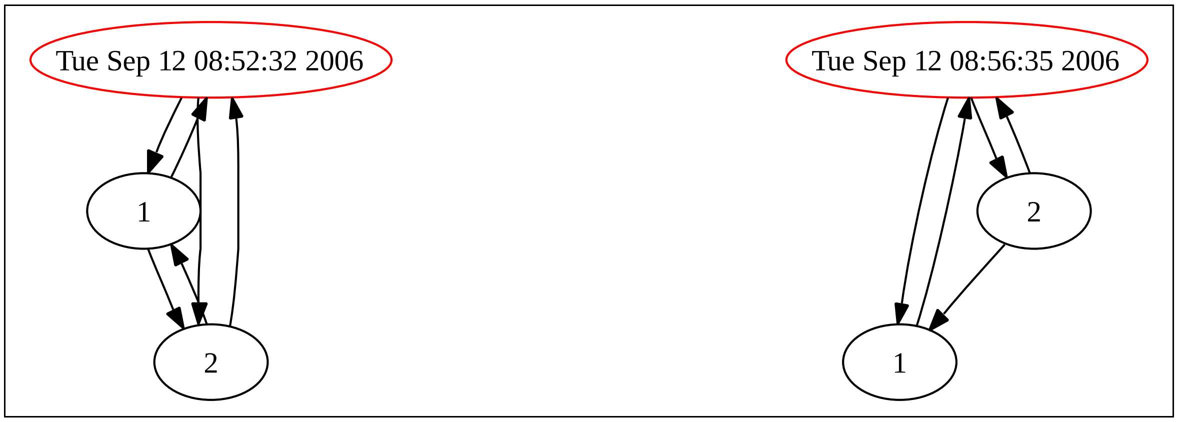


Figure 5.1: The neighbourhoods graphs (depth 4, left; depth 2, right).