Introduction to C++: Exercises
(with solutions)

Leo Liberti

Last update: December 12, 2011
Contents

1 Generalities 7
   1.1 Definition of *program* ........................................ 7
       1.1.1 Solution ............................................... 7
   1.2 Definition of *programming language* ......................... 7
       1.2.1 Solution ............................................... 7
   1.3 Indentation ................................................... 8
       1.3.1 Solution ............................................... 9

2 Basic syntax 11
   2.1 A minimal C++ program ....................................... 11
       2.1.1 Solution ............................................... 11
   2.2 Variables and pointers ....................................... 11
       2.2.1 Solution ............................................... 12
   2.3 The second word ............................................. 12
       2.3.1 Solution ............................................... 13
   2.4 Random numbers .............................................. 14
       2.4.1 Solution ............................................... 14
   2.5 The second column ........................................... 15
       2.5.1 Solution ............................................... 16

3 Classes 19
   3.1 Complex numbers ............................................ 19
       3.1.1 Solution ............................................... 20
   3.2 Virtual inheritance .......................................... 22
       3.2.1 Solution ............................................... 24
3.3 Virtual and nonvirtual inheritance ................................................. 24
  3.3.1 Solution .............................................................................. 25
3.4 Nonvirtual base class destructor ..................................................... 25
  3.4.1 Solution .............................................................................. 26
4 Debugging ......................................................................................... 27
  4.1 Segmentation fault ........................................................................ 27
    4.1.1 Solution .............................................................................. 27
  4.2 Pointer bug .................................................................................... 30
    4.2.1 Solution .............................................................................. 31
  4.3 Scope of local variables .................................................................... 33
    4.3.1 Solution .............................................................................. 33
  4.4 Erasing elements from a vector ....................................................... 34
    4.4.1 Solution .............................................................................. 34
5 A full-blown application .................................................................... 37
  5.1 The software architecture ............................................................. 38
    5.1.1 Description ......................................................................... 38
    5.1.2 Formalization of requirements ............................................... 38
    5.1.3 Modularization of tasks ......................................................... 38
    5.1.4 Main algorithm .................................................................... 39
    5.1.5 Fundamental data structures .................................................. 39
    5.1.6 Classes ............................................................................... 39
  5.2 The TimeStamp class ..................................................................... 40
    5.2.1 Solution .............................................................................. 41
  5.3 The FileParser class ...................................................................... 41
    5.3.1 Solution .............................................................................. 43
  5.4 The HTMLPage class ..................................................................... 43
    5.4.1 Solution .............................................................................. 44
  5.5 The URL class ............................................................................... 44
    5.5.1 Solution .............................................................................. 45
  5.6 The Vertex interface and VertexURL implementation ....................... 45
    5.6.1 Solution .............................................................................. 45
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.7</td>
<td>The Arc and Digraph classes</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>46</td>
</tr>
<tr>
<td>5.8</td>
<td>Putting it all together: main()</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>47</td>
</tr>
<tr>
<td>6.1</td>
<td>Debugging question 1</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>50</td>
</tr>
<tr>
<td>6.2</td>
<td>Debugging question 2</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>51</td>
</tr>
<tr>
<td>6.3</td>
<td>Debugging question 3</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>53</td>
</tr>
<tr>
<td>6.4</td>
<td>Debugging question 4</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>55</td>
</tr>
<tr>
<td>6.5</td>
<td>Debugging question 5</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>58</td>
</tr>
<tr>
<td>6.6</td>
<td>Debugging question 6</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>62</td>
</tr>
<tr>
<td>6.7</td>
<td>Specification coding question 1</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>64</td>
</tr>
<tr>
<td>6.8</td>
<td>Specification coding question 2</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>69</td>
</tr>
<tr>
<td>6.9</td>
<td>Task question 1</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>71</td>
</tr>
<tr>
<td>6.10</td>
<td>Task question 2</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>72</td>
</tr>
<tr>
<td>6.11</td>
<td>Task question 3</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>Solution</td>
<td>75</td>
</tr>
<tr>
<td>7</td>
<td>Questions from a banking C++ test</td>
<td>77</td>
</tr>
<tr>
<td>7.1</td>
<td>Question 1</td>
<td>77</td>
</tr>
<tr>
<td>7.2</td>
<td>Question 2</td>
<td>77</td>
</tr>
<tr>
<td>7.3</td>
<td>Question 3</td>
<td>78</td>
</tr>
</tbody>
</table>

CONTENTS
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4 Question 4</td>
<td>78</td>
</tr>
<tr>
<td>7.5 Question 5</td>
<td>78</td>
</tr>
<tr>
<td>7.6 Solutions</td>
<td>79</td>
</tr>
</tbody>
</table>
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.

1.1.1 Solution

(a) The washing machine is a mechanical computer insofar as it does not interpret its instructions but simply executes them. The instructions for 50 °C, “coloured cottons”, “half-load” and “start” are precisely defined by control knobs and button positions on the washing machines. Therefore the washing machine can correctly interpret the instructions given in the text. This means it is a program. (b) The object of the sentence “do the washing” has not been specified: the understanding rests upon the convention of the English language whereby “the washing” usually refers to washing clothes rather than anything else. In other words, interpretation of the sentence requires a human mind. Therefore, this is not a program.

1.2 Definition of *programming language*

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

1.2.1 Solution

The game of chess has a well-defined set of rules according to which it can be played. Furthermore, these rules can be correctly interpreted by a computer. However, these rules only say how to form a valid move, but not which move to choose. In other words, the rules of chess do not supply a computer with the sequence of operations to follow in order to interpret them. Rather, they restrict the choice to a subset of all possible sequences of operations (moves) to perform. In this sense, the rules of chess do not form valid instructions. Therefore, they are not a programming language. Computers programs that play chess only use the rules of chess as a constraint on the possible moves, but they use much more complex algorithms to determine which move to make.
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;
    return false;
  }
  std::cout << 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;
}

1.3.1 Solution

Definition
A C++ program is a set of instructions of the C++ programming language that can be interpreted by a computer into a sequence of basic operations that the computer can perform.

Axiom
A C++ program consists of a set of global declarations/definitions and several nested groups of instructions. Each group of instructions is delimited by curly brackets ({}).

Definition
The rank of a group of instructions in a C++ program is the nesting level of the group in the program. Global declarations/definitions are defined to be at rank 0.

Definition
Consider a C++ program with instructions $I_1, \ldots, I_n$. For a given $K > 0$, a group of $t$ instructions $(I_{h}, \ldots, I_{h+t})$ at rank $r$ in a C++ program is well-$K$-indented if:

1. no two instructions are on the same line of text;
2. the opening curly bracket is on a line of text above $I_h$ and the closing curly bracket is on a line of text below $I_{h+t}$;
3. $I_{h-1}$ (the last instruction at rank $r-1$ before the beginning of the group) starts $K$ columns of text before $I_{h}$;
4. the closing curly brace of the group is aligned with instruction $I_{h-1}$ (in other words, it is positioned $K$ columns of text before $I_{h+t}$).

Definition
A C++ program is well-indented if there is an integer $K > 0$ such that (a) the global declarations/definitions start at column 0, and (b) each of its instruction groups is well-$K$-indented.

Fact
Usual values for $K$ are 2, 4, 8. In this course, the value $K = 2$ is favoured as it allows for a greater number of nesting levels to be displayed in 80 characters (the usual screen width on UNIX systems).
Chapter 2

Basic syntax

2.1 A minimal C++ program

Edit, save and build the following program.

```cpp
/*******************************
* 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]`.

2.1.1 Solution

Insert the statement `cout << argv[0] << endl;` before the `return ret;` statement.

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?

2.2.1 Solution

(a) char endOfCharArray, int argc, char** argv, int ret, int i. (b) char** argv, char* argv[i] for all $i \in \{0, \ldots, argc - 1\}$. (c) Five pointers: argv and argv[i] with $0 \leq i \leq 3$. (d) The output is:

./variables_pointers
123
456
789

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.

```cpp
int main(int argc, char** argv) {
    // 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) {
```
2.3.1 Solution

```cpp
#include<iostream>

bool isSeparator(char c, char* separators) {
    char endOfString = '\0';
    char* sepPtr = separators;
    bool ret = false;
    while(*sepPtr != endOfString) {
        if (c == *sepPtr) {
            ret = true;
            break;
        }
        sepPtr++;
    }
    return ret;
}

int charArrayLength(char* a) {
    int ret = 0;
    char endOfString = '\0';
    while(*a != endOfString) {
        ret++;
        a++;
    }
    return ret;
}

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

    char endOfString = '\0';
    char theSentence[] = "Actually, couldn’t you come with me?";
    char separators[] = " 	,.;:?!";
    char* strPtr = theSentence;
    char* strEnd;

    // find the beginning of the second word
    bool isSecondWord = false;
    while(*strPtr != endOfString) {
        // while deals with case with consecutive separators
        while(isSeparator(*strPtr, separators)) {
            // while deals with case with consecutive separators
            strPtr++;
            isSecondWord = true;
        }
        if (isSecondWord) {
            // end of consecutive separators, means second word starts at strPtr
            break;
        } else {
            // character was not a separator, continue
            strPtr++;
        }
    }

    // find the end of the second word
    strEnd = strPtr;
    while(*strEnd != endOfString) {
        if (isSeparator(*strEnd, separators)) {
            // first separator found after second word, mark the end
            *strEnd = endOfString;
            break;
        }
    }
    return ret;
}
```

The second word
strEnd++;  
}
// print the second word and its length
cout << strPtr << "(" << charArrayLength(strPtr) << ")" << endl;
return ret;
}

2.4 Random numbers

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

```cpp
#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

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

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.

2.4.1 Solution

```cpp
#include<iostream>
#include<cstdlib>
#include<cmath>
#include<sys/time.h>

int main(int argc, char** argv) {
    using namespace std;
    // read sample size from command line
    if (argc < 2) {
        cerr << "mean_variance: sample size needed on command line" << endl;
        return 1;
    }
    int sampleSize = stoi(argv[1]);
    if (sampleSize <= 0) {
        cerr << "mean_variance: sample size must be strictly positive" << endl;
        return 2;
    }
    // initialize randomizer
    struct timeval theTV;
    struct timezone theTZ;
    gettimeofday(&theTV, &theTZ);
```
```cpp
srandom(theTV.tv_usec);
// generate the sample (numbers between 0 and 1)
double* sample = new double[sampleSize];
for(int i = 0; i < sampleSize; i++) {
    sample[i] = (double) random() / (double) RAND_MAX;
}

// compute the mean
double mean = 0;
for(int i = 0; i < sampleSize; i++) {
    mean += sample[i];
}
mean /= (double) sampleSize;

// compute the variance
double variance = 0;
double tmp = 0;
for(int i = 0; i < sampleSize; i++) {
    tmp = sample[i] - mean;
    tmp *= tmp;
    variance += tmp;
}
variance /= (double) sampleSize;

// compute the standard deviation
double stddev = sqrt(variance);

// output
cout << "mean = " << mean << " variance = " << variance
<< " std deviation = " << stddev << endl;
delete [] sample;
return 0;
```

Let \( x = (x_1, \ldots, x_N) \) a sample of size \( N \) of values of the random variable \( X \).

\[
\text{Var}(X) = \int_0^1 x^2 \, dx - \left( \int_0^1 x \, dx \right)^2 = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}.
\]

### 2.5 The second column

Write a program that reads in a text file and prints the sum of the numbers found in the second column (a word is in the second column of a file if its first character is the first non-space character after the first space character in the line). An example for opening and reading a text file is given below.

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

const int BUFSIZE = 1024;

int main(int argc, char** argv) {
    using namespace std;
    int ret = 0;
    if (argc < 2) {
        cerr << argv[0] << ": need filename on cmd line" << endl;
        exit(1);
    }
    char buffer[BUFSIZE];
    ifstream myStream(argv[1]);
    while(!myStream.eof()) {
        myStream.getline(buffer, (streamsize) BUFSIZE-1);
        cout << buffer << endl;
    }
    return ret;
}
```
2.5.1 Solution

```c++
/* name: secondColumn.cxx
author: Leo Liberti
purpose: sum the numbers in the second column of a given file
source: GNU C++
history: 0990119 work started */

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cstring>

const int BUFSIZE = 1024;
const int ARRAYSIZE = 50;

int main(int argc, char** argv) {
    using namespace std;
    int ret = 0;
    if (argc < 2) {
        // not enough arguments on cmd line
        cerr << argv[0] << ": need filename on cmd line" << endl;
        exit(1);
    }
    // used to hold lines of the text file
    char buffer[BUFSIZE];
    // these two ptrs are used to identify single words
    char* startPtr;
    char* endPtr;
    // this holds the data
    char myArray[ARRAYSIZE][ARRAYSIZE][BUFSIZE];
    // initialize myArray
    for(int i = 0; i < ARRAYSIZE; i++) {
        for(int j = 0; j < ARRAYSIZE; j++) {
            *(myArray[i][j]) = '\0';
        }
    }
    // open file
    ifstream myStream(argv[1]);
    // are we at the end of line?
    bool isEndLine = false;
    // is this char a space?
    bool isSpace = false;
    // row and column counters
    int row = 0;
    int col = 0;
    int maxRow = 0;
    int maxCol = 0;
    // until file ends
    while(!myStream.eof()) {
        // read a line in buffer
        myStream.getline(buffer, (streamsize) BUFSIZE-1);
        // set start and and pointers
        startPtr = buffer;
        endPtr = startPtr;
        // whilst not end of line
        isEndLine = false;
        while(!isEndLine) {
            // look for space
            endPtr++;
            isSpace = false;
            if (*endPtr == ' ') {
                // found, transform to null char
                *endPtr = '\0';
                isSpace = true;
            }
        }
    }
}
```

The second column
```cpp
if (*endPtr == '\0') {
    // we are at null char, copy this word into myArray
    isEndLine = true;
    strncpy(myArray[row][col], startPtr, BUFSIZE-1);
    col++;
}

if (isSpace) {
    // actually, the char was a space not and end of line, so reset to space
    *endPtr = ' ';
    // go to next non-space char
    while(*endPtr == ' ') {
        endPtr++;
    }
    // set start pointer to this char
    startPtr = endPtr;
    isEndLine = false;
}

if (col > maxCol) {
    // maximum column counter
    maxCol = col;
    row++;
    col = 0;
}
    maxRow = row;

double sum = 0;
for(int i = 0; i < maxRow; i++) {
    sum += atof(myArray[i][1]);
}
    cout << "sum = " << sum << endl;
#ifdef DEBUG
    cout << "maxRow = " << maxRow <<", maxCol = " << maxCol << endl;
    for(int i = 0; i < maxRow; i++) {
        for(int j = 0; j < maxCol; j++) {
            cout << myArray[i][j] << " ";
        }
        cout << endl;
    }
#endif
return ret;
```

Chapter 3

Classes

3.1 Complex numbers

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

```cpp
/*
** 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`.

```cpp

*/
** 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
```

### 3.1.1 Solution

```cpp  
/**  
** Name: complex.cxx  
** Author: Leo Liberti  
** Purpose: implementation of a complex numbers class  
** Source: GNU C++  
** History: 061019 work started
```
/*
#include<iostream>
#include<string>
#include<sstream>
#include<cstdlib>
#include<cmath>
#include "complex.h"

const char charPlusOp = '+';
const char charMinusOp = '-';
const char charIValue = 'i';

Complex::Complex() : real(0.0), imag(0.0) { }
Complex::Complex(double re, double im) : real(re), imag(im) { }
Complex::~Complex() { }

double Complex::getReal(void) {
    return real;
}

double Complex::getImaginary(void) {
    return imag;
}

void Complex::setReal(double re) {
    real = re;
}

void Complex::setImaginary(double im) {
    imag = im;
}

void Complex::fromString(const std::string& complexString) {
    using namespace std;
    string realString;
    stringstream stringBufReal;
    string imagString;
    stringstream stringBufImag;

    int opPos = complexString.find(charPlusOp);
    int iPos;
    if (opPos != complexString.npos) {
        // deal with cases re + im i
        realString = complexString.substr(0, opPos);
        stringBufReal.str(realString);
        stringBufReal >> real;
        imagString = complexString.substr(opPos + 1);
        stringBufImag.str(imagString);
        stringBufImag >> imag;
        if (imag == 0) {
            // case re + i
            imag = 1;
        }
    } else {
        opPos = complexString.find(charMinusOp);
        if (opPos != complexString.npos && opPos > 0) {
            // deal with cases re - im i
            realString = complexString.substr(0, opPos);
            stringBufReal.str(realString);
            stringBufReal >> real;
            imagString = complexString.substr(opPos + 1);
            stringBufImag.str(imagString);
            stringBufImag >> imag;
            if (imag == 0) {
                // case re - i
                imag = -1;
            }
            imag = -imag;
        } else {
            opPos = complexString.find(charIValue);
            if (opPos != complexString.npos) {
                // deal with case im i
                imagString = complexString.substr(0, opPos);
                stringBufImag.str(imagString);
                stringBufImag >> imag;
            } else {
                // deal with case re
                stringBufReal.str(complexString);
                stringBufReal >> real;
            }
        }
    }
}
*/
Complex Complex::operator+(Complex& theComplex) {
    return Complex(real + theComplex.getReal(), imag + theComplex.getImaginary());
}

Complex Complex::operator-(Complex& theComplex) {
    return Complex(real - theComplex.getReal(), imag - theComplex.getImaginary());
}

Complex Complex::operator*(Complex& theComplex) {
    double re = real * theComplex.getReal() - imag * theComplex.getImaginary();
    double im = imag * theComplex.getReal() + real * theComplex.getImaginary();
    return Complex(re, im);
}

Complex Complex::operator/(Complex& theComplex) {
    double c = theComplex.getReal();
    double d = theComplex.getImaginary();
    double denom = c*c + d*d;
    double re;
    double im;
    if (denom == 0) {
        re = 1e100;
        im = 1e100;
    } else { 
        re = (real*c + imag*d) / denom;
        im = (imag*c - real*d) / denom;
    }
    return Complex(re, im);
}

std::ostream& operator<<(std::ostream& out, Complex& theComplex) {
    using namespace std;
    double re = theComplex.getReal();
    double im = theComplex.getImaginary();
    if (im > 0) {
        if (im != 1) {
            out << re << " + " << im << "i";
        } else {
            out << re << "+ i";
        }
    } else if (im < 0) {
        if (im != -1) {
            out << re << " - " << fabs(im) << "i";
        } else {
            out << re << " - i";
        }
    } else if (re == 0) {
        out << im << "i";
    } else if (im == 0) {
        out << re;
    }
    return out;
}

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>

Virtual inheritance
class VirtualBase {
public:
    virtual ~VirtualBase() = 0;
    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*'

Virtual inheritance
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]

3.2.1 Solution

The problem on line 31 `Derived* d = v;` is given by the fact that we are attempting to cast a pure virtual base class into a derived class at compile-time: this is not possible. Instead, we have to employ the `dynamic_cast` operator and replace line 31 with:

```cpp
Derived* d = dynamic_cast<Derived*>(v);
```

This fixes the problem.

3.3 Virtual and nonvirtual inheritance

What is the output of the following code?

```cpp
#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();
```
3.3.1 Solution

The output is

A::f
A::g
A::f
B::g

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:

Nonvirtual base class destructor
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?

3.4.1 Solution

The main code allocates a Derived object on the heap but uses a Base pointer to store its address, exploiting polymorphism. Since the destructor is not declared virtual, the binding occurs at compile-time: since thePtr is of type Base*, the destructor called is the one for the Base class even though thePtr actually points to an object of type Derived.

As is said in the “C++ Gotchas” book by S. Dewhurst (gotcha #70), this bug causes unpredictable behaviour.

Chances are you’ll just get a call of the base class destructor for the derived class object: a bug. But the compiler may decide to do anything else it feels like (dump core? send nasty email to your boss? sign you up for a lifetime subscription to “This week in Object-Oriented COBOL”?).
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.

```c++
// 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;
    }
}
```

4.1.1 Solution

If we just compile and run the program, we obtain a lot of output, structured more or less as follows:

```
3.5
2.5
1.5
3.60739e-313
0
[...]
0
3.66616e-307
5.09279e-313
nan
nan
```
Exercises

Introduction to C++

L. Liberti

Segmentation fault

This is usually known as garbage output. The first three lines are the expected output, but we fail to understand why did the program not stop on the second loop when it was told. Furthermore, why the segmentation fault?

We build the program with the debugging flag set

```
c++ -g -o segmentationfault segmentationfault.cxx
```

and then debug it using valgrind:

```
valgrind -q ./segmentationfault
```

We obtain the usual garbage interspersed with meaningful valgrind messages. We can run it again saving the standard error stream to the standard output and redirecting it through a pager:

```
valgrind -q ./segmentationfault 2>&1 — less
```

valgrind's messages are:

```
==31179== Invalid write of size 8
==31179== at 0x8048681: main (segmentationfault.cxx:6)
==31179== Address 0x426A030 is 0 bytes after a block of size 8 alloc'd
==31179== at 0x401A878: operator new(unsigned) (vg_replace_malloc.c:163)
==31179== by 0x804864D: main (segmentationfault.cxx:4)
==31179==
==31179== Invalid read of size 4
==31179== at 0x804869E: main (segmentationfault.cxx:9)
==31179== Address 0x426A03C is 12 bytes after a block of size 8 alloc'd
==31179== at 0x401A878: operator new(unsigned) (vg_replace_malloc.c:163)
==31179== by 0x804864D: main (segmentationfault.cxx:4)
==31179==
```

The first error, Invalid write, is very serious. It means we are writing on heap memory we had not allocated. This usually results immediately in unpredictable behaviour and the fatal segmentation fault error. We can find out at which step of the loop the error arises by inserting a print statement

```
cout << "*** " << i << endl;
```

before line 6, rebuilding, and running valgrind again. We see immediately that the Invalid write occurs when \( i \) has value 1. It is important to remove the temporary print statement in order to keep the line numbers valid. The fact that \( d[0] \) exists in memory but \( d[1] \) is invalid should immediately tell us that we failed to allocate enough memory. In particular, we issued the C++ statement

```
double* d = new double;
```

which reserves space on the heap for just one double floating point number. What we really wanted to do, in fact, was to have space for an array of size 3:

```
double* d = new double[3];
```

We now rebuild and re-run the program. Again we get garbage output. valgrind reveals that the first error has been fixed:

```bash
==31348== Invalid read of size 4
==31348== at 0x804869E: main (segmentationfault.cxx:9)
==31348== Address 0x426A038 is 8 bytes after a block of size 8 alloc'd
==31348== at 0x401A878: operator new(unsigned) (vg_replace_malloc.c:163)
==31348== by 0x804864D: main (segmentationfault.cxx:4)
==31348==
==31348== Process terminating with default action of signal 11 (SIGSEGV)
==31348== Bad permissions for mapped region at address 0x4262FFC
==31348== at 0x804869E: main (segmentationfault.cxx:9)
```
Exercises

Introduction to C++

The Invalid read error is not as serious as the Invalid write, but almost as bad. It means that we are reading from unallocated memory.

To see what is happening we launch ddd and we set a breakpoint at line 9

```
break segmentationfault.cxx:9
```

Then we start debugging run. Since the problem is that the loop does not seem to terminate as it should, we want to keep track of the counter i. With

```
display i
```

we can have i's value printed after each step next. After a few steps, we find that i has value 0. Instead of terminating, the loop continues with i set at 4294967295. Naturally, the condition i >= 0 is still verified so that's the reason why the loop continues. The strange value in i is due to the fact that the counter was declared unsigned int, which means that it can never take negative values — in other words, 0 is the minimum value that i can take. When an unsigned int is set at 0 (as i in our case) and then a decrement operator is applied to it (i-- within the for statement), it is set to “one less than the minimum value”, which by definition is taken to be the maximum value (i.e. a large, positive value). Thus the bug is explained. It can be fixed by taking away the unsigned qualifier.

We rebuild, re-run the program and find that it works without any trouble whatsoever:

```
3.5
2.5
1.5
```

The third bug is much harder to find, since the behaviour of the program is now correct. We can identify it by enclosing the whole program into an endless loop:

```cpp
int main(int argc, char** argv) {  
    using namespace std;  
    while(true) {  
        double* d = new double[3];  
        for(int i = 0; i < 3; i++) {  
            d[i] = 1.5 + i;  
        }  
        for(int i = 2; i >= 0; i--) {  
            cout << d[i] << endl;  
        }  
    }  
}
```

We now build and run the program, and in another shell we launch the shell command

```
top -d 1
```

In the last column of this window we find some program names which are running on the computer. We find the line relative to segmentationfault:
If we let our program run for a few seconds (minutes) we notice that on the column marked \%MEM the percentage of memory dedicated to segmentationfault always increases. If we let this program run for a long time, it gobbles up the whole computer memory. On certain operating systems, this amounts to freezing the computer dead. On others, the process is automatically killed. In any case, we do not want any program of ours to follow this behaviour, called memory leak. It is easy to find that the cause of this bug is that we forgot to deallocate memory after the allocation. We insert the line \texttt{delete [] d;} right before the end of the program to fix this bug.

### 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?

```cpp
#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;
}
```
4.2.1 Solution

First, we compile `pointerbug.cxx` using the debug `-g` option:

```
c++ -g -o pointerbug pointerbug.cxx
```

To debug with `ddd`, we run: `ddd pointerbug`. We set command line arguments `set args e test` and set a breakpoint inside the `while` loop `break pointerbug.cxx:26`. We start the debugging session `run` which stops at the required breakpoint, on the line `*wordPtr = ' '`. Putting the mouse pointer on the symbol `wordPtr` reveals on the window status line that the array pointed at by `wordPtr` is "test". From here on, we debug one statement at a time by repeatedly issuing the command `next`. See Fig. 4.1.

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

// skip characters in the word and delete them, until we find the Char
while(*wordPtr != theChar)
{
  if(*wordPtr == '
')
    wordPtr+=

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

![Figure 4.1: The ddd frontend to the gdb debugger.](image)

At line 27, the array has become "est" because the 't' character has been replaced by a space (this is the effect of line 26). A `next` command later, we are on line 25 (the `while` test). As `theChar` is equal to 'e' and `*wordPtr` is also 'e', the test fails and the loop is exited. Three more `next` commands exit the debugging session without detecting any problem. We now change the command line arguments: `set args y test` and re-debug the program. We notice that, as the array pointed at by `wordPtr`, namely "test", does not contain the 'y' character, the `while` condition does not fail before the end of the array. It is not difficult to imagine that as the loop is repeated, characters in memory after the `wordPtr` array are set to contain spaces (as per line 26), so that eventually the whole memory adjacent to `wordPtr` is erased. As the variables are stored from the stack, the memory "after" the `wordPtr` array...
contains the values of the variables declared before the `wordPtr` array, namely the `testInt` variable. This is the reason why the program behaves incorrectly and unpredictably. This type of bug is a memory bug known as stack corruption, and it was engendered by the array bound overflow of the `wordPtr` pointer being increased past its legal limit.

The `valgrind` command-line debugging tool is usually very helpful in tracing memory bugs; in this case, however, its output is somewhat obscure. Running `valgrind -q ./pointerbug test` yields

```
==28373== Conditional jump or move depends on uninitialised value(s)
==28373==    at 0x8048752: main (pointerbug.cxx:25)
538976288
```

Line 25 is the `while` condition. What `valgrind` is telling us is that at a certain point of the program execution, the test `(*wordPtr != theChar)` occurs over an uninitialised portion of memory. Now, `theChar` is certainly initialised (line 17), so this leaves the `*wordPtr` as the sole possible source of the problem. We can shed further light on this matter by printing a loop counter (so we know when the problem occurs). We do this by inserting the C++ line `cout << (int) (wordPtr - buffer) << endl;` after line 27. The output is now,

```
1
[...]
21
22
23
24
==28407== Conditional jump or move depends on uninitialised value(s)
==28407==    at 0x8048783: main (pointerbug.cxx:25)
25
26
27
538976288
```

At this point we immediately realize that the loop is counting past the end of string, as our given string `test` is only 4 characters long, and the program stops well past it. It is also interesting to notice that the first 25 bytes after `buffer` are all initialised parts of the stack: this is consistent with `buffer` being 20 bytes long (i.e. the value of `bufSize`), `theChar` occupying only 1 byte of memory, and `testInt` being a 32 bit (i.e. 4 bytes) integer of type `int`.

One further point of interest is, why should the program terminate at all? Supposing there is no ‘y’ character in the memory starting from `buffer`, the `while` condition would never be satisfied. This is indeed the case, but the fact is that the overall distribution of byte values in memory is close to uniform, so that it is likely that each ASCII value from 0 to 255 will eventually be found in such a large chunk of memory.

In order to fix the bug, it suffices change the `while` condition to

```
(*wordPtr != theChar && wordPtr != '\0' && (int) (wordPtr - buffer) < bufSize)
```

so that the end-of-char-array delimiter (the NULL byte), as well as the maximum array size checking condition may also be able to stop the loop.

To answer the last part of the question, the value 1 contained in `testInt` will not be affected by the bug described above if `testInt` is declared after `buffer`, because the statement `*wordPtr = ' ';` will then overwrite memory after `buffer` (which, because the variables are on the stack, refers to variables declared before `buffer`).
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?

```cpp
#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;
}

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

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;
}
```

4.3.1 Solution

The ValueContainer::myInt pointer is always initialized to the address of a local variable, namely the i passed to the createContainer(int) method. Local variables are allocated on the stack and are
deallocated as soon as their scope is closed. In this case, the `i` variable local to `createContainer` is
deallocated at the end of the `createContainer` method, hence the pointer which has been stored in the
`ret` pointer to `ValueContainer` object points to an unused area of the stack. Chances are this stack area
will be overwritten when functions are called and returned from, so the output is unpredictable.

There are several ways to fix this bug. The most direct is to replace the statement `ret->set(&i);` with `ret->set(new int(i));` and to add the statement `delete theInt;` in the class destructor.

A more meaningful bugfix would involve recognizing that the semantics of the class is wrong. Set/get
pairs should behave with the same data, and the fact that the integer data within the class (theInt) has
a name which does not remind of a pointer are strong hints that the real bug was to design a class that
stores an integer pointer when all that was needed was simply an integer. Thus the “correct correction” is to make all changes so that `theInt` is an `integer variable` instead of a pointer to integer.

### 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.

```cpp
#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;
}
```

#### 4.4.1 Solution

The problem lies in the fact that erasing invalidates all subsequent iterators. In order to erase some
iterators subject to certain conditions, one must first remove them to the end of the vector and then
erase the end.

```cpp
#include<vector>
#include<iostream>
#include<algorithm>

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 theEnd = remove(theVector.begin(), theVector.end(), 1);
theVector.erase(theEnd, theVector.end());
for(int i = 0; i < theVector.size(); i++) {
    cout << theVector[i] << endl;
}
return ret;
}
Erasing elements from a vector
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. This application can be used to trace website maps or relations between websites (which often underline relations between the corresponding corporations); see for examples the maps of the IEEE (Fig. 5.1) and PSA (Fig. 5.2) websites.

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

Figure 5.1: The IEEE website map.
Figure 5.2: The PSA website map.

0 \rightarrow 1;
1 \rightarrow 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:

```c
#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:

```c
#include<string>
#include<cstdio>
#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 and verify that the output is:
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. 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.

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 ::

```
returnType ClassName::MethodName(args) { 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 strings 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. 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:

```cpp
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:

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

```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
```

### 5.3.1 Solution

### 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`. 

*The HTMLPage class*
Exercises

Introduction to C++ L. Liberti

• 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: 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
```

5.4.1 Solution

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, to allow access to the links contained therein, and to return the hostname of the URL.

• 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.

```cpp
#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.

```cpp
#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* 44
• 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: 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

5.5.1 Solution

5.6 The Vertex interface and VertexURL implementation

Design and implement a class VertexURL inheriting from the pure virtual class Vertex: 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 (and be able to set and get): its own integer ID, the IDs of the vertices adjacent to it, a text label associated to the vertex. 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: Run the program

  ./vurldriver http://kelen.polytechnique.fr

  and verify that the output is

  vertex 0: star 1 2.

5.6.1 Solution

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: and verify that the output is:

# graphviz output by WET (L. Liberti 2006)

digraph www_1158015497 {

The Arc and Digraph classes
### Exercises

Introduction to C++  L. Liberti

```plaintext
0 [ label = "name1" ];
1 [ label = "name2" ];
2 [ label = "name3" ];
3 [ label = "Thu Jan 10 19:24:53 2008", color = red ];
  1 -> 2;
  1 -> 3;
  2 -> 1;
  3 -> 3;
}
```

5.7.1 Solution

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:

  ```c
  // 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`.

**Putting it all together: main()**
Test your WET application by running

```bash
./wet http://www.enseignement.polytechnique.fr/profs/informatique/Leo.Liberti/test.html
```
and verifying that the output is

```
# graphviz output by WET (L. Liberti 2006)
digraph www_1199989821 {
  0 [ label = "www.enseignement.polytechnique.fr" ];
  1 [ label = "www.enseignement.polytechnique.fr" ];
  2 [ label = "www.enseignement.polytechnique.fr" ];
  3 [ label = "Thu Jan 10 19:30:21 2008", 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. 5.3, left. Reducing the depth level to 2 yields Fig. 5.3, right.

![Figure 5.3: The neighbourhoods graphs (depth 4, left; depth 2, right).](image)

5.8.1 Solution
Putting it all together: main()
Chapter 6

Exam questions

This section contains a set of typical exam questions.

6.1 Debugging question 1

The following code causes some compilation errors. Find them and correct them.

```cpp
// this code contains errors
#include <iostream>
#include <cassert>

int main(int argc, char** argv) {
    int ret = 0;
    if (argc < 2) {
        cerr << "error: need an integer on command line";
        return 1;
    }
    int theInt = atoi(argv[1]);
    if (isPrime(theInt)) {
        cout << theInt << "is a prime number!" >> endl;
    } else {
        cout << theInt << "is composite" << endl;
        return ret;
    }
}

bool isPrime(int num) {
    assert(num >= 2);
    if (num == 2) {
        return true;
    } else if (num % 2 == 0) {
        return false;
    } else {
        bool prime = true;
        int divisor = 3;
        int upperLimit = sqrt(num) + 1;
        while (divisor <= upperLimit) {
```

---

```cpp
```
### 6.1.1 Solution

```cpp
#include <iostream>
#include <cassert>
#include <cmath>

bool isPrime (int num) {
    assert(num >= 2);
    if (num == 2) {
        return true;
    } else if (num % 2 == 0) {
        return false;
    } else {
        bool prime = true;
        int divisor = 3;
        int upperLimit = (int) (sqrt(num) + 1);
        while (divisor <= upperLimit) {
            if (num % divisor == 0) {
                prime = false;
            }
            divisor +=2;
        }
        return prime;
    }
}

int main(int argc, char** argv) {
    using namespace std;
    int ret = 0;
    if (argc < 2) {
        cerr << "error: need an integer on command line" << endl;
        return 1;
    }
    int theInt = atoi(argv[1]);
    if (isPrime(theInt)) {
        cout << theInt << " is a prime number!" << endl;
    } else {
        cout << theInt << " is composite" << endl;
    }
    return ret;
}
```

**Debugging question 1**
The output is

1231 is a prime number!

6.2 Debugging question 2

The following code causes some compilation errors. Find them and correct them.

```cpp
// this code contains errors
#include <iostream>

class A {
    A() {
        std::cout << "constructing object of class A" << std::endl;
    }
    ~A() {
        std::cout << "destroying object of class A" << std::endl;
    }
    setDatum(int i) {
        theDatum = i;
    }
    getDatum(void) {
        return theDatum;
    }
private:
    int theDatum;
}

int main(int argc, char** argv) {
    A a;
    a.SetDatum(5);
    std::cout << a.GetDatum() << std::endl;
    return;
}
```

What is the corrected program output when executed with `./debug2`?

6.2.1 Solution

```cpp
#include <iostream>

class A {
public:
    A() {
        std::cout << "constructing object of class A" << std::endl;
    }
    ~A() {
        std::cout << "destroying object of class A" << std::endl;
    }
    void setDatum(int i) {
```
theDatum = i;
}
int getDatum(void) {
    return theDatum;
}
private:
    int theDatum;
};

int main(int argc, char** argv) {
    A a;
    a.setDatum(5);
    std::cout << a.getDatum() << std::endl;
    return 0;
}

The output is

constructing object of class A
5
destroying object of class A

6.3 Debugging question 3

The following code looks for a word in a text file and replaces it with another word:

// this code contains errors
// [search and replace a word in a text file]
#include<iostream>
#include<fstream>
#include<string>
int main(int argc, char** argv) {
    using namespace std;
    if (argc < 4) {
        cout << "need a filename, a word and its replacement on cmd line" << endl;
        return 1;
    }
    ifstream ifs(argv[1]);
    if (!ifs) {
        cout << "cannot open file " << argv[1] << endl;
        return 2;
    }
    string s;
    while(!ifs.eof()) {
        ifs >> s;
        if (ifs.eof()) {
            break;
        }
        if (s == argv[2]) {
            cout << argv[3];
        } else {
            break;
        }
    }
    return 0;
}
cout << s;
}
cout << " ";
}
cout << endl;
ifs.close();
return 0;
}

Test it on the following file, called \texttt{story.txt}, with the command \	exttt{./debug3 story.txt wolf teddy-bear}:

This is just a stupid little story with no meaning whatsoever. The wolf came out of his den and chased little red hood, found her and pounced on her. Little red hood cried, "the wolf!", just stepped aside and the wolf cracked his neck on the pavement, his brain in pulp. She was to hear of him no more.

The output is:

This is just a stupid little story with no meaning whatsoever. The teddy-bear came out of his den and chased little red hood, found her and pounced on her. Little red hood cried, "the wolf!", just stepped aside and the teddy-bear cracked his neck on the pavement, his brain in pulp. She was to hear of him no more.

Note that one occurrence of the word "wolf" was not actually changed to "teddy-bear". Fix this problem.

\subsection*{6.3.1 Solution}

```cpp
// search and replace a word in a text file
#include<iostream>
#include<fstream>
#include<string>
#include<sstream>

void replaceInString(std::string& s, std::string needle, std::string replace) {
    int m = needle.length();
    int n = s.length();
    for(int i = 0; i < n; i++) {
        if(s.substr(i, m) == needle) {
            s.replace(i, m, replace, 0, replace.length());
        }
    }
}

int main(int argc, char** argv) {
    using namespace std;
    if (argc < 4) {
        cout << "need a filename, a word and its replacement on cmd line" << endl;
        return 1;
    }
    string filename = argv[1];
    string word = argv[2];
    string replacement = argv[3];
    ifstream ifs(filename);
    string line;
    while (getline(ifs, line)) {
        replaceInString(line, word, replacement);
        cout << line;
6.4 Debugging question 4

Compile and run the following code:

```cpp
#include<iostream>
#include<string>

std::string method(std::string& s) {
    s = s.substr(1, s.npos);
    return s;
}

int main(int argc, char** argv) {
    using namespace std;
    string s("ciao");
    cout << s << endl;
    while(true) {
        string& t = s;
        // if you change t with s in the snippet, the code has the same effect
        method(s);
        cout << s << endl;
    }
    return 0;
}
```

Debugging question 4
```cpp
    t = method(t);
    if (t.length() == 0) {
        break;
    }
    cout << t << endl;
    /////////////// end snippet ///////////
}
return 0;
```

Now examine the part of the code marked “snippet”: it has the curious property that if you replace any occurrence of the `t` variable symbol with the `s` variable symbol, you still obtain the same output. Explain why.

### 6.4.1 Solution

Since `method` is passed its argument by reference, it changes the original parameter. `t` has type `string&` so it is just a reference to `s`; so all along `t` and `s` actually share the same memory.

### 6.5 Debugging question 5

The following code contains a simple list implementation and a `main()` which creates a list, prints it, finds a node and removes it, then prints the list again.

```cpp
#include <iostream>

class Node {
public:
    // constructors / destructor
    Node() : next(NULL), previous(NULL) { }
    Node(int a) : next(NULL), previous(NULL), data(a) { }
~Node() {
    if (next) {
        delete next;
    }
}

    // set/get interface
    void setData(int a) {
        data = a;
    }
}
```

`Debugging question 5`
int getData(void) {
    return data;
}

void setNext(Node* theNext) {
    next = theNext;
}

Node* getNext(void) {
    return next;
}

void setPrevious(Node* thePrevious) {
    previous = thePrevious;
}

Node* getPrevious(void) {
    return previous;
}

// list capabilities

// return true if node is the first of the list, false otherwise
bool isFirst(void) {
    return !previous;
}

// return true if node is the last of the list, false otherwise
bool isLast(void) {
    return !next;
}

// return the size of the sublist starting from this node
int size(void) {
    Node* t = this;
    int ret = 1;
    while(!t->isLast()) {
        t = next;
        ret++;
    }
    return ret;
}

// append a new given node at the end of the list
void append(Node* theNext) {
    Node* t = this;
    while(!t->isLast()) {
        t = next;
    }
    t->setNext(theNext);
    theNext->setPrevious(t);
}

// create a new node with value 'a' and append it at the end of the list
void appendNew(int a) {
    Node* t = this;
    while(!t->isLast()) {
        t = next;
    }

Debugging question 5
Node* theNewNode = new Node(a);
t->setNext(theNewNode);
theNewNode->setPrevious(t);

// remove this node from the list
void erase(void) {
    previous->setNext(next);
next->setPrevious(previous);
}

// replace this node with a given node
void replaceWith(Node* replacement) {
    previous->setNext(replacement);
next->setPrevious(replacement);
}

// find first node with a specified value in sublist starting from this node
// return NULL if not found
Node* find(int needle) {
    if (data == needle) {
        return this;
    }
    Node* t = this;
    while(!t->isLast()) {
        t = next;
        if (t->getData() == needle) {
            return t;
        }
    }
    return NULL;
}

// print the data in the sublist starting from this node
void print(void) {
    Node* t = this;
    while(!t->isLast()) {
        std::cout << t->getData() <<", ";
t = next;
    }
    std::cout << t->getData() << std::endl;
}

protected:
    // pointer to next node in list
    Node* next;

    // pointer to previous node in list
    Node* previous;

private:
    // the integer data stored in the node
    int data;
int main(int argc, char** argv) {
    using namespace std;
    int c = 1;
    // create a new list
    Node start(c);
    // add 10 nodes to the list
    while(c < 10) {
        c++;
        start.appendNew(c);
    }
    // print the list
    start.print();
    // find the node with value 5
    Node* t = start.find(5);
    // erase this node
    t->erase();
    // print the list again
    start.print();

    return 0;
}

The code contains:

1. one compilation error;
2. some run-time errors;
3. a memory leak.

Correct the compilation error, then compile the code — it should simply hang. Use the ddd debugger to find the run-time errors, find all instances thereof, and correct them (remember to compile with the -g option for debugging). Satisfy yourself that the code output is:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10
1, 2, 3, 4, 6, 7, 8, 9, 10

Now use the valgrind memory debugger to check for memory leaks. Run the command

`valgrind --leak-check=yes ./debug5`

and using its output find and fix the error.

6.5.1 Solution

The compilation error is `t.getData()`, it should be `t->getData()` instead. The run-time error is caused by the fact that the many statements `t = next;` simply re-assign the constant value of `next` to `t` at every iteration of the while loops. Since `next` is only `isFinal()` when the list has 2 elements, the code works for lists of 1 and 2 nodes, but fails to terminate for lists of more than 2 nodes. This error can be
corrected by simply replacing \texttt{t = next;} with \texttt{t = t->getNext();} throughout the code. The memory leak is caused by the \texttt{erase()} method, which does not deallocate the node itself before terminating. This can be fixed by adding the lines

\begin{verbatim}
next = NULL;
delete this;
\end{verbatim}

at the end of the \texttt{erase()} method. The same goes for the \texttt{replaceWith()} method which, although is not used in \texttt{main()}, would nonetheless cause another memory leak. The corrected code is as follows:

\begin{verbatim}
#include <iostream>
class Node {
public:
    // constructors / destructor
    Node() : next(NULL), previous(NULL) { }
    Node(int a) : next(NULL), previous(NULL), data(a) { }
    ~Node() {
        if (next) {
            delete next;
        }
    }

    // set/get interface
    void setData(int a) {
        data = a;
    }
    int getData(void) {
        return data;
    }
    void setNext(Node* theNext) {
        next = theNext;
    }
    Node* getNext(void) {
        return next;
    }
    void setPrevious(Node* thePrevious) {
        previous = thePrevious;
    }
    Node* getPrevious(void) {
        return previous;
    }

    // list capabilities

    // return true if node is the first of the list, false otherwise
    bool isFirst(void) {
        return !previous;
    }

    // return true if node is the last of the list, false otherwise
    bool isLast(void) {

Debugging question 5
\end{verbatim}
return !next;
}

// return the size of the sublist starting from this node
int size(void) {
    Node* t = this;
    int ret = 0;
    while(!t->isLast()) {
        t = t->getNext();
        ret++;
    }
    return ret;
}

// append a new given node at the end of the list
void append(Node* theNext) {
    Node* t = this;
    while(!t->isLast()) {
        t = t->getNext();
    }
    t->setNext(theNext);
    theNext->setPrevious(t);
}

// create a new node with value 'a' and append it at the end of the list
void appendNew(int a) {
    Node* t = this;
    while(!t->isLast()) {
        t = t->getNext();
    }
    Node* theNewNode = new Node(a);
    t->setNext(theNewNode);
    theNewNode->setPrevious(t);
}

// remove this node from the list
void erase(void) {
    previous->setNext(next);
    next->setPrevious(previous);
    next = NULL;
    delete this;
}

// replace this node with a given node
void replaceWith(Node* replacement) {
    previous->setNext(replacement);
    next->setPrevious(replacement);
    next = NULL;
    delete this;
}

// find first node with a specified value in sublist starting from this node
// return NULL if not found
Node* find(int needle) {

if (data == needle) {
    return this;
}
Node* t = this;
while(!t->isLast()) {
    t = t->getNext();
    if (t->getData() == needle) {
        return t;
    }
}
return NULL;

// print the data in the sublist starting from this node
void print(void) {
    Node* t = this;
    while(!t->isLast()) {
        std::cout << t->getData() << " ",
        t = t->getNext();
    }
    std::cout << t->getData() << std::endl;
}

protected:
    // pointer to next node in list
    Node* next;

    // pointer to previous node in list
    Node* previous;

private:
    // the integer data stored in the node
    int data;
};

int main(int argc, char** argv) {
    using namespace std;
    int c = 1;
    // create a new list
    Node start(c);
    // add 10 nodes to the list
    while(c < 10) {
        c++;
        start.appendNew(c);
    }
    // print the list
    start.print();
    // find the node with value 5
    Node* t = start.find(5);
    // erase this node
    t->erase();
    // print the list again
    start.print();
}

Debugging question 5
return 0;
}

6.6 Debugging question 6

Compile the following code with cc -o schnitzi schnitzi.c and run it with ./schnitzi 36 to test it, and try other numbers. Explain what it does and why.

```c
#include<stdio.h>
main(l,i,I)char**i;{l/=!(l>(I=atoi(*++i))||fork()&main(l+1,i-1)||I%l);
return printf("%d\n",l);}
```

6.6.1 Solution

Extremely hard question! Look for “schnitzi obfuscated” on Google.

6.7 Specification coding question 1

Write the implementation of the following SimpleString class.

```c
/* name: simplestring.h
 ** author: L. Liberti
 */

#ifndef _SIMPLESTRINGH
#define _SIMPLESTRINGH

#include<iostream>

class SimpleString {
 public:
  // default constructor: initialise an empty string
  SimpleString();

  // initialise a string from an array of characters
  SimpleString(char* initialiseFromArray);

  // destroy a string
  ~SimpleString();

  // return the size (in characters)
  int size(void) const;

  // return the i-th character (overloading of [] operator)
  char& operator[](int i);

  // copy from another string
```
void copyFrom(SimpleString& s);

// append a string to this
void append(SimpleString& suffix);

// does this string contain needle?
bool contains(SimpleString& needle) const;

private:

int theSize;
char* buffer;
int theAllocatedSize;
static const int theMinimalAllocationSize = 1024;
char theZeroChar;
};

std::ostream& operator<<(std::ostream& out, SimpleString& s);
#endif

Test it with with the following main program:

/* name: ssmain.cxx
   author: Leo Liberti
*/

#include<iostream>
#include "simplestring.h"

int main(int argc, char** argv) {
    using namespace std;
    SimpleString s("I'm ");
    SimpleString t("very glad");
    SimpleString u("so tired I could pop");
    SimpleString v("very");
    cout << s << endl;
    cout << t << endl;
    cout << u << endl;
    SimpleString w;
    w.copyFrom(s);
    s.append(t);
    w.append(u);
    cout << s << endl;
    cout << w << endl;
    if (s.contains(v)) {
        cout << "s contains very" << endl;
    } else {
        cout << "s does not contain very" << endl;
    }
    if (w.contains(v)) {
        cout << "w contains very" << endl;
    } else {
        cout << "w does not contain very" << endl;
    }
}
and check that the output is as follows.

I'm very glad so tired I could pop

I'm very glad I'm so tired I could pop

s contains very
w does not contain very

Write also a makefile for building the object file `simplestring.o` and the executable `ssmain`.

### 6.7.1 Solution

```cpp
/* name: simplestring.hxx
 ** author: L. Liberti
 */

#include<iostream>
#include<cstring>
#include "simplestring.h"

inline int max(int a, int b) {
    if (a >= b) {
        return a;
    } else {
        return b;
    }
}

SimpleString::SimpleString() : buffer(NULL), theSize(0), theAllocatedSize(0), theZeroChar(\0) { }

SimpleString::SimpleString(char* initialiseFromArray) {
    using namespace std;
    theSize = strlen(initialiseFromArray);
    int s = max(theSize, theMinimalAllocationSize);
    buffer = new char [s];
    theAllocatedSize = s;
    strcpy(buffer, initialiseFromArray);
}

SimpleString::~SimpleString() {
    if (buffer) {
        delete [] buffer;
    }
```
int SimpleString::size(void) const {
    return theSize;
}

char& SimpleString::operator[](int i) {
    if (i >= theSize || i < 0) {
        return theZeroChar;
    } else {
        return buffer[i];
    }
}

void SimpleString::copyFrom(SimpleString& s) {
    if (buffer) {
        delete[] buffer;
    }
    int ss = s.size();
    int ssalloc = max(ss, theMinimalAllocationSize);
    buffer = new char[ssalloc];
    for(int i = 0; i < ss; i++) {
        buffer[i] = s[i];
    }
    theSize = ss;
}

void SimpleString::append(SimpleString& suffix) {
    int ss = suffix.size();
    if (ss > theAllocatedSize - theSize) {
        // suffix does not fit, reallocate
        char* saveBuffer = buffer;
        int s = theSize + ss + 1;
        buffer = new char[s];
        theAllocatedSize = s;
        strcpy(buffer, saveBuffer);
        delete[] saveBuffer;
    }
    for(int i = 0; i < ss; i++) {
        buffer[i + theSize] = suffix[i];
    }
    theSize = theSize + ss;
    buffer[theSize] = '\0';
}

bool SimpleString::contains(SimpleString& needle) const {
    int s = needle.size();
    bool found;
    for(int i = 0; i < theSize - s; i++) {
        found = true;
        for(int j = i; j < s + i; j++) {
            if (buffer[j] != needle[j - i]) {
                found = false;
                break;
            }
        }
        if (found) {
            return true;
        }
    }
    return false;
}
6.8 Specification coding question 2

Write the implementation of the following IntTree class (a tree data structure for storing integers).

```cpp
/* Name: inttree.h
 * Author: Leo Liberti
 */

#ifndef _INTTREEH
#define _INTTREEH

#include<vector>

class IntTree {
public:
    // constructs an empty tree
    IntTree();

    // destroys a tree
    ~IntTree();

    // ... (other member functions)

private:
    // ... (private data)

};

```

The makefile is as follows.

```bash
all: ssmain

ssmain: ssmain.cxx simplestring.o
$(CXX) -o ssmain ssmain.cxx simplestring.o

simplestring.o: simplestring.h simplestring.cxx
$(CXX) -c simplestring.cxx

clean:
$(RM) -f simplestring.o ssmain
```
// add a new subnode with value n
void addSubNode(int n);

// get the number of subtrees
int getNumberOfSubTrees(void);

// get a pointer to the i-th subtree
IntTree* getSubTree(int i);

// removes a subtree without deleting it (and returning it)
IntTree* removeSubTree(int i);

// adds a tree as a subtree of the current tree
void addSubTree(IntTree* t);

// get pointer to the parent tree
IntTree* getParent(void);

// set parent pointer
void setParent(IntTree* p);

// get value of current tree
int getValue(void);

// set value in current tree
void setValue(int n);

// find the first subtree containing given value by depth-first search
// and put its pointer in 'found'
// PRECONDITION: found must be NULL on entry
// POSTCONDITION: if found is NULL on exit, value was not found
void findValue(int n, IntTree* &found);

// print a tree to standard output
void print(void);

protected:

// value stored in the node (tree)
int value;

// pointer to parent tree
IntTree* parent;

// vector of subtree pointers
std::vector<IntTree*> subTree;

private:

// iterator sometimes used in methods
std::vector<IntTree*>::iterator stit;
}
Test it with with the following main program:

```cpp
/* name: itmain.cxx
   author: Leo Liberti */
#include <iostream>
#include "inttree.h"

int print(IntTree& t) {
    using namespace std;
    cout << "T = ";
    t.print();
    cout << endl;
}

int main(int argc, char** argv) {
    using namespace std;
    IntTree myTree;
    myTree.setValue(0);
    myTree.addNode(1);
    myTree.addNode(2);
    myTree.addNode(3);
    print(myTree);

    int st = myTree.getNumSubTrees();
    for(int i = 0; i < st; i++) {
        IntTree* t = myTree.getSubTree(i);
        t->addNode(2*i + st);
        t->addNode(2*i + 1 + st);
    }
    print(myTree);

    IntTree* anotherTree = new IntTree;
    anotherTree->setValue(-1);
    anotherTree->addNode(-2);
    anotherTree->addNode(-3);

    IntTree* t = NULL;
    myTree.findValue(2, t);
    if (t) {
        st = t->getNumSubTrees();
        IntTree* u = t->removeSubTree(st - 1);
        print(myTree);
        delete u;
        t->addSubTree(anotherTree);
    }
    print(myTree);

    myTree.findValue(-3, t);
    int level = 1;
    while(t != NULL) {
```
t = t->getParent();
level++;
}
cout << "node (-3) is at level " << level << " (root has level 1)" << endl;
return 0;
}

and check that the output is as follows.

T = 0 ( 1 2 3 )
T = 0 ( 1 ( 3 4 ) 2 ( 5 6 ) 3 ( 7 8 ) )
T = 0 ( 1 ( 3 4 ) 2 ( 5 ) 3 ( 7 8 ) )
T = 0 ( 1 ( 3 4 ) 2 ( 5 -1 ( -2 -3 ) ) 3 ( 7 8 ) )
node (-3) is at level 3 (root has level 1)

6.8.1 Solution

/* Name: inttree.cxx
   Author: Leo Liberti
*/

#include <iostream>
#include "inttree.h"

IntTree::IntTree() : value(0), parent(NULL) { }

IntTree::~IntTree() {
  int ss = subTree.size();
  if (ss > 0) {
    for(int i = 0; i < ss; i++) {
      if (subTree[i]) {
        delete subTree[i];
      }
    }
  }
}

void IntTree::addSubNode(int n) {
  IntTree* newNode = new IntTree;
  newNode->setValue(n);
  newNode->setParent(this);
  subTree.push_back(newNode);
}

int IntTree::getNumberOfSubTrees(void) {
  return subTree.size();
}

IntTree* IntTree::getSubTree(int i) {
  if (i < 0 || i >= subTree.size()) {
    return NULL;
  } else {

 Specification coding question 2

 69
return subTree[i];
}
}

IntTree* IntTree::removeSubTree(int i) {
    IntTree* ret = NULL;
    if (i >= 0 && i < subTree.size()) {
        ret = subTree[i];
        ret->setParent(NULL);
        stit = subTree.begin();
        for (int j = 0; j < i; j++) {
            stit++;
        }
        subTree.erase(stit);
    }
    return ret;
}

void IntTree::addSubTree(IntTree* t) {
    t->setParent(this);
    subTree.push_back(t);
}

IntTree* IntTree::getParent(void) {
    return parent;
}

void IntTree::setParent(IntTree* p) {
    parent = p;
}

int IntTree::getValue(void) {
    return value;
}

void IntTree::setValue(int n) {
    value = n;
}

void IntTree::findValue(int n, IntTree* &found) {
    if (!found) {
        if (value == n) {
            found = this;
        } else {
            int ss = subTree.size();
            for (int i = 0; i < ss; i++) {
                IntTree* t = NULL;
                subTree[i]->findValue(n, t);
                if (t) {
                    found = t;
                }
            }
        }
    }
}
6.9 Task question 1

Write a program that prints its own (executable) file size in bytes and in number of printable characters (a character char c is printable if (isalnum(c) > 0 || ispunct(c) > 0 || isspace(c) > 0)).

6.9.1 Solution

/* name: task1.cxx
   author: Leo Liberti
*/

#include<iostream>
#include<fstream>

int main(int argc, char** argv) {
  using namespace std;
  ifstream ifs(argv[0]);
  char c;
  int characters = 0;
  int printables = 0;
  while(true) {
    ifs.get(c);
    if (ifs.eof()) {
      break;
    }
    characters++;
    if (isalnum(c) > 0 || ispunct(c) > 0 || isspace(c) > 0) {
      printables++;
    }
  }
  ifs.close();
  cout << "file size: bytes = " << characters << ", printable characters = "
  << printables << endl;
  return 0;
}
6.10 Task question 2

Write a program that prints the name of the image files referenced in the web page http://www.enseignement.polytechnique.fr/index.html. Image files are referenced in HTML both by the attribute HREF of the tag A (e.g. `<A REF="http://website.com/image.jpg">`) and by the attribute SRC of the tag IMG (e.g. `<IMG SRC="/images/logo.gif">`). The output should be as follows.

```
images/entete1.gif
images/remise.gif
images/amphi.gif
/icones/home.gif
```

6.10.1 Solution

```cpp
#include<iostream>
#include<fstream>
#include<sstream>
#include<string>
#include<cstring>
#include<cstdlib>
#include<unistd.h>

static const std::string jpg = "jpg";
static const std::string jpeg = "jpeg";
static const std::string gif = "gif";

int download(std::string inUrl, std::string outHtmlFile) {
    using namespace std;
    string wget = "wget ";
    string cmd = wget + " -O " + outHtmlFile + " " + inUrl + " > /dev/null 2>&1";
    int status = system(cmd.c_str());
    return status;
}

bool compareCaseInsensitive(const std::string& s1, const std::string& s2) {
    if (s1.size() != s2.size()) {
        return false;
    }
    int thesize = s1.size();
    for(int i = 0; i < thesize; i++) {
        if (tolower(s1[i]) != tolower(s2[i])) {
            return false;
        }
    }
    return true;
}
```
bool findLink(std::string& file, int tagsize, int i, std::string& link) {
    using namespace std;
    char terminator;
    if (file[i + tagsize] == '"') {
        terminator = '"';
    } else {
        terminator = ' ';
    }
    int j = i + tagsize + 1;
    while (file[j] != terminator) {
        j++;
    }
    if (terminator == '"') {
        link = file.substr(i + tagsize + 1, j - i - tagsize - 1);
    } else {
        link = file.substr(i + tagsize + 1, j - i - tagsize - 2);
    }
    int pos = link.find_last_of('.');
    if (pos != link.npos) {
        string ext = link.substr(pos + 1);
        if (compareCaseInsensitive(ext, jpg) ||
            compareCaseInsensitive(ext, jpeg) ||
            compareCaseInsensitive(ext, gif)) {
            return true;
        }
    }
    return false;
}

int main(int argc, char** argv) {
    using namespace std;
    stringstream htmlCode;
    string theUrl = "http://www.enseignement.polytechnique.fr/";
    string outFile = "index.html";
    const char* openFile;
    if (argc < 2) {
        download(theUrl, outFile);
        openFile = outFile.c_str();
    } else {
        openFile = argv[1];
    }
    ifstream ifs(openFile);
    if (!ifs) {
        cerr << "error: cannot open file " << openFile << endl;
        exit(1);
    }
    char c;
    while(!ifs.eof()) {
        ifs.get(c);
        htmlCode << c;
    }
    ifs.close();
    string file = htmlCode.str();

    Task question 2
string tag1 = "href=";
string tag2 = "src=";
string key1;
string key2;
int fileLength = file.size();
string link;
char terminator;

for(int i = 0; i < fileLength; i++) {
    key1 = file.substr(i, tag1.size());
    if (compareCaseInsensitive(key1, tag1)) {
        if (findLink(file, tag1.size(), i, link)) {
            cout << link << endl;
        }
    }
    key2 = file.substr(i, tag2.size());
    if (compareCaseInsensitive(key2, tag2)) {
        if (findLink(file, tag2.size(), i, link)) {
            cout << link << endl;
        }
    }
}

return 0;

6.11 Task question 3

Write a program that reads an \( m \times n \) matrix \( A = (a_{ij}) \) from a text file with \( m \) lines is a list of \( n \) double entries separated by a space, then output the homogeneous system \( Ax = 0 \) in explicit algebraic form with \( m \) lines of text having the following format:

\[
a_{i1} \cdot x[1] + \ldots + a_{in} \cdot x[n].
\]

The name of the file containing the matrix and the integer \( n \) should be read from the command line.

For example, for the text file

\[
1 -2.3 3 0.0
-9.123 2 -50 -1
\]

you should obtain the following output:

\[
x[1] - 2.3 \cdot x[2] + 3 \cdot x[3] = 0 \\
\]

Hint: if you have a file stream \( \text{ifs} \) and you want to read a double value off it, use the following construct.

\[
double t; \\
\text{ifs} \gg t;
\]
6.11.1 Solution

/* name: task3.cxx
   author: Leo Liberti
*/

#include <iostream>
#include <fstream>
#include <cmath>

int main(int argc, char** argv) {
    using namespace std;
    if (argc < 3) {
        cerr << "error: need filename containing m x n matrix, and the integer n"
             << " on the command line" << endl;
        exit(1);
    }
    ifstream ifs(argv[1]);
    if (!ifs) {
        cerr << "error: cannot open file " << argv[1] << endl;
        exit(2);
    }
    int n = atoi(argv[2]);

double t;
int i = 1;
char c;
bool firstTerm = true;
while(true) {
    ifs.get(c);
    ifs >> t;
    if (ifs.eof()) { break; }
    if (t != 0) {
        if (t < 0) {
            if (firstTerm) { cout << "-"; }
            else { cout << " - "; }
        }
        if (fabs(t) != 1) { cout << fabs(t) << "*"; }
        cout << "x[" << i << "]";
    } else {
        if (!firstTerm) { cout << " + "; }
        if (t != 1) { cout << t << "*"; }
        cout << "x[" << i << "]";
    }
    firstTerm = false;
    i++;
}
}
```cpp
firstTerm = false;
    i++;
    if (i > n) {
        i = 1;
        firstTerm = true;
        cout << " = 0" << endl;
    }
}
return 0;
```
Chapter 7

Questions from a banking C++ test

In this chapter we group some questions taken from a real online C++ test used by banks worldwide. You have 60 seconds to answer each multiple choice question.

7.1 Question 1

Code:

```cpp
char a[] = "ABCD\r\n";
```

Question: How many elements will be in the array created by the declaration in the sample above?

1. 6
2. 7
3. 8
4. 9
5. 4

7.2 Question 2

Question: Which of the following statements is true when a derivation inherits both a virtual and non-virtual instance of a base class?

1. Each base class object has derived objects only from the non-virtual instance.
2. Each base class object has derived objects only from the virtual instance.
3. Each derived class object has base objects only from the virtual instance.
4. Each derived class object has base objects only from the non-virtual instance.
5. Each derived class object has a base object from the virtual instance and a base object from non-virtual instance.
7.3 Question 3

Code:

```cpp
string somestring;
```

**Question**: Which of the following choices will convert a standard C++ string object "somestring" to a C string?

1. Copy.somestring();
2. somestring.c_str();
3. std::cstring(somestring)
4. (char*)somestring
5. &somestring[1]

7.4 Question 4

Code:

```cpp
#include <iostream>
```

**Question**: Referring to the sample code above, which of the following could you use to make the standard I/O stream classes accessible without requiring the scope resolution operator?

1. using iostream;
2. using namespace iostream;
3. using namespace std;
4. using namespace std::iostream;
5. using std::iostream;

7.5 Question 5

Code:

```cpp
class basex {
    int x;
public:
    void setx(int y) {x=y;}
};
class derived : basex {};
```

**Question**: What is the access level for the member function setx in the class derived above?
1. protected
2. public
3. local
4. private
5. global

### 7.6 Solutions

1. 2
2. 5
3. 2
4. 3
5. 4