8th Recitation

Hash Tables

Instructions:

- Download the file `aed2324_p08.zip` from the course page and unzip it (it contains the `lib` folder, the `Tests` folder with the files `funHashingProblem.h`, `funHashingProblem.cpp`, `jackpot.h`, `jackpot.cpp` and `tests.cpp`, and the files `CmakeLists.txt` and `main.cpp`);
- In CLion, open a project by selecting the folder containing the files from the previous point.

1. Consider the `FunHashingProblem` class which has only static methods. Implement the member-function as described below:

```
vector<int> FunHashingProblem::findDuplicates(const vector<int>& values, int k)
```

Given a vector of integer values and a positive number `k`, check whether the vector contains any duplicate values within any range of consecutive indices that span a maximum distance of `k` (i.e., the number of increments needed to go from the first to the last element of the range). If `k` is larger than the size of the vector, it should check whether there are duplicate elements in the entire vector. This function returns a vector with the duplicate elements (in the order it finds them) or an empty one if there are none, but it does not modify the original (input) vector.

Suggestion: Use a hash table (`unordered_set`) to record the vector elements that are in the index range `[i..i+k]`.

Expected time complexity: O(*n*)

Execution example:

input: *values* = `{5, 6, 8, 2, 4, 6, 7}` and k = `4`
output: `{6}`
The solution finds the duplicate element `6`, repeated at a maximum distance `4`, which is $\leq k$.

input: *values* = `{5, 6, 8, 2, 4, 6, 9}` and k = `2`
output: `{}`
Does not find duplicates: element 6 is duplicated, but it is repeated at a maximum distance $> k$.

input: *values* = `{1, 2, 3, 2, 1}` and k = `7`
output: `{1, 2}`
Duplicate element `1` is repeated at maximum distance `4`; duplicate element `2` at distance `2`, both $\leq k$.

2. You want to implement a program to help manage bets in the *Totoloto* game. Consider the `Jackpot` class which stores existing bets in a hash table.

```
class Bet {
    std::vector<int> numbers;
    std::string player;
public:
    Bet(vector<int> ns, std::string p);
    ...//
};
```

```
typedef std::unordered_set<Bet,
                           betHash,
                           betHash> tabHBet;
class Jackpot {
    tabHBet bets;
public:
    void addBet(const Bet& ap);
    unsigned betsInNumber(unsigned num) const;
    std::vector<string> drawnBets(
        std::vector<int> draw) const;
    unsigned getNumBets() const;
};
```

* **Note:** you are strongly advised to carry out each of the subexercises in the order listed below.

2.1 Implement the member function described below:

```
void Jackpot::addBet(const Bet& b)
```

This function adds a given `Bet b` to the set of existing bets. Consider that a player cannot place identical bets.

2.2 Implement the member function described below:

```
unsigned Jackpot::betsInNumber(unsigned num) const
```

This function determines how many times the number `num` appears in the total number of bets placed.

2.3 Implement the member function described below:

```
std::vector<std::string> Jackpot::drawnBets(std::vector<int> draw) const
```

This function returns a vector with the names of the players with winning bets (i.e., those with more than `3` values equal to the draw values).