Define an abstract class Work with a text description, a name of a worker, a method providing the number of Works available and the pure virtual method getCost computing the cost of a work.
Define the following classes inherited from the Work:

- PieceWork representing the Work involving the repeated execution of the same action (with a number of the executions and a cost of a single action),
- FieldWork representing the Work of making some action in the area (with a length and a width of the area in meters and a cost per square meter),
- TimeWork representing the Work paid per hour (with the number of hours and a cost per hour).

Override, for each of the above classes, the virtual method getCost, making it to return the cost of a work of the given class. For TimeWork, assume that each 9th hour is unpaid (i.e., if a person works 8 hours, he/she is paid for 8, if 9 – is paid for 8, if 17 – is paid for 16, if 20 – is paid for 18, if 28 – is paid for 25 etc.).
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 Schedule with a description, a total budget and a dynamic list of the works to be performed. Implement the following public methods of the class:

- a one enabling to insert a new task (Work) of an arbitrary type to the work list on the specified position (moving the items from this and the further positions towards the end of the list), or at the end of the list if the position doesn't exist. The position numbers should start from 1 (DeficitError should be thrown if the total budget could be exceeded),
- a one enabling to remove the first work from the schedule (throwing the EmptyError exception if the schedule is empty),
- a one enabling to remove all the tasks (Works) from the list,
- a method returning the summary cost of all the works planned in the schedule.

Overload the indexing operator ([]) for the Schedule to have a direct access to the task on the particular position in the schedule (throwing the IndexError exception if it doesn't exist). Overload the shift-left operator (<<) printing the data of the schedule and the details of all the works planned. 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:

```
Schedule repairs("Expected repairs of my room", 2000); //budget=2000€
cout << Work::count(); //0
try {
  repairs.insert(1, new FieldWork("floor", "John", 4.5, 6, 30)); //4.5x6, 30€
  repairs.insert(2, new FieldWork("walls", "Luke", 21, 2.5, 15)); //21x2.5, 15€
  repairs.insert(1, new PieceWork("electric points", "Ben", 7, 20)); //7pcs, 20€
  repairs.insert(4, new TimeWork("cleaning", "Mary", 18, 10)); //18h, 10€
  repairs.insert(4, new PieceWork("lighting", "Tom", 4, 30)); //4pcs, 30€
} catch(Schedule::DeficitError &e) {
  cout << e.what(); //lighting too expensive - only 102.50 free money
}
cout << repairs;
//Expected repairs of my room, total budget: 2000.00, remaining money: 102.50:
//1. electric points (Ben), cost: 140.00
//2. floor (John), cost: 810.00
//3. walls (Luke), cost: 787.50
//4. cleaning (Mary), cost: 160.00
cout << Work::count(); //4
cout << repairs.summaryCost() << endl; //1897.50
repairs.removeFirst();
cout << Work::count(); //3
cout << repairs.summaryCost() << endl; //1757.50
try {
  cout << repairs[1].getCost() << endl; //810.00
  cout << repairs[5].getCost() << endl; //IndexError exception
} catch(Schedule::IndexError &e) {
  cout << e.what(); //item no. 5 not found
}
repairs.clear();
cout << Work::count(); //0
```