Master of Data Science Online Programme
Course: C++
Employee Database console app: Release Notes

Student: Andrei Batyrov (Fall 2022)
`arbatyrov@edu.hse.ru`

National Research University
Higher School of Economics (HSE)
Faculty of Computer Science

March 8, 2024

Contents

List of Figures

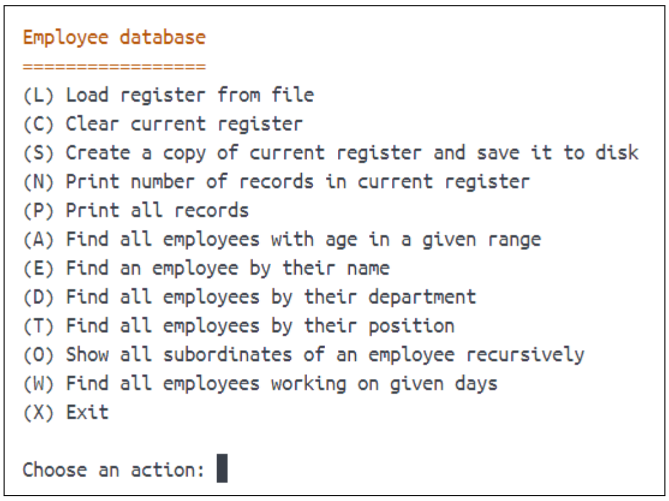
1 Description

The application allows to load employee records from a `csv` text file (deserialize) into a register stored in the computer memory. The register can be used to query records based on various filtering criteria and show the query results on the screen. The application can also create a copy of the register and save records from it to a `csv` text file (serialize).

2 Usage Instructions

2.1 Compile the application with `cmake`. Alternatively, you can compile the program by running `build.bat` (Windows) or `./build.sh` (Linux) without using `cmake`. These shell scripts use `g++` compiler.

2.2 Start the application by running the `main.exe` (Windows) or `./main` (Linux) file. The main menu will be displayed – see Figure 1 – inviting the user to choose an option.



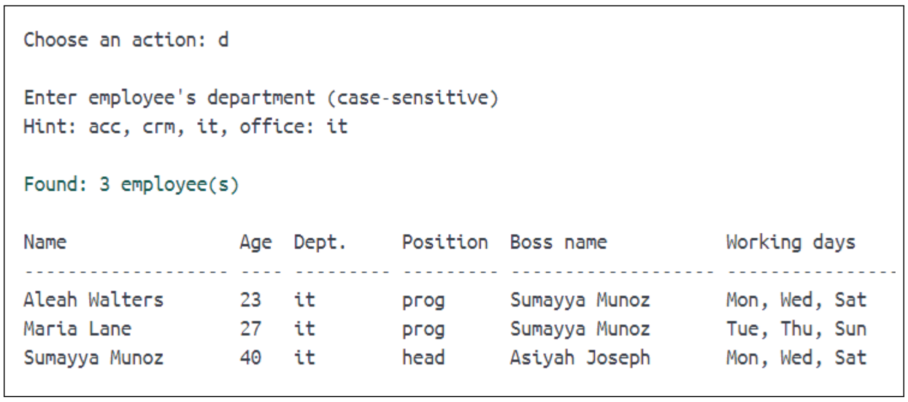
```
Employee database
=================
(L) Load register from file
(C) Clear current register
(S) Create a copy of current register and save it to disk
(N) Print number of records in current register
(P) Print all records
(A) Find all employees with age in a given range
(E) Find an employee by their name
(D) Find all employees by their department
(T) Find all employees by their position
(O) Show all subordinates of an employee recursively
(W) Find all employees working on given days
(X) Exit

Choose an action:
```

Fig. 1. Main menu

2.3 Choose `L` to load a `csv` text file. You can use `employees.csv` or your file.

2.4 After loading a file, you can proceed to querying the register. See Figure 2 for a sample result after querying for a specific department.



```
Choose an action: d

Enter employee's department (case-sensitive)
Hint: acc, crm, it, office: it

Found: 3 employee(s)

Name                 Age  Dept.     Position  Boss name            Working days
-------------------- ---- --------- --------- -------------------- ----------------
Aleah Walters        23   it        prog      Sumayya Munoz        Mon, Wed, Sat
Maria Lane           27   it        prog      Sumayya Munoz        Tue, Thu, Sun
Sumayya Munoz        40   it        head      Asiyah Joseph        Mon, Wed, Sat
```

Fig. 2. Sample query result

2.5 When finished, choose `X` to exit the application.

3 Implementation Notes

3.1 The source code is located in the `src` directory and is organized in 4 files:

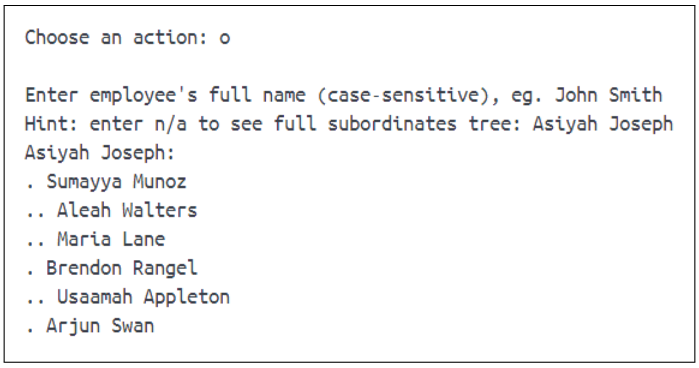
- `main.cpp`: handles user interface, query, load, and save operations;
- `printers.cpp`: handles printing query results operations;
- `record.cpp`: implements `Record` class;
- `register.cpp`: implements `Register` class;

The header files with classes and free functions declarations are located in the `include` directory: `printers.hpp`, `record.hpp`, `register.hpp`.

3.2 An object of the `Record` class is created by passing employee record fields to its constructor, i.e. deserialization is done in `main.cpp` before creating an employee record object. This was done to make both `Record` and `Register` classes more universal without including the parsing mechanics into them. These two classes are more focused on business logic rather than on loading, saving, and printing. All `Record` class attributes are private and there are no friend functions. To access the members, a number of getters is implemented, such as `getAge()`.

3.3 The main storage of the `Register` class is organized with a `std::vector` of pointers to constant employee record objects. When populating a register the employee record object are manually allocated on the heap. When cleaning the register by the user command or exiting the application, the record objects are manually deallocated from the heap. The `const` qualifier is used to make employee record objects immutable, since the program is not intended to change loaded employee records. Again, to control the integrity of the employee records, all `Register` class getters, including index getters return immutable objects or references, for example `const Register::EmpVec& Register::getStorage() const` which returns a read-only reference to the vector of pointers to read-only employee records.

3.4 For showing employee subordination, instead of returning a flat list of all direct and indirect subordinates, a tree is printed, since the graph data structure looks to be more suitable in this case – see Figure 3.


```
Choose an action: o

Enter employee's full name (case-sensitive), eg. John Smith
Hint: enter n/a to see full subordinates tree: Asiyah Joseph
Asiyah Joseph:
. Sumayya Munoz
.. Aleah Walters
.. Maria Lane
. Brendon Rangel
.. Usaamah Appleton
. Arjun Swan
```

Fig. 3. Subordination tree for a sample employee

3.5 In compliance with the RAII idiom [3], both `Record` and `Register` classes implement copy constructors, copy assignment operator overloads, and destructors.

4 Time Complexity

4.1 Adding a pointer to the register vector takes one `push_back` with $O(1)$ [1] and 4 index (`std::map`) inserts, each with $O(\log N)$, totaling $\sum O(\log N)$.

4.2 The time complexity of index getters is bounded by $O(\log N)$.

4.3 The getter of a set of employees working on given days has the worst time complexity $O(N \log N)$. That's because we need to iterate over every employee and check the intersection of their working days and given working days provided by the user. We use `std::set` (BST) to collect pointers to employee records that satisfy the working days condition, which has insert time complexity $O(\log N)$, so the total time complexity is $O(N \log N)$. We also might use `std::unordered_set` (hash map) with best insert time complexity $O(1)$, but its worst insert time complexity is $O(N)$ as opposed to the worst insert time complexity $O(\log N)$ for `std::set`. So, instead of $O(N \log N)$ we might end up with $O(N^2)$. That's why we eventually chose `std::set`.

4.4 The time complexity of a getter for age range is $O(N)$. That's because we need to iterate over every employee to check their age and then push the pointers to those employee records that satisfy the age condition, that is we create a new vector of filtered pointers. Vector `push_back` operation is $O(1)$, so the total time complexity is $O(N)$.

5 Memory Leaks Testing

The application was tested on a file with size around 300 MB having over 5,000,000 employee records, similar to those in the assignment: `Adam·Smith→50→finan→→Mon→Tue`. Conducted tests confirmed correct heap memory allocation and deallocation by monitoring the memory resources occupied by the running application. On Windows, memory was monitored in Task Manager – see Figure 4. On Linux, `valgrind` command [2] was used – see Figure 5.

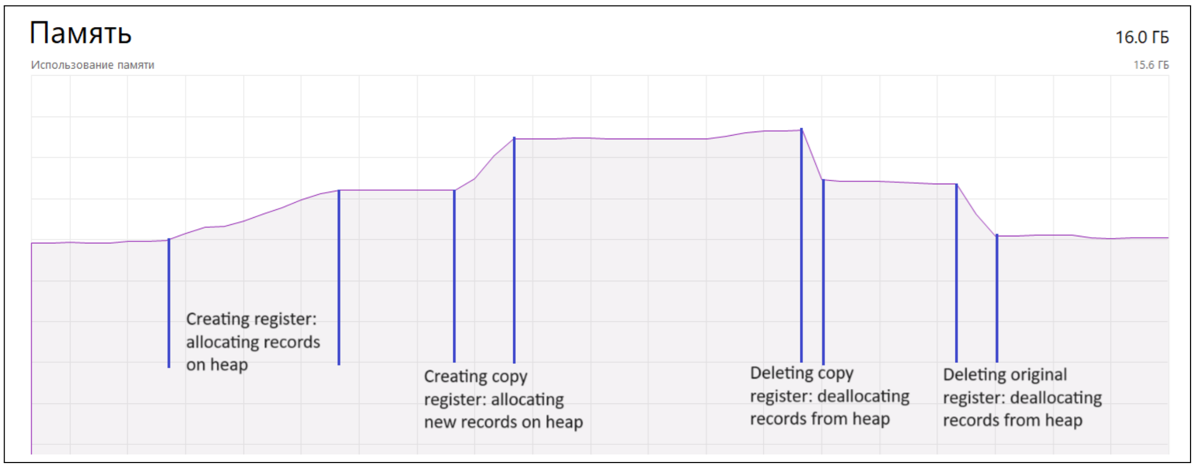
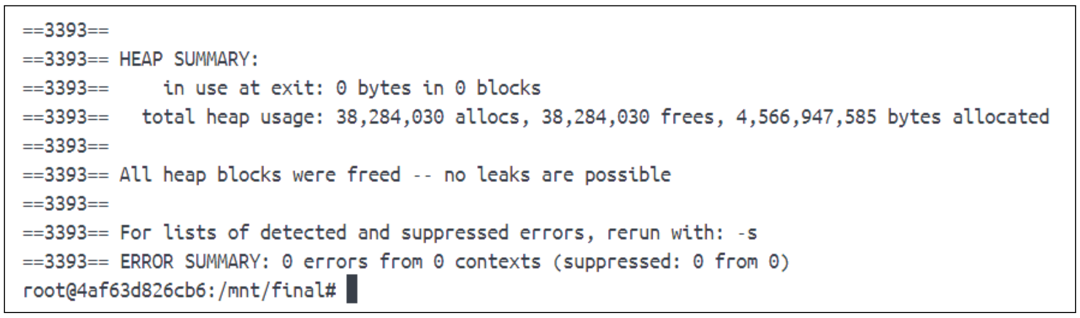

Память
Использование памяти
16.0 ГБ
15.6 ГБ
Creating register: allocating records on heap
Creating copy register: allocating new records on heap
Deleting copy register: deallocating records from heap
Deleting original register: deallocating records from heap

Fig. 4. Windows memory usage



```
==3393==
==3393== HEAP SUMMARY:
==3393==     in use at exit: 0 bytes in 0 blocks
==3393==   total heap usage: 38,284,030 allocs, 38,284,030 frees, 4,566,947,585 bytes allocated
==3393==
==3393== All heap blocks were freed -- no leaks are possible
==3393==
==3393== For lists of detected and suppressed errors, rerun with: -s
==3393== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@4af63d826cb6:/mnt/final#
```

Fig. 5. Linux valgrind report

References

[1] *C++11 reference*. URL: https://en.cppreference.com/w/cpp/11.

[2] Ilya Kosarev. *C++ webinars and study materials*. Higher School of Economics. URL: https://github.com/PersDep/cpp-basics-for-mds-2024/tree/main.

[3] Sergey Shershakov. *C++ course*. Higher School of Economics. URL: https://edu.hse.ru/course/view.php?id=180241.