|  |  |
| --- | --- |
| Name |  |
| KUID |  |

# EECS 168 2020 Spring: Final Exam

Possible points: 150pts

Due Date: Wednesday May 13th 11:59:59am (Noon)

Submit via email to your Lab TA

Place an X in the box that applies to you:

|  |  |
| --- | --- |
| I am in the MWF section (officially) |  |
| I am in the TR section (officially) |  |
| I am an EECS 169 student (leave blank if not in 169) |  |

# Rules (Please Read These!)

* DO NOT alter the formatting of the test. Any changes to formatting could result in grading errors
* Only mark your answers within designated answer boxes
* Read and sign below
* Put your name and KUID on every page

Hi EECS 168 or 169 students. I know you just got this take home exam off the internet, but can you promise me, or more importantly, promise yourself that you'll take this exam without any unauthorized aid?

* Unauthorized aid: google searches, the materials or help of other students, past exams, help from the undergrad staff or GTAs, **any compilers** (e.g. don't just put the code tracing problems in a compiler and run them)
* For the code tracing problem, you are to only read them. They are valid algorithms, though they might not be sensible ones.
* Any strange characters you see are there on purpose.

But since we're in such a crazy world and our class didn't go as planned, I am fine with you using the following authorized aid:

* Authorized resources: your notes, your labs (the code, not the compiler), the online lectures I made for you, your amazing brain

I'd like this exam to still be an assessment of your skill and understanding.

If you agree to this, then please type your name in the box below. If you can't agree to it, then I'm afraid I won't grade your exam. Afterall, I'm not here to test how well you can google things. That's what your job will do.

|  |  |
| --- | --- |
| Type your name |  |

# [25pts] True/False

**Place an 'X' in the appropriate column. For example, if you think the answer is true, then put an 'X' in the true column.**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Question** | **True** | **False** |
| 1 | The == operator can only compare the type *char* |  |  |
| 2 | Public members of a class are directly accessible in main() |  |  |
| 3 | A member method has access to all public AND private members |  |  |
| 4 | Every if statement must have an else statement |  |  |
| 5 | The keyword *destroy* deallocates heap allocated memory |  |  |
| 6 | While loops repeat forever regardless of looping condition |  |  |
| 7 | The / operator only works with doubles; any other type gives a compiler error |  |  |
| 8 | Two variables in two different functions can have the same name |  |  |
| 9 | Passing parameters by value is identical to passing by reference |  |  |
| 10 | Pointers (e.g. int\*) and references (e.g. int&) are identical in every way |  |  |
| 11 | Only char pointers (e.g. char\*) can be set to nullptr |  |  |
| 12 | Two pointers can point to the same object |  |  |
| 13 | A Circle\* and a char\* can both point to an array of doubles |  |  |
| 14 | Constructors are called exactly one time per object’s life |  |  |
| 15 | Destructors have the option of taking parameters |  |  |
| 16 | All uninitialized variables automatically receive a default value of nullptr. |  |  |
| 17 | Strings are surrounded by single quotes |  |  |
| 18 | Main's *argc* contains a count of the command line arguments |  |  |

## True/False (continued)

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Question** | **True** | **False** |
| 19 | The destructor for a class named Pizza MUST be named ~Pizza() |  |  |
| 20 | All elements in a single array must be the same type |  |  |
| 21 | Memory leaks are caused by stack allocated memory not being deallocated |  |  |
| 22 | Do-while loops check their looping condition before running the first time |  |  |
| 23 | Every function must take at least one parameter |  |  |
| 24 | Every function must return a value |  |  |
| 25 | The keyword *delete* is used to deallocate stack allocated arrays |  |  |

# [32pts] Conceptual

**Provide your answers in the given boxes.**

1. [3pts] When is a stack allocated variable that was declared in a function named *foo* deallocated?

|  |
| --- |
|  |

1. [3pts] When is a heap allocated array that was created in a function named *foo* deallocated?

|  |
| --- |
|  |

1. [3pts] How many times, in a single object’s life, is a constructor called?

|  |
| --- |
|  |

1. [3pts] How many times, in a single object’s life, is a destructor called?

|  |
| --- |
|  |

1. [3pts] If an object is passed by value to a function, what special constructor is called?

|  |
| --- |
|  |

1. [3pts] If a class has exactly one constructor and that constructor requires a parameter, can you make an array of objects of that type, yes or no?

|  |
| --- |
|  |

1. [3pts] If we do not create our own copy constructor, does the default copy constructor create a deep or shallow copy?

|  |
| --- |
|  |

1. [3pts] Why will the equation *area = (1/2)\*base\*height* always result in zero even if the base and height are greater than zero?

|  |
| --- |
|  |

1. [8pts, 2pts each] Assume a Circle class has a private member variable named m\_radius. Assume "Circle.h" is included as needed. Answer the following questions below indicating whether or not the attempted access is legal or illegal by placing an 'X' in the appropriate column.

|  |  |  |
| --- | --- | --- |
| **Code** | **Legal** | **Illegal** |
| int main()  {  Circle c1;  c1.m\_radius = 5.5.;  //main continues... |  |  |
| int main()  {  Circle\* c1 = new Circle();  c1->m\_radius = 5.5.;  //main continues... |  |  |
| void changer(Circle& c)  {  c.m\_radius = 5.5;  } |  |  |
| void changer(Circle c)  {  c.m\_radius = 5.5;  } |  |  |

# [14pts] Code Snippets

**For these questions you'll only need to write enough code to answer a given question.**

[3pts] Write only the needed code to create a heap allocated array of 1000 doubles. You do not have to initialize them.

|  |
| --- |
|  |

[3pts] Write the needed code to deallocate your array of doubles. (Variable names must be consistent with the previous question!)

|  |
| --- |
|  |

[3pts] Assume a Circle class exists and is already defined. Write the needed code to create a heap allocated Circle and give it a radius of 2.5. For this question, assume the constructor takes no parameters. Instead, Circles have a setRadius that takes a double.

|  |
| --- |
|  |

[5pts] Assume there is a class called Car. This class only has one constructor which requires an int. Below, write only the needed code to create 1000 Car objects. You may pass in a value of zero to all Car constructors.

|  |
| --- |
|  |

# [33pts] Code Tracing

Remember, DO NOT just put this code in a compiler!

|  |  |
| --- | --- |
| Code [3pts] | Output to terminal |
| int n = 5;  //Remember bools print as 1 and 0  //true => 1, false => 0  std::cout <<(n == 5) << std::endl;  std::cout <<(n > 3) << std::endl;  std::cout <<(n < 4) << std::endl;  std::cout <<(n >= 3) << std::endl;  std::cout <<(n <= 5) << std::endl;  for(int i=(n-(2\*n)); i<0; i=i+1)  {  n=n+i;  }  std::cout << n << std::endl; |  |

|  |  |
| --- | --- |
| Code [4pts] | Output |
| bool q = false;  bool z = true;  bool r = (q || z) && (z != false);  z = !(r || !(r != !(r)));  q = ((z==true) && (`a`!=`A`)) || (`a` == `A`);  //Remember bools print as 1 and 0  //true => 1, false => 0  std::cout << q << `\n`;  std::cout << z << `\n`;  std::cout << (!r) << `\n`;  std::cout << r << `\n`; |  |

|  |  |
| --- | --- |
| Code [5pts] | Output |
| int n = 4;  for(int i=0; i<n; i=i+1)  {  for(int j=0; j<n; j=j+1)  {  if(j==i)  {  std::cout << `M`;  }  else if(i==0 || i==n-1)  {  std::cout << `U`;  }  else  {  std::cout << `K`;  }  }  std::cout << `\n`;  } |  |

|  |  |
| --- | --- |
| Code [3pts] | Output to terminal |
| int amount = 3;  do  {  std::cout << amount;  amount = amount + 1;  }while (amount < 5); |  |

|  |  |
| --- | --- |
| Code [5pts] | Output to terminal |
| int score = 95;  if( score > 90 )  {  std::cout << `A`;  }  if( score > 80 )  {  std::cout << `B`;  }  if( score > 70 )  {  std::cout << `C`;  }  if( score < 70 )  {  std::cout << `D`;  }  if( score < 60 )  {  std::cout << `F`;  } |  |

|  |  |
| --- | --- |
| Code [3pts] | Output to terminal |
| int choice = 5;  if( choice != 5 && choice == 10 )  {  std::cout << “Correct\n“;  }  else if( choice != 5 || choice == 10 )  {  std::cout << “Wrong\n“;  }  else if( choice == 50)  {  std::cout << “Failure\n“;  }  else  {  std::cout << “Good job!\n“;  } |  |

|  |  |
| --- | --- |
| Code [3pts] | Output to terminal |
| //reminder, this is legal  //and written as intended  void doAllTheWork()  {  int x = 2;  int y = 3;  int z = 1;  if(z <= 0)  {  return;  }  std::cout << x << `\n`;  std::cout << y << `\n`;  std::cout << z << `\n`;  x = y + 10;  y = x+x;  z = y + x + z + 3;  z = z + 1;  std::cout << x << `\n`;  std::cout << y << `\n`;  std::cout << z << `\n`;  }  int main()  {  doAllTheWork();  std::cout << “Goodbye.\n“;  return(0);  } |  |

|  |  |
| --- | --- |
| Code [3pts] | Output to terminal |
| //code written as intended  void matrix(int n)  {  n = n-1;  std::cout << “Hahahah!\n“;  }  int main()  {  int x = 10;    std::cout << x << `\n`;  matrix(x);  std::cout << x << `\n`;  } |  |

|  |  |
| --- | --- |
| Code [4pts] | Output to terminal |
| int\* nums = new int[3];  int\* ptr2 = nullptr;  ptr2 = nums;  ptr2[2] = 99;  ptr2[1] = 11;  ptr2[0] = 44;  for(int i=0; i<3; i++)  {  std::cout << nums[i] << `\n`;  nums[i] = nums[i] + 1;  }  std::cout << ptr2[0] << `\n`; |  |

# [26pts] Cool String: Part 1 - Memory

Below is the header file from our CoolString class. You may assume the methods listed here are working properly.

|  |
| --- |
| #ifndef COOL\_STRING\_H  #define COOL\_STRING\_H  #include <iostream>  class CoolString  {  private:  char\* m\_array;  int m\_size;    public:  //creates array of given size, stores size  CoolString(int size);  //makes deep copy  CoolString(const CoolString& original);  //delete array  ~CoolString();  //returns the size of the array  int size() const;  //return the character at an index  char getAt(int index) const;  //stores character at an index  bool setAt(int index, char entry);  // returns true is same size and all  //value are in the same order  bool operator==(const CoolString& rhs) const;    //returns true if not the same (either differing size or  //values)  bool operator!=(const CoolString& rhs) const;  };  #endif |

Using the CoolString header file for reference, carefully trace the following code then answer the questions below. You will trace the code up to the FREEZE POINT at which you may assume the program pauses (functions are paused in the middle of running and everything stays where it is in memory).

|  |
| --- |
| Code |
| void func(CoolString cs)  {  cs.setAt(0, 'P');  //FREEZE POINT, func has NOT returned yet!    return;  }  int main()  {  std::string word = "dogs";  CoolString myCS( word.length() );  for(int i=0; i< myCS.size(); i++)  {  myCS.setAt(i, word.at(i) );  }  myCS.setAt(0, 'L');  func(myCS);    return(0);  } |

1. [3pts] How many CoolString objects (not including std::string) are allocated?

|  |
| --- |
|  |

1. [3pts] How many character arrays are allocated?

|  |
| --- |
|  |

1. [3pts] How many CoolString objects are on the call stack?

|  |
| --- |
|  |

1. [3pts] How many CoolString objects are on the heap?

|  |
| --- |
|  |

1. [3pts] How many character arrays are on the calls stack?

|  |
| --- |
|  |

1. [3pts] How many character arrays are on the heap?

|  |
| --- |
|  |

1. [4pts] For the instance named myCS, list the values within its array (e.g. if its array contained the characters C, A, T, S you would write those characters below).

|  |
| --- |
|  |

1. [4pts] For the instance named cs, list the values within its array (e.g. if its array contained the characters C, A, T, S you would write those characters below).

|  |
| --- |
|  |

# [20pts] Cool String: Part 2 - Implementation

1. [10pts] Assume you are adding a new method to the CoolString class called *reverse()*. Assume its signature is the following in the header file:

|  |
| --- |
| #ifndef COOL\_STRING\_H  #define COOL\_STRING\_H  #include <iostream>  class CoolString  {  //ASSUME ALL OTHER METHODS AND VARIABLES ARE STILL HERE BUT  //OMITTED FOR SPACE SAKE  void reverse();  };  #endif |

Reverse will reverse the order of values in the characters array (e.g. if the CoolString contained C,A,T,S then after a call to reverse it would contain S,T,A,C. Reverse does NOT print anything. You may NOT alter the parameter list. Write the implementation below:

|  |
| --- |
| //CoolString.cpp  void CoolString::reverse()  {  //Your code below |

1. [10pts] Assume you are in main.cpp, write a function definition for a function named *userFill* that takes an CoolString by reference (not const, just plain reference). It then lets the user fill the CoolString's array with values.

|  |
| --- |
| //main.cpp  void userFill(CoolString& cs)  {  //your definition below |

This should be page 22.

If it is not, place make sure you didn't change the size of any of the answer fields.

The exam is now over.

Congratulations!

Email your exam to your TA by 11:59:59am (noon) Wednesday May 13th