TERM	COURSE NAME	COURSE CODE	VERSION
OOP345-Winter2020-Quiz-1	Object-Oriented Software Development using C++	OOP345	**A**

1. Encapsulation
    a. refers to the integration of data and logic within a class' implementation that establishes the crisp interface between the implementation and any client
    b. maintains high cohesion within a class and low coupling between the class' implementation and any one of its clients.
    c. All of the above
    d. None of the above
2. Inheritance:
    a. one class inherits the structure of another class
    b. a single interface provides multiple implementations
    c. All of the above
    d. None of the above
3. Polymorphism:
    a. one class inherits the structure of another class
    b. a single interface provides multiple implementations
    c. All of the above
    d. None of the above
4. A Class:
    a. describes the structure common to a set of similar objects
    b. doesn’t describes the structure common to a set of similar objects
    c. All of the above
    d. None of the above
5. Encapsulation:
    a. Internal linkage
    b. External linkage
    c. None of the above
6. A module consists of:
    a. Header file
    b. Implementation file
    c. All of the above
    d. None of the above
7. A module's header file:
    a. declares the names that are exposed to client modules
    b. defines the module's logic
    c. All of the above
    d. None of the above
8. A module’s implementation file:
    a. declares the names that are exposed to client modules
    b. defines the module's logic
    c. All of the above
    d. None of the above
9. A module's implementation file:
    a. Needs the header files of those modules that define classes or functions used in the implementation file
    b. Doesn’t need the header files of those modules that define classes or functions used in the implementation file
    c. All of the above
    d. None of the above
10. The stages of creating an executable are:
    a. preprocessing,
    b. compiling,
    c. linking
    d. All of the above
    e. None of the above
11. it is good practice to:
    a. write the suite of unit tests for each module of an application before coding the module's implementation
    b. not to write the suite of unit tests for each module of an application before coding the module's implementation
    c. All of the above
    d. None of the above

12. the memory available to an application at run-time consists of:
    a. static memory
    b. dynamic memory
    c. All of the above
    d. None of the above
13. Which statement is correct:
    a. the operating system provides dynamic memory to an application at run-time upon request
    b. the linker provides dynamic memory to an application at run-time upon request
    c. All of the above
    d. None of the above
14. the keyword **new []** allocates a contiguous region of dynamic memory and returns its starting address at:
    a. compile time
    b. linking time
    c. run-time
    d. All of the above
    e. None of the above
15. allocated memory must be deallocated within the scope of the pointer that holds its address:
    a. Yes
    b. Otherwise, might produce a memory leak
    c. All of the above
    d. None of the above
16. object-oriented classes may contain:
    a. data members
    b. member functions
    c. both data members and member functions
    d. None of the above
17. the keyword **private** identifies subsequent members [data & functions] as:
    a. inaccessible to any client
    b. Accessible to all clients
    c. All of the above
    d. None of the above
18. the keyword **public** identifies subsequent members [data & functions] as:
    a. inaccessible to any client
    b. Accessible to all clients
    c. All of the above
    d. None of the above
19. data members:
    a. hold the information about an object's state
    b. describe the logic that an object performs on its data members
    c. All of the above
    d. None of the above
20. member functions:
    a. hold the information about an object's state
    b. describe the logic that an object performs on its data members
    c. All of the above
    d. None of the above
21. a query report:
    a. the state of an object without changing its state
    b. the state of an object and changes its state
    c. All of the above
    d. None of the above
22. a modifier:
    a. changes the state of an object
    b. reports the state of the object without changing its state
    c. All of the above
    d. None of the above
23. an empty state:
    a. is the set of data values that identifies the absence of valid data in an object
    b. is the set of data values that identifies the presence of valid data in an object
    c. All of the above
    d. None of the above

24. a helper function:
    a. is a global function that supports a class
    b. refers to the class that it supports through its explicit parameter(s)
    c. All of the above
    d. None of the above
25. a helper operator is typically:
    a. an operator that does change the value of its operands
    b. Implement it as a helper function if it doesn't require to change the value of its operand
    c. All of the above
    d. None of the above
26. a friend function:
    a. has direct access to the private members of the class that granted the function friendship
    b. doesn't have direct access to the private members of the class that granted the function friendship
    c. All of the above
    d. None of the above
27. The access modifier **protected:**
    a. identifies subsequent members [data & functions] as inaccessible to any client code
    b. limits access to derived classes, and denies access to client code
    c. All of the above
    d. None of the above
28. any member function of a derived class may access:
    a. Any **protected** or **public** member of its base class
    b. Any **protected** or **public** or **private** member of its base class
    c. All of the above
    d. None of the above
29. The keyword **virtual** on a member function's declaration specifies:
    a. dynamic dispatch
    b. Static dispatch
    c. All of the above
    d. None of the above
30. It is important to deallocate the assigned dynamic memory allocation:
    a. To avoid memory leak
    b. To avoid memory corruption
    c. All of the above
    d. None of the above
31. an abstract base class:
    a. contains at least one pure virtual function
    b. contains no pure virtual function
    c. All of the above
    d. None of the above
32. an interface:
    a. is an abstract base class with no data members
    b. Is an abstract base class with data members
    c. All of the above
    d. None of the above
33. The scope of a name defined in local scope, refers to:
    a. The name has been declared within a function
    b. The name has been declared as a member of a class
    c. The name has been declared as a member of a named block
34. The scope of a name defined in class, refers to:
    a. The name has been declared within a function
    b. The name has been declared as a member of a class
    c. The name has been declared as a member of a named block
35. The scope of a name defined in namespace, refers to:
    a. The name has been declared within a function
    b. The name has been declared as a member of a class
    c. The name has been declared as a member of a named block
36. External linkage refers to:
    a. Connected across scopes in different modules
    b. Connected across scopes within its own module

37. Internal linkage refers to:
    a. Connected across scopes in different modules
    b. Connected across scopes within its own module
38. The keyword **static** identifies
    a. Internal linkage
    b. External linkage
39. The keyword **extern** identifies
    a. Internal linkage
    b. External linkage
40. Shadowing refers to:
    a. The entire region of a program
    b. A name declared within the scope of an identical shadows the entity that has the broader scope.
    c. None of the above

Code1.0

Main.cpp

```
1.  #include "generic.h"
2.  void increment();
3.  void display();
4.  int main() {
5.      increment();
6.      display();
7.      counter++;
8.      std::cout << " [main]    counter = " << counter << std::endl;
9.      std::cout << " [main]    Address of counter = " << &counter << std::endl;
10. }
```

Generic.h

```
1.  #include <iostream>
2.  static int counter = 0;
```

Unit_a.cpp

```
1.  #include "generic.h"
2.  void increment()
3.  {
4.      int counter = 2;
5.      counter = counter + 2;
6.      std::cout << " [increment]   Address of counter = " << &counter << std::endl;
7.  }
8.  void display( )
9.  {
10.     std::cout << "[display]    counter = " << counter << std::endl;
11.     std::cout << "[display]    Address of counter = " << &counter << std::endl;
12. }
```

41. Code 1.0 - Line 8 of Main.cpp, prints:
    a. [main] counter = 4
    b. [main] counter = 1
    c. [main] counter = 0
42. Code 1.0 - Line 10 of Unit_a.cpp, as a result of executing line 6 in Main.cpp prints:
    a. [display] counter = 4
    b. [display] counter = 1
    c. [display] counter = 0
43. Code 1.0 - The address printed in line 9 Main.cpp, is the same as the address printed in line 6 of unit_a.cpp
    a. YES
    b. NO

Code2.0

Main.cpp

```
1.  #include "generic.h"
2.  int  id = 20;
3.  void increment();
4.  void display();
5.  int main() {
6.      increment();
7.      display();
8.      counter++;
9.      std::cout << " [main]    counter = " << counter << std::endl;
10.     std::cout << " [main]    Address of counter = " << &counter << std::endl;
11. }
```

Generic.h

```
1.  #include <iostream>
2.  static int counter = 0;
```

Unit_a.cpp

```
1.  #include "generic.h"
2.  int id = 50;
3.  void increment()
4.  {
5.     int counter = 2;
6.     counter = counter + 2;
7.     std::cout << " [increment]   Address of counter = " << &counter << std::endl;
8.  }
9.  void display( )
10. {
11.     std::cout << "[display]    counter = " << counter << std::endl;
12.     std::cout << "[display]    Address of counter = " << &counter << std::endl;
13. }
```

Unit_b.cpp

```
1.  #include <iostream>
2.  extern int counter;
3.  extern int id;
4.  void present()
5.  {
6.      std::cout << "[present]     counter = " << counter << std::endl;
7.      std::cout << "[present]     Address of counter = " << &counter << std::endl;
8.      std::cout << "[present]     Address of id = " << id << std::endl;
9.  }
```

44. Code 2.0 - Building code 2.0 will result in duplicate symbol linker error because of:
    a. _id
    b. _counter
    c. All of the above
    d. None of the above
45. Code 2.0 - Building code 2.0 will result in undefiled symbol linker error because of
    a. _id
    b. _counter
    c. All of the above
    d. None of the above

Code3.0

Main.cpp

```
1. #include "generic.h"
2. int main (int argc, char *argv[]) {
3. int i;

4. std::cout << "Course : " << argv[0] << std::endl;
5. for (i = 1; i < argc; i++)
6.     std::cout << " - [" << argv[i][0] <<"]["<< argv[i][3] <<"]"<< std::endl;
7. }
```

Assume the following command line arguments are passed to the program in Code 3.0

Assignments Workshops Tests Exam

46. The first iteration of Line 6 code 3.0 will print

    a. [A][i]
    b. [W][k]
    c. [T][t]
    d. [E][m]
    e. [Assignments][Tests]

47. The second iteration of Line 6 code 3.0 will print

    a. [A][i]
    b. [W][k]
    c. [T][t]
    d. [E][m]
    e. [Assignments][Tests]

48. The third iteration of Line 6 code 3.0 will print

    a. [A][i]
    b. [W][k]
    c. [T][t]
    d. [E][m]
    e. [Assignments][Tests]