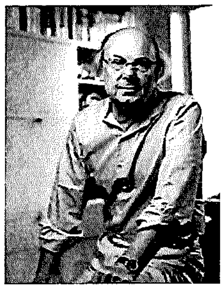
Name SAULL SINGH Net ID: SKS9379

CS-UY 2124 - Object Oriented Programming
MID-TERM EXAM #1 – October 24, 2023

- **Do not open this test booklet until you are instructed to do so.**
- Duration: 1 hour, 15 minutes
- Do not separate any pages. Do not pull the test apart from the staple.
- Ensure your name and Net ID is printed at the top of every page.
- This is a closed book exam, no calculators, computers, or phones are allowed.
- Anyone found cheating on this exam will receive a zero for the exam.
- Anyone who is found writing after time has been called will receive a zero for this exam.
- If you have a question, please ask the proctor of the exam.
- Note that we have omitted any **#includes** or **using namespace std;** statements in all questions in order to save space and to save your time thinking about them. You may assume that all such statements that are needed are present. And you don't have to write them either!!!
- You also do not need to write any comments in any of your code.
- Please read all questions carefully! They may look familiar and yet be completely different.
- Answering the short-answer questions, in particular, requires that you read and understand the programs shown. You need to read them carefully if you are going to understand them.
- If a question asks you to write a class or a function and provides you with test code, **be sure your class / function works with that test code**. If the question provides you with sample output, then your answer should match that output.
- Print your name and Net ID on the top of **EACH** page.

Name SAHIL SINGH Net ID: SKS9779

1. **EXTRA CREDIT (3 points):** Who created C++? Fill in the circle that corresponds to your answer. Fill in only one circle.



- ○ Bill Gates
- ○ Sergey Brin
- ○ Guido van Rossum
- ○ Ada Lovelace
- ○ Alan Turing
- ○ James Gosling
- [✓ Bjarne Stroustrup]
- ○ Dennis Ritchie
- ○ Claude Shannon
- ○ None of the above

2. **(5 pts.)** Consider the following class Circle:

```
const double PI = 3.14; // Yes, a global constant. No, not a problem.

class Circle {
public:
    Circle(double radius): theRadius(radius) {}
    double calc_area() {
        return PI * theRadius * theRadius;
    }
private:
    double theRadius;
};
```

If a prototype for a function named print is declared as follows,

```
void print(Circle* cir_ptr);
```

Which of the following expressions can be a **call** to the Circle's `calc_area` method using the `cir_ptr` parameter?

Completely fill the circle next to your Choice.

- ○ `cir_ptr.calc_area()`
- ○ `cir_ptr-<calc_area()`
- ○ `cir_ptr>-calc_area()`
- ○ `(*cir_ptr.calc_area())`
- [✓ `cir_ptr->calc_area()`]
- ○ `*(cir_ptr.calc_area())`
- ○ None of the above.

Name Sali Net ID: ____________

3. **[5 points]:** Given the following code:

```
class Holodeck {
public:
    Holodeck(int val) : program(val) {}

    void display() {
        cout << "Holodeck program: " << program << endl;
    }
private:
    int program = 17;
};

class Galaxy {
public:
    void display() { cout << "Galaxy class ship\n"; }
private:
    Holodeck something;
};

int main() {
    Galaxy gal;
    gal.display();
}
```

What is the result of compiling and running the above code? Completely fill the circle next to your choice.

- ○ `Holodeck program: 17`
- ○ `Galaxy class ship`
- ○ `Holodeck program: 17`
  `Galaxy class ship`
- ○ `Galaxy class ship`
  `Holodeck program: 17`
- ○ Compile time error
- [✓ Run time error (or undefined behavior)]
- ○ None of the above

Name Salil Net ID: ____________

4. **(5 points):** Consider the existence of a class named **`Paint`**. In a program, two **`Paint`** objects have been instantiated by the statements below:

```
Paint pnt1("red");
Paint pnt2("blue");
```

After the objects are created, a subsequent line in the program's source code contains the following:

```
pnt1 = pnt2;
```

The statement above produces which of the following function calls? Completely fill the circle next to your choice.

- O `pnt1.assignment=(pnt2)`
- [✓ `pnt1.operator=(pnt2)`]
- O `pnt2.operator=(pnt1)`
- O `pnt2.assignment=(pnt1)`
- O `operator=(pnt1, pnt2)`
- O the **`Paint`** copy constructor
- O `Paint(pnt2.color)`
- O `~Paint()`
- O None of the above

5. **(5 pts)** Consider a program defining a class named **`Pirate`**. An overloaded output operator function exists that accepts a **`Pirate`** instance as its second parameter. The program creates a vector of **`Pirate`** objects named **`pirates`**:

```
vector<Pirate> pirates;
```

The vector is populated with **`Pirate`** objects. Write a **ranged for loop** that outputs each Pirate on a separate line:

```
for(Size_t i= 0; i< pirates.size()) ; i++{
    Cout<< pirates[i]<<endl;

}
```

Name [illegible] Net ID: ______________

0 6. [5 pts.] Consider the code below. What would be the output when attempting to compile-and-run the program? Completely fill the circle next to your choice.

```
void doSomething(int& num)
{
    num = 0;
    cout << num << ", ";
}

int main()
{
    int x = 2;
    cout << x << ", ";
    doSomething(x);
    cout << x << endl;
    return 0;
}
```

int m = 0

- ○ 0, 0, 0
- ○ 2, 0, 2
- ○ 2, 0, 0
- [✓ 2, 2, 2]
- ○ Undefined
- ○ None of the above

Name Saril Net ID: ____________

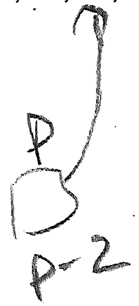
7. **[5 pts.]** Given an **array of integers** whose address is stored in the variable **data**.

The array holds the following ten values: 0, 1, 4, 9, 16, 25, 36, 49, 64 and 81.

What will the result of the following code be:

25

```
int* p = &data[5];
cout << p[-2] << endl;
```



p[-2]

-2

Given the code below. What would be the output when attempting to compile-and-run the program? Completely fill the circle next to your choice.

○ 0	○ 36
○ 1	○ 49
○ 4	○ 64
● 9	○ 81
○ 16	○ Undefined
○ 25	○ None of the above

Name Sahil ______________________ Net ID: ______________

8. **[10 pts.]** Consider the class below.

```
struct Room {
    int size;
    int num_beds;
};

class Hotel {
public:
    Hotel(string hotel_name, vector<Room> v) {
        name = hotel_name;
        num_rooms = v.size();
        rooms = new Room[num_rooms];
        for (int i = 0; i < num_rooms; i++) {
            rooms[i].size = v[i].size;
            rooms[i].num_beds = v[i].num_beds;
        }
    }

    void display() {
        cout << name << ": ";
        for (int i = 0; i < num_rooms; i++) {
            cout << "Room: " << i << ": Room size: "
                 << rooms[i].size << ", number of beds: "
                 << rooms[i].num_beds << endl;
        }
    }

private:
    string name;
    int num_rooms;
    Room* rooms;
};
```

Implement the destructor and copy constructor on the next page.

Name Salil Net ID:

a. **Write a destructor for the Hotel class.**

```
~Hotel(){
    delete rooms

        }
```

b. **Write a copy constructor for the Hotel class.**

```
Hotel( Hotel* rhs) {
    if (this != &rooms){
        hotel.operator= rhs;



    }

}
```

Name Sahil Net ID: S

9. **[20 pts.]** Write two functions:
   - one will read a tab-separated (`\t`; tabs are whitespace) input data file that contains entries pertaining to soccer teams, and which populates a vector that stores the data structures that hold each row of data, and
   - one that will display those vector items which match a filter requirement.

The Input File

Each line of the file is formatted as shown below. There is no header line in the input file. Each line of the input file represents an **`Entry`**. Note that there is no whitespace within the *team*, *ranking*, or *state* values. (See the example data file.)

```
team\tranking\tstate
```

You can assume that an **`Entry`** struct is defined containing the 3 fields: `team`, `rank` and `state`.

read_data Function

Implement the **`read_data`** function. Your implementation should accept the input stream and a vector for the data. The function will fill the vector and return nothing.

You may assume the input stream has been correctly opened already!

display_match Function

Implement the **`display_match`** function. Your implementation should accept the data vector and a string value. Print any vector items which have a state that is equal to the string parameter. This function does not return a value. See the Example Output for formatting.

Example of calling the **`read_data`** and **`display_match`** functions from `main`

As an example, the functions could be called as:

```
int main() {
    vector<Entry> entries;
    ifstream ifs("entries.txt");
    read_data(ifs, entries);
    ifs.close();
    display_match(entries, "NY");
    display_match(entries, "MN");
    display_match(entries, "MA");
}
```

Example of the input file

```
LAFC   25    CA
NER    20    MA
NYFC   19    NY
RB     23    NY
DFC    10    IL
TFC    27    FL
```

Example output

```
NY: NYFC(19), RB(23),
MN: -
MA: NER(20),
```

Name Salil Net ID: ______

Implement the read_data and display_match functions below.

```
Struct Entry{
        string team;
        int ranking;
        string state;

Void readdata(ifstream ifs, vector<Entry>&ent){
        string t_team;
        int t_rank;
Entry e;  string t_state;
        while (ifs>>t_team>>t_rank>>t_state){
                e.team = t_team;
                e.rank = t_rank;
                e.State = t_state;
        }
        ent.push_back(e);
}

Void display_match(vector<Entry>&ent, string name){
        string st.
        vector<string> new;
        for (size_t i=0; i<ent.size; i++){
                if (e[i].state == name){
                        cout << name << ": " << ent.team
                                << "(" << ent.ranking <<
                                ")";
                }
                else{
                        cout << name << ": -";
                }
        cout<<endl;
}
```

Name Salil Net ID:

10. [**40 pts.**] **You will define a single class**, SoccerTeam, defined as follows:

A SoccerTeam has

- a ranking (int)
- a name (string),
- a city where it is located (string)
- a state (2 letters) where it is located (string)
- a collection of opponent SoccerTeam(s) that it intends to play friendly (unofficial) games with, for pre-tournament practice.

Provide the definition of the SoccerTeam class. This is the ONLY class you need to write.

Implement the following functions

- an appropriate constructor for the SoccerTeam class (see the test code below)
- an output operator for a SoccerTeam object providing the information about the team. This must include the SoccerTeam's ranking, city, state and the list of opponent SoccerTeam names that it will face in the friendly games, if applicable. (See example output below.)
- a member function (aka method) **add_opponent** which adds a SoccerTeam to the collection of opponent SoccerTeam(s).
- a member function (aka method) **can_play** that enforces the rules, below, determining if a team can play another team, i.e. the team that is passed in. **can_play** will return true if the two teams are allowed to play and false otherwise. You probably want to use this method in your implementation of **add_opponent** to save a lot of duplicate writing.

Enforce the following rules

- A SoccerTeam cannot be added to its own collection of opponents
- An opponent SoccerTeam may be added if, and only if:
  - Its ranking differs by less than "6" (i.e. 5 or smaller) from the ranking of the SoccerTeam it's being added to, and
  - The two SoccerTeams are not located in the same state.
- This is a **reciprocal** relationship. That means that when SoccerTeam A adds SoccerTeam B to its collection of opponents, then SoccerTeam B will also have SoccerTeam A in its collection of opponents.
- a SoccerTeam cannot be added more than once to the collection of another SoccerTeam,
  - Of course, a SoccerTeam may be added to *multiple* SoccerTeams, e.g. SoccerTeam A might be added to SoccerTeam B, as well as to SoccerTeam C's collection of opponents.
- the add_opponent method should not fail silently. That means that if it fails, return false. If it succeeds, return true.
- **It is possible for two different teams to have the same name!**

Note

- This problem does not involve copy control or the heap. **Do not** allocate a SoccerTeam on the heap!

There is Sample test code on the next page

Name Sahil Net ID:

Sample test code
You should consider the following code to test your implementation:

```
int main() {
    SoccerTeam p1(22, "Revolution", "Boston", "MA");
    SoccerTeam p2(19, "NYFC", "New York", "NY");
    SoccerTeam p3(20, "Red Bulls", "New York", "NY");
    SoccerTeam p4(20, "Red Devils", "Chicago", "IL");
    SoccerTeam p5(25, "LAFC", "Los Angeles", "CA");
    SoccerTeam p6(10, "DFC", "Dallas", "TX");
    SoccerTeam p7(27, "TFC", "Tampa", "FL");

    p1.add_opponent(p2);      // returns true
    p1.add_opponent(p1);      // returns false, can't add to self
    p2.add_opponent(p1);      // returns false, already added to p1
    p2.add_opponent(p3);      // returns false, same state
    p4.add_opponent(p1);      // returns true
    p5.add_opponent(p6);      // returns false, ranking diff >5
    p7.add_opponent(p1);      // returns true

    cout << p1 << endl << endl
         << p2 << endl;
}
```

Sample output
The following output was produced from executing our sample test code on our solution:

```
Team Revolution (Boston, MA), ranked 22 faces: NYFC, Red Devils, TFC,

Team NYFC (New York, NY), ranked 19 faces: Revolution,
```

Begin your answer to this question on the next page.

Name Sahil Net ID:

```
Class SoccerTeam{
    Friend ostream& operator(ostream& os, Const SoccerTeam p);
    public:
        bool add_play (Soccerteam* p1){
            if( this!==&p1 || state ==&p1.t-stat
                || (ranking-p.ranking) >6){
                return false;
                };

            return true;

        }
        bool add component(Soccerteam* p1){
            string Found;
            if (Found==true){
                Col.push_back(P1.name);
                return true;
                }
            else{
                return false;
            };
```

Name Sol'n Net ID:

```
ostream& operator (ostream &os, const SoccerTeam p){
    os << "Team " << name << " (" << city
    << ", " << state << "), ranked " << rank <<
     " focus:";
     for(string x : col){
         cout << x << ", ";

     }

 };

private:
   int ranked;
   string name;
   string city;
   string state;
   vector<string> col;



};
```