ANTRAS NAMŲ DARBAS

Kontekstas ir problematika

Dauguma taikomųjų programų vienu ar kitu būdu dirba su duomenimis. Kad programa veiktų efektyviai, pasirinktas duomenų saugojimo būdas irgi turi būti efektyvus. To efektyvumo užtikrinimas dažnai reikalauja papildomo darbo, realizuojant vieną ar kitą sprendimą. Siekiant neužkrauti to papildomo darbo ant visų programuotojų pečių, tipiškai stengiamasi duomenų struktūras ir su jomis susijusius algoritmus realizuoti atskirame modulyje, taip, kad jomis būtų galima naudotis nežinant realizacijos detalių.

Užduoties sąlyga

Sukurti efektyvų konteinerio ADT, įgalinantį efektyviai įterpti, gauti, bei trinti jame saugomus duomenis.

Užduoties terminas

Pradinis pateikimas **iki kovo 18** imtinai (bent viena konteinerio realizacija (nebūtinai su rodyklėmis), gali būti konkrečiam elementų tipui (pvz. int)), tobulinimas iki semestro galo. Pradinis variantas atsiunčiamas el.paštu irmantas.radavicius@mif.vu.lt su laiško tema „INFO?_Vardas_Pavarde_ND2“. Pradiniame variante ADT projekto apiforminimui reikalavimų nėra, galutiniame etape reikės pateikti pagal (griežtas) instrukcijas, įgalinančias dalinai automatizuotą patikrinimą (bus paskelbta vėliau); tai gali reikalauti nežymių kodo modifikacijų (pvz. funkcijų pervadinimo, etc).

Užduoties reikalavimai

1) sukurti konteinerio ADT ir jam taikomų bazinių operacijų **aprašus**. Rašant kodą C kalba, pateikiamas naujo tipo apibrėžimas (typedef) ir funkcijų prototipų rinkinys; rašant kodą C++ kalba, pateikiama klasė su atitinkamais metodais.

   Sukurtam ADT **privaloma** aprašyti:
   a) duomenų elemento įterpimo operaciją(-as), pvz. *add(pos, val), addFirst(val), addLast(val), add(val)*
   b) duomenų elemento ištrynimo operaciją(-as), pvz. *remove(pos), removeFirst(), removeLast(), remove(val)*
   c) duomenų elemento gavimo operaciją(-as), pvz. *get(pos), getFirst(), getLast(), get(val)*
   d) sąrašo dydžio patikrinimo operaciją(as), pvz. *getSize(), getMaxSize()*

   Galima **papildomai** įgyvendinti:
   a) duomenų elemento redagavimo operaciją(-as), pvz. *edit(pos, val), editFirst(val), editLast(val), edit(old, new)*
   b) perėjimo nuo vieno elemento prie kito operaciją(-as), pvz. *next(current), prev(current)*
   c) kitas operacijas, pvz. *create(), clear(), destroy(),* etc.

   Operacijų pasirinkimas priklauso nuo pasirinktos ADT logikos. Jeigu elemento pozicija laikoma reikšminga (t.y. jei pvz. {1; 2; 3} ir {3; 2; 1} laikomi skirtingais duomenų rinkiniais), tikslinga parūpinti operacijas, reikalaujančias modifikuojamų/gaunamų duomenų elemento pozicijos (pvz. *add(pos,val)*). Jeigu elemento pozicija laikoma nereikšminga, tokiu atveju operacijos tos pozicijos kaip parametro gali ir nereikalauti (pvz. *add(val)*). Jeigu duomenų struktūra efektyviai dirba su pradžioje arba pabaigoje esančiais elementais, tikslinga parūpinti operacijas skirtas būtent tik pradžiai (pvz. *addFirst(val)*) ar pabaigai (pvz. *addLast(val)*). Jeigu programuojama objektiškai, duomenų struktūros sukūrimas ir sunaikinimas įgyvendinamas per konstruktorius ir destruktorius.

2) sukurti bent dvi (skirtingas) efektyvias šio ADT **realizacijas**, atsižvelgiant į a) naudojamą **atminties** kiekį ir b) **greitaveiką**. Viena iš realizacijų **privalo** būti paremta rodyklėmis, t.y. duomenys negali būti išdėstomi nuosekliame atminties bloke (masyve arba jo analoge). Bendru atveju, realizacija gali būti bet kokia (sąrašas, medis, etc), tol kol realizacijos detalės nepradeda atsispindėti ADT apraše. Realizacija turi būti įgyvendinta taip, kad ADT viduje būtų galima naudoti **bet kokio tipo** elementus (naudojant netipizuotas rodykles (C) arba šablonus (C++)). Realizuojant, standartiniais C/C++ konteineriais (*vector*, *list*, etc) naudotis negalima.

3) būtina **apsaugoti** ADT nuo nekorektiško panaudojimo. Programuojant su C, klaidų kodams saugoti naudojamas specialus (globalus arba struktūros viduje esantis) kintamasis, kurio reikšmė 0 jei ką tik vykdyta operacija buvo sėkminga; kitu atveju išsaugomas klaidos kodas. Programuojant su C++, nesėkmingų operacijų rezultatas turi būti atitinkamas išimties (angl. exception) objektas.

Užduoties rezultatai

Kiekvienas ADT variantas realizuojamas atskirame modulyje, vertinimui pateikiamas jo antraštės failas (.h), realizacijos failas (.cpp) bei testavimui skirtas kodo failas (.cpp). Idealiu atveju, visos realizacijos turi naudoti tą patį antraštės failą (naudojant *pointer-to-implementation* idėją (C++); kas tai yra ir kaip tai galima įgyvendinti žr. pavyzdžiui https://en.wikibooks.org/wiki/C%2B%2B_Programming/Idioms#Pointer_To_Implementation_.28pImpl.29). Kitu, įprastu atveju – turi (idėjiškai) atitikti pateikiamų operacijų prototipai, kitaip tariant, turint fiksuotą testavimui skirtą *main* funkciją, vienos naudojamos ADT realizacijos pakeitimas į kitą turi reikalauti minimalių pakeitimų, o geriausia – visai nereikalauti.

Užduoties vertinimas

Savarankiškai ir sėkmingai atilkta užduotis vertinama iki 0,5 balo prie galutinio pažymio. Už efektyvią ADT realizaciją gali būti skiriami papildomai balai.