**Name:** _____

| Question | Points | Score |
|---|---|---|
| 1 | 10 | |
| 2 | 10 | |
| 3 | 10 | |
| 4 | 10 | |
| 5 | 10 | |
| 6 | 41 | |
| Total: | 91 | |

# Name: _____

# 2143 OOP Exam 1
# Fall 2025

1. (10 points) Below is some usage dealing with a typical college student. Based on the usage, create the class definition as seen on the study guide and as discussed in class.

**Usage:**

```
Student S1("John", "Doe", 20);
Student S2("Aarya", "Chakrabarti", 19);
S1.changeMajor("Computer Science");
S2.increaseTotalHours(12);
```

**Solution:**

**Rubric (10 points)**

- **Basic Class Structure** (3 Points)
  - Full class declaration, correct syntax for class and member declaration
  - Data members correctly declared as private
  - Data members have correct data type
  - Public access specifier used for methods
- **Constructors** (3 Points)
  - Implemented base, and whatever usage showed you to imppleement.
- **Setters and Getters** (2 Points)
  - All setters correctly declared with appropriate parameter types
  - All getters correctly declared, returning appropriate types
- **Additional Methods** (2 Points)
  - Correct implementation of `changeMajor`
  - Correct implementation of `increaseTotalHours`

```
class Student{
// default private
    string fname;   // shown in both constructors
    string lname;   // shown in both constructors
    int age;        // shown in both constructors
    string major;   // discovered from changeMajor method
    int hours;      // discovered from setHours method
public:
    //setters
    // setters return nothing, but recieve the value to be assigned to the data member
    void setFname(string);
    void setLname(string);
    void setAge(int);
    void setMajor(string);
    void setHours(int);
    // getters
    // getters return the value currently in the data member but have no parameters
    string getFname(string);
    string getLname(string);
    int getAge(string);
    string getMajor(string);
    int getHours(int);
    // additional methods discovered from usage
    void changeMajor(string);
    void increaseTotalHours(int);
};
```

2. (10 points) Below is some usage dealing with popular books. Based on the usage, create the class definition as seen on the study guide and as discussed in class.

**Usage:**

```
Book B0;
Book B1("1984", "George Orwell", 328, 1949);
Book B2("To Kill a Mockingbird", "Harper Lee", 281);
B2.setIsbn("978-1-4028-9462-6");
B2.setPrice(23.99);
```

**Solution:**

**Rubric (10 points)**

- **Basic Class Structure** (3 Points)
  - Full class declaration, correct syntax for class and member declaration
  - Data members correctly declared as private
  - Data members have correct data type
  - Public access specifier used for methods
- **Constructors** (3 Points)
  - Implemented base, and whatever usage showed you to impplement.
- **Setters and Getters** (2 Points)
  - All setters correctly declared with appropriate parameter types
  - All getters correctly declared, returning appropriate types
- **Additional Methods** (2 Points)
  - Correct implementation of `changeMajor`
  - Correct implementation of `increaseTotalHours`

```
class Book {
private:
    float price;        // discovered with the setPrice method
    int pages;          // In both constructors
    int year;           // In constructor B1
    string author;      // In both constructors
    string isbn;        // discovered with the setIsbn method
    string title;       // In both constructors
public:
    // Constructors
    // Usage showed default plus 2 overloaded constructors:
    Book();
    Book(string , string , int , int);
    Book(string , string , int);

    // Getters and Setters
    // No additional methods to add since the setIsbn and setPrice methods were just setters. However
    //    they did inform us that the class needed two more data members.
    float getPrice();
    int getPages();
    int getYear();
    string getAuthor();
    string getISBN();
    string getTitle();
    void setPrice(float);
    void setPages(int);
    void setYear(int);
    void setAuthor(string);
    void setISBN(string);
    void setTitle(string);
};
```

3. (10 points) Assume there is a class called **Pixel** and it has three data members:

```
class Pixel{
    int Red;   //possible values are 0-255
    int Green; //possible values are 0-255
    int Blue;  //possible values are 0-255
public:
    // constructor(s)
    // methods
};
```

**Where:**

- If we want a **red** pixel we set Red=255 Green=0 and Blue=0.
- If we want a **purple** pixel we set Red=128 Green=0 and Blue=128.
- If we want a **pink** pixel we set Red=255,Green=192,Blue=203.
- If we want to **GrayScale** the pixel we set Red,Green, and Blue = the average of the three color channels. In this case: Red=255,Green=192,Blue=203 would now be Red=216,Green=216,Blue=216

I would like you to write a method for the Pixel class called **grayScale** that turns a pixel into a shade of gray. To do this you can simply average the three color values together, then replace the Red, Green, and Blue data members with the average of all three color channels, so that each color channel has the same value. Assume you are defining this method outside of the class definition.
**A)** Show me what goes inside the class definition.
**B)** Show me how to define the method outside the class definition.

## Solution:

**Rubric (10 points)**

- **Inline Method (function) Header** (3 Points)
  - Method header had correct return type.
  - Had no params.
  - Had correct method name.
- **Function (method) Header** (4 Points)
  - Used the scope resolution operator tying external definition to class.
  - Correct return type and location.
  - Correct params (none).
- **Logic** (3 Points)
  - Averaged the three color channels and assigned value to each channel.

### A)

```
class Pixel{
  ...
    void grayScale(); // This is it! That's all.
  ...
};
```

### B)

```
/**
* Long version...
* The description implied that we are converting the pixel to gray scale
* internally by changing each channel to the avg. So we return void. But
* we also calculate the average and assign to each color channel.
*/
void Pixel::grayScale(){
    int sum = 0;    // sum of all three channels
    int avg = 0;    // not really needed but makes readable code
    sum = (Red + Green + Blue); // total up all color chanels
    avg = sum / 3;  // Calculate the avg;
    Red = Green = Blue = avg; // Assign same value to all 3 channels
}

/**
* Short version ...
*/
void Pixel::grayScale(){
    // Is this line less readable?? Or readable enough?
    Red = Green = Blue = (Red + Green + Blue) / 3;
}
```

4. (10 points) Using the same **Pixel** class as above, overload the **+** operator where adding two colors is really just averaging each color channel together.So if I added the following:

```
Pixel p1(100,100,100);
Pixel p2(50,50,50);
Pixel p3 = p1 + p2; // p3 would now contain (75,75,75) for its 3 color channels.
```

Assume you are defining this method inline.

**Solution:**

**Rubric (10 points)**

- **Function (method) Header** (4 Points)
  - Method header had correct return type (Pixel).
  - Used $operator+$ in header.
  - Passed in a const Pixel by address.
- **Function logic** (3 Points)
  - Averaged each channel separately and assigned avg to appropriate channel.
- **Logic** (3 Points)
  - Returned a Pixel type

```
// Short answer (not always the most readable
Pixel operator+(const Pixel &rhs){
    return Pixel((this->Red+rhs.Red)/2,this->Green+rhs.Green)/2this->Blue+rhs.Blue)/2;
}

// Longer answer
Pixel operator+(const Pixel &rhs){
    int r = (this->Red+rhs.Red)/2;
    int g = (this->Green+rhs.Green)/2;
    int b = (this->Blue+rhs.Blue)/2;
    Pixel p(r,g,b);
    return p;
}

// Another way
Pixel operator+(const Pixel &rhs){
    int r = (Red+rhs.Red)/2;
    int g = (Green+rhs.Green)/2;
    int b = (Blue+rhs.Blue)/2;
    return Pixel(r,g,b);
}
```

5. (10 points) Overload the **ostream** operator for a 3D point class which has the data data members: X, Y, Z and the output for a point would look like:

```
[X:30,Y:21,Z:0]
```

The numbers would obviously change for each point.

**Solution:**

**Rubric (10 points)**

1. **Method Header (5 points)**
   - Used $friend$ keyword.
   - Returned ostream&.
   - Used operator<<.
   - Passed in ostream&.
   - Passed in other instance correctly.
2. **Return value (3 points)**
   - Wrote a proper string to the ostream variable passed in by address and returned it.
3. **Formatting/Output Clarity (2 points)**
   - Data formatted correctly with square brackets and colons.

```
class Pixel3D {
private:
   // stuff
public:

    // Declare the operator<< as a friend. The signature must match the definition if you
    // do define the method outside of the class.
    // Doing all inline is ok as well.
    friend std::ostream& operator<<(std::ostream& os, const Pixel3D& p);
};

std::ostream& operator<<(std::ostream& os, const Pixel3D& p) {
    // write everything to the os instance we passed in:
    os << "[X:"<<p.X<<", "<<"Y:"<<p.Y<<", "<<"Z:"<<p.Z<<"]";
    // then return os to be printed to stdout.
    return os;
}
```

6. Find the appropriate word for each definition (blank) and write the words **corresponding letter** in the blank.

| | | | | |
|---|---|---|---|---|
| **A.** Method | **B.** Constructor | **C.** Overload | **D.** Instance | **E.** Object |
| **F.** Shallow Copy | **G.** Deep Copy | **H.** Private | **I.** Public | **J.** Destructor |
| **K.** LIFO | **L.** FIFO | **M.** Inheritance | **N.** Polymorphism | **O.** Encapsulation |
| **P.** Copy Constructor | **Q.** Assignment Operator | **R.** Definition | **S.** Pointers | **T.** Self Assignment |
| **U.** Terminates | **V.** Friend Keyword | **W.** New | **X.** Delete | |

(1) (5 points) When assigning one object to another, a _____**F**_____ only duplicates the memory addresses of dynamically allocated resources, while a _____**G**_____ allocates new memory and duplicates the actual data.

(2) (5 points) In a class, members that should only be accessible by methods of the same class are declared as _____**H**_____ , while members that should be accessible from outside the class are declared as _____**I**_____ .

(3) (5 points) Stacks use the _____**K**_____ principle, while a queues use the _____**L**_____ principle.

(4) (3 points) The ability of a function or operator to behave differently based on the types or number of arguments is known as _____**N**_____ .

(5) (7 points) The **"Rule of Three"** states that if a class requires a _____**J**_____ , a _____**P**_____ , or an _____**Q**_____ operator, it likely needs all three due to resource management needs such as dynamic memory.

(6) (3 points) The assignment operator should check for _____**T**_____ to avoid issues like overwriting oneself, which can lead to unexpected behavior or errors such as freeing memory that's still in use.

(7) (3 points) When defining a class in C++, a method with the same name as the class is called a _____**B**_____.

(8) (3 points) In C++, the _____**V**_____ keyword allows a function or class to access the private and protected members of another class, thus bypassing the concept of _____**O**_____, which normally restricts direct access to an object's internal state.

(9) (7 points) When you create a class the code that blueprints the class is the class's _____**R**_____ and when you create an _____**D**_____ of that class we call it an _____**E**_____.