/\* Filename: Chap04ProgrammingChallenge07.cpp

Author: S. Rimal

Purpose: To solve the Chapter 4 programming challenge 7 (Time Calculator).

\*/

#include <iostream>

using namespace std;

int main()

{

// Constants

const int SECONDSPERMINUTE = 60; // seconds in a minute

const int SECONDSPERHOUR = 3500; // seconds in an hour

const int SECONDSPERDAY = 86400; // seconds in a day

// Variables

double seconds; // to store seconds

double minutes; // to store minutes

double hours;   // to store minutes

double days;    // to store days

// Get the input

cout << "Enter the number of seconds: ";

cin >> seconds;

// Display the number of minutes in that many seconds

if (seconds >= SECONDSPERMINUTE)

{

minutes = seconds / SECONDSPERMINUTE;

cout << "There are " << minutes << " minutes in " << seconds << " seconds.\n";

} // end if

  // Display the number of hours in that many seconds

if (seconds >= SECONDSPERHOUR)

{

hours = seconds / SECONDSPERHOUR;

cout << "There are " << hours << " hours in " << seconds << " seconds.\n";

} // end if

  // Display the number of days in that many seconds

if (seconds >= SECONDSPERDAY)

{

days = seconds / SECONDSPERDAY;

cout << "There are " << days << " days in " << seconds << " seconds.\n";

} // end if

cin.ignore();

cin.get();

return 0;

}

\*Filename: Arrays01.cpp

Author: S. Rimal

Puropse: To demonstrate how to declare an array, reference an array

element, store values in array elements, and retrieve values

in array elements.

\*/

#include <iostream>

using namespace std;

int main()

{

const int SIZE = 4; // constant that will serve as size of array

double a[SIZE]; // array with SIZE elements

cout << "Enter 4 numbers: \n";

for (int i = 0; i < SIZE; i++)

{

cout << i + 1 << ": ";

cin >> a[i];

} // end for

cout << endl;

cout << "Here are the numbers in reverse order: \n";

for (int i = SIZE - 1; i >= 0; i--)

{

cout << "\ta[" << i << "] = " << a[i] << endl;

} // end for

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

} // end main()

/\*Filename: Arrays02.cpp

Author: S. Rimal

Date: Oct. 13, 2016

Puropse: To demonstrate array intitialization.

\*/

#include <iostream>

using namespace std;

int main()

{

const int SIZE = 4;  // constant to size array

double a[SIZE];  // uninitialized array

for (int i = 0; i < SIZE; i++)

cout << "a[" << i << "] = " << a[i] << endl;

cout << endl;

// completely initialized array

double b[SIZE] = { 22.2, 33.3, 44.4, 55.5 };

for (int i = 0; i < SIZE; i++)

cout << "b[" << i << "] = " << b[i] << endl;

cout << endl;

// paritially initialized array

double c[SIZE] = { 22.2, 33.3 };

for (int i = 0; i < SIZE; i++)

cout << "c[" << i << "] = " << c[i] << endl;

cout << endl;

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

} // end main()

/\*Filename: Arrays07.cpp

Author: Santoshi Rimal

Purpose: To demonstrate the use of two-dimensional arrays.

\*/

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

float sales[3][4]; // has three rows, four columns

float totalSales = 0; // accumulator that holds total sales

int div, qtr;

cout << "This program will calculate the total sales of all the\n"

<< "company's divisions.\n\n";

cout << "Enter the following sales information: \n\n";

// nested loops to fill array with quarterly sales

// figures for each division of the company

for (div = 0; div < 3; div++) // loop for row dimension

{

for (qtr = 0; qtr < 4; qtr++) // loop for column dimension

{

cout << "Division " << (div + 1);

cout << ", Quarter " << (qtr + 1) << ": ";

cin >> sales[div][qtr];

} // end inner loop

} // end outer loop

// nested loops to add all elements

for (div = 0; div < 3; div++)

for (qtr = 0; qtr < 4; div++)

totalSales += sales[div][qtr];

cout << fixed << showpoint;

cout << setprecision(2);

cout << "The total sales for the company are: $";

cout << totalSales << endl;

// date: Oct. 20, 2016

// alogrithm to sum rows; totals for divisions

for (div = 0; div < 3; div++)

{

float rowTotalSales = 0;  // accumulator for row totals

for (qtr = 0; qtr < 4; qtr++)

rowTotalSales += sales[div][qtr]; // end inner for

cout << "Sales for Division " << (div + 1) << " are: "

<< rowTotalSales << endl;

} // end outer for

// algorithm to sum the columns; totals for quarters

for (qtr = 0; qtr < 4; qtr++)

{

float colTotalSales = 0;  // accumulator for column totals

for (div = 0; div < 3; div++)

colTotalSales += sales[div][qtr];  // end inner for

cout << "Sales for Quarter " << (qtr + 1) << " are: "

<< colTotalSales << endl;

} // end outer loop

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

} // end main()

/\*Filename: Arrays06.cpp

Author: Santoshi Rimal

Purpose: To demonstrate the use of parallel arrays

and also processing of array elements.

- One of the arrays will store the number of hours

worked by an employee

- The second will hold the pay rate of the employee.

\*/

#include <iostream>

using namespace std;

const int NUMEMPS = 5;

int main()

{

int hours[NUMEMPS]; // holds hours worked

float payRate[NUMEMPS]; // holds pay rate

// above two arrays are parallel arrays

cout << "Enter the hours worked by " << NUMEMPS << " employees "

<< "and their hourly rates.\n\n";

for (int index = 0; index < NUMEMPS; index++)

{

cout << "Hours worked by employee # " << (index + 1) << ": ";

cin >> hours[index];

cout << "Hourly pay rate for employee # " << (index + 1) <<

": ";

cin >> payRate[index];

} // end for

cout << "\n\nHere is the gross pay for each employee: \n\n";

cout.precision(2);

cout.setf(ios::fixed | ios::showpoint);

// the above statements set output formatting

for (int index = 0; index < NUMEMPS; index++)

{

float grossPay = hours[index] \* payRate[index];

cout << ": $" << grossPay << endl;

} // end for

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

} // end main()

/\*Filename: Arrays05.cpp

Author: Santoshi Rimal

Purpose: To demonstrate passing an array to a function.

- Cannot cout a numeric array just by using the array name.

- Cannot assign one array to another to copy elements.

- Character arrays are treated differently than numeric

arrays by cout.

- Complete arrays are passed by reference to functions.

- Individual elements of an array may be passed by value

or by reference.

\*/

#include <iostream>

using namespace std;

const int SIZE = 100;

// function prototypes

void getArray(double[], int &);

void printArray(const double[], const int);

int main()

{

double a[SIZE], b[SIZE];

int n;

// read the array

getArray(a, n);

cout << "\nThe array has " << n << " elements.\nThey are: \n";

printArray(a, n);

// cannot assign one numeric array to another, if one wants to

// copy elements of one to the other

// Examples: b = a will not work

cout << a; // will not print out array

char msg[] = "Hello";

cout << endl << endl << msg;

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

} // end main()

void getArray(double a[], int & n)

{

cout << "Enter data. Terminate with a zero: \n";

for (n = 0; n < SIZE; n++)

{

cout << n + 1 << ": ";

cin >> a[n];

if (a[n] == 0) break;

} // end for

} // end getArray()

void printArray(const double a[], const int n)

{

for (int i = 0; i < n; i++)

cout << "\t" << i << ": " << a[i] << endl;

} // end printArray

// Filename: Program.cs

// Author: Santoshi Rimal

// Purpose: To demonstrate the creation of a console project in C#.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BarChart

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] array = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 }; // distribution

            Console.WriteLine("Grade distribution");

            // for each array element, output bar of chart

            for(int count = 0; count < array.Length; count++)

            {

                // output bar Labels ("00-09 ", ..., "90-99: ", "100: ")

                if (count == 10)

                    Console.Write(" 100: ");

                else

                    Console.Write("{0:D2}-{1:D2}:", count \* 10, count \* 10 + 9);

                // display bar of asterisks

                for (int stars = 0; stars < array[count]; ++stars)

                    Console.Write("\*");

                Console.WriteLine();  // start a new line of output

            } // end outer for

        } // end Main()

    } // end class

} // end namespace

/\* Filename: ClassesEx15.cpp

Author: Santoshi Rimal

Purpose: Demonstrate nesting of one class inside another;

this is called class aggregation (or class composition)

\*/

#include <iostream>

using namespace std;

// first class

class Rectangle

{

private:

double width, length;

public:

// mutators

void setLength(double len) { length = len; }

void setWidth(double wid) { width = wid; }

// accessors

double getLength() { return length; }

double getWidth() { return width; }

double getArea() { return length \* width; }

};  // end first class

// second class

class Carpet

{

private:

double pricePerYd;

Rectangle size;  // an object of first class

public:

// mutators

void setPricePerYd(double p) { pricePerYd = p; }

void setDimensions(double len, double wid)

{

size.setLength(len / 3);

size.setWidth(wid / 3);

}  // end setDimensions

double getTotalPrice()

{

return size.getArea() \* pricePerYd;

}  // end getTotalPrice()

};  // end second class

// main() program

int main()

{

Carpet purchase;

double pricePerYd;

double length, width;

cout << "Room length in feet: ";

cin >> length;

cout << "Room width in feet: ";

cin >> width;

cout << "Carpet price per sq. yard: ";

cin >> pricePerYd;

purchase.setDimensions(length, width);

purchase.setPricePerYd(pricePerYd);

cout << "\nThe total price of the new " << length << "x " <<

width << " carpet is $" << purchase.getTotalPrice() << endl;

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

} // end main()

/\* Filename: ClassesEx14.cpp

Author: Santoshi Ramirez

Purpose: Demonstrate existence of "this" pointer

(sometimes called hidden argument).

Will use "this" pointer to refer to

object members in this program

\*/

#include <iostream>

using namespace std;

// class Test

class Test

{

private:

int x;

public:

Test(int = 0);  // constructor

void print() const;

};

Test::Test(int value)

{

x = value;

} // end constructor

void Test::print() const

{

cout << "   x = " << x;

// explicitly use "this" pointer and arrow operator to access

// member x

cout << "\n    this->x = " << this->x;

// explicitly use the dereferenced "this" pointer and the dot operator

// to access member x

cout << "\n    (\*this).x = " << (\*this).x << endl;

} // end print()

int main()

{

Test testObj(12);    // instantiate and initialize testObj

testObj.print();

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

} // end main()

/\* Filename: ClassesEx13.cpp

Author: Santoshi Rimal

Purpose: Demonstrate how to return objects from a function

\*/

#include <iostream>

#include <iomanip>

#include <string>

#include "InventoryItem.h"

using namespace std;

// function prototypes

inventoryItem createItem();  // returning an object

void showValues(inventoryItem);  // passing an object

// main() function

int main()

{

inventoryItem part = createItem();  // instantiation of an object

showValues(part);

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

}  // end main()

// showValues() function

void showValues(inventoryItem item)

{

cout << fixed << showpoint << setprecision(2);

cout << "Part number: " << item.getPartNum() << endl;

cout << "Description: " << item.getDescription() << endl;

cout << "Units on Hand: " << item.getOnHand() << endl;

cout << "Price: " << item.getPrice() << endl;

cout << endl;

} // end showValues()

// createItem() function

inventoryItem createItem()

{

// local objects and variables

inventoryItem tempItem;

int partNum;

string description;

int qty;

double price;

// get data from user

cout << "Enter the data for the new part number: \n";

cout << "Part number: ";

cin >> partNum;

cin.ignore();

cout << "Description: ";

getline(cin, description);

cout << "Quantity on Hand: ";

cin >> qty;

cout << "Unite Price: ";

cin >> price;

cout << endl;

// store data in inventory Item object and return it

tempItem.storeInfo(partNum, description, qty, price);

return tempItem;

} // end createItem()

/\* Filename: InventoryItem.cpp

Author: Santoshi Rimal

Purpose: Implementation of class InventoryItem

\*/

#include "InventoryItem.h"

void inventoryItem::storeInfo(int p, string d, int oH, double cost)

{

partNum = p;

description = d;

onHand = oH;

price = cost;

} // end storeInfo

/\*Filename: ClassesEx12.cpp

Author: Santoshi Rimal

Purpose: Demonstrate use of friend functions;

In this example, friend is a non-member

function

\*/

#include <iostream>

using namespace std;

// class

class Count

{

private:

int x;

public:

Count() { x = 0; }  // constructor

// Count(Count&);  // definition of copy constructor

void print() const { cout << x << endl; }

friend void setX(Count&, int);  // non-member friend function

}; // end of class Count

void setX(Count& c, int val)

{

c.x = val;

 } // end setX()

int main()

{

Count counter, count;

cout << "Counter.x after instantiation: ";

counter.print();

setX(counter, 8);

cout << "counter.x after call to setX friend function: ";

counter.print();

count = counter;  // memberwise assignment in projects

cout << "The value of x in count = ";

count.print();

cout << endl;

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

} // end main()

/\* Filename: ClassesEx11.cpp

Author:Santoshi Rimal

Date: Dec. 01, 2016

Purpose: To demonstrate the use of static member variables

and static member functions

\*/

#include <iostream>

#include "Employee.h"

using namespace std;

int main()

{

// use class name and binary scope operator to access

// the static member function getCount()

cout << "Number of employees before instantiation of any object is "

<< Employee::getCount() << endl;

// use "new" operator to create two new employees

Employee \*e1ptr = new Employee("Susan", "Baker");

Employee \*e2ptr = new Employee("Robert", "Smith");

// call static member function on first employee

cout << "Number of employees after objects are instantiation is "

<< e1ptr->getCount();

cout << "\n\nEmployee 1: " << e1ptr->getFirstName() << "   "

<<  e1ptr->getLastName() << endl;

cout << "Employee 2: " << e2ptr->getFirstName() << "   "

<< e2ptr->getLastName();

// deallocate memory

delete e1ptr;

e1ptr = 0;

delete e2ptr;

e2ptr = 0;

cout << "\nNumber of employees after objects are deleted is "

<< Employee::getCount() << endl;

cin.ignore();

cout << "\nPress Enter to continue...\n";

cin.get();

return 0;

} // end main()