Technical University of Denmark

Written examination date: 22 December 2022

Course title: Programming in C++ Page 1 of 17 pages

Course number: 02393

Aids allowed: All aids allowed

Exam duration: 4 hours

Weighting: pass/fail

Exercises: 4 exercises with 3 or 4 tasks each, for a total of 14 tasks

Submission details:

1. You must **submit your solution on DTU Digital Eksamen**. You can do it **only once**, so submit only when you have completed your work.
2. You must submit your solution as **one ZIP archive** containing the following files, with **these exact names**:
   - `ex`*`ZZ`*`-library.cpp`, where *ZZ* ranges from `01` to `04` (i.e., one per exercise);
   - `ex04-library.h` (additionally required for exercise 4).
3. You can test your solutions by uploading them on CodeJudge, under "`Exam December 2022`" at:

   `https://dtu.codejudge.net/02393-e22/exercises`
4. You can test your solutions on CodeJudge as many times as you like. *Uploads on CodeJudge are not official submissions* and will not affect your grade.
5. Additional tests may be run on your submissions after the exam.
6. Feel free to add comments to your code.
7. **Suggestion:** read all exercises before starting your work; start by solving the tasks that look easier, even if they belong to different exercises.

Exercise 1. The Wumpus

Alice is implementing a variant of the old game *Hunt the Wumpus*: the player explores a randomly-generated cavern with a map made of tiles. A map of size 6 × 16 might look like **Fig. 1** on the right:

- 'X' is the current position of the player;
- ' ' is an empty tile: the player can move there;
- '#' is a rock: the player cannot move onto that tile;
- 'W' is the Wumpus: a monster living in the cavern.

```
 # ##   #  #
 ##      W#  ##
      #  #   ##
   #   #    #
##     X #
###      ###  #
```

Fig. 1. Randomly-generated 6 × 16 cavern with all tiles visible.

During the game, the player explores the cavern in the dark, with a torch that only illuminates the nearby tiles. The dark cavern is drawn as shown in **Fig. 2** on the right:

- when a tile is close enough to the player, it is lit by the torch, hence its content is visible and appears as described above. Also, the tile is marked as "explored";
- when a tile is too far from the player, it is *not* lit by the torch. Therefore:
  - if the tile was *not* explored before, its content is unknown to the player, and it is shown as '?';
  - if the tile was previously explored by the player, then:
    - ∗ if the tile contains a rock, it is shown as '#';
    - ∗ otherwise, it is shown as '-'.

    In other words: the game map always shows already-explored rocks '#', even when they are not lit; instead, other explored but unlit tiles are shown as '-' (such tiles may be empty, or contain the Wumpus).

```
----#-##????????
-##--    W??????
----  #  # ?????
---#   #   ?????
##--   X # ?????
###-     ##?????
```

Fig. 2. The cavern above in the dark, with some explored tiles.

Alice has written some code: a test program is in the file `ex01-main.cpp`, and an (incomplete) implementation is in `ex01-library.h` and `ex01-library.cpp`. Such files are available in the ZIP archive for this exam, and in the next pages.

Structure of the code. A tile of the cavern is represented as a `struct Tile` with two fields named `content` and `explored`, which represent, respectively:

- the content of the tile: and `enum` value between `nothing`, `player`, `rock`, or `wumpus`;
- whether the tile has been already explored by the player.

Alice's code already includes two functions (that you must not modify):

- `void setupCavern(Tile **cav, unsigned int m, unsigned int n, unsigned int seed)`
- `void deleteCavern(Tile **c, unsigned int m)`

These functions respectively insert random contents in a cavern `cav` with `m` rows and `n` columns (created with `createCavern()`, see task **(a)**), or deallocate the cavern.

Tasks. Help Alice by completing the following tasks. You need to edit and submit the file `ex01-library.cpp`.

This exercise continues on the next page...

(a) Implement the function:

```
Square** createCavern(unsigned int m, unsigned int n)
```

It must return an array of `m` × `n` `Tile`s, i.e., `Tile**`. It must allocate the required memory, and initialise each tile with `nothing` as `content`, and `explored` set to `false`.

(b) Implement the function:

```
void revealCavern(Tile **cav, unsigned int m, unsigned int n)}
```

The function must print on screen the contents of all the tiles of the cavern `cav` of size `m` × `n`, according to the explanation at the beginning of this exercise: see page 2 and **Fig. 1**. Spaces must only be used to represent empty tiles.

Hint. This function just prints all the tiles contents *without* considering the player position and light. Still, some operations are similar to Task **(d)** below: with a bit of planning, you can avoid code duplication. . .

(c) Implement the function:

```
bool movePlayer(Tile **cav, int m, int n, int r, int c)
```

where `cav` is a cavern of size `m` × `n`, and `r` and `c` are a row and column position.

The function tries to change the player position to (`r`, `c`) in the cavern `cav`. The function must find the position of the player in `cav`, and check whether:

- the target position (`r`,`c`) is within the boundaries of the cavern;
- the distance between the player position and coordinate (`r`,`c`) is *less* than 5;
- the tile at position (`r`,`c`) is empty, or already contains the player.

If any of these conditions is *not* satisfied, the function must return `false` without altering the cavern. If all conditions are satisfied, the function must update the cavern `cav` and return `true`: the old player tile becomes `empty`, and the tile at position (`r`, `c`) contains the player. You can assume that there is one player in the cavern.

Example. Assume that `cav` is the cavern shown on page 2, **Fig. 1**: it has size 6×16 and the player is at position (4, 7).

- `movePlayer(cav, 6, 16, 6, 7)` must return `false` (out of boundary);
- `movePlayer(cav, 6, 16, 0, 0)` must return `false` (destination too far);
- `movePlayer(cav, 6, 16, 4, 7)` must return `true` (after updating the cavern);
- `movePlayer(cav, 6, 16, 3, 7)` must return `false` (destination tile not empty).

Hints. Remember that the distance between two points at coordinates (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. A similar operation is also needed in Task **(d)** below: with a bit of planning, you can avoid code duplication. . .

This exercise continues on the next page. . .

(d) Implement the function:

```
void drawCavern(Tile **cav, unsigned int m, unsigned int n)
```

The function must print on screen the cavern `cav` of size `m` × `n` taking into account the position of the player and the torch light, according to the explanation at the beginning of this exercise: see page 2 and **Fig. 2**.

More in detail, the function must find the position of the player in the cavern `cav`, and then print the tiles on screen. When printing a tile on screen, the function must check its distance from the player's position:

- if the distance between the player position and the tile position is *less* than 4, then the tile is lit by the player's torch, hence the function must:
  - print the tile content as in Task **(b)** above, and
  - mark the tile as "explored," by setting its `explored` field to `true`;
- otherwise, the tile is *not* lit by the player's torch. Therefore, the function must:
  - if the tile has *not* been already explored, print it as '`?`';
  - if the tile has been already explored, print it as '`#`' if it contains a rock, or '`-`' otherwise.

Spaces must only be used to represent empty tiles.

You can assume that there is exactly one player in the cavern.

Hint: this function must perform several checks that are also needed in Tasks **(b)** and **(c)** above (see also their hints). With a bit of planning, you can avoid code duplication...

File ex01-main.cpp

```
#include <iostream>
#include "ex01-library.h"
using namespace std;

int main() {
    unsigned int rows = 6, cols = 16;
    Tile **cav = createCavern(rows, cols);
    setupCavern(cav, rows, cols, 5);

    cout << "The␣cavern␣is:" << endl;
    drawCavern(cav, rows, cols);

    cout << endl << "Moving␣to␣row␣3,␣column␣0..." << endl;
    if (!movePlayer(cav, rows, cols, 3, 0)) {
        cout << "Cannot␣move␣to␣row␣3,␣column␣0!" << endl;
    }
    drawCavern(cav, rows, cols);

    cout << endl << "Moving␣to␣row␣3,␣column␣7..." << endl;
    if (!movePlayer(cav, rows, cols, 3, 7)) {
        cout << "Cannot␣move␣to␣row␣3,␣column␣7!" << endl;
    }
    drawCavern(cav, rows, cols);

    cout << endl << "Moving␣to␣row␣2,␣column␣4..." << endl;
    if (!movePlayer(cav, rows, cols, 2, 4)) {
        cout << "Cannot␣move␣to␣row␣2,␣column␣4!" << endl;
    }
    drawCavern(cav, rows, cols);

    cout << endl << "Moving␣to␣row␣4,␣column␣7..." << endl;
    if (!movePlayer(cav, rows, cols, 4, 7)) {
        cout << "Cannot␣move␣to␣row␣4,␣column␣7!" << endl;
    }
    drawCavern(cav, rows, cols);

    cout << endl << "Revealed␣cavern:" << endl;
    revealCavern(cav, rows, cols);

    deleteCavern(cav, rows);
    return 0;
}
```

File ex01-library.h

```
#ifndef EX01_LIBRARY_H_
#define EX01_LIBRARY_H_

enum Content { nothing, player, rock, wumpus };

struct Tile {
    Content content;
    bool explored;
};

Tile **createCavern(unsigned int m, unsigned int n);
void revealCavern(Tile **cav, unsigned int m, unsigned int n);
bool movePlayer(Tile **cav, int m, int n, int r, int c);
void drawCavern(Tile **cav, unsigned int m, unsigned int n);

void setupCavern(Tile **cav, unsigned int m, unsigned int n,
                 unsigned int seed);
void deleteCavern(Tile **cav, unsigned int m);

#endif /* EX01_LIBRARY_H_ */
```

File ex01-library.cpp

```
#include <iostream>
#include <random>
#include "ex01-library.h"

using namespace std;

// Task 1(a). Implement this function
Tile **createCavern(unsigned int m, unsigned int n) {
    // Replace the following with your code
    return nullptr;
}

// Task 1(b). Implement this function
void revealCavern(Tile **cav, unsigned int m, unsigned int n)
    // Write your code here
}

// Task 1(c). Implement this function
bool movePlayer(Tile **cav, int m, int n, int r, int c) {
    // Replace the following with your code
    return false;
}

// Task 1(d). Implement this function
void drawCavern(Tile **cav, unsigned int m, unsigned int n) {
    // Write your code here
}

// Do not modify the following function.
// This code (that you don't need to read) places the player
// at location (0,0) and pseudo-randomly places some rocks
// and a Wumpus. The pseudo-random placement depends on the
// value of 'seed'.
void setupCavern(Tile **cav, unsigned int m, unsigned int n,
                 unsigned int seed) {
    mt19937 e; // Pseudo-random number generator
    e.seed(seed);

    // 1/3rd of the tiles are rocks
    unsigned int rocks = (m * n) / 3;
    for (unsigned int i = 0; i < rocks; i++) {
        cav[e()%m][e()%n].content = rock;
    }

    // We never place the Wumpus on row 0 or column 0
    unsigned int row = (e() % (m-1)) + 1;
    unsigned int col = (e() % (n-1)) + 1;
    cav[row][col].content = wumpus;

    cav[0][0].content = player;
}

// Do not modify the following function.
void deleteCavern(Tile **c, unsigned int m) {
    for (unsigned int i = 0; i < m; i++) {
        delete[] c[i];
    }
    delete[] c;
}
```

Exercise 2. DTU Handball Tournament

Bob is writing a program to manage the DTU Handball Tournament. Each DTU department can form a team, and play a match against another team; the winner of a match proceeds to the next match. Bob's plan is to represent the DTU Handball Tournament data as a tree, where the leaves are the teams, and each internal node represent a handball match between a "left" and "right" team, and its winner.

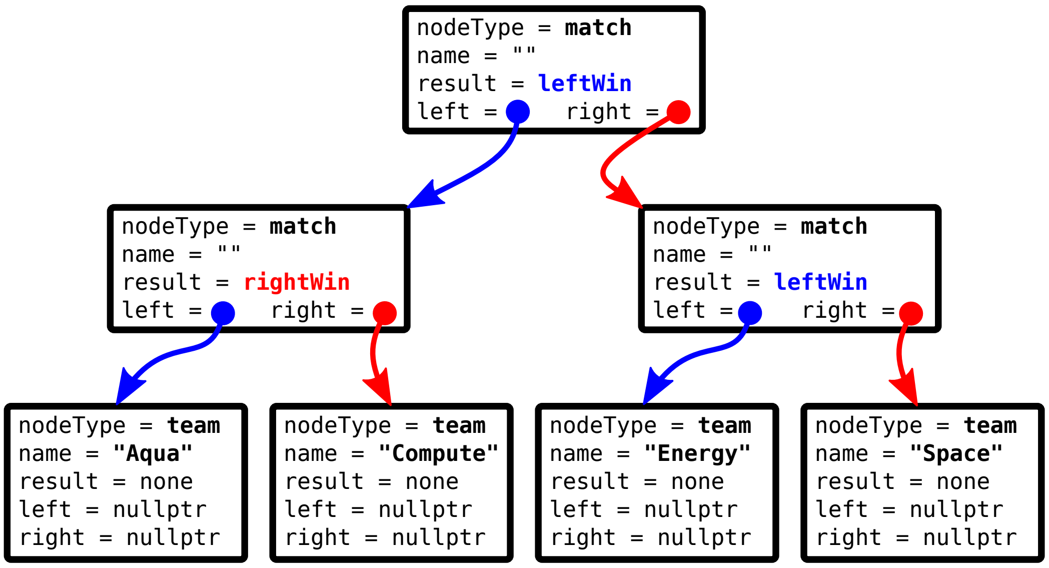
Bob has already written some code. His first test program is in file `ex02-main.cpp` and the (incomplete) code with some functions he needs is in files `ex02-library.h` and `ex02-library.cpp`. Such files are available with this exam paper (in a separate ZIP archive), and they are also reported in the next pages.

Structure of the code. A tournament tree node is represented with a `struct TournamentNode` having 5 fields:

- `nodeType` can be either `team` or `match`;
- `name` *(only meaningful if* `nodeType` *is* `team`*)* is the name of the team;
- `result` *(only meaningful if* `nodeType` *is* `match`*)* can be `leftWin` or `rightWin` or `none`;
- `left` and `right` *(only meaningful if* `nodeType` *is* `match`) point to two subtrees.

An empty tournament is represented as a `TournamentNode*` pointer equal to `nullptr`.

Example. A small tournament with 4 teams is represented as follows:



The leaves contain the 4 teams: Aqua, Compute, Energy, and Space. The node with "`nodeType` = `match`" pointing to Aqua and Compute means: the result of the match was a win for the team on the "right" (i.e. Compute), which proceeds to the next match. Instead, the match between Energy and Space was won by the team on the "left" (i.e. Energy) which proceeds to the next match. The topmost node is the final match between the winners of the two matches below (i.e. Compute on the left, and Energy on the right); the result was a win for the team on the "left," so DTU Compute won the handball tournament.

Tasks. Help Bob by completing the following tasks. You need to edit and submit the file `ex02-library.cpp`. **NOTE:** some tasks may be easier to solve using recursion, but you can use iteration if you prefer.

(a) Implement the function:

```
void displayTeams(TournamentNode *t)
```

which shows the teams participating in the tournament `t`. The function must traverse the tree `t` in-order, from left to right, printing on screen one team name per line.

Special case. If `t` is `nullptr`, the function must not print anything on screen.

Example. Consider the tournament tree on page 6, and assume that `t` points to its topmost node. Then, `displayTeams(t)` must print:

```
Aqua
Compute
Energy
Space
```

(b) Implement the function:

```
unsigned int matches(TournamentNode *t)
```

which returns the number of matches that took place in the tournament tree `t`.

Special cases. If `t` is `nullptr` or `t` points to a team node, then the function must return 0.

Example. Consider the tournament tree on page 6, and assume that `t` points to its topmost node. Then, `matches(t)` must return 3.

(c) Implement the function:

```
string winner(TournamentNode *t)
```

which returns the name of the team that won the tournament `t`.

The function must traverse the match nodes of the tournament tree `t` by descending either on the "left" or "right" (depending on who won the match), until it reaches a team node (which is the winner of `t`).

Special cases. If `t` is `nullptr`, the function must return the empty string `""`.

Example. Consider the tournament tree on page 6, and assume that `t` points to its topmost node. Then, `winner(t)` must return `"Compute"`.

This exercise continues on the next page...

(d) Implement the function:

```
bool wonAnyMatch(TournamentNode *t, string teamName)
```

which returns `true` if the team named `teamName` won *at least* one match in the tournament `t`. Otherwise (i.e. if the team did not participate in the tournament, or lost its very first match and was eliminated) the function returns `false`.

Special cases. If `t` is `nullptr` or `t` points to a team node, then the function must return `false`.

Example. Consider the tournament tree on page 6, and assume that `t` points to its topmost node. Then:

- `wonAnyMatch(t, "Energy")` must return `true`;
- `wonAnyMatch(t, "Space")` must return `false`;
- `wonAnyMatch(t, "Physics")` must return `false`.

Hint. To find out whether a team called `"X"` has won at least one match in a tournament tree `t`, you can traverse the match nodes of `t`, looking for one where the winner between `left` and `right` directly points to a team node for `"X"`. For example, consider the tournament tree on page 6: we can determine that `"Energy"` has won at least one match by traversing the tree until we encounter the match node on the right, whose `left` field (i.e. the winner) directly points to the `"Energy"` team.

File `ex02-library.h`

```cpp
#ifndef EX02_LIBRARY_H_
#define EX02_LIBRARY_H_

#include <string>

enum NodeType { match, team };
enum Result { leftWin, rightWin, none };

struct TournamentNode {
    NodeType nodeType;

    std::string name; // Only used if nodetype == team

    Result result; // Only used if nodetype == match;
    TournamentNode *left; // Only used if nodetype == match;
    TournamentNode *right; // Only used if nodetype == match;
};

void displayTeams(TournamentNode *t);
unsigned int matches(TournamentNode *t);
std::string winner(TournamentNode *t);
bool wonAnyMatch(TournamentNode *t, std::string teamName);

#endif /* EX02_LIBRARY_H_ */
```

File `ex02-main.cpp`

```cpp
#include <iostream>
#include "ex02-library.h"
using namespace std;

int main() {
    TournamentNode t0 = {team, "Aqua", none, nullptr, nullptr};
    TournamentNode t1 = {team, "Compute", none, nullptr, nullptr};
    TournamentNode t2 = {team, "Energy", none, nullptr, nullptr};
    TournamentNode t3 = {team, "Space", none, nullptr, nullptr};

    // Initial matches
    TournamentNode m0 = {match, "", rightWin, &t0, &t1};
    TournamentNode m1 = {match, "", leftWin, &t2, &t3};

    // Final match
    TournamentNode t = {match, "", leftWin, &m0, &m1};

    cout << "The␣teams␣in␣the␣tournament␣are:␣" << endl;
    displayTeams(&t);

    cout << endl << "The␣number␣of␣matches␣is:␣" << matches(&t) << endl;

    cout << "The␣winner␣of␣the␣tournament␣is:␣" << winner(&t) << endl;

    cout << "Did␣DTU␣Energy␣win␣any␣match?␣";
    if (wonAnyMatch(&t, "Energy")) { cout << "Yes!" << endl; }
    else { cout << "No!" << endl; }

    cout << "Did␣DTU␣Space␣win␣any␣match?␣";
    if (wonAnyMatch(&t, "Space")) { cout << "Yes!" << endl; }
    else { cout << "No!" << endl; }

    cout << "Did␣DTU␣Physics␣win␣any␣match?␣";
        if (wonAnyMatch(&t, "Physics")) { cout << "Yes!" << endl; }
    else { cout << "No!" << endl; }

    return 0;
}
```

File `ex02-library.cpp`

```cpp
#include <iostream>
#include "ex02-library.h"
using namespace std;

// Task 2(a). Implement this function
void displayTeams(TournamentNode *t) {
    // Write your code here
}

// Task 2(b). Implement this function
unsigned int matches(TournamentNode *t) {
    // Replace the following with your code
    return 0;
}

// Task 2(c). Implement this function
string winner(TournamentNode *t) {
    // Replace the following with your code
    return "";
}

// Task 2(d). Implement this function
bool wonAnyMatch(TournamentNode *t, string teamName) {
    // Replace the following with your code
    return false;
}
```

Exercise 3. Smart Lockers

Claire works in a logistics company, which has recently installed several smart lockers. She is writing a class `Locker` to manage the information about the lockers and the packages they contain. She has already written some code: her first test program is in file `ex03-main.cpp` and the (incomplete) code of the class is in files `ex03-library.h` and `ex03-library.cpp`. Such files are available with this exam paper (in a separate ZIP archive), and they are also reported in the next pages.

Structure of the code. Claire has represented the information about a package using a `struct Package`, with 3 fields:

- `sender`: the name of the sender;
- `recipient`: the name of the recipient;
- `unlockCode`: a numeric code used by the recipient to unlock the locker and retrieve the package.

Claire knows that the `map` and `vector` containers of the C++ standard library provide many functionalities she needs. *(See hints on page 11.)* Therefore, she has decided to use the following internal (`private`) representation for the library:

- `vector<string> lockerIDs` — the identifiers of the installed lockers;
- `map<string,Package> lockerOccupancy` — a mapping from `strings` (locker IDs) to instances of `Package` (info about the package occupying the locker, if any). When a locker ID does not appear in this mapping, it means that the locker is empty and available.

Claire has already implemented the default constructor of `Locker`, which creates an internal database with all the lockers installed. She has also implemented the method `display()`, which shows whether a locker is occupied, and what package it contains.

Tasks. Help Claire by completing the following tasks. You need to edit and submit the file `ex03-library.cpp`.

(a) Implement the following method to put a package inside a locker:

```
void Locker::putPackage(string lockerID, string sender, string recipient, int unlockCode)
```

The method must work as follows:

(a) if `lockerID` is *not* in `lockerIDs`, do nothing;

(b) otherwise:

- if the locker `lockerID` is already occupied by some package, do nothing;
- if the locker `lockerID` is available, then update the map `lockerOccupancy` to assign the given locker to the given package.

This exercise continues on the next page...

(b) Implement the following method to retrieve a package:

```
void Locker::retrievePackage(string lockerID, int unlockCode)
```

The method must check whether `lockerID` is occupied by a package with the given `unlockCode`; if so, remove the occupancy (so the locker is available again); otherwise, do nothing.

(c) Implement the method:

```
void Locker::findPackagesByRecipient(vector<string> recipients)
```

This method displays the ID(s) of the locker(s) with a package whose recipient is contained in the given collection `recipients`.

The locker IDs must be displayed one-per-line, by following their order in `lockerIDs`.

For example, suppose that we have a vector `v` containing the strings `"Alice"` and `"Bob"`. Then, `hotel.findPackagesByRecipient(v)` will display the IDs of all lockers whose package has recipient equal to either `"Alice"` or `"Bob"`.

Hints on using `maps` and `vectors` (See also: `https://en.cppreference.com/w/cpp/container/map` and `https://en.cppreference.com/w/cpp/container/vector`)

- To remove an element from a map or a vector, you can use their `erase(...)` methods:
  - `https://en.cppreference.com/w/cpp/container/map/erase`
  - `https://en.cppreference.com/w/cpp/container/vector/erase`
- A key `k` in a map `m` can be mapped to `v` with: `m[k] = v;` with this operation, the entry for `k` in `m` is created (if not already present) or updated (if already present).
- To check if key `k` is present in map `m`, you can check: `m.find(k) != m.end()`.
- The value mapped to a key `k` in a map `m` is obtained with: `m[k]`:
- To loop on all (key, value) pairs in a map `m`, you can use: `for (auto p: m) { ... }`. The loop variable `p` is a `pair` with the map key as `p.first`, and the corresponding value as `p.second` (see `https://en.cppreference.com/w/cpp/utility/pair`).

File `ex03-main.cpp`

```
#include <iostream>
#include "ex03-library.h"
using namespace std;

int main() {
    Locker locker = Locker();

    cout << "Initial␣locker␣occupancy:" << endl;
    locker.display();

    locker.putPackage("LYNGBY04", "Alice", "Bob", 951);
    cout << endl << "After␣putting␣a␣package␣from␣Alice␣to␣Bob␣inside␣an␣empty␣locker:" << endl;
    locker.display();

    locker.retrievePackage("LYNGBY01", 1234);
    cout << endl << "After␣retrieving␣a␣package:" << endl;
    locker.display();

    cout << endl << "Package(s)␣addressed␣to␣Bob␣or␣Daisy:" << endl;
    vector<string> v;
    v.push_back("Bob");
    v.push_back("Daisy");
    locker.findPackagesByRecipient(v);

    return 0;
}
```

File `ex03-library.h`

```
#ifndef EX03_LIBRARY_H_
#define EX03_LIBRARY_H_

#include <string>
#include <vector>
#include <map>
using namespace std;

struct Package {
    string sender;
    string recipient;
    int unlockCode;
};

class Locker {
private:
    vector<string> lockerIDs;
    map<string,Package> lockerOccupancy;
public:
    Locker();
    void putPackage(string lockerID, string sender, string recipient, int unlockCode);
    void retrievePackage(string lockerID, int unlockCode);
    void findPackagesByRecipient(vector<string> recipients);
    void display();
};

#endif /* EX03_LIBRARY_H_ */
```

File `ex03-library.cpp`

```cpp
#include <iostream>
#include "ex03-library.h"
using namespace std;

// Do not modify
Locker::Locker() {
    this->lockerIDs.push_back("LYNGBY01");
    this->lockerOccupancy["LYNGBY01"] = {"Alice", "Daisy", 1234};

    this->lockerIDs.push_back("LYNGBY02");
    this->lockerOccupancy["LYNGBY02"] = {"Claire", "Alice", 567};

    this->lockerIDs.push_back("LYNGBY03");
    this->lockerOccupancy["LYNGBY03"] = {"Daisy", "Bob", 890};

    this->lockerIDs.push_back("LYNGBY04");

    this->lockerIDs.push_back("LYNGBY05");
    this->lockerOccupancy["LYNGBY05"] = {"Bob", "Daisy", 159};

    this->lockerIDs.push_back("LYNGBY06");
}

// Task 3(a). Implement this method
void Locker::putPackage(string lockerID, string sender, string recipient, int unlockCode) {
    // Write your code here
}

// Task 3(b). Implement this method
void Locker::retrievePackage(string lockerID, int unlockCode) {
    // Write your code here
}

// Task 3(c). Implement this method
void Locker::findPackagesByRecipient(vector<string> recipients) {
    // Write your code here
}

// Do not modify
void Locker::display() {
    for (auto it = this->lockerIDs.begin(); it != this->lockerIDs.end(); it++) {
        cout << "Locker␣'" << *it << "'␣";
        if (this->lockerOccupancy.find(*it) == this->lockerOccupancy.end()) {
            cout << "is␣empty" << endl;
        } else {
            cout << "contains␣a␣package␣from␣" << this->lockerOccupancy[*it].sender;
            cout << "␣to␣" << this->lockerOccupancy[*it].recipient;
            cout << "␣(unlock␣code:␣" << this->lockerOccupancy[*it].unlockCode << ")" << endl;
        }
    }
}
```

Exercise 4. Counting Sensor Data Buffer

Daisy is writing a program that reads integer values from a sensor.

Her program may need to read either the most frequent value obtained from the sensor, or the frequency (i.e., the number of occurrences being observed by the sensor) of a specific value. Therefore, she plans a `CountingBuffer` class with the following interface:

- `write(v)`: appends value `v` (obtained from the sensor) into the buffer;
- `frequency(v)`: returns the frequency of value `v`;
- `mostFrequent()`: returns the value with highest frequency. If two or more values meet this condition, it returns the newest value. If no value is in the buffer, it returns the default value specified in the `CountingBuffer` constructor (see below).
- `clear()`: empties the buffer, and resets the frequency counters to 0.

Daisy's first test program is in the file `ex04-main.cpp` and the (incomplete) code of the class is in files `ex04-library.h` and `ex04-library.cpp`. Such files are available with this exam paper (in a separate ZIP archive), and they are also reported in the next pages.

Structure of the code. Daisy has defined a high-level abstract class `Buffer` with the pure virtual methods `write(v)` and `clear()`. She wants to implement `CountingBuffer` as a subclass of `Buffer`, with the additional methods `frequency(v)` and `mostFrequent()`.

Example. Once completed, the class `CountingBuffer` must work as follows:

- suppose that we create `buf = CountingBuffer(-1)` (i.e. `buf` has a default value `-1`);
- suppose that `buf.write(9)` is invoked, followed by `buf.write(7)`. Then, a call to `buf.mostFrequent()` must return `7`. Also, calls to `buf.frequency(9)` and `buf.frequency(7)` must return `1`;
- then, suppose that `buf.write(3)` is invoked, followed by `buf.write(9)`. Since the frequency of `9` is now greater than the frequency of `7`, a call to `buf.mostFrequent()` must now return `9`. Also, a call to `buf.frequency(9)` must now return `2`;
- finally, suppose that `buf.clear()` is invoked. Then, `buf.mostFrequent()` must now return the default value `-1`, and `buf.frequency(9)` must now return `0`.

Tasks. Help Daisy by completing the following tasks. You need to edit and **submit two files**: `ex04-library.h` and `ex04-library.cpp`.

NOTE: you are free to define the `private` members of `CountingBuffer` however you see fit. For instance, you may choose to use a `map<int,int>` to keep track of the frequencies of values written in the buffer. *(See hints on page 11.)* Also, you are free to (internally) compute the frequencies and the most frequent value either when new values are appended into the buffer, or when the `frequency(v)` and `mostFrequent()` methods are invoked. The tests will only consider the behaviour of the public methods `write()`, `frequency(v)`, `mostFrequent()`, and `clear()`.

This exercise continues on the next page...

(a) Declare in `ex04-library.h` and sketch in `ex04-library.cpp` a class `CountingBuffer` that extends `Buffer`. This task is completed (and passes CodeJudge tests) when `ex04-main.cpp` compiles without errors. To achieve this, you will need to:

1. define a `CountingBuffer` constructor that takes one parameter: an `int` value representing the default value (returned by `mostFrequent()` when the buffer is empty);
2. in `CountingBuffer`, override the *pure virtual methods* of `Buffer` (i.e., those with "`=0`"), and add the following `public` methods to the class interface:
   - `unsigned int frequency(int v)`
   - `int mostFrequent()`
3. finally, write a placeholder implementation of all the `CountingBuffer` methods above (e.g. they may do nothing and/or just return `0` when invoked).

(b) This is a follow-up to task **(a)** above. In `ex04-library.cpp`, write a working implementation of the methods:

```
void CountingBuffer::write(int v)
unsigned int CountingBuffer::frequency(int v)
```

The intended behaviour of `write(v)` is to store value `v` (so it can be used by `frequency(v)` and `mostFrequent()`). The method `frequency(v)` returns the frequency of value `v` (i.e., the number of times v was stored in the buffer).

(c) This is a follow-up to tasks **(a)** and **(b)** above. In `ex04-library.cpp`, write a working implementation of the method:

```
int CountingBuffer::mostFrequent()
```

When invoked, `mostFrequent()` returns the value with the highest frequency. If two (or more) values meet this condition, then the method returns the newest (i.e. the last one inserted using the method `write(...)` above).

Special case: if the buffer is empty, then `mostFrequent()` must return the default value specified in the `CountingBuffer` constructor.

This exercise continues on the next page...

(d) This is a follow-up to task **(a)**, **(b)**, and **(c)** above. In `ex04-library.cpp`, write a working implementation of the method:

```
void CountingBuffer::clear()
```

When invoked, `clear()` empties the buffer and resets the frequency counters. Consequently, invoking `clear()` and then `mostFrequent()` returns the default buffer value (specified in the `CountingBuffer` constructor); also, after invoking `clear()`, `frequency(v)` returns 0 for any value `v`, until `write(v)` is invoked.

File `ex04-main.cpp`

```
#include <iostream>
#include "ex04-library.h"
using namespace std;

int main() {
    CountingBuffer *cb = new CountingBuffer(-1);
    Buffer *b = cb; // Just an alias for 'cb' above, but using the superclass

    cout << "Most␣frequent␣value:␣" << cb->mostFrequent() << endl;
    cout << "Finding␣the␣frequency␣for␣value␣0␣returns:␣" << cb->frequency(0) << endl;

    b->write(9); b->write(7);
    cout << "Wrote␣9␣and␣7.␣Most␣frequent␣value:␣" << cb->mostFrequent() << endl;
    cout << "Finding␣the␣frequency␣for␣value␣9␣returns:␣" << cb->frequency(7) << endl;
    cout << "Finding␣the␣frequency␣for␣value␣7␣returns:␣" << cb->frequency(9) << endl;

    b->write(3); b->write(9);
    cout << "Wrote␣3␣and␣9.␣Most␣frequent␣value:␣" << cb->mostFrequent() << endl;
    cout << "Finding␣the␣frequency␣for␣value␣3␣returns:␣" << cb->frequency(3) << endl;
    cout << "Finding␣the␣frequency␣for␣value␣9␣returns:␣" << cb->frequency(9) << endl;

    b->clear();
    cout << "Buffer␣cleared.␣Most␣frequent␣value:␣" << cb->mostFrequent() << endl;
    cout << "Finding␣the␣frequency␣for␣value␣9␣now␣returns:␣" << cb->frequency(9) << endl;

    delete cb;
    return 0;
}
```

File `ex04-library.h`

```
#ifndef EX04_LIBRARY_H_
#define EX04_LIBRARY_H_

class Buffer {
public:
    virtual void write(int v) = 0;
    virtual void clear() = 0;
    virtual ~Buffer();
};

// Task 4(a). Declare the class CountingBuffer, by extending Buffer
// Write your code here

#endif /* EX04_LIBRARY_H_ */
```

File `ex04-library.cpp`

```
#include "ex04-library.h"

// Task 4(a). Write a placeholder implementation of CountingBuffer's
// constructor and methods

// Task 4(b). Write a working implementation of write() and frequency()

// Task 4(c). Write a working implementation of mostFrequent()

// Task 4(d). Write a working implementation of clear()

// Do not modify
Buffer::~Buffer() {
    // Empty destructor
}
```