

EECS 168 Final Exam Review Session Solution

True/False and Multiple Choice

1. What kind of return statement is valid for a void function?

- a. `return 0;`
- b. `return "";`
- c. `return;`
- d. No return statement is valid.

A void doesn't *have to* have a "return" statement. But we can add one, as long as we don't add anything between "return" and the semicolon.

2. (T/F) All functions must have parameters.

No function is required to have parameters. Functions can have as many parameters as you want them to have (0 or more parameters). The one exception is destructors, which cannot have any parameters at all.

3. The first element of an array is at index...

- a. -1
- b. 0
- c. 1
- d. 2

Array indexes always start at 0.

4. The last index of an array of size 5 is...

- a. 5
- b. -1
- c. 0
- d. 4

Array indexes always end at size - 1.

5. (T/F) Each parameter of a function requires a type.

Function parameters are regular variables. Variables in C++ require a type. When you *pass in* a variable to a function you don't have to explicitly state the type though (because you should have declared the variable and its type earlier in code).

6. For loops are preferred when...

- a. The number of iterations is defined by a number or variable
- b. We want the loop to continue until a specific event happens
- c. The loop should be guaranteed to run at least once
- d. Never

For loops start at a number and end at a number, so we usually use them for things like iterating through an array. Option b is related to while loops and option c is related to "do while" loops. In practice, we can almost always use loops interchangeably. This is just a "rule of thumb", not an actual strict programming rule.

7. (T/F) Passing an array to a function makes a copy of the entire array.

When you "pass an array" you're actually passing a *pointer to an array*. What this does is it makes the pointer that is the parameter point to whatever the pointer that was passed in was pointing to.

8. The size of a stack-allocated array must be known at...
- a. The beginning of the program
 - b. Compile time (by the time we're compiling the program)
 - c. Runtime (by the time we're running the program)
 - d. It doesn't matter

The size of a stack-allocated must be known at compile time (for example, `int arr[4]` or `int arr[size]` where "size" has value 5 at compile time). However, for a heap-allocated array we can do something like `int* arr = new int[size]`, where size is not known until runtime (because the user inputs the size). The size of a heap-allocated array can also be set at compile time, but it is not necessary. For stack-allocated arrays, it is required.

9. What is the correct syntax for creating a heap-allocated array of size "size"?
- a. `int[] arr = new int* [size]`
 - b. `int[] * arr = new int [size]`
 - c. `int * arr = int [size]`
 - d. `int * arr = new int [size]`

Option d creates a heap-allocated array of integers. A, b, and c are not valid syntax.

10. (T/F) Passing a double (by value) to a function makes a copy of that double. Passing any type of variable value makes a copy. If you pass it by reference, you make the variable that is a parameter *reference* the variable that was passed in. So any changes to the function's parameter variable also changes the variable that was passed in.

11. What is the correct syntax for creating a heap-allocated array (of size "size") of pointers to "Triangle" objects?
- a. `Triangle* arr = new Triangle*[size];`
 - b. `Triangle* arr = new Triangle[size];`
 - c. `Triangle** arr = new Triangle*[size];`
 - d. `Triangle* arr = new Triangle();`

Option a is not valid syntax, 'b' is a heap-allocated array of Triangle objects, 'c' is a heap-allocated array of pointers, and 'd' is not an array but a heap-allocated object. C is the only good option. However, the array of pointers could have each pointer point to an array of Triangle objects rather than pointing to Triangle objects directly. Judging from this single line of code we can't know. There is an example of the distinction between these two in the Memory Allocation section.

12. When would you want to create a heap-allocated array of pointers to objects?
- a. Never.
 - b. When your class doesn't have a constructor with 0 parameters.
 - c. When your class has a constructor with at least one parameter.
 - d. When your class doesn't have a destructor.

Some people were asking why would we ever want to use an array of pointers to objects (such as option c in question 11) vs just an array of objects (option b in question 11). When you don't have a default (parameterless) constructor and you want to create an array, you have to use an array of pointers to objects rather than an array of objects.

13. Which of the following tells us we're allocating something in the heap rather than the stack?

- a. We are using pointers
- b. We are creating an array
- c. We are using the word "new"
- d. We are creating a 2D array

Using "new" is the only good way to know when we're allocating something in the heap rather than the stack.

14. How do we get rid of stack-allocated arrays?

- a. We don't have to, the computer does it by itself
- b. Use the keyword delete[]
- c. Empty the array
- d. Set it to nullptr

Any variable that is stack-allocated gets "deleted" by the computer once we are done with the function that holds that variable. We don't need to manually "delete" anything that is stack-allocated.

15. What is the best way to avoid memory leaks?

- a. Not using pointers at all
- b. Using "delete" on everything created using "new"
- c. Setting our array pointer(s) to nullptr
- d. Using valgrind

The best way is to use "delete" for every "new". Using valgrind is good for determining whether we have leaks or not but it is best to avoid leaks entirely by deleting everything created using "new".

Memory Management

Finish the definition for the main function such that there are no memory leaks. Assume Car is class with a single, default constructor (no parameters) and getters and setters for “year” and “make”. The Car class is defined in problem 4 of Code Writing. **Suggestion: Do Memory Allocation first (next page).**

```
int main()
{
    int size = 11;
    int nums[4];
    Car myCar; //Stack-allocated car object
    Car * myCar2 = new Car(); //Heap-allocated Car object

    Car arrayOfCars1[size]; //Stack-allocated array of Car objects
    Car * arrayOfCars2 = new Car[size]; //Heap-allocated array of Car objects

    Car ** arrayOfCarPointers = new Car*[size]; //Array of pointers
    for (int i = 0; i < size; i++)
    {
        arrayOfCarPointers[i] = new Car(); //Each pointer points to a Car (so now
        arrayOfCarPointers is an array of pointers to objects)
    }

    Car ** arrayOfCars2D = new Car*[size]; //Array of pointers
    for (int i = 0; i < size; i++) {
        arrayOfCars2D[i] = new Car[size]; //Each pointer points to an array of Cars (since
        each pointer points to an array, arrayOfCars2D is considered a 2D array)
    }

    bool temp = false;
    double myHeapAllocatedvariable = 1.5;

    //Your code here

    delete myCar2;
    delete[] arrayOfCars2;

    for (int i = 0; i < size; i++)
    {
        delete arrayOfCarPointers[i]; //Here we're deleting objects, so we don't need the
        [] after delete
    }
    delete[] arrayOfCarPointers; //Deleting an array of pointers to car objects

    for (int i = 0; i < size; i++)
    {
        delete[] arrayOfCars2D[i]; //Here we're deleting arrays, so we Do need the []
        after delete
    }
    delete[] arrayOfCars2D; //Deleting an array of pointers to arrays of car objects

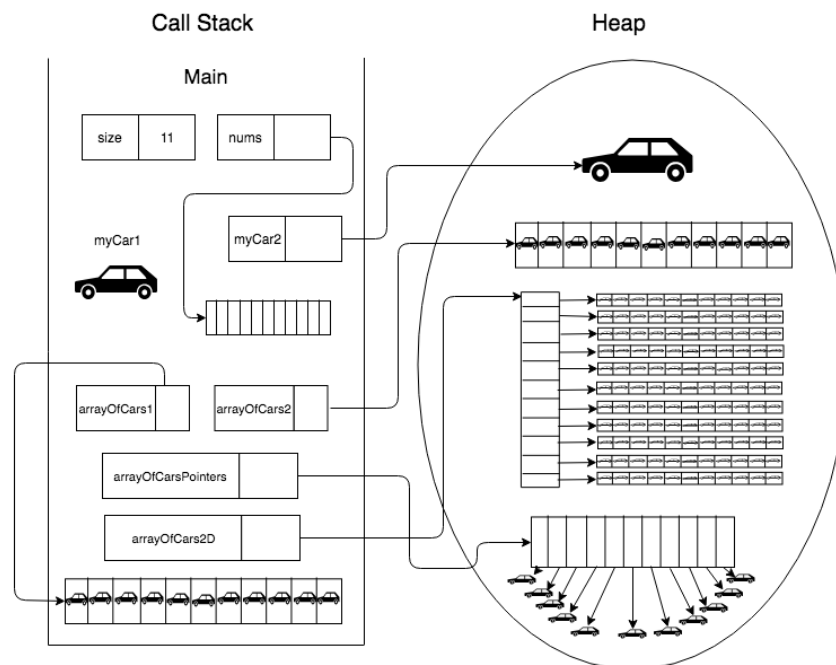
    return 0;
}
```

Memory Allocation

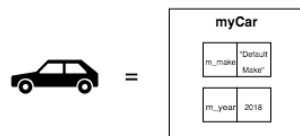
Fill in the table regarding the variables in the Memory Management question (previous page).

Variable	Type	Stack or Heap?
size	integer	stack
nums	array of integers	stack
nums[1]	integer	stack
myCar	Car object	stack
arrayOfCars1	array of Cars	stack
arrayOfCars1[0]	Car object	stack
myCar2	Car object	heap
arrayOfCars2	array of Cars	heap
arrayOfCars2[1]	Car object	heap
arrayOfCarPointers	array of pointers	heap
arrayOfCarPointers[3]	pointer to a Car object	heap
arrayOfCars2D	array of pointers	heap
arrayOfCars2D[2]	pointer to an array of Cars	heap
temp	boolean	stack
myHeapAllocatedVariable	double	stack

***Note:** All the variables declared with a “*” are *actually* stack-allocated pointers that point to something in the stack or heap. However, for “type” I’m filling in what they’re pointing to and for “stack/heap” I’m deciding whether **what they point to** is in the stack or the heap. This is most likely what you would want to answer in the exam if a question like this comes up.



Notes:



If you have to draw an object on the test don't draw the actual thing it's representing, draw a box with its member variables inside. I just did it here because I had a ton of Car objects.

Code Tracing

Code that runs	Output to terminal
<pre>//Circle.h class Circle { public: Circle(double radius) { //Constructor takes in a radius and sets the member variable</pre>	<pre>m, 3.1416 4</pre>

<pre>ptr2->setRadius(z); //Attempts to set the radius to -2. However, if you see the</pre>	
---	--

Code that runs	Output to terminal
<pre>char* reverse(char* arr, int&size) { size = 7; //Since size is passed by reference, the "size" in main is affected by anything we do with the int size = 7; //This was an error in the original question, it should have been 7 not 0.</pre> <div data-bbox="1062 1467 1283 1778"> <pre> 0 1 2 3 4 5 6 0 j 1 j a 2 j a y 3 j a y h 4 j a y h a 5 j a y h a w 6 j a y h a w k</pre> </div>	jayhawk

Code Writing

Note: I added significantly more details to these questions since the original ones were a little vague.

1. Write a function that takes in a 2D array of integers and returns an array of the even integers in the 2D array. The function should take in the 2D array, the number of rows, the number of columns, and a variable for the size of the 1D array. The size of the 1D array can be initialized to 0 in main, but the function should modify the “size” variable in main to be equal to the amount of even integers.
2. Write a class StringClass that holds a heap-allocated array of characters.
 - a. The function must have the following methods:
 - **StringClass(int length);** //Constructor that initializes the size and initializes the array
 - **~StringClass();**
 - **char getCharacter(int index) const;**
 - **void setCharacter(int index, char c);**
 - **int getLength() const;**
 - b. Overload the assignment operator (write a function to use the assignment operator (the “=” operator) with StringClass objects and strings).

bool operator==(std::string rhs) const;

Example:

```
StringClass myStrObject(8); //Pass in the size of the string
myStrObject = "a string"; //In your function you can assume you'll get a string of the right
size (in this case a string of length 8).
```

Also, overload the comparison operator for equality (the “==” operator) so that we can verify whether two strings a StringClass and a string are equal.

bool operator=(std::string rhs) const;

Example:

```
StringClass myStrObject(5);
If (myStrObject == "Hello")
{
    cout << "Equal!\n";
}
```

- c. Write a copy constructor for a StringClass object. The copy constructor should copy the length, create a new array, and copy character by character.

StringClass(const StringClass& strClassObject);

3. Write a class SocialMedia that holds an array of Users (User is a class). Obtain the number of users and each user’s email, password, name, and age from a file called “data.txt”.

Data.txt:

5

myEmail@socialmedia.com password Spongebob 20

emailAddresss@socialmedia.com 12345 Patrick 15

badEmail@socialmedia.com insecurePassword Squidward 40

UserEmail@socialmedia.com 54321 Gary 23

runningOutOfEmailNames@socialmedia.com abcde Plankton 32

The User class must contain a constructor with parameters for email, password, name, and age as well as getters and setters.

In the SocialMedia class make a “login” function that takes in an email and a password. The function should first verify if a user with that email exists. If such a user exists, the function should verify if that user’s password matches the password passed in to the login function. Print a message telling the user whether the login was successful or not.

In the SocialMedia class also make a “search” function that takes in a char and prints all the Users whose name starts with that character and prints their name, age, and email address.

In main.cpp create a SocialMedia object and give the user options (login, search, or quit) until the user decides to quit.

4. Write code to print every Car object created in the program in the “Memory Allocation” question (including the Car objects that are inside arrays). Below are the Car.h and Car.cpp files.

```
//Car.h
#ifndef CAR_H
#define CAR_H

#include <string>

Class Car {
    private:
        string m_make;
        string m_model;
        int year;
    public:
        Car();
        void setMake(string make);
        string getMake() const;
        void setYear(int year);
        int getYear() const;
};

#endif
```

As stated in the “Memory Allocation” question, the Car constructor initializes make and year to “default” values. You won’t necessarily use all the functions defined in this class. The goal of this problem is to use whatever functions you need to use to do all the printing.

```
//Car.cpp
#include "Car.h"
Car::Car()
{
    m_make = "Default Make";
    m_year = 2018;
}
void Car::setMake(string make)
{
    m_make = make;
}
string Car::getMake() const
{
    return m_make;
}
void Car::setYear(int year)
{
    m_year = year;
}
int Car::getYear() const
{
    return m_year;
}
```

Code Writing Solutions

The solutions are posted in the following link:

<https://github.com/brianquiroz216/EECS-168-SI-Final-Review-Session-Solution>

For some questions that didn't require a main.cpp I created one just so the program had some context in which it could be ran and tested. In the actual exam don't do anything extra unless they ask you to because you'll run out of time if you try to write more code than you have to.