**C++程式設計HW5(Chapter 8) Due: 5/21/2020**

1. **True/False 是非題:15%**
   1. True/False:Array is a variable that can store multiple values of the different type and its values are stored in adjacent memory locations.//same type
   2. True/False: The following statement initializes all five elements of the number array to 1.

int number[5] = {1};

* 1. True/False: To assign the entire contents of one array to another, you can use the assignment operator.
  2. True/False: If we want to add 1 to a[ i ], we can use a[ i++ ] to reach our goal.
  3. True/False: We will get the errorMessage because the array has its boundary.

int main()

{

int a[] = {1,2,3,4,5};

a[9] = a[1]+a[3];

cout<<a[9];

}

* 1. True/False: When you pass an array as an argument to a function, the function can modify the contents of the array.
  2. True/False: A one-dimensional array can only store elements of a single data type, but a two-dimensional array can hold data of two different data types.
  3. True/False: An element of a two-dimensional array is referenced by the array name and two subscripts, first the element row number and then the element column number.
  4. True/False: When you create a vector it is unnecessary to specify how many elements it will hold because it will expand in size as you add new values to it.
  5. True/False: We can use a while loop with a Boolean variable to observe that the two arrays are equal or not.
  6. True/False: C++ allows a vector to be created with no elements in it.
  7. True/False: C++ allows arrays to have more than three dimensions.
  8. True/False: After carrying out the following two statements, sales will have

been created as a one-dimensional array that can hold 20 double values.

typedef salesArray double[20];

salesArray sales;

* 1. True/False: It’s necessary that all objects in the array use the same constructor.
  2. True/False: If there are fewer initializer calls in the list than there are objects in the array, the default constructor will be called for all the remaining objects.

1. **Choice 選擇題:15%**

1. To access an array element, use the array name and the element's \_\_\_\_\_\_\_\_.

A) name

B) data type

C) subscript

D) value

E) size declarator

2. The statement

int grades[ ] = { 100, 90, 99, 80 };

is an example of \_\_\_\_\_\_\_\_.

A) default arguments

B) an illegal array declaration

C) an illegal array initialization

D) implicit array sizing

E) data encapsulation

3. If you leave out the size declarator in an array declaration \_\_\_\_\_\_\_\_.

A) the array will contain no elements

B) the array size defaults to 100 elements

C) the value of each array element is set to a default value of 0

D) you must furnish an initialization list

E) the array cannot be created

4. The statement

typedef int oneDArray[20];

does which of the following?

A) creates an array of 20 integers

B) makes oneDArray a copy of another 20-integer array

C) makes oneDArray an alias for a data type that holds 20 integers

D) creates a one-dimensional integer array with all elements initialized to 20

E) None of the above

5. A two-dimensional array can be viewed as \_\_\_\_\_\_\_\_.

A) two rows of values

B) two columns of indexes

C) a table with rows and columns

D) any of the above

E) None of the above

6. An array of 10 integers named myArray can have its contents displayed with which of the following statements?

A) cout<<myArray;

B) cout<<myArray[];

C) cout<<myArray[10];

D) cout<<myArray[0-9];

E) None of the above

7. You can assign the contents of one array to another by using \_\_\_\_\_\_\_\_.

A) the assignment operator

B) the equality operator

C) both array names

D) A and C together

E) None of the above

8. To add up all the values in a two-dimensional array it would be best to use \_\_\_\_\_\_\_\_.

A) one for loop

B) two separate for loops

C) a nested for loop

D) no loop

E) one sentinel controlled loop

9. If the array defined as int myArray[20][10] is being passed to a function named displayArray, along with information on the number of rows and number of columns, which of the following function **calls** is correct?

A) displayArray(myArray, 20, 10);

B) displayArray(myArray[ ][ ], 20, 10);

C) displayArray(int myArray, 20, 10);

D) displayArray(myArray[20][10]);

E) None of the above

10. When a relationship is established between two or more arrays by using the same subscript to relate entries between the arrays, the arrays are called \_\_\_\_\_\_\_\_ arrays.

A) linked

B) indexed

C) parallel

D) brother

E) paired

11. If employee is an array of objects with a public member function named setHoursWorked, which of the following statements correctly calls that function for the employee object in array element 5?

A) employee.setHoursWorked[5] = 40;

B) employee[5].setHoursWorked = 40;

C) employee.setHoursWorked[5](40);

D) employee[5].setHoursWorked(40);

E) setHoursWorked(employee[5], 40);

12. Consider the following function definition:

void too2(int a[], int N)

{ for(int i=0;i<N;i++)a[i]=2; }

Which of the following is(are) acceptable function call?

int myArray[29];

A) too2(myArray,10);

B) too2(myArray,29);

C) too2(myArray,50);

D) too2(myArray[],10);

E) too2(myArray[3],29);

13. Consider the following structure declaration:

struct Sam

{ int a;

double b;

Sam(int n=0, double d=0.0){a=n;b=d;}

};

Which of the following is(are) acceptable?

A) Sam sam[3] = {Sam(10,9.5),Sam(5,10.5),Sam(15,8.5)};

B) Sam sam[3] = {Sam(10,9.5),Sam(5,10.5) };

C) Sam sam[3] = {Sam(10,9.5), ,Sam(15,8.5)};

D) Sam sam[3] = {{10,9.5},{5,10.5},{15,8.5}};

\*\*E) Sam sam[3] = {10,9.5,5,10.5,15,8.5};

14. Regarding arrays, which of the following statement is incorrect?

A) An array can be used to store and manipulate a collection of data that is all of the same type. //T

B) The indexed variables of an array can be used just like any other variables of the base type of the array.

C) The most common programming error made when using array is attempting to access a nonexistent array index since array does not provide range check of its index. //T

D) An array formal parameter is a call-by-reference parameter, not a call-by-value parameter, therefore any change made to the formal parameter in the function body will be made to the array argument when the function is called. //T

E) When the array is passed as an argument to a function, only the address of the first indexed variable (array[0]) is given to the called function. Therefore a function with an array parameter usually needs another formal parameter of type int to give the size of the array. //T

15. About arrays of objects, which of the following statement is(are) correct?

A) If you do not use an initialization list when an array of objects is created, the default constructor will be invoked for each object in the array. //T

B) If you do use an initialization list when an array of objects is created, the correct constructor will be called for each object, depending on the number and type of arguments used

C) If a constructor requires more than one argument, the initializer must take the form of a constructor function call

D) If there are fewer initializer calls in the list than there are objects in the array, the default constructor will be called for all the remaining objects

E) It is best to always provide a default constructor; but if there is none you must be sure to furnish an initializer for every object in the array

1. **Programming 程式題: 70% (10% each)**

**1. Chips and Salsa**

Write a program that lets a maker of chips and salsa keep track of their sales for five different types of salsa they produce: mild, medium, sweet, hot, and zesty. It should use **two parallel five-element arrays**: an array of strings that holds the five salsa names and an array of integers that holds the number of jars sold during the past month for each salsa type. The salsa names should be stored using an **initialization list** at the time the name array is created. The program should prompt the user to enter the number of jars sold for each type. Once this sales data has been entered, the program should produce a report that displays sales for each salsa type, total sales, and the names of the highest selling and lowest selling products.

Input validation: Do not accept user inputs negative integer or non-integer for number of jars sold.

**2. Driver’s License Exam**

The State Department of Motor Vehicles (DMV) has asked you to write a program that grades the written portion of the driver’s license exam, which has 20 multiple choice questions. Here are the correct answers:



To do this you should create a **TestGrader** class. The class will have an **answers** array of 20 characters, which holds the correct test answers. It will have two public member functions that enable user programs to interact with the class: **setKey** and **grade**. The **setKey** function receives a 20-character string holding the correct answers and copies this information into its **answers** array. The **grade** function receives a 20-character array holding the test taker’s answers and compares each of their answers to the correct one. An applicant must correctly answer 15 or more of the 20 questions to pass the exam. After “grading” the exam, the **grade** function should create and return to the user a string that includes the following information:

* a message indicating whether the applicant passed or failed the exam

• the number of right answers and the number of wrong answers

• a list of the question numbers for all incorrectly answered questions.

The client program that creates and uses a **TestGrader** object should first make a single call to **setKey**, passing it a string containing the 20 correct answers. Once this is done it should allow a test taker’s 20 answers to be entered, making sure only answers of A–D are accepted, and store them in a 20-character array. Then it should call the **grade** function to grade the exam and should display the string the function returns. The program should loop to allow additional tests to be entered and graded until the user indicates a desire to quit.

Input validation: Do not accept test taker inputs other than A – D for answers.

**3. Lottery**

Write a program that simulates a lottery. The program should have an array of 5 integers named winningDigits, with a randomly generated number in the range of 0 through 9 for each element in the array. The program should ask the user to enter 5 digits and should store them in a second integer array named player or quit to quit the game. The program must compare the corresponding elements in the two arrays and count how many digits match. For example, the following shows the winningDigits array and the player array with sample numbers stored in each. There are two matching digits, elements 2 and 4.



Once the user has entered a set of numbers, the program should display the winning digits and the player’s digits and tell how many digits matched. Then the program repeats the game.

Input validation: Do not accept player inputs less than 0 or greater than 9.

**4.Rainfall Statistics**

Create a **Stats class** whose member data includes a string for storing the county (city) name, a string for storing staring year- month: yyyymm (e.g., 201901, 199812, 201003, etc.) an array capable of storing **30** double data values for consecutive monthly rainfall, and whose member functions include total, average, lowest, and highest functions for returning information about the data to the client program. The total and average functions return the total and average rainfalls of the rainfall values stored in the data member, respectively. The lowest and highest functions return the month-year and rainfall of the lowest and highest rainfall that fell in those months. In addition to these functions, the Stats class should have a Boolean storeValue function that accepts a double value for rainfall amount from the client program and stores it in the array. Input validation: Do not accept rainfall amounts less than 0. It is the job of this function to keep track of how many values are currently in the array, so it will know where to put the next value it receives and will know how many values there are to process when it is carrying out its other functions. It is also the job of this function to make sure that no more than 30 values are accepted. If the storeValue function is able to successfully store the value sent to it, it should returntrue to the client program. However, if the client program tries to store a thirty-first value, the function should not store the value and should returnfalse to the client program. The class should also have a displayReport function to print a summary rainfall report similar to the following:

January 2010 –April 2012 Rain Report for xxxxxxx County

Total rainfall in this period: 23.19 inches

Average monthly rainfall: 1.93 inches

The least rain fell in January, 2011 with 0.24 inches

The most rain fell in May, 2010 with 4.29 inches

The client program should create (using a constructor with initial values for starting yyyymm and county name) and use a Stats object to carry out the rainfall analysis. Then generate a report. Notice that the Stats object does no I/O. All input and output is done by the client program.

**5. Drink Machine Simulator**

Create a class that simulates and manages a soft drink machine. Information on each drink type should be stored in a structure that has data members to hold the drink name, the drink price, and the number of drinks of that type currently in the machine. The class should have an array of five of these structures, initialized with the following data.



The class should have two public member functions, displayChoices (which displays a menu of drink names and prices) and buyDrink (which handles a sale). The class should also have at least two private member functions, inputMoney, which is called by buyDrink to accept, validate, and return (to buyDrink) the amount of money input, and dailyReport, which is called by the destructor to report how many of each drink type remain in the machine at the end of the day and how much money was collected. You may want to use additional functions to make the program more modular.

The client program that uses the class should have a main processing loop which calls the displayChoices class member function and allows the patron to either pick a drink or quit the program. If the patron selects a drink, the buyDrink class member function is called to handle the actual sale. This function should be passed the patron’s drink choice.

Here is what the buyDrink function should do:



Input Validation: Only accept valid menu choices. Do not deliver a beverage if the money inserted is less than the price of the selected drink.

**6. Bin Manager Class**

Design and write an object-oriented program for managing inventory bins in a warehouse. To do this you will use two classes: InvBin and BinManager. The InvBin class holds information about a single bin. The BinManager class will own and manage an array of InvBin objects. Here is a skeleton of what the InvBin and BinManager class declarations should look like:





**Client Program**

Once you have created these two classes, write a menu-driven client program that uses a BinManager object to manage its warehouse bins. It should initialize it to use 9 of the bins, holding the following item descriptions and quantities. The bin index where the item will be stored is also show here.

The modular client program should have functions to display a menu, get and validate the user’s choice, and carry out the necessary activities to handle that choice. This includes adding items to a bin, removing items from a bin, and displaying a report of all bins. Think about what calls the displayReport client function will need to make to the BinManager object to create this report. When the user chooses the “Quit” option from the menu, the program should call its displayReport function one last time to display the final bin information. All I/O should be done in the client class. The BinManager class only accepts information, keeps the array of InvBin objects up to date, and returns information to the client program.

Input Validation in the BinManager class: The class functions should not accept numbers less than 1 for the number of parts being added or removed from a bin. They should also not allow the user to remove more items from a bin than it currently holds.

**7. Theater Ticket Sales**

Create a TicketManager class and a program that uses it to sell tickets for a single performance theater production. Here are some suggestions:

* Design and write the client program that uses the class, the TicketManager class and all of its functions.
* The class design and the names, parameters, and return types of each function should be decided in advance.
* The project can be implemented either as a multi-file program, or all the functions can be cut and pasted into a single file.

Here are the **specifications**:

* The theater’s auditorium has **15 rows**, with **30 seats** **in each row**. To represent the seats, the **TicketManager** class should have a two-dimensional array of **SeatStructures**. Each of these structures should have data members to keep track of the seat’s price and whether or not it is available or already sold.
* The seat prices should be read in from the **SeatPrices.dat file**. It contains 15 values representing the price for each row. All seats in a given row are the same price, but different rows have different prices. The seat availability information should be read in from the **SeatAvailability.dat file**. It contains 450 characters (15 rows with 30 characters each), indicating which seats have been sold ('\*') and which are available ('#'). Initially all seats are available. However, once the program runs and the **file is updated**, some of the seats will have been sold. The obvious function to read in the data from these files and set up the array is the constructor that runs when the **TicketManager** object is first created.
* The client program should be a **menu-driven program** that provides the user with a menu of box office options, accepts and validates user inputs, and calls appropriate class functions to carry out desired tasks. The menu should have options to display the seating chart, request tickets, print a sales report, and exit the program.
* When the user selects the display seats menu option, a **TicketManager** function should be called that creates and returns a string holding a chart, similar to the one shown here. It should indicate which seats are already sold (\*) and which are still available for purchase (#). The client program should then display the string.

Seats

123456789012345678901234567890

Row 1 \*\*\*###\*\*\*###\*########\*\*\*\*\*####

Row 2 ####\*\*\*\*\*\*\*\*\*\*\*\*\*####\*\*\*\*\*\*\*##

Row 3 \*\*###\*\*\*\*\*\*\*\*\*\*########\*\*\*\*###

Row 4 \*\*######\*\*\*\*\*\*\*\*\*\*\*\*\*\*##\*\*\*\*\*\*

Row 5 \*\*\*\*\*\*\*\*#####\*\*\*\*\*\*\*\*\*########

Row 6 ##############\*\*\*\*\*\*\*\*\*\*\*\*####

Row 7 #######\*\*\*\*\*\*\*\*\*\*\*\*###########

Row 8 \*\*\*\*\*\*\*\*\*\*\*\*##\*\*\*\*############

Row 9 #########\*\*\*\*\*############\*\*\*\*

Row 10 #####\*\*\*\*\*\*\*\*\*\*\*\*\*############

Row 11 #\*\*\*\*\*\*\*\*\*\*#################\*\*

Row 12 #############\*\*\*\*\*\*\*\*########\*

Row 13 ###\*\*\*\*\*\*\*\*\*\*\*########\*\*######

Row 14 ##############################

Row 15 ##############################

* When the user selects the request tickets menu option, the program should prompt for the number of seats the patron wants, the desired row number, and the desired starting seat number. A **TicketManager** ticket request function should then be called and passed this information so that it can handle the ticket request. If any of the requested seats do not exist, or are not available, an appropriate message should be returned to be displayed by the client program. If the seats exist and are available, a string should be created and returned that lists the number of requested seats, the price per seat in the requested row, and the total price for the seats. Then the user program should ask if the patron wishes to purchase these seats.
* If the patron indicates they do want to buy the requested seats, a **TicketManager** purchase tickets module should be called to handle the actual sale. This module must be able to accept money, ensure that it is sufficient to continue with the sale, and if it is, mark the seat(s) as sold, and create and return a string that includes a ticket for each seat sold (with the correct row, seat number, and price on it).
* When the user selects the sales report menu option, a **TicketManager** report module should be called. This module must create and return a string holding a report that tells how many seats have been sold, how many are still available, and how much money has been collected so far for the sold seats. Think about how your team will either calculate or collect and store this information so that it will be available when it is needed for the report.
* When the day of ticket sales is over and the quit menu choice is selected, the program needs to be able to write the updated seat availability data back out to the file. The obvious place to do this is in the **TicketManager** destructor.