Exam #1 – Solutions

CS 162 – Winter 2019

The questions in this exam are all either multiple choice or true/false. There are 35 questions, and each one has only one correct answer. You may take the entire 50 minute lecture period to complete the exam. When you are finished, you may keep this copy of the exam, so feel free to write on it.

1. What is the git operation that makes a permanent snapshot of all staged changes and saves that snapshot in the local repository?
    a. add
    b. commit
    c. push
    d. log

2. What is the git operation that downloads an entire remote git repository including its history, changelogs, and files into a new local directory?
    a. clone
    b. commit
    c. pull
    d. add

3. True or false? The following snippet of code correctly frees all of the memory it allocates.

```
int* a = new int[8];
int* b =
for (int i = 0; i < 8; i++) {
     a[i] = i;
}
b = a;
delete[] a;
```

    a. True
    b. False

For questions 4–8, consider the following structure definitions:

```
struct point {
   float x;
   float y;
};

struct scatterplot {
   struct point* points;
   int n;
}
```

4. In the snippet below, is the `struct point` represented by `p` statically allocated or dynamically allocated?

   ```
   struct point* p;
   ```

   a. Statically allocated
   b. Dynamically allocated
   c. No `struct point` is allocated by the snippet above.
   d. ¯\_(ツ)_/¯

5. In the snippet below, is the `struct point` represented by `p` statically allocated or dynamically allocated?

   ```
   struct point p;
   ```

   a. Statically allocated
   b. Dynamically allocated
   c. No `struct point` is allocated by the snippet above.
   d. ¯\_(ツ)_/¯

6. Which line below correctly allocates a `struct point` on the heap?
   a. `struct point p;`
   b. `struct point* p;`
   c. `struct point p = new struct point;`
   d. `struct point* p = new struct point;`

7. Which snippet below correctly allocates and initializes a `struct scatterplot` (where “`std::rand() % 128`” generates a random number between 0 and 128)?

   a.
   ```
   struct scatterplot* plot = new struct scatterplot[10];
   for (int i = 0; i < 10; i++) {
     plot->points[i].x = std::rand() % 128;
     plot->points[i].y = std::rand() % 128;
   }
   plot->n = 10;
   ```

   b.
   ```
   struct scatterplot plot[10];
   for (int i = 0; i < 10; i++) {
     plot->points[i].x = std::rand() % 128;
     plot->points[i].y = std::rand() % 128;
   }
   plot->n = 10;
   ```

   c.
   ```
   struct scatterplot* plot = new struct scatterplot;
   plot->points = new struct point[10];
   for (int i = 0; i < 10; i++) {
     plot->points[i].x = std::rand() % 128;
     plot->points[i].y = std::rand() % 128;
   }
   plot->n = 10;
   ```

   d.
   ```
   struct scatterplot* plot = new struct point[10];
   for (int i = 0; i < 10; i++) {
     plot[i].x = std::rand() % 128;
     plot[i].y = std::rand() % 128;
   }
   plot->n = 10;
   ```

8. What is the correct way to free the memory allocated by the correct initialization of `plot` in the previous question?

   a. `delete[] plot->points;`

   b. `delete plot->points;`
      `delete plot;`

   c. `delete[] plot;`

   d. **`delete[] plot->points;`**
      `delete plot;`

For questions 9–12, assume you have a file named `course_data.txt` with the following contents:

```
cs162  141  87.4
cs391  60   86.4
cs492  106  93.6
cs480  90   86.1
cs493  84   94.2
cs290  108  95.7
cs261  278  84.3
```

9. After running the snippet below, what would be a valid way to make sure the file was successfully opened?

```
std::ifstream infile;
infile.open("course_data.txt");
```

   a. **`if (infile.fail()) { /* Handle error */ }`**
   b. `if (!infile.eof()) { /* Handle error */ }`
   c. `if (!infile.success()) { /* Handle error */ }`
   d. `if (!infile.open()) { /* Handle error */ }`

10. Assuming the file is correctly opened in the question above, what would be the correct way to read the data from the file, one line at a time, into the variables defined by the following declarations?

```
std::string course;
int n_students;
float avg_grade;
```

a. `while (!infile.fail()) {`
   `  infile.getline(course, n_students, avg_grade);`
   `}`

b. `while (!infile.fail()) {`
   `  infile >> course >> n_students >> avg_grade;`
   `}`

c. `while (!infile.fail()) {`
   `  infile.get(course, n_students, avg_grade);`
   `}`

d. `while (!infile.fail()) {`
   `  getline(infile, course, n_students, avg_grade);`
   `}`

11. If you wanted to add new data to the end of `course_data.txt`, after the data that was already in the file, how would you need open the file?

a. `std::ofstream outfile;`
   `outfile.open("course_data.txt");`

b. `std::ifstream outfile;`
   `outfile.open("course_data.txt");`

c. `std::ofstream outfile;`
   `outfile.open("course_data.txt", std::ofstream::app);`

d. `std::ifstream outfile;`
   `outfile.open("course_data.txt", std::ifstream::app);`

12. True or false? The following snippet will erase the current contents of `course_data.txt`.

```
std::ofstream file;
file.open("course_data.txt");
file.close();
```

    a. **True**
    b. False

13. True or false? Both `std::ifstream::fail()` and `std::ifstream::eof()` can be used to determine when you've reached the end of a file while reading from it.
    a. **True**
    b. False

14. What `#include` statement is needed to be able to use the class `std::ifstream`?
    a. `#include <string>`
    b. `#include <cstdlib>`
    c. `#include <iostream>`
    d. **`#include <fstream>`**

15. Assuming a correct definition of a `Point` class, what term best applies to the following method?

```
Point::Point(int x, int y);
```

    a. Default constructor
    b. **Parameterized constructor**
    c. Copy constructor
    d. Assignment operator

For questions 16–19, refer to the class definition below:

```
class Glass {
private:
  float fullness;
};
```

16. How would you initialize a half-full `Glass`, given the class definition above?
    a. `Glass g(0.5);`

    b. `Glass g;`
       `g.fullness = 0.5;`

    c. `Glass g = fullness(0.5);`

    d. It is not possible to initialize a half-full `Glass` given the definition above.

17. What would be the correct prototype for a copy constructor for the `Glass` class?
    a. `Glass();`
    b. `Glass(float fullness);`
    c. `Glass(const Glass& g);`
    d. It is not possible to write a copy constructor for the `Glass` class.

18. What would be the correct syntax for a constructor that uses an initialization list to initialize a new `Glass` object?
    a. `Glass::Glass(float fullness) : fullness(fullness) {}`
    b. `Glass::Glass(float fullness) -> fullness(fullness) {}`
    c. `Glass::Glass(float fullness) : fullness {}`
    d. `Glass::Glass(float fullness) { fullness = fullness; }`

19. What term best describes the constructor written in the previous question?
    a. Default constructor
    b. Parameterized constructor
    c. Copy constructor
    d. Assignment operator

20. What is “encapsulation” in the context of object-oriented programming?
    a. The practice of exactly capturing the characteristics of a real-world object in a class definition.
    b. The practice of using dynamically-allocated memory for class members instead of statically-allocated memory.
    c. The practice of limiting all definitions for a class, including definitions of member functions, to a single header file.
    d. The practice of hiding a class’s implementation details behind a well-defined public interface.

21. What does the `private` keyword do within a class definition?
    a. **Indicates that certain members and methods of the class cannot be accessed outside the class.**
    b. Indicates that the definitions of certain class methods will be made directly within the same `.hpp` file.
    c. Separates the declarations of a class's data members from the declarations of its class methods.
    d. Indicates that the class can only be used within another class and not directly within the `main()` function or any other function with global scope.

22. In the following class definition, what kind of method is `get_value()`?

```
class Integer {
private:
     int value;
public:
     int get_value();
};

int Integer::get_value() {
     return value;
}
```

    a. Mutator
    b. **Accessor**
    c. Constructor
    d. Operator

23. Assuming we have a correctly defined class `Point` and an existing object of that class called `p1`, what method is called by the following statement?

```
Point p2 = p1;
```

    a. Accessor
    b. Mutator
    c. **Copy constructor**
    d. Assignment operator

24. True or false? The following statement results in the default constructor for the `Point` class being called.

```
Point* line = new Point[2];
```

    a. **True**
    b. False

25. True or false? The compiler automatically creates an assignment operator for a class if one isn't explicitly defined.
    a. **True**
    b. False

26. Which of the following snippets results in a call to the assignment operator for the `Point` class?
    a. `Point p2 = p1;`
    b. `Point p2(p1);`
    c. `Point p2 = Point(p1.x, p1.y);`
    d. **`Point p2;`**
       `p2 = p1;`

27. What is the purpose of the `this` keyword?
    a. It provides a way to access the definition of a class from within the class's methods.
    b. It provides a way to call a class's constructor from another of that class's methods.
    c. **It provides a way to access the object on which a class method was called from within that class method.**
    d. It provides a way to access a specific class object from within the `main()` function or another function with global scope.

28. Assuming we have a correctly defined class `Point`, what is the correct way to dynamically allocate an array of points?
    a. `Point points[10];`
    b. `Point* points[10];`
    c. `Point points = new Point(10);`
    d. **`Point* points = new Point[10];`**

For questions 29–31, consider the following class declaration:

```
class Matrix {
private:
  double** data;
  int rows;
  int cols;
public:
  Matrix(int r, int c);
};
```

29. What would be the correct way to write the constructor declared above?

a.
```
Matrix::Matrix(int r, int c) {
    rows = r;
    cols = c;
    data = new double[rows][cols];
  }
```

b.
```
Matrix::Matrix(int r, int c) {
    rows = r;
    cols = c;
    data = new double**[rows][cols];
  }
```

c.
```
Matrix::Matrix(int r, int c) {
    rows = r;
    cols = c;
    data = new double(rows, cols);
  }
```

d.
```
Matrix::Matrix(int r, int c) {
    rows = r;
    cols = c;
    data = new double*[rows];
    for (int i = 0; i < rows; i++) {
      data[i] = new double[cols];
    }
  }
```

30. Assuming the constructor in the previous question is correctly defined, what would be the correct way to implement a destructor for the `Matrix` class?

a.
```
Matrix::~Matrix() {
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      delete data[i][j];
    }
    delete[] data[i];
  }
  delete[] data;
}
```

b.
```
Matrix::~Matrix() {
  for (int i = 0; i < rows; i++) {
    delete[] data[i];
  }
  delete[] data;
}
```

c.
```
Matrix::~Matrix() {
  delete[] data;
}
```

d.
```
Matrix::~Matrix() {
  delete[][] data;
}
```

31. Assuming the constructor and destructor in the previous questions are implemented correctly, what is the proper way to dynamically allocate a `Matrix` object and then free all of its memory?

a.
```
Matrix* m = new Matrix(3, 4);
delete[] m;
```

b.
```
Matrix* m = new Matrix(3, 4);
delete m;
```

c.
```
Matrix** m = new Matrix(3, 4);
delete[][] m;
```

d.
```
Matrix m = new Matrix(3, 4);
delete m;
```

32. Which of the reasons below is the best for choosing to implement an overloaded assignment operator.
    a. To be able to make shallow copies of objects.
    b. To be able to make copies of any dynamically allocated members of an object when it's being copied via assignment.
    c. To be able to pass objects of the class in question as arguments to functions.
    d. To be able to implement a copy constructor for the class.

33. What is the order of the steps of compilation?
    a. Preprocessing → Compilation → Assembly → Linking
    b. Preprocessing → Linking → Assembly → Compilation
    c. Preprocessing → Assembly → Linking → Compilation
    d. Preprocessing → Assembly → Compilation → Linking

34. Which step of the compilation process converts C++ code to assembly code?
    a. Preprocessing
    b. Linking
    c. Compilation
    d. Assembly

35. What is the purpose of a header guard, like the one below?

```
#ifndef __POINT_HPP
#define __POINT_HPP

#endif
```

    a. To indicate to the compiler that the current file is a header file.
    b. To make definitions within the header guard usable in other files.
    c. To prevent definitions within the header guard from being made multiple times, in case a header file is included multiple times.
    d. To show off your knowledge about how to write preprocessor directives.