Структуре података и алгоритми 2024/25 – Први домаћи задатак

*Никола Марковић, 1230/23*

За израду је одабран језик С++ и уз њега један строго објектно-оријентисан приступ у којем је свака структура имплементирана у облику класе, најчешће као композиција више класа, код чијих инстанци се сви подобјекти уништавају у случају уништења ње саме. Свакако, треба нагласити да се неки од њих, нпр. *CargoStack*, могу инстанцирати ван оквира других објеката како би се користили у сврхе конструктора или додатне логике (нпр. истоварања робе из приколице теретног возила на сто).

Конкретно, у складу са потребама задатка, коришћене су сљедеће структуре података:

* поредак возила на граничном прелазу – **приоритетни ред (PVQueue)** (путничка возила), **ред (CVQueue)** (теретна возила)
* предаја и преглед пасоша за путничка возила – **ред (PassengerQueue)** [није било потребе за *Passport* класом]
* роба у приколици теретних возила – **стек (CargoStack)**

**Сваки ред** је имплементиран користећи једноструко уланчану листу готово идентичних карактеристика, уз мале разлике нпр. методу за испис тренутног поретка теретних возила у реду. Редови се овдје користе због њихове *FIFO* дисциплине приступа која савршено одговара потребама задатка. Приоритетни ред се користи како би се омогућило својеврсно обилажење *FIFO* логике уколико је неки критеријум (попут година старости једног од путника) задовољен.

**Стек** робе је имплементиран користећи динамички низ који карактерише могућност самосталне алокације и реалокације (у оба смјера) меморије у складу са траженом величином. Стек карактерише *LIFO* дисциплина приступа која одговара руковању робом у складу са потребама задатка. Код конкретне имплементације (позиције у приколици), роба поред који стоји највећи индекс нумерације се узима да је најближа вратима (на врху је стека), те се она прва истовара тј. уклања са стека.

Главна функција служи искључиво за примање почетног корисничког уноса, те у складу с њим позива одговарајућу функцију из *Simulation* простора имена. Одавде се код своди на растављање на што је више могуће функција (стандардних и анонимних ламбда) и класа како би се логика већине петљи које великим дијелом сачињавају ове симулације свела на неколико линија које позивају концизно именоване функције простора имена и методе одговарајућих класа. Како није било сигурно у ком смјеру ће тећи даљи рад, на почетку сам одлучио да сваку структуру имплементирам скупом њој-одговарајућих класа умјесто да користим насљеђивање због потенцијалних ограничења у даљем току рада која би захтијевала рефакторизацију.

Меморијска сигурност је добра, у смислу да не може никад доћи до цурења без обзира на кориснички унос, претежно јер се потенцира да се што је више могуће објеката инстанцира на стеку без сувишне динамичке алокације. Оно што се мора инстанцирати у динамичку зону меморије ће бити адекватно деалоцирано, искључиво „низводно“, тј. објекти реда изнад ће правилним коришћењем увијек деалоцирати сваки динамички објекат који их сачињава у облику раније наведених композиција. На примјер, *CVListNode* ће бити алоциран и деалоциран позивањем метода за додавање и брисање чворова њему надлежне *CVLinkedList* класе.

Генерално, програм без проблема јасно и прецизно исписује све што текст задатка од њега тражи уз неке додатке попут навођења имена улазне датотеке са робом теретних возила. Могућност даљег проширења функционалности (ван оквира симулације) је упитна, претежно због меморијске сигурности.