Define an abstract class **Case** with a text label, a numerical colour, a method providing the number of Cases and the pure virtual method getCapacity computing the capacity (volume) of a case.

Define the following classes inherited from the Case:

- **BrickCase** being a cuboid with a length, a width and a height,
- **TubeCase** being a cylinder with a base radius and a height,
- **PrismCase** being a regular triangular prism with a base side length and a height.

Override, for each of the above classes, the virtual method getCapacity, making it to return the volume of a case of the given class. Implement all the constructors, destructors, getters, setters and exceptions which make the functionality of the classes complete, and all the other methods and exceptions necessary to run the code provided below.

Define the class **Repository** with a description, a total capacity and a dynamic list of the cases stored in the repository. Implement the following public methods of the class:

- a one enabling to add a new case of an arbitrary type to the repository (with the control of the total capacity of the repository, CapacityError should be thrown if the total capacity could be exceeded),
- a one enabling to remove the case (by its label) from the repository (throwing the NameError exception if it doesn't exist),
- a one enabling to remove all the cases from the repository,
- a method returning the summary volume of all the cases stored in the repository.

  Overload the indexing operator ([]) for the Repository to have a direct access to the item on the particular position in the list (throwing the IndexError exception if it doesn't exist). Overload the shift-left operator (<<) printing the data of the repository and the details of all the cases stored. Add all the other members which are necessary to make the functionality of the class complete or are required to run the code below.

Write a program which tests all the class capabilities, in particular the following code should be enabled:

```
Repository repo("Warehouse", 100);
cout << Case::count() << endl; //0
try {
repo.add(new BrickCase("books", 0x000000, 5, 3.5, 4)); //5 x 3.5 x 4
repo.add(new TubeCase("posters", 0x00ff00, 1, 3)); //radius=1, height=3
repo.add(new PrismCase("toys", 0x800080, 3, 2)); //base=3, height=2
repo.add(new BrickCase("jewels", 0xffff00, 2, 2, 2)); //2 x 2 x 2
repo.add(new BrickCase("trinkets", 0x000080, 5, 1, 1)); //5 x 1 x 1
} catch(Repository::CapacityError &e) {
cout << e.what() << endl; //trinkets too large - only 4.8 free space
}
cout << repo;
//Warehouse, total capacity: 100.0, free space: 4.8:
//1. books, 0x000000, volume: 70.0
//2. posters, 0x00ff00, volume: 9.4
//3. toys, 0x800080, volume: 7.8
//4. jewels, 0xffff00, volume: 8.0
cout << Case::count() << endl; //4
cout << repo.summaryVolume() << endl; //95.2
repo.remove("toys");
cout << Case::count() << endl; //3
cout << repo.summaryVolume() << endl; //87.4
try {
cout << repo[1].getCapacity() << endl; //70.0
cout << repo[5].getCapacity() << endl; //IndexError exception
} catch(Repository::IndexError &e) {
cout << e.what() << endl; //item no. 5 not found
}
repo.clear();
cout << Case::count() << endl; //0
```

Define an abstract class **Measure** with a number, a name, a method providing the number of Measures and the pure virtual method getResult returning the final result for the measure.
Define the following classes inherited from the Measure:

- **MinMeasure** representing a measure with three values given (the result for the measure is the minimal one),
- **MaxMeasure** representing a measure with three values given (the result for the measure is the maximal one),
- **AvgMeasure** representing a measure with three values given (the result for the measure is the average value of the given values).

Override, for each of the above classes, the virtual method getResult, making it to return the final result of a measure of the given class.
Implement all the constructors, destructors, getters, setters and exceptions which make the functionality of the classes complete, and all the other methods and exceptions necessary to run the code provided below.

Define the class **WeatherStation** with a location, an institution name, a total number of measure devices allowed and a dynamic list of the measure devices installed. Implement the following public methods of the class:

- a one enabling to add a new measure of an arbitrary type to the station (with the control of the measures limitation, LimitError should be thrown if the number of measures could be exceeded),
- a one enabling to remove the measure (by its name) from the results (throwing the NameError exception if it doesn't exist),
- a one enabling to remove all the measures from the station,
- a method returning the maximal spread of all the measures made (the spread for a single measure is the difference between the minimal and the maximal value given for the measure).
  Overload the indexing operator ([]) for the WeatherStation to have a direct access to the item on the particular position in the list (throwing the IndexError exception if it doesn't exist). Overload the shift-left operator (<<) printing the data of the weather station and the details of all the measures made. Add all the other members which are necessary to make the functionality of the class complete or are required to run the code below.

Write a program which tests all the class capabilities, in particular the following code should be enabled:

```
WeatherStation st(“Lodz”, “Weather Institute”, 3); //max 3 measure devices
cout << Measure::count() << endl; //0
try {
 st.add(new MinMeasure(1, “Temperature”, 2.3, 2.5, 2.0)); //degrees, spread 0.5
 st.add(new MaxMeasure(2, “Wind”, 20, 18.5, 19)); //km/h, spread 1.5
 st.add(new AvgMeasure(3, “Preasure”, 1015, 1010, 1008)); //hPa, spread 7
 st.add(new MaxMeasure(4, “Humidity”, 75, 73, 77)); //percents, spread 4
} catch(WeatherStation::LimitError &e) {
  cout << e.what() << endl; //only 3 measure devices allowed
}
cout << st;
//Weather Institute, Lodz, max. spread: 7.0:
//1. Temperature, result: 2.0
//2. Wind, result: 20.0
//3. Preasure, result: 1011.0

cout << Measure::count() << endl; //3
cout << st.maxSpread() << endl; //7.0
st.remove(“Preasure”);
cout << Measure::count() << endl; //2
cout << st.maxSpread() << endl; //1.5

try {
  cout << st[1].getResult() << endl; //2.0
  cout << st[5].getResult() << endl; //IndexError exception
} catch(WeatherStation::IndexError &e) {
  cout << e.what() << endl; //item no. 5 not found    }
st.clear();
cout << Measure::count() << endl; //0
```