

UMBC

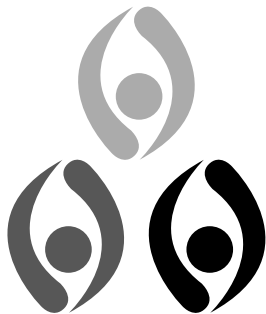
AN HONORS UNIVERSITY IN MARYLAND

Student's Name _____

Date _____ Subject _____

Professor's name: _____

Comments: _____



Integrity:
A Value That Endures

“By enrolling in this course, each student assumes the responsibilities of an active participant in UMBC’s scholarly community in which everyone’s academic work and behavior are held to the highest standards of honesty. Cheating, fabrication, plagiarism, and helping others to commit these acts are all forms of academic dishonesty, and they are wrong. Academic misconduct could result in disciplinary action that may include, but is not limited to, suspension or dismissal. To read the full Student Academic Conduct Policy, consult the UMBC Student Handbook, the Faculty Handbook, or the UMBC Policies section of the UMBC Directory.”

UMBC Faculty Senate
February 13, 2001

THIS PAGE INTENTIONALLY BLANK

1. The following code implements a function to find the minimum value of an integer array.

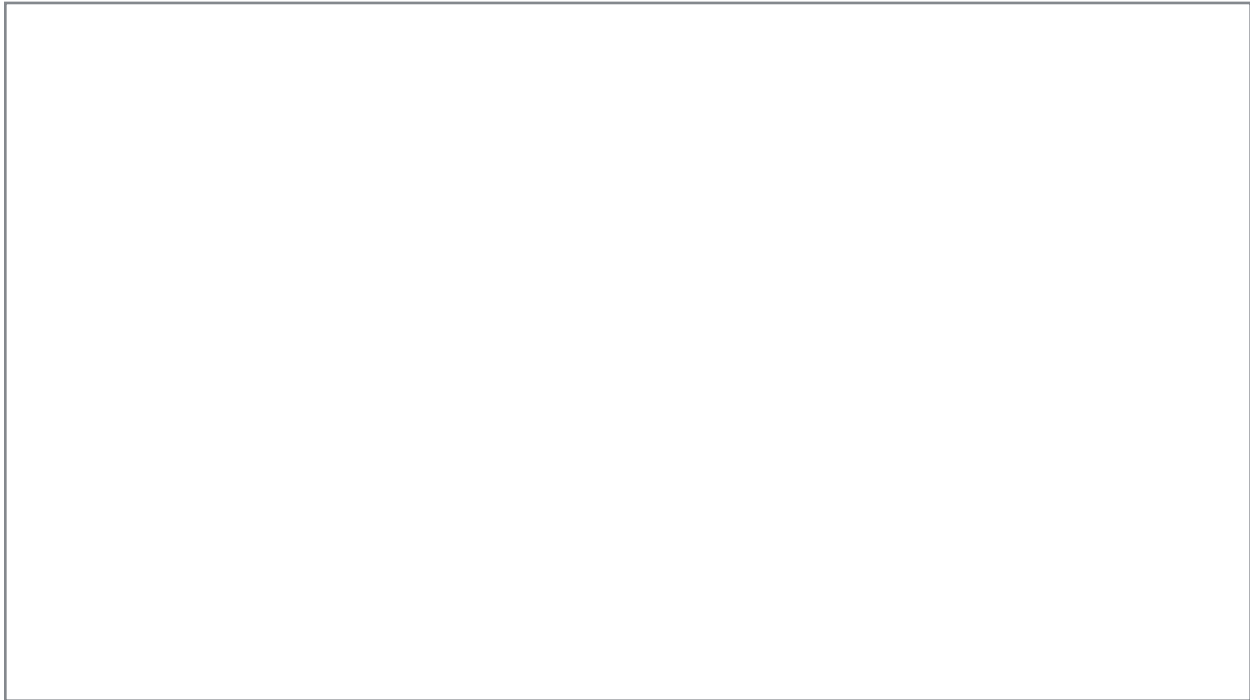
```
int min(int data[], int length) {  
    int smallest = data[0];  
    for (int i = 0; i < length; i++)  
        if (data[i] < smallest)  
            smallest = data[i];  
    return smallest;  
}
```

- a. (5 points) Define a simple exception class `EmptyArrayEx` and write a version of the function that throws the exception if the value of `length` is not positive.

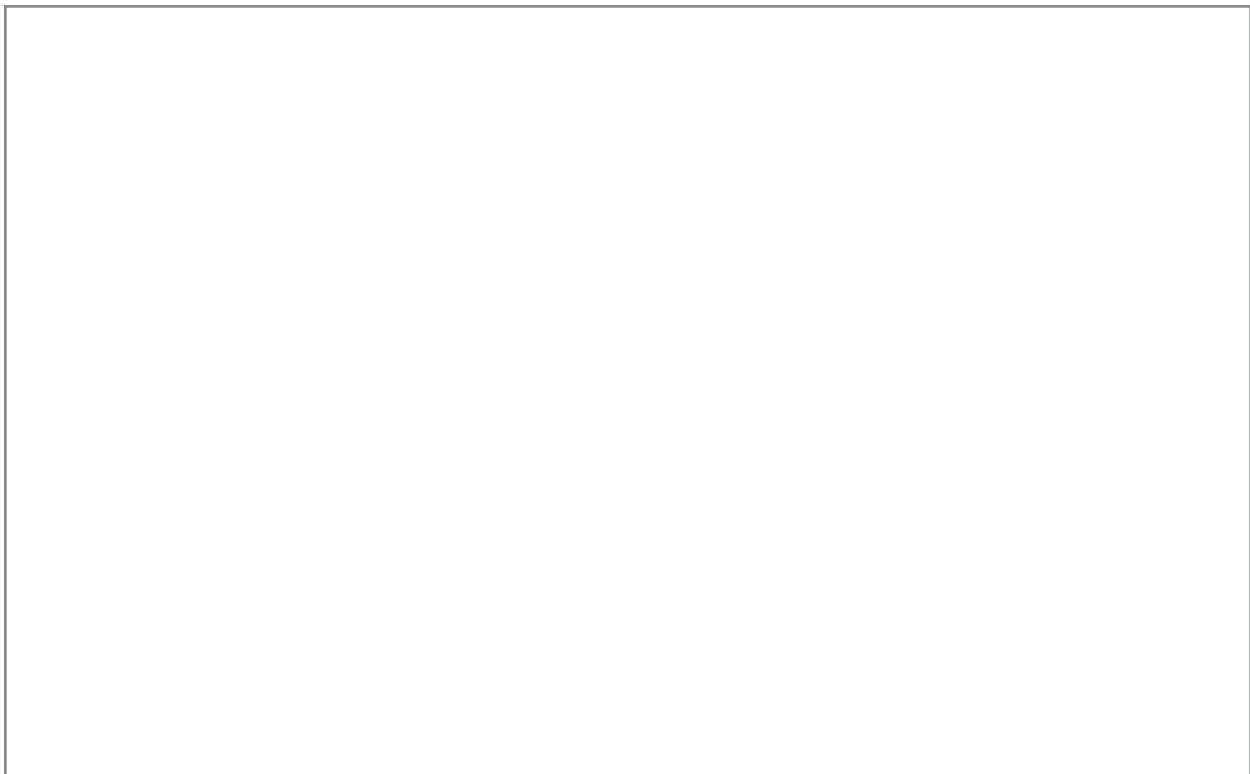
- b. (5 points) Write a simple `main()` function that calls `min()` and handles the `EmptyArrayEx` exception, should it be thrown.

2. Continuing with the `min()` code from the previous problem:

a. (8 points) Write a templated version of the function to find the minimum value in an array of type `T` (`T` is the template variable).



b. (2 points) Write a simple `main()` to demonstrate the use of the templated `min()`.



3. (5 points) Consider the following program that calls the `min()` function:

```
#include <iostream>
using namespace std;

class Donut {
public:
    Donut() : m_type("plain") {}
    Donut(string type) : m_type(type) {}
    string getType() const { return m_type; }
private:
    string m_type;
};

ostream& operator<<(ostream& sout, Donut &d) {
    sout << d.getType();
    return sout;
}

int main() {
    Donut donuts[3];
    donuts[0] = Donut("glazed");
    donuts[1] = Donut("boston cream");
    donuts[2] = Donut("cinnamon");

    Donut smallest = min(donuts, 3);
    cout << "The smallest donut is " << min(donuts, 3) << endl;

    return 0;
}
```

The program will *not* work with the templated `min()` function since `Donut` does not overload `operator<`. Write the overloaded `operator<` for the `Donut` class.

4. (20 points) Complete the code:

a. Declare the function `eat()` to be *pure virtual*:

```

[ ] void eat() [ ];

```

b. Declare the function `toUpperCase()` which has a single parameter `str`, a C-string:

```

void toUpperCase( [ ] str);

```

c. Declare a constant iterator `itr` for the double vector `dVector`:

```

[ ] itr = dVector.begin();

```

d. I want to insert the value of the integer variable `aScore` into the integer vector `scores`:

```

scores. [ ];

```

e. I want to call the `Bride()` function of the `Princess` object pointed to by `pPtr`:

```

Princess *pPtr = new Princess;

```

```

pPtr [ ] Bride();

```

f. I am overloading the assignment operator. I need to be sure I handle self-assignment (e.g. `x = x`) properly and that I return the appropriate value:

```

MyClass& MyClass::operator=(const MyClass& rhs) {
    if (this != [ ]) {
        /* execute only if NOT self-assignment */
    }
    return [ ];
}

```

g. Declare an integer pointer `iPtr` and initialize it to point to the third element of the integer array `data`:

```

[ ] iPtr = [ ];

```

5. (10 points) What output is produced by the following code?

```
1  int *p1, *p2;
2  int x = 2, y = 7;
3  int z[3] = {2, 3, 5};
4  p1 = &x;
5  p2 = &y;
6  cout << *p1 + *p2 << endl;
7  p2 = p1;
8  cout << *p1 * *p2 << endl;
9  p1 = z;
10 cout << *(p1++) * *p2 << endl;
11 cout << *p1 * (*p2)++ << endl;
12 cout << (*p1 + 1) / *p2 << endl;
```

6. (30 points) True or False?

- a. There are different iterator types for use with different STL containers.
- b. *Polymorphism* is synonymous with virtual functions and static binding.
- c. It is a good idea to throw an exception if an error occurs in a constructor.
- d. It is a good idea to throw an exception if an error occurs in a destructor.
- e. Multithreading allows several subcomputations to execute simultaneously.
- f. In Late Binding (also called Dynamic Binding), decisions as to which version of an overridden function should be called are made at run-time.
- g. Smart pointers help eliminate memory leaks.
- h. A class is *abstract* if at least one of its functions is pure virtual.
- i. `const` can be used to prevent a function from modifying its arguments.
- j. When `&` is used in a variable or return type declaration, it means "address of."
- k. *Inheritance* allows one data type to acquire properties of other data types.
- l. The nodes of a linked list are stored in successive memory locations.
- m. A base class pointer can point to a derived class object.
- n. A derived class pointer can point to a base class object.
- o. The order of precedence is the same for multiplication (`*`) and division (`/`).

7. (10 points) Consider the following program:

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 class Pastry {
6 public:
7     Pastry() {}
8     virtual string Bite() { return "Take a bite of the pastry."; }
9 };
10
11 class Donut : public Pastry {
12 public:
13     Donut() : Pastry(), m_type("glazed") {}
14     Donut(string type) : Pastry(), m_type(type) {}
15     void Dunk() { cout << m_type << " donut is dunked!" << endl; }
16     virtual string Bite() {
17         return "Take a bite of the " + m_type + " donut.";
18     }
19 private:
20     string m_type;
21 };
22
23 int main() {
24     Pastry *pPtr = new Donut("powdered sugar");
25     Donut *dPtr = new Donut("jelly filled");
26
27     pPtr->Dunk();
28     cout << pPtr->Bite() << endl;
29
30     dPtr->Dunk();
31     cout << dPtr->Bite() << endl;
32
33     return 0;
34 }
```

a. One line prevents the program from compiling. Which line and why is it incorrect?

b. If the incorrect line is deleted and the program is run, what output will it produce?

8. (10 points) The following code partially defines the `Car` class (constructors, `addPassenger()` function, and prototype `operator<<`) and includes a sample `main()` function. On the following page, write a destructor for the `Car` class. *Note:* details of the `Passenger` class are not needed.

```
class Car {
public:
    Car() : m_make("Toyota"), m_model("Corolla"), m_seats(5) {
        m_passengers = new Passenger*[ m_seats ];
        for (int i = 0; i < m_seats; i++)
            m_passengers[i] = NULL;
    }

    Car(string make, string model, int seats) :
        m_make(make), m_model(model), m_seats(seats), m_seatsTaken(0) {
        m_passengers = new Passenger*[ m_seats ];
        for (int i = 0; i < m_seats; i++)
            m_passengers[i] = NULL;
    }

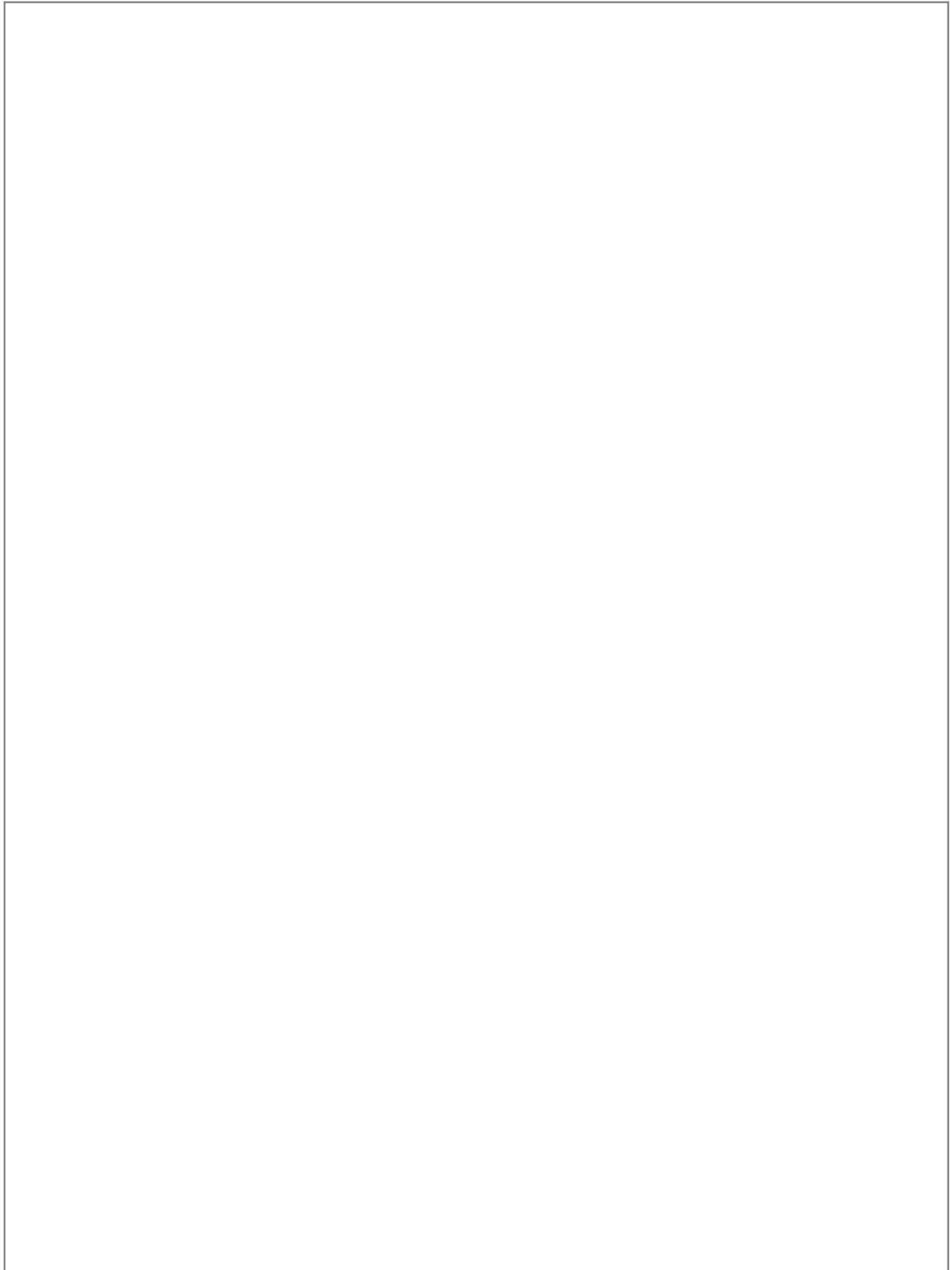
    void addPassenger(Passenger *passenger) {
        if (m_seatsTaken < m_seats) {
            for (int i = 0; i < m_seats; i++)
                if (m_passengers[i] == NULL) {
                    m_passengers[i] = passenger;
                    break;
                }
        }
    }
}

friend ostream& operator <<(ostream& sout, const Car& car);

private:
    string m_make;
    string m_model;
    int m_seats;
    int m_seatsTaken;
    Passenger **m_passengers;
};

int main() {
    Car c("Toyota", "Sienna", 7);
    c.addPassenger( new Passenger("Alice") );
    c.addPassenger( new Passenger("Bob") );
    cout << c << endl;
    return 0;
}
```

Problem 8 Solution:



9. Let n be the length of the input array to the `min()` function from Problem 1:

```
int min(int data[], int length) { // n = length
    int smallest = data[0];
    for (int i = 0; i < length; i++)
        if (data[i] < smallest)
            smallest = data[i];
    return smallest;
}
```

- a. (5 points) What is the Big-O complexity of `min()` as a function of the input size n ? Justify your answer.

- b. (5 points) An array of length n can be sorted with complexity $O(n \ln n)$. Describe a more efficient algorithm (than `min()`) to find the minimum value of an integer array. What is the Big-O complexity of the algorithm?

THIS PAGE INTENTIONALLY BLANK

Page	Points	Earned
1	10	
2	10	
3	5	
4	20	
5	10	
6	30	
7	10	
8/9	10	
10	10	
Total	115	