All multiple choice questions are equally weighted. You can generally assume that code shown in the questions is intended to be syntactically correct, unless something in the question or one of the answers suggests otherwise.

1. **CharQueue is a Queue ADT which stores characters. After the following statements have executed, what values are at the front and rear of the queue, respectively? ["enqueue" and "dequeue" correspond to "insert" and "remove" as used in some lecture examples.]**

```
CharQueue CQ;
CQ.enqueue('a');
CQ.enqueue('b');
CQ.enqueue('c');
CQ.enqueue('d');
char c1 = CQ.dequeue();
char c2 = CQ.dequeue();
CQ.enqueue(c2);
if (CQ.front() == 'b')
   CQ.enqueue('e');
else
   CQ.enqueue('f');
```

A. front c
rear f
B. front c
rear e
C. front b
rear f
D. front b
rear e
E. front e
rear b

2.
```
double d0, d1, d2, d3;
double * dp = new double[4];
double da[4];
```

Which statement is FALSE:
A. dp = da;
is legal, but probably illogical
B. dp[2] = 3.14;
is legal
C. *dp = 3.14;
is legal
D. da = &d0;
is legal
E. d0[2] = 3.14;
is illegal

3. **If you see this line inside a class declaration:**

   ```
   ~X ( );
   ```

   Then:
   A. You know that the class being declared is named X.
   B. You known that the programmer is really inexperienced, because the character ~ cannot be used in identifiers.
   C. It's likely, although not certain, that the class uses dynamic memory.

   A. A only
   B. B only
   C. C only
   D. A and C
   E. None of A, B, or C

4. **Assume that q is an instance of a C++ class IntQueue, and q currently contains 15, 20, 10, and 30 from front to rear (shown left to right):**

   (15, 20, 10, 30)

   If the code segment

   ```
   while (q.getFront( ) != 10)
      q.Dequeue( );
   ```

   is now executed, the resulting contents is:

   A. (30)
   B. (10, 30)
   C. (15, 20, 10)
   D. (15, 20)
   E. (10)

5. **Choose the true statement about this function:**

   ```
   void Xchange (double & a, double & b) {
     //swap the values of the two parameters

   double *temp;   //line A
   temp = new double;   //line B

   *temp = a;  //line C
   a = b;
   b = *temp;
   }
   ```

   A. syntax error on line A
   B. Line B is not an error, but it is completely unnecessary.
   C. Lines A and B are fine, but line C has a semantic error, because the pointer used is not initialized.
   D. Lines C and the following 2 lines are bad, because instead of a and b you need to write *a and *b
   E. There is no syntax error in the function, but there is a memory leak.

6. **Examine this code. Will the compiler complain (give a warning or error?)**

```
int compute (int & A, int & B) {
    A++;  //line  A
    B++;  //line B
    return (A + B);
}
...
int x = 1;
const int y = 2;
const int z = compute (x, y);  //line D
```

A. compiler complaint on line A

B. compiler complaint on line B

C. compiler complaint on line D, because of the arguments passed to compute

D. compiler complaint on line D, because of having z on the left-hand side of the = sign.

E. No complaints.

7. **The "delete" operator in C++ ...**

A. can be applied to any variable which is a class instance
B. can be used as an easy way to clear memory (set it to zeros or NULL)
C. can only be used in a destructor

A. A only

B. B only

C. C only

D. A, B, and C

E. none of A, B, or C

8. **What is the best description of the result of the following lines of code:**

```
int * ip;
...
ip = new int [100];  // line A
```

A. The default constructor for ip is called 100 times.

B. An area equal in size to an integer has been allocated on the heap.

C. An area equal in size to 100 integers has been allocated on the heap.

D. An area equal in size to 99 integers has been allocated on the heap.

E. The area pointed to by ip has been initialized.

9. Here is a partial declaration of listClass, which is intended to implement a list (vector) ADT:

```
class listClass {
  public:
void ListInsert (int NewPosition, listItemType NewItem, bool & success);

  ...
  private:
    int  size;      //number of items in the list
    listNode * head;    //ptr to the 1st item in the list, or NULL if size is 0
...};
```

Suppose that the following assert occurs inside the implementation of the ListInsert method:

```
assert ((success && (head != NULL)) || !success);  //line A
```

What is an appropriate use of this assertion?

A. As a precondition (but not as a postcondition)
B. As a postcondition (but not as a precondition)
C. As both a postcondition and a precondition
D. line A would be a syntax error inside ListInsert, because head is not a parameter of the function
E. The situation head != NULL is always true, so it is pointless (but harmless) to have such an assert.

10. int size, front, rear;

Above are some private member variables from a array-based inplementation of a queue (with wrap-around). In this implementation, an empty queue is detected by the condition (size == 0)
and after an insert (enqueue), the rear index is updated by the following statement:
rear = (rear+1) % MAX;

What would be an appropriate test for the queue being full?

A. (rear == size)
B. (rear == MAX)
C. (size == MAX - 1)
D. (size == MAX)
E. (front == size)

11. Suppose you see this in a C++ header file:

```
class X: public Y {   ...
  };
```

Which of the following are true?

A. X is a derived class of Y
B. Y is a base class (with respect to X)

A. A only
B. B only
C. Both A and B
D. Neither A nor B
E. This can't be from a valid header file, because the keyword "public" belongs inside the { }.

12. **Class X has a method with the following implementation:**

```
X::X (const X & Y) {
   *this = Y;
}
```

Assume this compiles without error.

You ask a colleague to read and comment on this part of your program. Which of these comments about the code is accurate, based on what is given?

A. One problem is that there are no comments. Otherwise, the code looks OK, although it doesn't seem useful unless the class X has an overloaded operator =.
B. Causes an infinite recursion, because the assignment statement (*this = Y) will cause this same method to be called again (and again, etc.).
C. Assigning one class object to another (as with *this = Y) is always a shallow copy, so there's no point in defining such a method at all.
D. Since the code compiles without errror, you must have decleared "this" as a member variable of class X.
E. There will be a run-time error, because this is a const method.

13. **Referencing data through a pointer is called…**

A. deferencencing
B. decomposition
C. deconstruction
D. devolution
E. disgusting

14. **Suppose you see this in a C++ header file:**

```
class aard: public vark {

};
```

What's the best way to describe this?

A. aard has a vark
B. aard is a kind of vark
C. vark has a aard
D. vark is a kind of aard
E. aard is an instance of vark

15. Here is a fragment of a program (assume the program compiles and runs correctly). What will be the last value of temp (which is then printed)?

```
.............................
intStack S;  // a stack of integers (notes below)
int value;
cout << "Enter two positive numbers: ";
cin >> num1 >> num2;
S.push(num1);
S.push(num1);

int temp;
while (!S.isEmpty( )) {
   temp = S.pop() + S.pop();
   if (temp < num2) {
       S.push(temp);
       S.push(num1);
   }
}
cout << temp;
```

For this particular implementation of the stack, assume the prototypes of pop and push are

```
void push(int);
int   pop( );
```

A. It will be num1 * num2
B. It will be a multiple of num1
C. It will be a multiple of 2
D. It will be num2 raised to a power
E. It will be num1 raised to a power

16. (5+10 points) Dynamically allocated arrays are an appropriate implementation technique for lists, stacks, and queues. As new data is inserted, the array should "grow" if needed. Start from the following partial class declaration for a stack of cards. Note that there are few comments, but you should be able to make reasonable inferences from your knowledge of stacks and from seeing other implementations of stack. There is a class called Card (you shouldn't need any information about it except its name).

```
class CardStack {
public:
    cardStack( );
    bool  push  (const Card & thisCard);
    Card    pop   ( );
...
private:
    int size; // number of elements currently in the stack
    int capacity; // current size of the array. capacity >= size at all times
    int stackTop; // array index of the top element
    Card * elements; // the array
...};
```

Implement
A. the constructor carStack() and
B. the method push( ).
You should not call any functions or methods, or need to use any member variables other than those defined here. Try to write complete, compilable code.

Hints: the new operator can be used to allocate an entire array of values.

```
int * ip;
ip = new int[10]; //gets a pointer to an array of 10 ints
ip[3] = 15; // use the pointer like an array
delete [] ip; // syntax for deleting the whole array
```

A.
B.
C.
D.
E.

TRUE AND FALSE (2 pts. each). Circle your answer on this page – do not bubble in on the mark-sense form

17. True or False (circle): An event list is a LIFO (last in, first out) data structure.

18. True or False (circle): If a class has dynamic memory and a copy constructor, but does not overload operator=, assignments can result in memory leaks.

19. True or False (circle): Local variables can be explicitly deallocated using delete.

20. True or False (circle): A const method is allowed to modify its local variables.

21. True or False (circle): For a linked list-based Stack, assuming isEmpty() and pop() are implemented properly, the following code is a sufficient implementation for the destructor ~Stack:

```
Stack::~Stack() {
   while (!isEmpty())
      pop();
}
```