**EE2310 C++程式設計 HW 7 (Chapter 10 Pointers) due: 6/4/2020**

**Part 1, Choice(s) 選擇題 (30%)**

1) The \_\_\_\_\_\_\_\_, also known as the address operator, returns the memory address of a variable.

A) asterisk ( \* )

B) ampersand ( & )

C) percent sign (%)

D) exclamation point ( ! )

E) None of the above

2) With pointer variables, you can \_\_\_\_\_\_\_\_ manipulate data stored in other variables.

A) never

B) seldom

C) indirectly

D) All of the above

E) None of the above

3) The code segment int \*ptr; has the same meaning as \_\_\_\_\_\_\_\_.

A) int ptr;

B) \*int ptr;

C) int ptr\*;

D) int\* ptr;

E) None of the above

4) When you work with a dereferenced pointer, you are actually working with \_\_\_\_\_\_\_\_.

A) a variable whose memory has been deallocated

B) a copy of the value pointed to by the pointer variable

C) the variable whose address is stored in the pointer variable

D) All of the above

E) None of the above

5) Which arithmetic operations can be performed on pointers?

A) All arithmetic operations that are legal in C++

B) Multiplication, division, addition, and subtraction

C) Addition , subtraction , preincrement, and postincrement

D) Only multiplication and addition

E) None of the above

6) A pointer may be initialized with \_\_\_\_\_\_\_\_.

A) the address of an existing object of the appropriate type

B) the value of a floating-point constant

C) the value of a floating-point variable

D) All of the above

E) None of the above

7) The statement double \*num; \_\_\_\_\_\_\_\_.

A) defines a variable of type double called num

B) defines and initializes a pointer variable called num

C) initializes a variable called \*num

D) defines a pointer variable called num

E) None of the above

8) When the less than ( < ) operator is used between two pointer variables, the expression is testing whether \_\_\_\_\_\_\_\_.

A) the value pointed to by the first is less than the value pointed to by the second

B) the value pointed to by the first is greater than the value pointed to by the second

C) the address of the first variable comes before the address of the second variable in the computer's memory

D) the first variable was declared before the second variable

E) None of the above

9) Assuming that arr is an array identifier, the statement sum += \*arr; \_\_\_\_\_\_\_\_.

A) is illegal in C++

B) will always result in a compiler error

C) adds the value stored in arr[0] to sum

D) adds the address of the pointer arr to sum

E) None of the above

10) The delete operator should only be used on pointers that \_\_\_\_\_\_\_\_.

A) have not yet been used

B) have been correctly initialized

C) point to storage allocated by the new operator

D) are appropriately dereferenced

E) None of the above

11) A function may return a pointer, but the programmer must ensure that the pointer \_\_\_\_\_\_\_\_.

A) is pointing to an object that is still valid after the return of the function

B) has been assigned an address

C) was received as a parameter by the function

D) has not previously been returned by another function

E) None of the above

12) Which of the following statements is not valid C++ code?

A) int ptr = &num1;

B) int ptr = int \*num1;

C) float num1 = &ptr2;

D) All of the above are valid

E) All of the above are invalid

13) Which of the following statements correctly deletes a dynamically-allocated array pointed to by p?

A) delete p;

B) p delete[ ];

C) delete [ ] p;

D) delete array p;

E) None of the above

14) You may use a pointer to a structure as a \_\_\_\_\_\_\_\_.

A) function parameter

B) structure member

C) function return type

D) All of the above

E) None of the above

15) If arr is an array identifier and k is an integer, the expression arr[k] is equivalent to

A) \*(arr + k)

B) \*arr + k

C) &arr[k]

D) arr + k

E) None of the above

16) To dereference a structure pointer and simultaneously access a member of the structure, the appropriate operator to use is \_\_\_\_\_\_\_\_.

A) the ampersand, &

B) an asterisk, \*

C) the structure pointer operator, ->

D) the dereference operator, <-

E) None of the above

17) When the \_\_\_\_\_\_\_\_ is placed in front of a variable name, it returns the address of that variable.

A) asterisk ( \* )

B) conditional operator

C) ampersand ( & )

D) semicolon ( ; )

E) None of the above

18) The statement cout << &num1; will output \_\_\_\_\_\_\_\_.

A) the value stored in the variable called num1

B) the memory address of the variable called num1

C) the number 1

D) the string "&num1"

E) None of the above

19) A pointer variable is designed to store \_\_\_\_\_\_\_\_.

A) any legal C++ value

B) only floating-point values

C) a memory address

D) an integer

E) None of the above

20) The statement int \*ptr; means \_\_\_\_\_\_\_\_.

A) the variable called ptr will store an integer value

B) the variable called \*ptr will store an asterisk and an integer value

C) ptr is a pointer variable that will store the address of an integer variable

D) All of the above

E) None of the above

21) The statement cout << \*ptr; will output \_\_\_\_\_\_\_\_.

A) the value stored in the variable whose address is contained in ptr

B) the string "\*ptr"

C) the address of the variable stored in ptr

D) the address of the variable whose address is stored in ptr

E) None of the above

22) The \_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_ operators can respectively be used to increment and decrement a pointer variable.

A) dereferencing, indirection

B) modulus, division

C) ++, --

D) All of the above

E) None of the above

23) The statement cin >> \*p; \_\_\_\_\_\_\_\_.

A) stores the keyboard input into the variable p

B) stores the keyboard input into the pointer called p

C) is illegal in C++

D) stores the keyboard input into the variable pointed to by p

E) None of the above

24) Dynamic memory allocation occurs \_\_\_\_\_\_\_\_.

A) when a variable is created by the compiler

B) when a variable is created at run-time

C) when a pointer fails to dereference the right variable

D) when a pointer is assigned an incorrect address

E) None of the above

25) The statement int \*ptr = new int; acquires memory to hold an integer and then \_\_\_\_\_\_\_\_.

A) initializes the allocated memory to 0

B) assigns an integer value to the variable called ptr

C) sets ptr to point to the allocated memory

D) creates a new pointer called int

E) None of the above

26) Any time you use the new operator, it is good practice to \_\_\_\_\_\_\_\_.

A) use delete afterwards to free the memory allocated by new

B) use a preprocessor directive

C) clear the data from the old operator

D) All of the above

E) None of the above

27) If dynamically allocated memory is not freed, \_\_\_\_\_\_\_\_.

A) the system may run out of memory

B) it results in a compiler error

C) a run-time error informs your user that the program did not free memory space

D) the source code will not link correctly

E) None of the above

28) A reason for passing a pointer to a function is \_\_\_\_\_\_\_\_.

A) to avoid the overhead of copying large data structures

B) to allow the called function to modify a variable accessible to the calling function

C) to allow easy access to data in the function that is being called

D) A and B are both true

E) None of the above

29) A pointer variable may be initialized with \_\_\_\_\_\_\_\_.

A) any non-zero integer value

B) the address of an existing variable of the appropriate type

C) A and B are both true

D) None of the above

30) If a variable occupies more than one byte of memory, its address is \_\_\_\_\_\_\_\_.

A) the address of the last byte of storage allocated to it

B) the average of the addresses used to store the variable

C) the address of the first byte of storage allocated to it

D) general delivery

E) None of the above

**Part 2, True/False 是非題 (10%)**

1. The statement Rectangle \* boxPtr; defines a variable boxPtr to be a pointer pointing to a type Rectangle. T
2. With pointer variables you can access, but you cannot modify, data in other variables. F
3. An array name is a pointer constant because the address it represents cannot be changed during run-time. F
4. Variables cannot be created when a program is already running. F
5. The expression s->m has the same meaning as (\*s).m F
6. A pointer can be passed as an argument to a function. T
7. Any arithmetic operation may be performed on pointers. F
8. The ampersand (&) is used to dereference a pointer variable in C++. F
9. The expression \*s->p; is only meaningful if s is a pointer to a structure and p is a pointer that is a member of that structure. T
10. It is possible for a structure to contain as a member a pointer to its own structure type. T

**Part 3, Code 程式題 (60%)**

**1) Money Donation (10%)**

Modify Program Example in pptx pp.10-108 (the United Cause case study program) so it can be used with any set of donations. The program should **dynamically allocate the donations array** and ask the user to input its values. Then modify the program so the arrPtr array is sorted in descending order instead of ascending order.

**2) Test Scores (10%)**

Write a program that **dynamically allocates an array** large enough to hold a user-defined number of test scores.

Once all the scores are entered, the array should be passed to a function that sorts them in ascending order.

Another function should be called that calculates the average score.

The program should display the sorted list of scores and averages with appropriate headings.

Use pointer notation rather than array notation whenever possible.

<note> Input Validation: Do not accept negative numbers for test scores.

**3) Pie a la Mode (10%)**

In statistics, the **mode** of a set of values is the value that occurs most often and the **median** is the value that lies in the middle when the values are arranged in sorted order.. Write a program that determines how many pieces of pie most people eat in a year. Set up an integer array that can hold responses from **30 people**. For each person, enter the number of pieces they say they eat in a year. Then write a function that finds the mode of these 30 values. This will be the number of pie slices eaten by the most people. The function that finds and returns the mode should accept two arguments, an array of integers, and a value indicating how many elements are in the array. Then write a function that determines the median of a sorted array. The function should take an array of numbers and an integer indicating the size of the array and return the median of the values in the array. You should check if the array is already sorted. If not, sort the array. **Use pointer notation whenever possible**.

4) **Statistics Report of Monthly Rainfall (10%)**

Based on programming #4 in HW5, redesign your program so that (a) it uses **dynamic storage instead of fixed size array to store rainfall data**, (b) in addition to the original function and report, it also displays the name of each month in the period (at least a year) and its rainfall amount, **sorted** in order of rainfall from highest to lowest. You should modify your program accordingly. Make the program modular by calling on different functions to input the rainfall amounts, to sort the data, and to display the data.

**5) Modified Bin Manager Class (10%)**

Modify the BinManagerclass you wrote for Programming #6 in HW5 to overload its getQuantity, addParts, and removePartsfunctions as shown here:

bool addParts(string itemDescription, int q);

bool removeParts(string itemDescription, int q)

int getQuantity(string itemDescription);

These new functions allow parts to be added, parts to be removed, and the quantity in stock for a particular item to be retrieved by using an **item description**, rather than a bin number, as an argument. In addition to writing the three overloaded functions, you will need to create a private BinManagerclass function that uses the item description as a search key to locate the index of the desired bin.

Test the new class functions with the similar client program you wrote for Programming #6 in HW5 but using **dynamic array** to store your objects (at least allocate 20 objects), modifying it to call the new functions. Be sure to use some descriptions that match bins in the array and some that do not.

As you did in the previous Bin Manager program, if an add or remove operation is successfully carried out, make the function return true. If it cannot be done—for example, because the string passed to it does not match any item description in the array—make the function return false. If the getQuantityfunction cannot locate any item whose description matches the one passed to it, make it return −1.

**6) Television (10%)**

Create a class called Television that has the member variables:

displayType – a string that store display types

dimension -- a double variable for storing dimension of the TV in inches

connectivitySupport – a **dynamic array of strings** that stores the different connectivity modes supported by the TV.

The class should have mutator and accessor to set and get the member variables. Include a constructor that takes arguments of type string, double, and an array of strings to assign the three member variables.

Embed your class in a test program that reads the input from the user to set displayType, dimension, and connectivitySupport values by default for a television. Your program should then read in input for the number of televisions. Your program should create as many Television objects as required (demo at least 5 TVs) (with default member variables values) and ask the user to if customization is required for any television. If so, use the mutator methods to set the values accordingly.