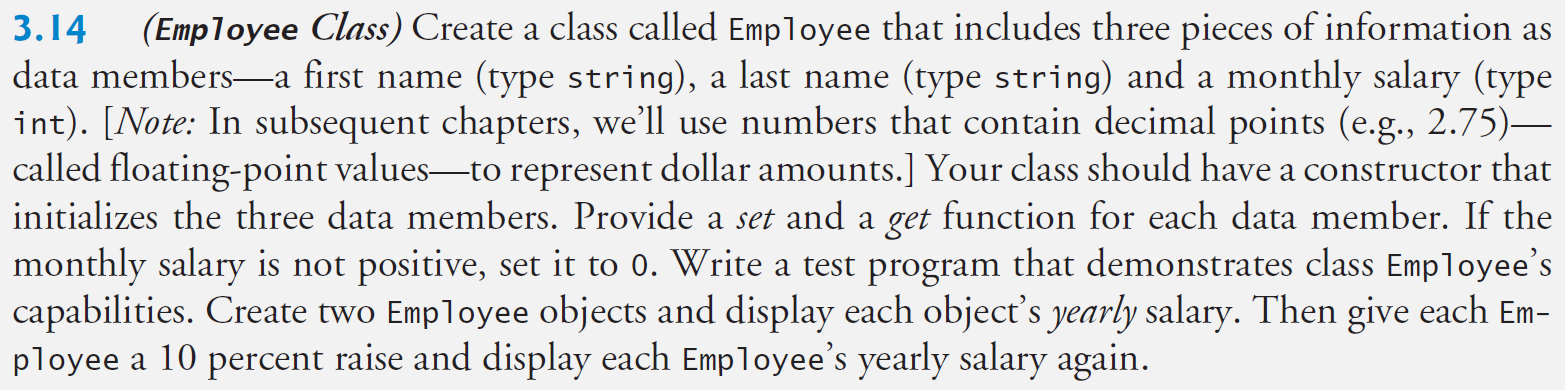
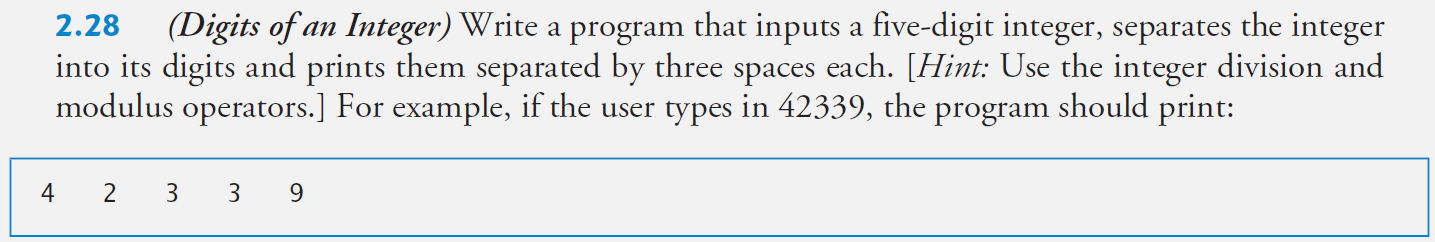
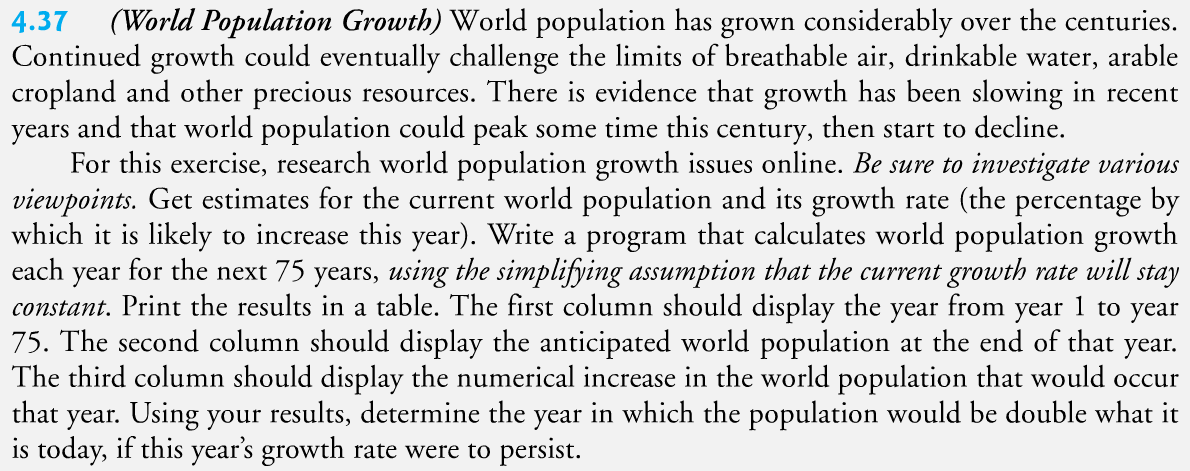
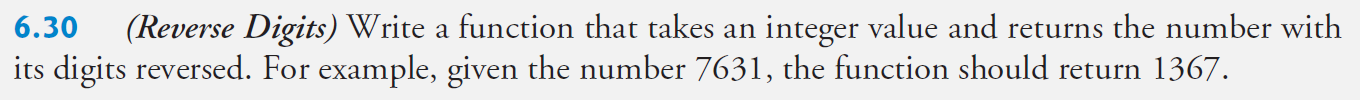
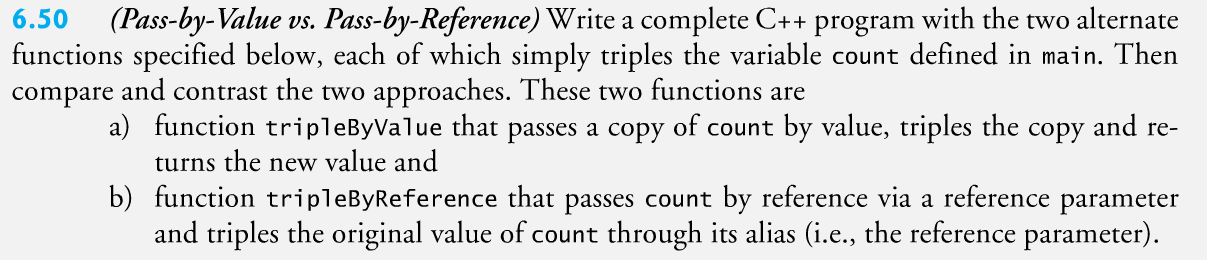
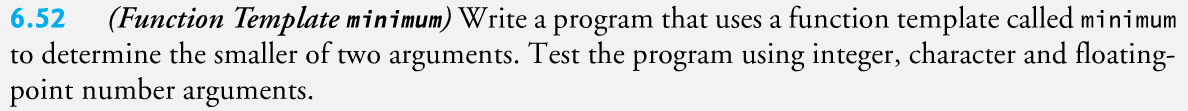
Course C++ - EXCERCISES

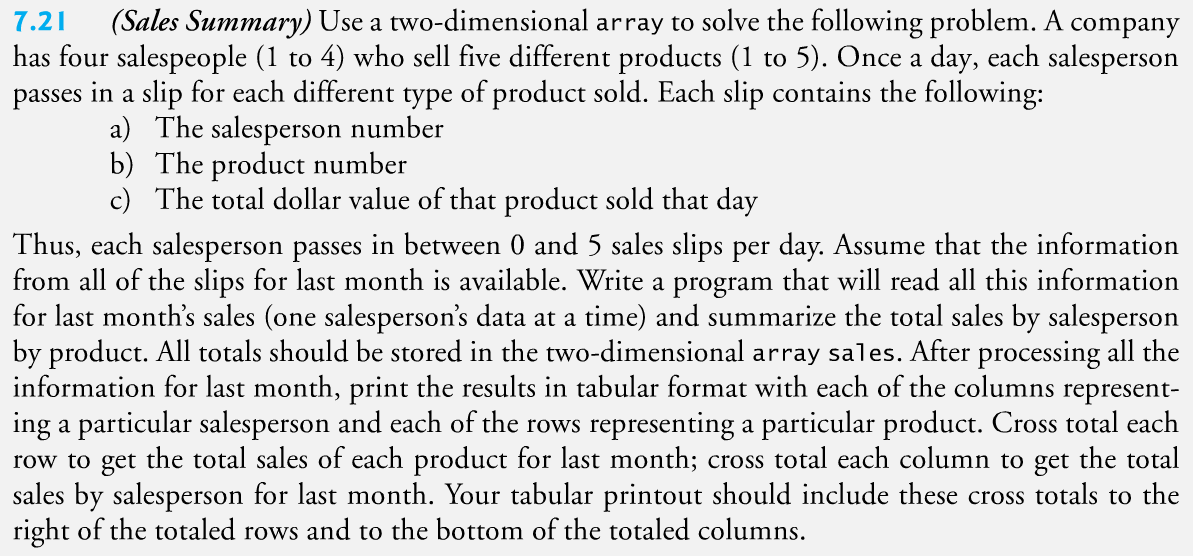












**Chapter 8: Excercises about pointers**

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;

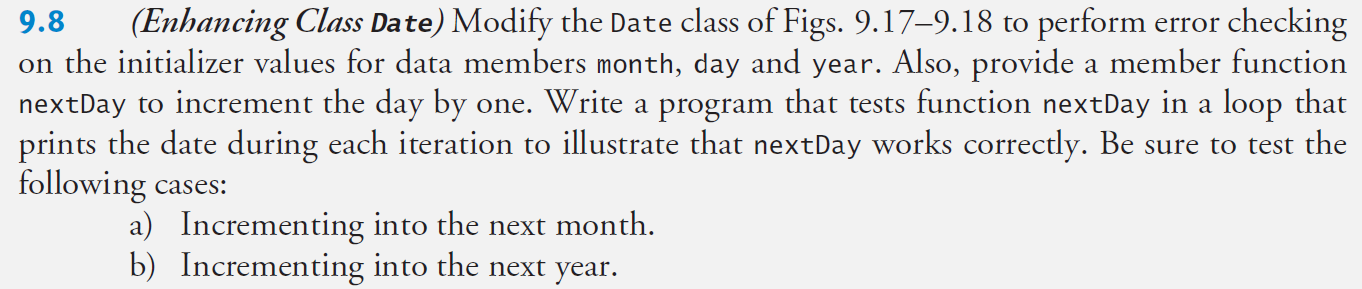
\*ip++;

cout <<endl<< i << j << d;

cout <<endl<< \*ip << \*jp << \*dp;

return 0;

}



// 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 ||

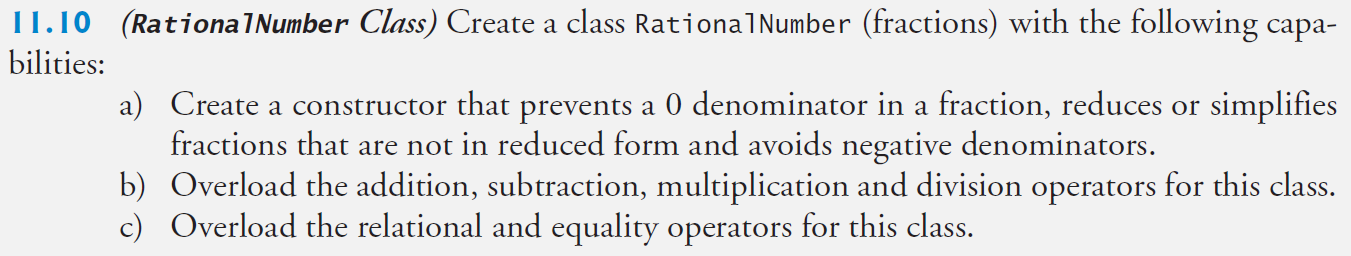
( year % 4 == 0 && year % 100 != 0 ) ) )

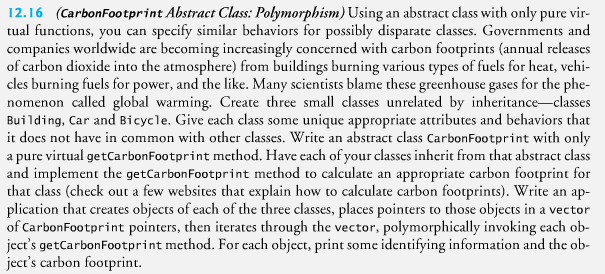
return testDay;

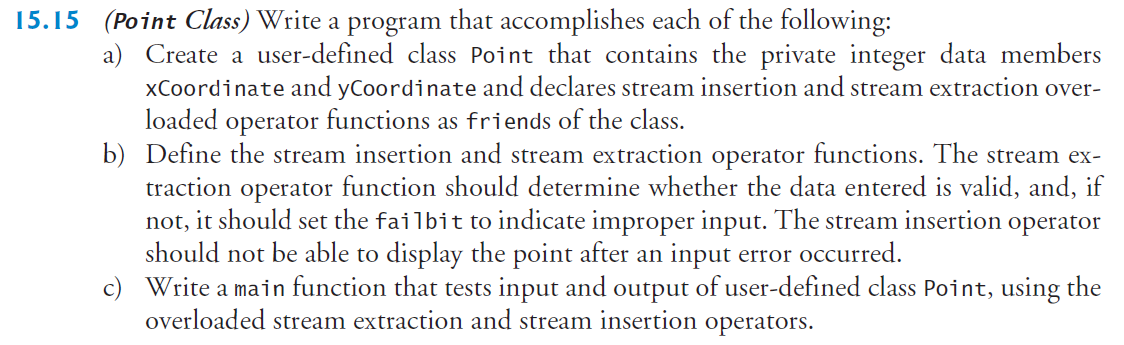
throw invalid\_argument( "Invalid day for current month and year" );

} // end function checkDay

**10.10**

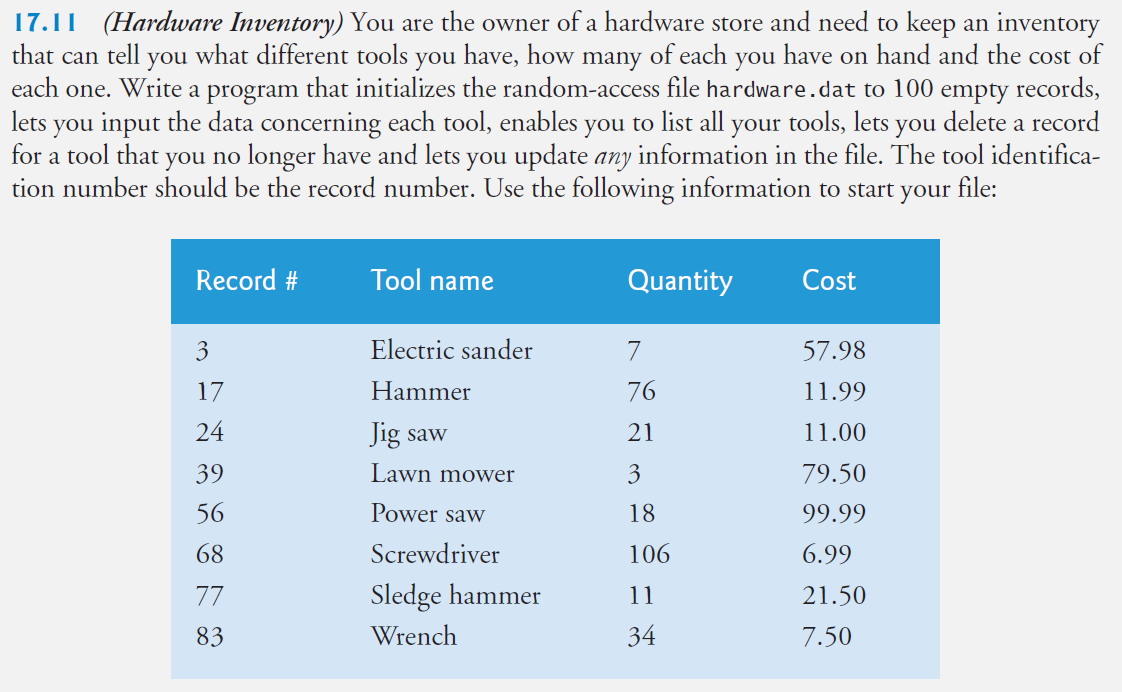


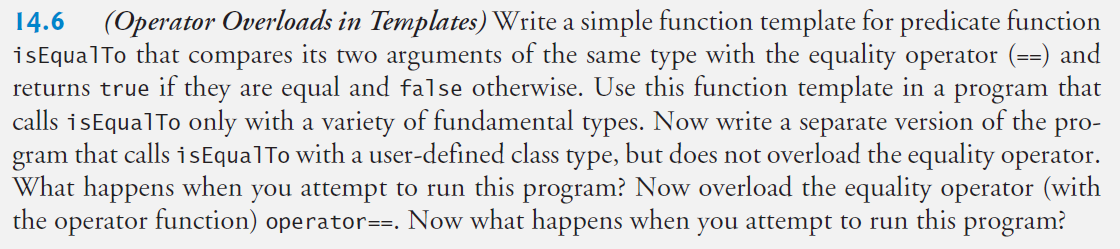




**14.11**

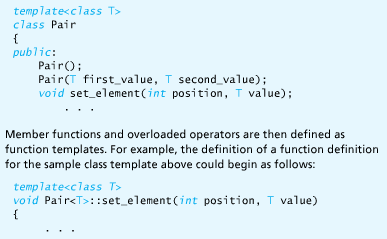
**13.15**





**18.3**

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



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 m²). Voeg een ok-button toe die na klikken in een message box de uitstoot van de gekozen CO2-bron toont. Tip: gebruik QComboBox voor de dropdown.
2. Vertrek van het project “TemperatureLogger”.
   1. Toon een error dialog (op basis van exceptions) als ergens een fout optreedt. Test door de readings file te hernoemen.
   2. 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”
   1. Pas de stijlen aan volgens de huisstijl van je bedrijf
   2. 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.
   3. Voeg een kolom toe aan de foutentabel op het scherm met een volgnummer: 1, 2, 3, …