**CPSC 1020 FALL 2018**

**FINAL EXAM**

This is a Closed Book exam. Please keep your notes and your computers closed. You have 2 hours to complete the Exam. There is a total of 70 points. Although the points total 70 your grade will reflect 100%. EX. A total score of 70 will result in a grade of 100.

**1. (9 points) Matching**

1. Base Class 15. Binary Search
2. Derived Class 16. Recursion Function
3. Dynamically Allocated Memory 17. Polymorphic
4. Pass by Value 18. Stack
5. Pass by Reference 19. Queue
6. Inheritance 20. Dynamic Stacks
7. Override of Function 21. Static Stacks
8. Virtual Function 22. Composition
9. Size 23. Operator Overloading
10. STL 24. Static Member Variable
11. Linear Search 25. Setw(n)
12. Fixed 26. Showpoint
13. Setprecision(n) 27. Left or Right
14. Template

There are more choices than there are definitions.

\_\_\_\_ A function parameter that points to another variable and any changed made to the parameter in the function is made to the variable it points to.

\_\_\_\_ Allows us to set the number of significant digits.

\_\_\_\_ describes a general class that other classes can inherit data members and functions, also known as a parent class

\_\_\_\_ When an argument is passed in a parameter, only a copy of the argument’s value is passed. Changes to the parameter do not affect the original argument.

\_\_\_\_ Initialized outside of the class declaration and is shared by all instances of the class.

\_\_\_\_ Used by C++ to redefine how standard operators work when used with class objects.

\_\_\_\_ Sets the size of a print field

\_\_\_\_ Code that produces different behavior when executing code with different types.

\_\_\_\_ Returns the number of elements in a vector.

\_\_\_\_ causes the output to go to one side or another.

\_\_\_\_ Forces a floating point number to display in a particular point format.

\_\_\_\_ This is when a derived class provides functionality of a function with the same name, same parameters, and same return value of a function in the base class.

\_\_\_\_ describes a class that inherits functionality and data members from another class

\_\_\_\_ Allows us to define a class in terms of another class. Creates an “is-a” relationship.

\_\_\_\_ Allows the most specific version of a member function in an inheritance hierarchy to be selected for execution. This is what makes polymorphism possible.

\_\_\_\_ Causes a decimal point and trailing zeros to be displayed for floating-point numbers, even if there is no fractional part.

\_\_\_\_ this is when a program, while running, asks the computer to set aside a chunk of unused memory large enough to hold a variable of a specific data type.

\_\_\_\_ Occurs when an object of one class owns an object of another class (has-a) relationship.

1. **This is a 3 part question: (9 points total) Memory Allocation and Vectors**

Consider the following struct:

typedef Pixel{

unsigned char red;

unsigned char green;

unsigned char blue;

}

**Part A.**

In **C,** write the code to **dynamically allocate memory** for a 2D array of **Pixels** of size **width** and **height.** You can assume width and height have been declared and initialized. **3 points**

**Part B.**

In **C++,** write the code to **dynamically allocate memory** for a 2D array of **Pixels** of size **width** and **height.** You can assume width and height have been declared and initialized. **3 points**

**Part C:**

Create a **2D Vector** of Pixels. Use **resize** to set the size of the vector to width and height. **3 points**

1. **(2 points – each) In-Line Function**

Consider the following code:

Class A{

public:

//This function returns a + b

int add(int a, int b);

};

In class we discussed two examples of how to make the function “add” an inline function.

Provide the code --for both examples-- to make the function “add” inline.

**Example 1:**

**Example 2:**

1. **(1 point) Pointer**

What is the output of the following code:

#include <stdio.h>

Output:

int f(int , int \*, int \*\*);

int main()

{

int c, \*b, \*\*a;

c = 6;

b = &c;

a = &b;

printf("%d \n", f(c, b, a));

return 0;

}

int f(int x, int \*py, int \*\*ppz)

{

int y, z;

\*\*ppz += 3;

z = \*\*ppz;

\*py += 2;

y = \*py;

x += 1;

return x + y + z;

}

**5. (3 points) Basic C to C++**

Rewrite the following C program in C++:

#include <stdio.h>

int main(int argc, char\* argv[])

{

FILE\* input = fopen(argv[1], "r" );

int num;

while(fscanf(input, "%d", &num)== 1){

printf("%d\n", num);

}

fclose(input);

return 0;

}

Code goes here:

**6. (1 points) Initialization List**

On attachment 1 you will find the class declaration for Address, Date, and Person. Complete the Person default constructor using the **member initialization list technique**.

Person::Person()

**7. (2 points) Linked List**

With respect to a linked list. How can you tell if the list is empty?

How can we tell if a node is the last node in a list?

**8. (3 points) Function Pointers**

Given the following function prototypes, complete the code listed in main.

void printNum(int);

int adder(int, int);

int main(int argc, char\* argv[])

{

      1.Declare a function pointer called "fp" that can be pointed to printNum.

      2.Initialize the function pointer from 1 above to point to printNum.

      3.Call printNum using the function pointer passing the value of 54.

      return 0;

}

**9. (1 points) Abstract Classes**

The following class is not considered abstract.

Class MyClass

{

public:

virtual void myFunction();

}

Change the above class so that MyClass is an abstract class.

**10. (1 points) True / False**

\_\_\_\_\_\_\_\_\_ A reference variable can be initialized at any time, but a pointer must be initialized when created.

\_\_\_\_\_\_\_\_\_ One advantage a vector has over a linked list is it is easier to insert an element in a vector than it is to insert a node in a linked list.

**11. (1 point) Pointers**

What is the output of the following program?

#include <stdio.h>

Output:

void f(int\*, int)

int main()

{

int i = 5, j = 10;

f(&i, j);

printf(“%d\n”, i+j);

return 0;

}

void f(int\* p, int m)

{

m = m + 5;

\*p = \*p + m;

}

**12. (1 points) Multiple choice**

With respect to inheritance:

**Part 1:**

When both a base class and a derived class have destructors. If an instance of the derived class goes out of scope the base class's destructor is called \_\_\_\_\_\_\_\_\_\_.

1. First
2. Last
3. Never

**Part 2:**

When both a base class and a derived class have constructors and an instance of the derived class is created the base class's constructor is called \_\_\_\_\_\_\_\_.

1. First
2. Last
3. Never

**13. (2 points) Copy Constructor**

Which of the following describe when a copy constructor is called? There may be more than one answer.

1. When an object is initialized with another objects’ data.
2. When an object is passed by reference to a function.
3. When an object is passed by value.
4. When an object is returned by value.
5. When an object is returned from a function by reference.
6. All of the above.
7. None of the above

**14. (1 point) Pointers**

Consider the following:

int first = 10;

int second = 15;

int third = 20;

int \*sptr = &first;

int \*sptr2 = &second;

int \*\*dptr =  &sptr;

         \*dptr = sptr2;

         After the following statement:

         \*dptr = &third;

         What will the following print statement print?

         printf(“%d”, \*\*dptr);

Output

**15. (1 point) Freeing Memory**

Assume we have dynamically allocated the memory for a pointer, called pix, to 10 objects of type Pixel.  I am now done with the memory and need to return the memory to the operating system.  In C++ which of the following is the appropriate way to give the memory back to the operating system?

1. free(pix);
2. pix.delete();
3. delete Pixel[ ];
4. delete [ ] pix;

**16. (1 point) Vector**

Assume I create an empty vector of type integer and call it tiger.  Now I want to add an element with a value of 15 to this vector.  Which of the following is the appropriate way to add the element.

1. tiger[0].push\_back(15);
2. tiger.push\_back(15);
3. tiger.at(0).add(15);
4. tiger(15);

**17. (15 points) Classes**

**This is a multi-part question. First, fill in the code as instructed in each function below. Second, according to the code in main give the output of the code in the box provided.**

**This class creates an array of doubles and sets each element in the array to a value passed into the constructor**

#include <iostream>

using namespace std;

class NumberArray

{

private:

double \*aPtr;

int arraySize;

public:

NumberArray operator=(const NumberArray &right); //overloaded operator function

NumberArray(const NumberArray &);

NumberArray(int size, double value);

~NumberArray();

void print();

void setValue(double value);

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Destructor

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NumberArray::~NumberArray()

{ cout << "Destructing" << endl;

**//WRITE THE CODE FOR THE DESTRUCTOR**

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//The overloaded operator function for assignment

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NumberArray NumberArray::operator=(const NumberArray &right)

{

cout<< "operator =" << endl;

**//YOU CAN ASSUME THIS CODE IS ALREADY WRITTEN**

return \*this;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Copy Constructor

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NumberArray::NumberArray(const NumberArray &obj)

{ cout << "copy" << endl;

**//WRITE THE CODE FOR THE COPY CONSTRUCTOR**

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Constructor

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NumberArray::NumberArray(int size, double value)

{ cout << "regular" << endl;

**//WRITE THE CODE FOR THE REGULAR CONSTRUCTOR**

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Sets the value stored in all entries of the array

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void NumberArray::setValue(double value)

{

for(int index = 0; index < arraySize; index++)

aPtr[index] = value;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Print out all entries in the array

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void NumberArray::print()

{

for(int index = 0; index < arraySize; index++)

cout << aPtr[index] << " ";

OUTPUT:

cout << endl;

}

**What is the output of the main?**

int main()

{

NumberArray first(3, 10.5);

first.print();

NumberArray second(5, 20.5);

second.print();

NumberArray third(7, 30.5);

third.print();

first = second = third;

return 0;

}

**18. (2 points) Manipulators**

Given the following program, what is the output.

OUTPUT:

#include <iostream>       
#include <iomanip>        
using namespace std;  
  
int main () {  
  double f =3.14159;  
  cout << setprecision(5) << f << endl;  
  cout << setprecision(9) << f << endl;  
  cout << fixed;  
  cout << setprecision(5) << f << endl;  
  cout << setprecision(9) << f << endl;  
  return 0;  
}

**19. (12 points) Member initialization and operator overloading**

**Consider the class listed on Attachment 2.**

**Implement the functions indicated below:**

#include "Point.h"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Complete the constructors **using Member Initization List Technique**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Point::Point ()

Point::Point (int x, int y)

Point::Point (const Point& p)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*You do not need to implement the destructor, getters, nor the setters

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Point::~Point () {}

int Point::get\_x() const { return x; }

int Point::get\_y() const { return y; }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Setters

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Point::set\_x(int x)

{

this->x = x;

}

void Point::set\_y(int y)

{

this->y = y;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\***Implement the +, = operators**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// + operator

Point Point::operator+(const Point& rhs) const

{

}

// - operator

Point Point::operator-(const Point& rhs) const

{

//you can assume this is written

}

// = operator

Point& Point::operator=(const Point& rhs)

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\***Implement the << operator such that it will print the points in the**

**\*following format [3,4]**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// ostream

std::ostream& operator<<(std::ostream& out, const Point& p)

{

}