Course C++ - EXCERCISES

2.28 *(Digits of an Integer)* Write a program that inputs a five-digit integer, separates the integer into its digits and prints them separated by three spaces each. [*Hint:* Use the integer division and modulus operators.] For example, if the user types in 42339, the program should print:

```
4   2   3   3   9
```

3.14 *(`Employee` Class)* Create a class called `Employee` that includes three pieces of information as data members—a first name (type `string`), a last name (type `string`) and a monthly salary (type `int`). [*Note:* In subsequent chapters, we'll use numbers that contain decimal points (e.g., 2.75)—called floating-point values—to represent dollar amounts.] Your class should have a constructor that initializes the three data members. Provide a *set* and a *get* function for each data member. If the monthly salary is not positive, set it to `0`. Write a test program that demonstrates class `Employee`'s capabilities. Create two `Employee` objects and display each object's *yearly* salary. Then give each `Employee` a 10 percent raise and display each `Employee`'s yearly salary again.

4.37 *(World Population Growth)* World population has grown considerably over the centuries. Continued growth could eventually challenge the limits of breathable air, drinkable water, arable cropland and other precious resources. There is evidence that growth has been slowing in recent years and that world population could peak some time this century, then start to decline.

For this exercise, research world population growth issues online. *Be sure to investigate various viewpoints.* Get estimates for the current world population and its growth rate (the percentage by which it is likely to increase this year). Write a program that calculates world population growth each year for the next 75 years, *using the simplifying assumption that the current growth rate will stay constant*. Print the results in a table. The first column should display the year from year 1 to year 75. The second column should display the anticipated world population at the end of that year. The third column should display the numerical increase in the world population that would occur that year. Using your results, determine the year in which the population would be double what it is today, if this year's growth rate were to persist.

6.30 *(Reverse Digits)* Write a function that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the function should return 1367.

6.50 *(Pass-by-Value vs. Pass-by-Reference)* Write a complete C++ program with the two alternate functions specified below, each of which simply triples the variable `count` defined in `main`. Then compare and contrast the two approaches. These two functions are

a) function `tripleByValue` that passes a copy of `count` by value, triples the copy and returns the new value and

b) function `tripleByReference` that passes `count` by reference via a reference parameter and triples the original value of `count` through its alias (i.e., the reference parameter).

6.52 *(Function Template `minimum`)* Write a program that uses a function template called `minimum` to determine the smaller of two arguments. Test the program using integer, character and floating-point number arguments.

7.21 *(Sales Summary)* Use a two-dimensional `array` to solve the following problem. A company has four salespeople (1 to 4) who sell five different products (1 to 5). Once a day, each salesperson passes in a slip for each different type of product sold. Each slip contains the following:

a) The salesperson number
b) The product number
c) The total dollar value of that product sold that day

Thus, each salesperson passes in between 0 and 5 sales slips per day. Assume that the information from all of the slips for last month is available. Write a program that will read all this information for last month's sales (one salesperson's data at a time) and summarize the total sales by salesperson by product. All totals should be stored in the two-dimensional `array` `sales`. After processing all the information for last month, print the results in tabular format with each of the columns representing a particular salesperson and each of the rows representing a particular product. Cross total each row to get the total sales of each product for last month; cross total each column to get the total sales by salesperson for last month. Your tabular printout should include these cross totals to the right of the totaled rows and to the bottom of the totaled columns.

Chapter 8: Excercises about pointers

Indicate in red the code lines that are syntactically incorrect. Indicate in green the expressions that are syntactically correct, but that are useless or might cause problems during execution.

Finally, what will appear on the display? Do not use your computer.

```
int main()
{
      int i,j;
      double d;
      int *ip = nullptr, *jp = nullptr;
      double *dp = nullptr;
      i = 7;
      ip = &7;
      jp = &i;
      *jp = j;
      *ip = i;
      ip = &j;
      &i = ip;
      j=4;
      (*ip)++;
      &d = dp;
      *ip*= i;
      *jp=*&j;
      ip++;
      i = ip-&i;
      dp = &i;
      dp = ip;
      &dp = &&d;
      *ip+=1;
      *ip++;
      cout <<endl<< i << j << d;
      cout <<endl<< *ip << *jp << *dp;
      return 0;
}
```

9.8 *(Enhancing Class* **`Date`***)* Modify the `Date` class of Figs. 9.17–9.18 to perform error checking on the initializer values for data members `month`, `day` and `year`. Also, provide a member function `nextDay` to increment the day by one. Write a program that tests function `nextDay` in a loop that prints the date during each iteration to illustrate that `nextDay` works correctly. Be sure to test the following cases:

a) Incrementing into the next month.
b) Incrementing into the next year.

```
// Fig. 9.17: Date.h
// Date class definition; Member functions defined in Date.cpp
#ifndef DATE_H
#define DATE_H

class Date
{
public:
   static const unsigned int monthsPerYear = 12; // months in a year
   explicit Date( int = 1, int = 1, int = 1900 ); // default constructor
   void print() const; // print date in month/day/year format
   ~Date(); // provided to confirm destruction order
private:
   unsigned int month; // 1-12 (January-December)
   unsigned int day; // 1-31 based on month
   unsigned int year; // any year

   // utility function to check if day is proper for month and year
   unsigned int checkDay( int ) const;
}; // end class Date

#endif
```

```
// Fig. 9.18: Date.cpp
// Date class member-function definitions.
#include <array>
#include <iostream>
#include <stdexcept>
#include "Date.h" // include Date class definition
using namespace std;

// constructor confirms proper value for month; calls
// utility function checkDay to confirm proper value for day
Date::Date( int mn, int dy, int yr )
{
   if ( mn > 0 && mn <= monthsPerYear ) // validate the month
      month = mn;
   else
      throw invalid_argument( "month must be 1-12" );

   year = yr; // could validate yr
   day = checkDay( dy ); // validate the day

   // output Date object to show when its constructor is called
   cout << "Date object constructor for date ";
   print();
   cout << endl;
} // end Date constructor

// print Date object in form month/day/year
void Date::print() const
{
   cout << month << '/' << day << '/' << year;
} // end function print
```

```
// output Date object to show when its destructor is called
Date::~Date()
{
   cout << "Date object destructor for date ";
   print();
   cout << endl;
} // end ~Date destructor

// utility function to confirm proper day value based on
// month and year; handles leap years, too
unsigned int Date::checkDay( int testDay ) const
{
   static const array< int, monthsPerYear + 1 > daysPerMonth =
      { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

   // determine whether testDay is valid for specified month
   if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
      return testDay;

   // February 29 check for leap year
   if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
      ( year % 4 == 0 && year % 100 != 0 ) ) )
      return testDay;

   throw invalid_argument( "Invalid day for current month and year" );
} // end function checkDay
```

10.10 ***(RationalNumber Class)*** Create a class `RationalNumber` (fractions) with the following capabilities:

a) Create a constructor that prevents a 0 denominator in a fraction, reduces or simplifies fractions that are not in reduced form and avoids negative denominators.
b) Overload the addition, subtraction, multiplication and division operators for this class.
c) Overload the relational and equality operators for this class.

12.16 ***(CarbonFootprint Abstract Class: Polymorphism)*** Using an abstract class with only pure virtual functions, you can specify similar behaviors for possibly disparate classes. Governments and companies worldwide are becoming increasingly concerned with carbon footprints (annual releases of carbon dioxide into the atmosphere) from buildings burning various types of fuels for heat, vehicles burning fuels for power, and the like. Many scientists blame these greenhouse gases for the phenomenon called global warming. Create three small classes unrelated by inheritance—classes `Building`, `Car` and `Bicycle`. Give each class some unique appropriate attributes and behaviors that it does not have in common with other classes. Write an abstract class `CarbonFootprint` with only a pure virtual `getCarbonFootprint` method. Have each of your classes inherit from that abstract class and implement the `getCarbonFootprint` method to calculate an appropriate carbon footprint for that class (check out a few websites that explain how to calculate carbon footprints). Write an application that creates objects of each of the three classes, places pointers to those objects in a `vector` of `CarbonFootprint` pointers, then iterates through the `vector`, polymorphically invoking each object's `getCarbonFootprint` method. For each object, print some identifying information and the object's carbon footprint.

13.15 *(`Point` Class)* Write a program that accomplishes each of the following:

a) Create a user-defined class `Point` that contains the private integer data members `xCoordinate` and `yCoordinate` and declares stream insertion and stream extraction overloaded operator functions as `friend`s of the class.

b) Define the stream insertion and stream extraction operator functions. The stream extraction operator function should determine whether the data entered is valid, and, if not, it should set the `failbit` to indicate improper input. The stream insertion operator should not be able to display the point after an input error occurred.

c) Write a `main` function that tests input and output of user-defined class `Point`, using the overloaded stream extraction and stream insertion operators.

14.11 *(Hardware Inventory)* You are the owner of a hardware store and need to keep an inventory that can tell you what different tools you have, how many of each you have on hand and the cost of each one. Write a program that initializes the random-access file `hardware.dat` to 100 empty records, lets you input the data concerning each tool, enables you to list all your tools, lets you delete a record for a tool that you no longer have and lets you update *any* information in the file. The tool identification number should be the record number. Use the following information to start your file:

Record #	Tool name	Quantity	Cost
3	Electric sander	7	57.98
17	Hammer	76	11.99
24	Jig saw	21	11.00
39	Lawn mower	3	79.50
56	Power saw	18	99.99
68	Screwdriver	106	6.99
77	Sledge hammer	11	21.50
83	Wrench	34	7.50

Chapter 15

Rewrite the carbon footprint exercise in such a way that

1. The carbon footprint is calculated in the constructor of each class and set in a member variable.
2. The function GetCarbonFootprint() no longer calculates the carbon footprint but serves merely as a getter.
3. Add the name (“Bicyle”, “Building”, “Car”) as a new member to the class and initialize it in the constructor.
4. Choose an appropriate collection that allows sorting. Sort the collection by carbon footprint.
5. Store the objects (and not pointers to the objects) in the collection
6. Print name and footprint before and after sorting:
   BEFORE SORTING:
   Bicycle: 0
   Building: 335000
   Car: 200
   AFTER SORTING:
   Bicycle: 0
   Car: 200
   Building: 335000
7. Save name and footprint of the sorted collection to a text file.

8. Are virtual functions still needed?

18.3 *(Operator Overloads in Templates)* Write a simple function template for predicate function `isEqualTo` that compares its two arguments of the same type with the equality operator (==) and returns `true` if they are equal and `false` otherwise. Use this function template in a program that calls `isEqualTo` only with a variety of fundamental types. Now write a separate version of the program that calls `isEqualTo` with a user-defined class type, but does not overload the equality operator. What happens when you attempt to run this program? Now overload the equality operator (with the operator function) `operator==`. Now what happens when you attempt to run this program?

Please find below the start of a template class “Pair”.

```
template<class T>
class Pair
{
public:
    Pair();
    Pair(T first_value, T second_value);
    void set_element(int position, T value);
        . . .
```

Member functions and overloaded operators are then defined as function templates. For example, the definition of a function definition for the sample class template above could begin as follows:

```
template<class T>
void Pair<T>::set_element(int position, T value)
{
     . . .
```

Complete the implementation for set_element, give the implemenation for get_element and the constructor without parameters.

Make a new class HeterogeneousPair, in which the two elements can have a different type. Design the appropriate getters and setters.

SYNTHESE-OEFENING

Schrijf een aanzet voor een facturatieprogramma. De gebruiker krijgt een menu met 4 opties:

1. print factuur (bij keuze hiervan wordt het factuurnummer opgevraagd), factureer en lijst.

2. “Factureer” roept een dummy functie aan die “Factureren !” op het scherm zet.

3. “Lijst” toont alle in deze run geprinte facturen

4. EINDE

“Print” zal “print factuur” + de gegevens van de factuur op het scherm tonen. Let er evenwel op dat slechts één object van een bepaalde factuur kan geconstrueerd worden. Als eenzelfde factuur een volgende keer wordt gevraagd om te printen, zal met hetzelfde, reeds geconstrueerde object, gewerkt worden.

De gegevens over de factuur (bestaande uit factuurnummer, datum, klant en bedrag) lees je uit een fixed records size tekstfile. Meld aan de gebruiker als de gevraagde factuur niet aanwezig is.

Voorzie ook een optionele command-line parameter die, indien meegegeven, het menu niet toont maar:

- indien commandline-parameter = “f” → factureer
- indien commandline-parameter = <een factuurnummer> → print van deze factuur

Qt

1. Vorm de method CarbonFootPrint uit de oefening CarbonFootPrint om zodat die een *double* teruggeeft i.p.v. een *cout* uitvoert. Maak een Qt Widget Applicatie die de keuze laat (via een dropdown) tussen Car, Bicyle en Building (neem voor car 20 gallons en voor building 200 m^2). Voeg een ok-button toe die na klikken in een message box de uitstoot van de gekozen CO_2-bron toont. Tip: gebruik QComboBox voor de dropdown.

2. Vertrek van het project "TemperatureLogger".
    a. Toon een error dialog (op basis van exceptions) als ergens een fout optreedt. Test door de readings file te hernoemen.
    b. Voeg een refresh button toe m.b.v. de designer die zorgt voor een onmiddellijke refresh van de data.

3. Vertrek van het project "CheckExcel"
    a. Pas de stijlen aan volgens de huisstijl van je bedrijf
    b. Wijzig de "Save"-functionaliteit zodat deze, indien een parameter wordt meegegeven bij het opstarten van de applicatie, deze als bestandsnaam geïnterpreteerd wordt en er automatisch naar dit bestand wordt opgeslagen in dezelfde directory als de Excel-file.
    c. Voeg een kolom toe aan de foutentabel op het scherm met een volgnummer: 1, 2, 3, …