Algoritmid ja andmestruktuurid (IAS0090)

Laboratoorsete tööde juhend

2023

Sisukord

Sissejuhatus

Laboratoorseks tööks saab iga üliõpilane eritüübilisi objekte sisaldava ning viitadega ühendatud andmestruktuuri skeemi, *.h failid kõigi vajalike deklaratsioonidega ja *.obj failid lähtestruktuuri genereerivate funktsioonidega. Üliõpilase ülesandeks on kirjutada 6 etteantud prototüübiga C funktsiooni, mis võimaldavad andmete otsimist ja lähtestruktuuri modifitseerimist. Töövahendiks on Microsoft Visual Studio alates 2017 aasta väljalaskest[1].

Lähtestruktuure on kokku 7 erinevat varianti. Struktuuris paiknevaid objektid võivad olla 10 erinevast tüübist. Variandi ja objekti tüübi määrab juhendaja igale üliõpilasele eraldi..

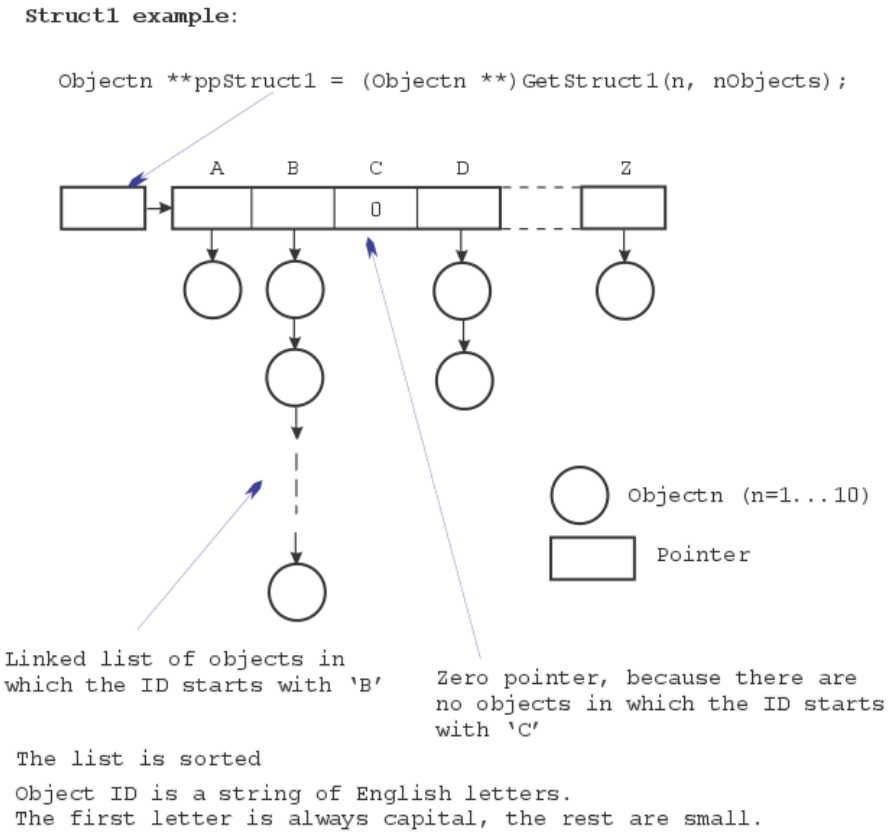
Käesolev juhend formuleerib täpselt laboratoorse töö ülesanded ja selgitab, kuidas Visual Studio keskkonnas projektiga alustada. Esitatud on ka kaitsmise kord.

[1] Tasulise Microsoft Visual Studio asemel võib kasutada vabavara Visual Studio Community, vt. https://visualstudio.microsoft.com/ . Kui Teie arvuti operatsioonisüsteemiks ei ole Windows (7 või 10), siis lugege esmalt *Lisa 2: isikliku arvuti kasutamise* kord.

1. Variandid 1A ja 1B

1.1. Lähteandmed

Lähtestruktuur koosneb viitade vektorist ja ahelloenditesse ühendatud objektidest. Objektid võivad olla tüüpidest *Object1*, *Object2*,…*Object10*. Objekti tüübi annab ette juhendaja. Objektide kui C *struct*-ide deklaratsioonid on esitatud failis *Objects.h* ja neid selgitavad skeemid asuvad käesoleva juhendi lisas. Lähtestruktuuri näide (rõhutame, et tegemist on üksnes selgitava näitega) on toodud alljärgneval joonisel:



Objekti identifikaatoriks on kõikidel juhtudel C string, mis koosneb inglise tähestiku väiketähtedest. Stringi esimene sümbol on aga inglise tähestiku suurtäht. Stringi pikkuseks võib olla mistahes nullist suurem arv.

Viitade vektori positsioonist 0 lähtuv ahelloend koosneb objektidest, millede identifikaator algab tähega ‘A’, positsioonilt 1 lähtuvas ahelloendis on objektid, millede identifikaator algab tähega ‘B’ jne. Ahelloend on identifikaatori järgi sorditud (s.t. kõige väiksema identifikaatoriga objekt on kõige esimene). Kui mingi algustähega seotud objekte ei ole, on viitade vektoris vastaval positsioonil null.

Lähtestruktuuri genereerib juhendaja poolt antud funktsioon *GetStruct1* (asetseb objektmoodulis *Structs.obj*, prototüüp on failis *Structs.h*).

1.2. Ülesanne

Laboratoorse töö ülesandeks on kirjutada 6 järgmist funktsiooni:

1.2.1. Esimene funktsioon

*void PrintObjects(Objectn **ppStruct1);*

Objectn asemel tuleb kirjutada loomulikult *Object1, Object 2,* jne. Sisendparameetriks on viit lähtestruktuuri viitade vektorile. Funktsiooni ülesandeks on väljastada kõikide struktuuris paiknevate objektide kirjeldused (iga objekt eraldi reas). Väljastavaks funktsiooniks on *printf*. Väljastamise formaatimise string (*printf* esimene parameeter) on toodud failis *Objects.h.* Loendur formaatimise stringi alguses peab esimese väljatrükitava objekti puhul olema 1 ja edasi pidevalt kasvama.

See funktsioon tuleb kirjutada kõige esimesena, sest ilma temata ei ole võimalik ülejäänud funktsioonide tööd kontrollida.

1.2.2. Teine funktsioon

*int InsertNewObject(Objectn **ppStruct1, char *pNewID, int NewCode);*

Sisendparameetriteks on viit lähtestruktuuri viitade vektorile, viit uue objekti identifikaatorile ja uue objekti kood. Funktsiooni ülesandeks on luua uus objekt ja lülitada ta andmestruktuuri.

Funktsioon peab esmalt kontrollima, kas uus identifikaator ikka vastab eespool toodud formaadile. Kui see tingimus ei ole täidetud, väljastab funktsioon lihtsalt nulli. Edasi peab funktsioon kontrollima, kas sellise identifikaatoriga objekt ikka tõesti puudub. Kui ta on olemas, väljastab funktsioon samuti lihtsalt nulli. Kui sellise identifikaatoriga objekti ei ole, tuleb ta luua ning lülitada õigesse ahelloendisse. Sealjuures sõltuvalt variandist:

A. Uued objektid võib paigutada suvalisele kohale, s.t. esialgne sorditud järjestus ei pea säiluma.

B. Ka uute objektidega täiendatud ahelloend peab olema sorditud.

Kõik vajalikud mäluväljad tuleb funktsiooni *malloc* abil tellida. Ka identifikaatori jaoks on vaja omaette mäluvälja, kuhu sisendparameetriks olev string tuleb kopeerida. Kellaaeg või kuupäev tuleb lugeda arvuti kellalt ja teisendada (vastavad funktsioonid on objektmoodulis *DateTime.obj*, prototüübid on failis *DateTime.h*). Kui funktsioon täitis oma ülesande, väljastab ta suuruse 1.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Uue objekti identifikaator algab sellise tähega, millele vastavat ahelloendit siiamaani ei olnud.
2. Uue objekti identifikaator algab sellise tähega, millele vastav ahelloend on juba genereeritud. Variandi B puhul lisanduvad siia olukorrad, kus:
    a. Uus objekt tuleb oma ahelloendi esimeseks.
    b. Uus objekt tuleb oma ahelloendi viimaseks.
    c. Uus objekt tuleb kusagile oma ahelloendi keskele.
3. Uue objekti identifikaator on ebaõiges formaadis.
4. Uue objekti identifikaator on sama mis mõnel juba olemasoleval objektil.

1.2.3. Kolmas funktsioon

Objectn RemoveExistingObject(Objectn **ppStruct1, char *pExistingID);*

Sisendparameetriteks on viit lähtestruktuuri viitade vektorile ja viit stringile, mis peab kokku langema ühe eeldatavalt olemasoleva objekti identifikaatoriga. Funktsiooni ülesandeks on leida see objekt ja eemaldada ta andmestruktuurist. Eemaldatud objekti kustutada ei tohi. Väljundsuuruseks on viit eemaldatud objektile. Kui objekti ei leitud, on väljundsuuruseks null.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:
1. Eemaldatav objekt on oma ahelloendis esimene.
2. Eemaldatav objekt on oma ahelloendis viimane.
3. Eemaldatav objekt on oma ahelloendis kusagil keskel.
4. Eemaldatav objekt on oma ahelloendis ainukene.
5. Eemaldatavat objekti ei olegi.

1.2.4. Neljas funktsioon

*Node *CreateBinaryTree(Objectn **ppStruct1);*

Sisendparameetriks on viit lähtestruktuuri viitade vektorile. Funktsiooni ülesandeks on ehitada kahendotsingu puu, mille tipud viitavad lähtestruktuuris olevatele kirjetele. Võtmeks on seejuures *Objectn* liige *Code*. Väljundparameetriks on viit puu juurtipule. Puu tippu esitav C *struct* on defineeritud failis *Headers.h*.

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Kirje lisamine kahendpuule"*.

1.2.5. Viies funktsioon

*void TreeTraversal(Node *pTree);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule. Funktsiooni ülesandeks on meetodil "vasak-juur-parem" käia läbi kõik puu tipud ja iga tipu puhul trükkida välja tema juurde kuuluva objekti kirjeldus (samuti nagu esimeses funktsioonis).

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Puu läbikäik (3)"*. Magasiniga opereerimiseks kasutage vastavat C *struct*-i failist *Headers.h* ja võtke eeskujuks funktsioonide push ning pop koodid slaididelt "*Näide magasini realisatsioonist (1)"* ja "*Näide magasini realisatsioonist (2)"*

1.2.6. Kuues funktsioon

*Node *DeleteTreeNode(Node *pTree, unsigned long int Code);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule ja võimalikule koodile. Funktsiooni ülesandeks on eemaldada puust tipp, mis viitab etteantud liiget *Code* sisaldavale objektile. Väljundparameetriks on viit pärast eemaldamist saadud puu juurtipule.

Funktsioon peab olema mitterekursiivne. Tema kirjutamisel lähtuge slaididest *"Kirje eemaldamine kahendpuust (1)"* ja *"Kirje eemaldamine kahendpuust (2)"*.

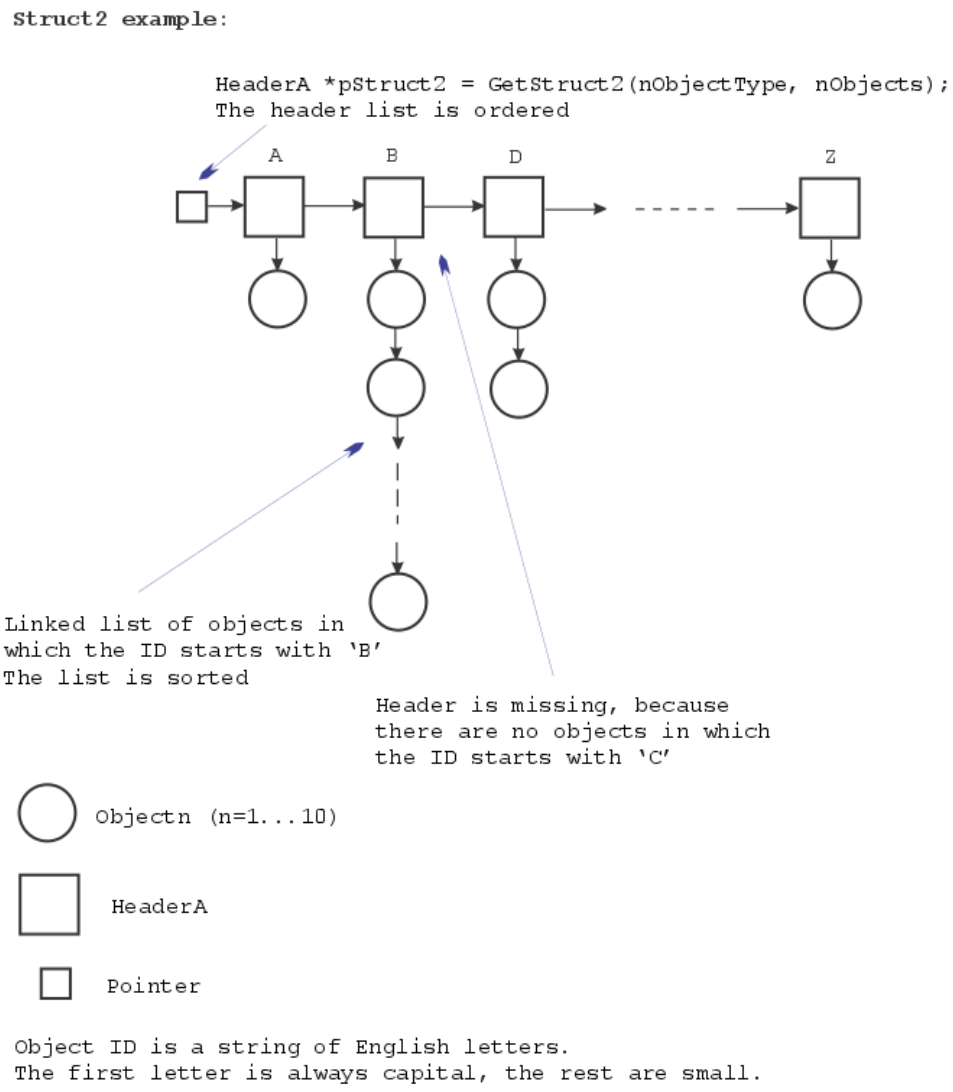
Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:
1. Eemaldatav tipp on puu juureks.
2. Eemaldataval tipul ei ole tütartippe.

3. Eemaldataval tipul on ainult parempoolne tütartipp.
4. Eemaldataval tipul on ainult vasakpoolne tütartipp.
5. Eemaldataval tipul on mõlemad tütartipud.
6. Etteantud koodiga kirjet ei olegi.

2. Variandid 2A ja 2B

2.1. Lähteandmed

Lähteandmeteks on ahelloenditesse ühendatud objektid, mis on *HeaderA* tüüpi sidujatest koosneva ahelloendiga ühendatud terviklikuks andmestruktuuriks. Objektid võivad olla tüüpidest *Object1*, *Object2*,…*Object10*. Objekti tüübi annab ette juhendaja. Objektide kui C *struct*-ide deklaratsioonid on esitatud failis *Objects.h* ja neid selgitavad skeemid asuvad käesoleva juhendi lisas. *HeaderA* deklaratsiooni võib leida failist *Headers.h*. Lähtestruktuuri näide (rõhutame, et tegemist on üksnes selgitava näitega) on toodud alljärgneval joonisel:



Objekti identifikaatoriks on kõikidel juhtudel C string, mis koosneb inglise tähestiku väiketähtedest. Stringi esimene sümbol on aga inglise tähestiku suurtäht. Stringi pikkuseks võib olla mistahes nullist suurem arv.

Ahelloend on identifikaatori järgi sorditud (s.t. kõige väiksema identifikaatoriga objekt on kõige esimene).

Iga *HeaderA* tüüpi siduja vastab ühele kindlale identifikaatori algustähele e. teisiti väljendades – kõik ühe ja sama sidujaga seotud objektid omavad sedasama identifikaatori algustähte. Nii näiteks objektid identifikaatoritega *Jaan*, *Juhan* ja *Johannes* asuvad kõik selles objektide ahelas, mis väljub tähega 'J' seotud sidujast. Sidujad paiknevad *HeaderA* ahelas vastavalt tähestiku järjekorrale. Oluline on rõhutada, et kui mingi algustähega seotud objekte ei ole, puudub ka vastav siduja.

Lähtestruktuuri genereerib juhendaja poolt antud funktsioon *GetStruct2* (asetseb objektmoodulis *Structs.ob*j, prototüüp on failis *Structs.h*).

2.2. Ülesanne

Laboratoorse töö ülesandeks on kirjutada 6 järgmist funktsiooni:

2.2.1. Esimene funktsioon

*void PrintObjects(HeaderA *pStruct2);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on väljastada kõikide struktuuris paiknevate objektide kirjeldused (iga objekt eraldi reas). Väljastavaks funktsiooniks on *printf*. Väljastamise formaatimise string (*printf* esimene parameeter) on toodud failis *Objects.h.*

See funktsioon tuleb kirjutada kõige esimesena, sest ilma temata ei ole võimalik ülejäänud funktsioonide tööd kontrollida.

2.2.2. Teine funktsioon

*int InsertNewObject(HeaderA **pStruct2, char *pNewID, int NewCode);*

Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderA* tüüpi sidujale; viit uue objekti identifikaatorile ja uue objekti kood. Funktsiooni ülesandeks on luua uus objekt ja lülitada ta andmestruktuuri.

Funktsioon peab esmalt kontrollima, kas uus identifikaator ikka vastab eespool toodud formaadile. Kui see tingimus ei ole täidetud, väljastab funktsioon lihtsalt nulli. Edasi peab funktsioon kontrollima, kas sellise identifikaatoriga objekt ikka tõesti puudub. Kui ta on olemas, väljastab funktsioon samuti lihtsalt nulli. Kui sellise identifikaatoriga objekti ei ole, tuleb ta luua ning lülitada õigesse ahelloendisse. Sealjuures sõltuvalt variandist:

A. Uued objektid võib paigutada suvalisele kohale, s.t. esialgne sorditud järjestus ei pea säiluma.

B. Ka uute objektidega täiendatud ahelloend peab olema sorditud.

Kõik vajalikud mäluväljad tuleb funktsiooni *malloc* abil tellida. Ka identifikaatori jaoks on vaja omaette mäluvälja, kuhu sisendparameetriks olev string tuleb kopeerida. Kellaaeg või kuupäev tuleb lugeda arvuti kellalt ja teisendada (vastavad funktsioonid on objektmoodulis *DateTime.obj*, prototüübid on failis *DateTime.h*). Kui funktsioon täitis oma ülesande, väljastab ta suuruse 1.

Võib juhtuda, et etteantud uue identifikaatori algustähega objekte varem ei olnudki ja seetõttu puudub ka vastav siduja. Sellisel juhul tuleb kõigepealt luua uus siduja ja paigutada ta sidujate ahelloendis tema jaoks ettenähtud kohale. Kui uus siduja tuleb kõige esimeseks, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Uus objekt tuleb juba olemasolevasse ahelloendisse. Variandi B puhul lisanduvad siia olukorrad, kus:
    a. Uus objekt tuleb oma ahelloendi esimeseks.
    b. Uus objekt tuleb oma ahelloendi viimaseks.
    c. Uus objekt tuleb kusagile oma ahelloendi keskele.
2. Ahelloendit uue objekti jaoks (ja seega ka temale vastavat *HeaderA* tüüpi sidujat) siiamaani ei olnud, kusjuures:
    a. Uus siduja tuleb *HeaderA* ahelloendis kõige esimeseks.
    b. Uus siduja tuleb *HeaderA* ahelloendis kõige viimaseks.
    c. Uus siduja tuleb *HeaderA* ahelloendis kusagile keskele.
3. Uue objekti identifikaator on ebaõiges formaadis.
4. Uue objekti identifikaator on sama mis mõnel juba olemasoleval objektil.

2.2.3. Kolmas funktsioon

Objectn RemoveExistingObject(HeaderA **pStruct2, char *pExistingID);*

Objectn asemel tuleb kirjutada loomulikult *Object1*, *Object2* jne. Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderA* tüüpi sidujale ja viit stringile, mis peab kokku langema ühe eeldatavalt olemasoleva objekti identifikaatoriga. Funktsiooni ülesandeks on leida see objekt ja eemaldada ta andmestruktuurist. Eemaldatud objekti kustutada ei tohi. Väljundsuuruseks on viit eemaldatud objektile. Kui objekti ei leitud, on väljundsuuruseks null.

Võib juhtuda, et eemaldatav objekt on oma ahelloendis ainuke olemasolev. Sellisel juhul tuleb pärast objekti eemaldamist eemaldada ning kustutada ka tema algustähele vastav *HeaderA* tüüpi siduja. Kui kustutatud siduja oli kõige esimene, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Eemaldatav objekt paikneb ahelloendis, kus on rohkem kui üks objekt, kusjuures:
    a. Eemaldatav objekt on oma ahelloendis esimene.
    b. Eemaldatav objekt on oma ahelloendis viimane.
    c. Eemaldatav objekt on oma ahelloendis kusagil keskel.
2. Eemaldatav objekt on oma ahelloendis ainukene, kusjuures:
    a. Kustutatav siduja on *HeaderA* ahelloendis kõige esimene.
    b. Kustutatav siduja on *HeaderA* ahelloendis kõige viimane.
    c. Kustutatav siduja on *HeaderA* ahelloendis kusagil keskel.
3. Etteantud identifikaatoriga objekti ei ole olemas.

2.2.4. Neljas funktsioon

*Node *CreateBinaryTree(HeaderA *pStruct2);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on ehitada kahendotsingu puu, mille tipud viitavad lähtestruktuuris olevatele kirjetele. Võtmeks on seejuures *Objectn* liige *Code*. Väljundparameetriks on viit puu juurtipule. Puu tippu esitav C *struct* on defineeritud failis *Headers.h*.

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Kirje lisamine kahendpuule"*.

2.2.5. Viies funktsioon

*void TreeTraversal(Node *pTree);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule. Funktsiooni ülesandeks on meetodil "vasak-juur-parem" käia läbi kõik puu tipud ja iga tipu puhul trükkida välja tema juurde kuuluva objekti kirjeldus (samuti nagu esimeses funktsioonis).

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Puu läbikäik (3)"*. Magasiniga opereerimiseks kasutage vastavat C *struct*-i failist *Headers.h* ja võtke eeskujuks funktsioonide push ning pop koodid slaididelt "*Näide magasini realisatsioonist (1)"* ja "*Näide magasini realisatsioonist (2)"*

2.2.6. Kuues funktsioon

*Node *DeleteTreeNode(Node *pTree, unsigned long int Code);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule ja võimalikule koodile. Funktsiooni ülesandeks on eemaldada puust tipp, mis viitab etteantud liiget *Code* sisaldavale objektile. Väljundparameetriks on viit pärast eemaldamist saadud puu juurtipule.

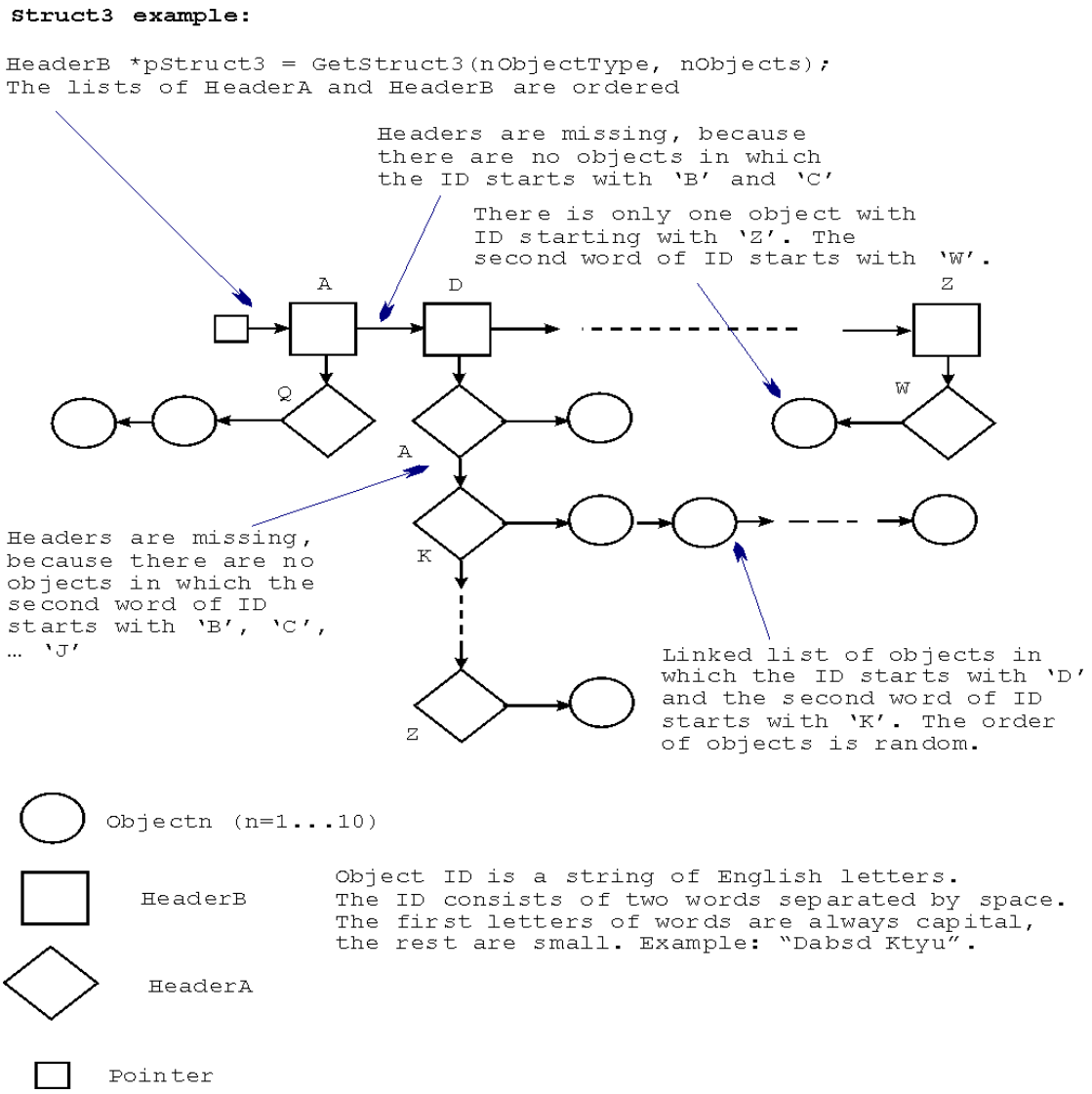
Funktsioon peab olema mitterekursiivne. Tema kirjutamisel lähtuge slaididest *"Kirje eemaldamine kahendpuust (1)"* ja *"Kirje eemaldamine kahendpuust (2)"*.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:
1. Eemaldatav tipp on puu juureks.
2. Eemaldataval tipul ei ole tütartippe.
3. Eemaldataval tipul on ainult parempoolne tütartipp.
4. Eemaldataval tipul on ainult vasakpoolne tütartipp.
5. Eemaldataval tipul on mõlemad tütartipud.
6. Etteantud koodiga kirjet ei olegi.

3. Variant 3

3.1. Lähteandmed

Lähteandmeteks on ahelloenditesse ühendatud objektid, mis on *HeaderA* ning *HeaderB* tüüpi sidujatest koosnevate ahelloenditega ühendatud terviklikuks andmestruktuuriks. Objektid võivad olla tüüpidest *Object1*, *Object2*,…*Object10*. Objekti tüübi annab ette juhendaja. Objektide kui C *struct*-ide deklaratsioonid on esitatud failis *Objects.h* ja neid selgitavad skeemid asuvad käesoleva juhendi lisas. *HeaderA* ning *HeaderB* deklaratsiooni võib leida failist *Headers.h*. Lähtestruktuuri näide (rõhutame, et tegemist on üksnes selgitava näitega) on toodud alljärgneval joonisel:



Objekti identifikaatoriks on kõikidel juhtudel C string, mis koosneb kahest sõnast. Kumbki sõna koosneb inglise tähestiku väiketähtedest, kuid sõna esimene sümbol on inglise tähestiku suurtäht. Sõnade vahel paikneb üks tühik. Sõnade pikkusteks võivad olla mistahes nullist suuremad arvud.

Objektide järjekord ahelloendis on juhuslik.

Iga *HeaderB* tüüpi siduja vastab ühele kindlale identifikaatori esimese sõna algustähele e. teisiti väljendades – kui me liigume mööda andmestruktuuri, siis kõik objektid, milledeni me võime antud sidujast lähtudes jõuda, omavad sedasama identifikaatori esimese sõna algustähte.

Iga *HeaderA* tüüpi siduja vastab ühele kindlale identifikaatori teise sõna algustähele e. teisiti väljendades – kõik ühe ja sama *HeaderA* tüüpi sidujaga seotud objektid omavad sedasama identifikaatori teise sõna algustähte. Nii näiteks objektid identifikaatoritega *Kask Jaan, Kuusk Juhan* ja *Kadakas Johannes* asuvad kõik selles objektide ahelas, mis väljub tähega 'J' seotud *HeaderA* tüüpi sidujast ning antud siduja omakorda asub ahelas, mis väljub tähega 'K' seotud *HeaderB* tüüpi sidujast.

Nii *HeaderB* kui ka *HeaderA* ahela lülid paiknevad vastavalt tähestiku järjekorrale. Oluline on rõhutada, et kui mingi algustähega seotud objekte ei ole, puudub ka vastav siduja.

Lähtestruktuuri genereerib juhendaja poolt antud funktsioon *GetStruct3* (asetseb objektmoodulis *Structs.ob*j, prototüüp on failis *Structs.h*).

3.2. Ülesanne

Laboratoorse töö ülesandeks on kirjutada 6 järgmist funktsiooni:

3.2.1. Esimene funktsioon

*void PrintObjects(HeaderB *pStruct3);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on väljastada kõikide struktuuris paiknevate objektide kirjeldused (iga objekt eraldi reas). Väljastavaks funktsiooniks on *printf*. Väljastamise formaatimise string (*printf* esimene parameeter) on toodud failis *Objects.h.*

See funktsioon tuleb kirjutada kõige esimesena, sest ilma temata ei ole võimalik ülejäänud funktsioonide tööd kontrollida.

3.2.2. Teine funktsioon

*int InsertNewObject(HeaderB **pStruct3, char *pNewID, int NewCode);*

Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderB* tüüpi sidujale; viit uue objekti identifikaatorile ja uue objekti kood. Funktsiooni ülesandeks on luua uus objekt ja lülitada ta andmestruktuuri.

Funktsioon peab esmalt kontrollima, kas uus identifikaator ikka vastab eespool toodud formaadile. Kui see tingimus ei ole täidetud, väljastab funktsioon lihtsalt nulli. Edasi peab funktsioon kontrollima, kas sellise identifikaatoriga objekt ikka tõesti puudub. Kui ta on olemas, väljastab funktsioon samuti lihtsalt nulli. Kui sellise identifikaatoriga objekti ei ole, tuleb ta luua ning lülitada õigesse ahelloendisse. Kõik vajalikud mäluväljad tuleb funktsiooni *malloc* abil tellida. Ka identifikaatori jaoks on vaja omaette mäluvälja, kuhu sisendparameetriks olev string tuleb kopeerida. Kellaaeg või kuupäev tuleb lugeda arvuti kellalt ja teisendada (vastavad funktsioonid on objektmoodulis *DateTime.obj*, prototüübid on failis *DateTime.h*). Uue objekti asukoht oma ahelloendis ei ole oluline. Kui funktsioon täitis oma ülesande, väljastab ta suuruse 1.

Võib juhtuda, et etteantud uue identifikaatori esimese ja/või teise sõna algustähega objekte varem ei olnudki ja seetõttu puudub (puuduvad) ka vastav(ad) siduja(d). Sellisel juhul tuleb puuduv(ad) siduja(d) kõigepealt luua ja paigutada ta (nad) sidujate ahelloendis (ahelloendites) ettenähtud kohale (kohtadele). Kui uus *HeaderB* tüüpi siduja tuleb kõige esimeseks, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Uus objekt tuleb juba olemasolevasse ahelloendisse.
2. Sellise esimese sõna algustähega objekte siiani ei olnud, mistõttu puuduvad ka vajalikud *HeaderB* ja *HeaderA* sidujad:
    a. Uus siduja tuleb *HeaderB* ahelloendis kõige esimeseks.
    b. Uus siduja tuleb *HeaderB* ahelloendis kõige viimaseks.
    c. Uus siduja tuleb *HeaderB* ahelloendis kusagile keskele.
3. Sellise esimese sõna algustähega objekte küll oli (s.t. vastav HeaderB siduja on olemas), kuid teise sõna algustähega objekte mitte, seega puudub ka vajalik *HeaderA* siduja:
    a. Uus siduja tuleb *HeaderA* ahelloendis kõige esimeseks.
    b. Uus siduja tuleb *HeaderA* ahelloendis kõige viimaseks.
    c. Uus siduja tuleb *HeaderA* ahelloendis kusagile keskele.
4. Uue objekti identifikaator on ebaõiges formaadis.
5. Uue objekti identifikaator on sama mis mõnel juba olemasoleval objektil.

3.2.3. Kolmas funktsioon

Objectn RemoveExistingObject(HeaderB **pStruct3, char *pExistingID);*

Objectn asemel tuleb kirjutada loomulikult *Object1*, *Object2* jne. Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderB* tüüpi sidujale ja viit stringile, mis peab kokku langema ühe eeldatavalt olemasoleva objekti identifikaatoriga. Funktsiooni ülesandeks on leida see objekt ja eemaldada ta andmestruktuurist. Eemaldatud objekti kustutada ei tohi. Väljundsuuruseks on viit eemaldatud objektile. Kui objekti ei leitud, on väljundsuuruseks null.

Võib juhtuda, et eemaldatav objekt on oma ahelloendis ainuke olemasolev. Sellisel juhul tuleb pärast objekti eemaldamist eemaldada ning kustutada ka tema algustähele vastav *HeaderA* tüüpi siduja. Sellele omakorda võib järgneda ka *HeaderB* tüüpi siduja kustutamine. Kui näiteks *Vaher Peeter* on ainukene objekt, mille esimene sõna algab ‘V’ tähega, siis selle eemaldamisega koos tuleb eemaldada ka veel nii tähele ‘P’ vastav *HeaderA* kui ka tähele ‘V’ vastav *HeaderB*.

Kui kustutatud *HeaderB* tüüpi siduja oli kõige esimene, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Eemaldatav objekt paikneb ahelloendis, kus on rohkem kui üks lüli, mistõttu sidujaid kaotada ei ole vaja:
    a. Eemaldatav objekt on oma ahelloendis esimene.
    b. Eemaldatav objekt on oma ahelloendis viimane.
    c. Eemaldatav objekt on oma ahelloendis kusagil keskel.

2. Eemaldatav objekt on oma ahelloendis ainukene, kuid sama esimese sõna esitähega objekte on rohkem, mistõttu *HeaderA* tüüpi siduja küll kaob, kuid *HeaderB* tüüpi siduja jääb alles:
    a. Kustutatav *HeaderA* siduja on ahelloendis kõige esimene.
    b. Kustutatav *HeaderA* siduja on ahelloendis kõige viimane.
    c. Kustutatav *HeaderA* siduja on ahelloendis kusagil keskel.
3. Eemaldatav objekt on oma ahelloendis ainukene, kuid sama esimese sõna esitähega objekte rohkem ei ole, mistõttu kaovad nii *HeaderA* kui ka *HeaderB* tüüpi sidujad:
    a. Kustutatav *HeaderB* siduja on ahelloendis kõige esimene.
    b. Kustutatav *HeaderB* siduja on ahelloendis kõige viimane.
    c. Kustutatav *HeaderB* siduja on ahelloendis kusagil keskel.
4. Etteantud identifikaatoriga objekti ei ole olemas.

3.2.4. Neljas funktsioon

*Node *CreateBinaryTree(HeaderB *pStruct3);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on ehitada kahendotsingu puu, mille tipud viitavad lähtestruktuuris olevatele kirjetele. Võtmeks on seejuures *Objectn* liige *Code*. Väljundparameetriks on viit puu juurtipule. Puu tippu esitav C *struct* on defineeritud failis *Headers.h.*

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Kirje lisamine kahendpuule"*.

3.2.5. Viies funktsioon

*void TreeTraversal(Node *pTree);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule. Funktsiooni ülesandeks on meetodil "vasak-juur-parem" käia läbi kõik puu tipud ja iga tipu puhul trükkida välja tema juurde kuuluva objekti kirjeldus (samuti nagu esimeses funktsioonis).

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Puu läbikäik (3)"*. Magasiniga opereerimiseks kasutage vastavat C *struct*-i failist *Headers.h* ja võtke eeskujuks funktsioonide push ning pop koodid slaididelt "*Näide magasini realisatsioonist (1)"* ja "*Näide magasini realisatsioonist (2)"*

3.2.6. Kuues funktsioon

*Node *DeleteTreeNode(Node *pTree, unsigned long int Code);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule ja võimalikule koodile. Funktsiooni ülesandeks on eemaldada puust tipp, mis viitab etteantud liiget *Code* sisaldavale objektile. Väljundparameetriks on viit pärast eemaldamist saadud puu juurtipule.

Funktsioon peab olema mitterekursiivne. Tema kirjutamisel lähtuge slaididest *"Kirje eemaldamine kahendpuust (1)"* ja *"Kirje eemaldamine kahendpuust (2)"*.

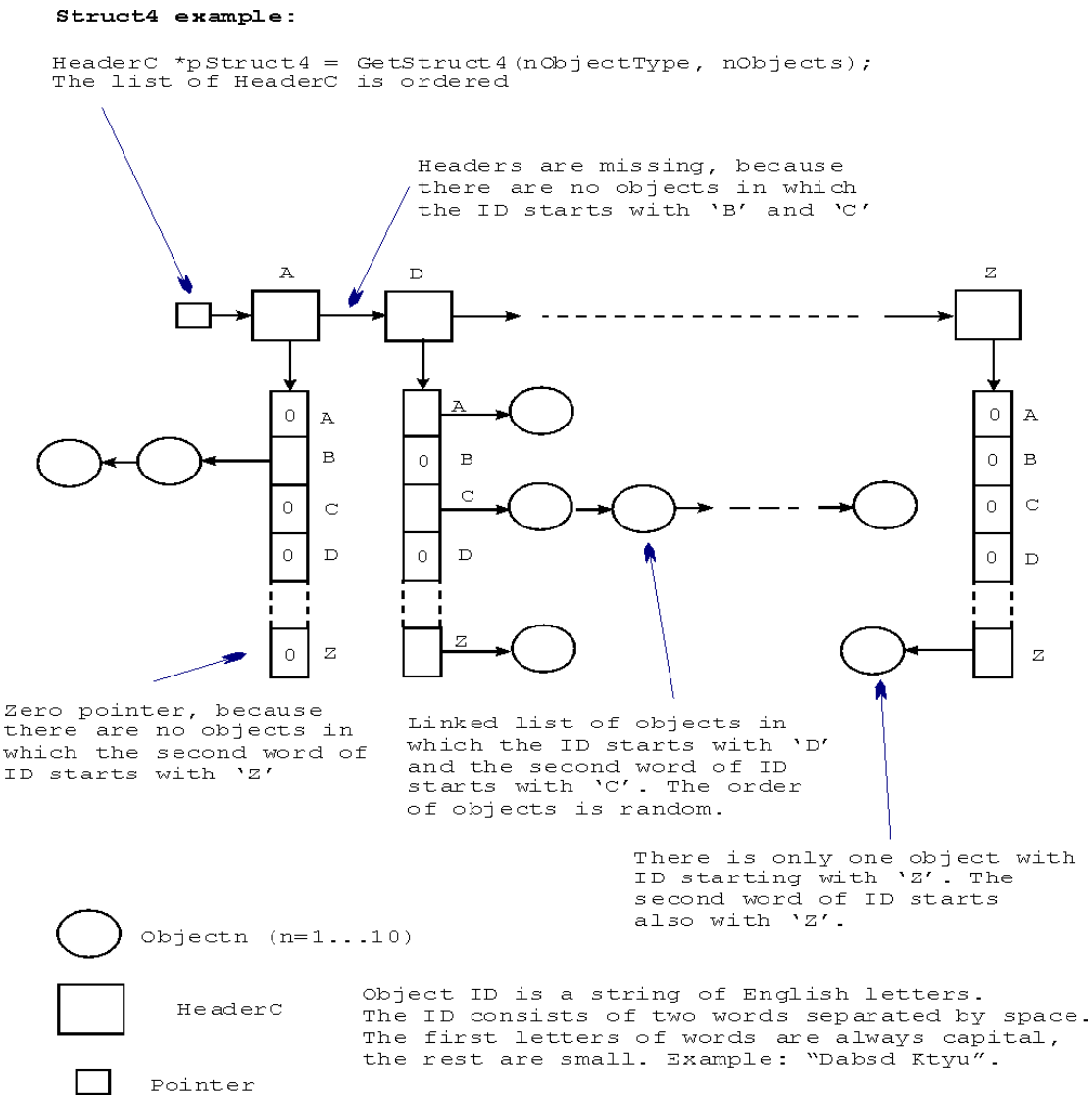
Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:
1. Eemaldatav tipp on puu juureks.
2. Eemaldataval tipul ei ole tütartippe.
3. Eemaldataval tipul on ainult parempoolne tütartipp.
4. Eemaldataval tipul on ainult vasakpoolne tütartipp.

5. Eemaldataval tipul on mõlemad tütartipud.
6. Etteantud koodiga kirjet ei olegi.

4. Variant 4

4.1. Lähteandmed

Lähteandmeteks on ahelloenditesse ühendatud objektid, mis on *HeaderC* sidujatest koosnevate ahelloenditega ning viitade vektoritega ühendatud terviklikuks andmestruktuuriks. Objektid võivad olla tüüpidest *Object1*, *Object2*,…*Object10*. Objekti tüübi annab ette juhendaja. Objektide kui C *struct*-ide deklaratsioonid on esitatud failis *Objects.h* ja neid selgitavad skeemid asuvad käesoleva juhendi lisas. *HeaderC* deklaratsiooni võib leida failist *Headers.h*. Lähtestruktuuri näide (rõhutame, et tegemist on üksnes selgitava näitega) on toodud alljärgneval joonisel:



Objekti identifikaatoriks on kõikidel juhtudel C string, mis koosneb kahest sõnast. Kumbki sõna koosneb inglise tähestiku väiketähtedest, kuid sõna esimene sümbol on inglise tähestiku suurtäht. Sõnade vahel paikneb üks tühik. Sõnade pikkusteks võivad olla mistahes nullist suuremad arvud.

Objektide järjekord ahelloendis on juhuslik.

Iga *HeaderC* tüüpi siduja vastab ühele kindlale identifikaatori esimese sõna algustähele e. teisiti väljendades – kui me liigume mööda andmestruktuuri, siis kõik objektid, milledeni me võime antud sidujast lähtudes jõuda, omavad sedasama identifikaatori esimese sõna algustähte.

Iga viitade vektori positsioon vastab ühele kindlale identifikaatori teise sõna algustähele. Kui mingi teise sõna algustähega seotud objekte ei ole, on viitade vektoris vastaval positsioonil null. Nii näiteks objektid identifikaatoritega *Kask Jaan*, *Kuusk Juhan* ja *Kadakas Johannes* asuvad kõik selles objektide ahelas, mis väljub tähele 'J' vastavast positsioonist viitade vektoris, viimane omakorda on aga seotud tähele 'K' vastava *HeaderC* tüüpi sidujaga.

HeaderC ahela lülid paiknevad vastavalt tähestiku järjekorrale. Oluline on rõhutada, et kui mingi algustähega seotud objekte ei ole, puudub ka vastav siduja.

Lähtestruktuuri genereerib juhendaja poolt antud funktsioon *GetStruct4* (asetseb objektmoodulis *Structs.ob*j, prototüüp on failis *Structs.h*).

4.2. Ülesanne

Laboratoorse töö ülesandeks on kirjutada 6 järgmist funktsiooni:

4.2.1. Esimene funktsioon

*void PrintObjects(HeaderC *pStruct4);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on väljastada kõikide struktuuris paiknevate objektide kirjeldused (iga objekt eraldi reas). Väljastavaks funktsiooniks on *printf*. Väljastamise formaatimise string (*printf* esimene parameeter) on toodud failis *Objects.h.*

See funktsioon tuleb kirjutada kõige esimesena, sest ilma temata ei ole võimalik ülejäänud funktsioonide tööd kontrollida.

4.2.2. Teine funktsioon

*int InsertNewObject(HeaderC **pStruct4, char *pNewID, int NewCode);*

Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderC* tüüpi sidujale; viit uue objekti identifikaatorile ja uue objekti kood. Funktsiooni ülesandeks on luua uus objekt ja lülitada ta andmestruktuuri.

Funktsioon peab esmalt kontrollima, kas uus identifikaator ikka vastab eespool toodud formaadile. Kui see tingimus ei ole täidetud, väljastab funktsioon lihtsalt nulli. Edasi peab funktsioon kontrollima, kas sellise identifikaatoriga objekt ikka tõesti puudub. Kui ta on olemas, väljastab funktsioon samuti lihtsalt nulli. Kui sellise identifikaatoriga objekti ei ole, tuleb ta luua ning lülitada õigesse ahelloendisse. Kõik vajalikud mäluväljad tuleb funktsiooni *malloc* abil tellida. Ka identifikaatori jaoks on vaja omaette mäluvälja, kuhu sisendparameetriks olev string tuleb kopeerida. Kellaaeg või kuupäev tuleb lugeda arvuti kellalt ja teisendada (vastavad funktsioonid on objektmoodulis *DateTime.obj*, prototüübid on failis *DateTime.h*). Uue objekti asukoht oma ahelloendis ei ole oluline. Kui funktsioon täitis oma ülesande, väljastab ta suuruse 1.

Võib juhtuda, et etteantud uue identifikaatori esimese sõna algustähega objekte varem ei olnudki ja seetõttu puudub ka vastav siduja. Sellisel juhul tuleb puuduv siduja

kõigepealt luua ja paigutada ta sidujate ahelloendis ettenähtud kohale. Samuti tuleb luua viitade vektor ja täita ta nullidega. Kui uus *HeaderC* tüüpi siduja tuleb kõige esimeseks, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Uus objekt tuleb juba olemasolevasse ahelloendisse.
2. Sellise esimese sõna algustähega objekte siiani ei olnud, mistõttu puudub ka vajalik *HeaderC* siduja ja viitade vektor:
    a. Uus siduja tuleb *HeaderC* ahelloendis kõige esimeseks.
    b. Uus siduja tuleb *HeaderC* ahelloendis kõige viimaseks.
    c. Uus siduja tuleb *HeaderC* ahelloendis kusagile keskele.
3. Sellise esimese sõna algustähega objekte küll oli, kuid teise sõna algustähega objekte mitte.
4. Uue objekti identifikaator on ebaõiges formaadis.
5. Uue objekti identifikaator on sama mis mõnel juba olemasoleval objektil.

4.2.3. Kolmas funktsioon

Objectn RemoveExistingObject(HeaderC **pStruct4, char *pExistingID);*

Objectn asemel tuleb kirjutada loomulikult *Object1, Object2* jne. Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderC* tüüpi sidujale ja viit stringile, mis peab kokku langema ühe eeldatavalt olemasoleva objekti identifikaatoriga. Funktsiooni ülesandeks on leida see objekt ja eemaldada ta andmestruktuurist. Eemaldatud objekti kustutada ei tohi. Väljundsuuruseks on viit eemaldatud objektile. Kui objekti ei leitud, on väljundsuuruseks null.

Võib juhtuda, et eemaldatav objekt on oma ahelloendis ainuke olemasolev ning rohkem sama esimese sõna algustähega objekte pole. Sellisel juhul tuleb pärast objekti eemaldamist eemaldada ning kustutada nii tema algustähele vastav *HeaderC* tüüpi siduja kui ka sellega seotud viitade vektor. Kui näiteks *Vaher Peeter* on ainukene objekt, mille esimene sõna algab ‘V’ tähega, siis selle eemaldamisega koos tuleb eemaldada ka veel tähele ‘V’ vastav *HeaderC* koos viitade vektoriga.

Kui kustutatud *HeaderC* tüüpi siduja oli kõige esimene, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Eemaldatav objekt paikneb ahelloendis, kus on rohkem kui üks lüli:
    a. Eemaldatav objekt on oma ahelloendis esimene.
    b. Eemaldatav objekt on oma ahelloendis viimane.
    c. Eemaldatav objekt on oma ahelloendis kusagil keskel.
2. Eemaldatav objekt on oma ahelloendis ainukene, kuid sama esimese sõna esitähega objekte on rohkem, mistõttu *HeaderC* tüüpi siduja jääb alles.
3. Eemaldatav objekt on oma ahelloendis ainukene ja sama esimese sõna esitähega objekte rohkem ei ole, mistõttu *HeaderC* tüüpi siduja kaob:
    a. Kustutatav *HeaderC* siduja on ahelloendis kõige esimene.
    b. Kustutatav *HeaderC* siduja on ahelloendis kõige viimane.
    c. Kustutatav *HeaderC* siduja on ahelloendis kusagil keskel.
4. Etteantud identifikaatoriga objekti ei ole olemas.

4.2.4. Neljas funktsioon

*Node *CreateBinaryTree(HeaderC *pStruct4);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on ehitada kahendotsingu puu, mille tipud viitavad lähtestruktuuris olevatele kirjetele. Võtmeks on seejuures *Objectn* liige *Code*. Väljundparameetriks on viit puu juurtipule. Puu tippu esitav C *struct* on defineeritud failis *Headers.h.*

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Kirje lisamine kahendpuule"*.

4.2.5. Viies funktsioon

*void TreeTraversal(Node *pTree);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule. Funktsiooni ülesandeks on meetodil "vasak-juur-parem" käia läbi kõik puu tipud ja iga tipu puhul trükkida välja tema juurde kuuluva objekti kirjeldus (samuti nagu esimeses funktsioonis).

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Puu läbikäik (3)"*. Magasiniga opereerimiseks kasutage vastavat C *struct*-i failist *Headers.h* ja võtke eeskujuks funktsioonide push ning pop koodid slaididelt "*Näide magasini realisatsioonist (1)"* ja "*Näide magasini realisatsioonist (2)"*

4.2.6. Kuues funktsioon

*Node *DeleteTreeNode(Node *pTree, unsigned long int Code);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule ja võimalikule koodile. Funktsiooni ülesandeks on eemaldada puust tipp, mis viitab etteantud liiget *Code* sisaldavale objektile. Väljundparameetriks on viit pärast eemaldamist saadud puu juurtipule.

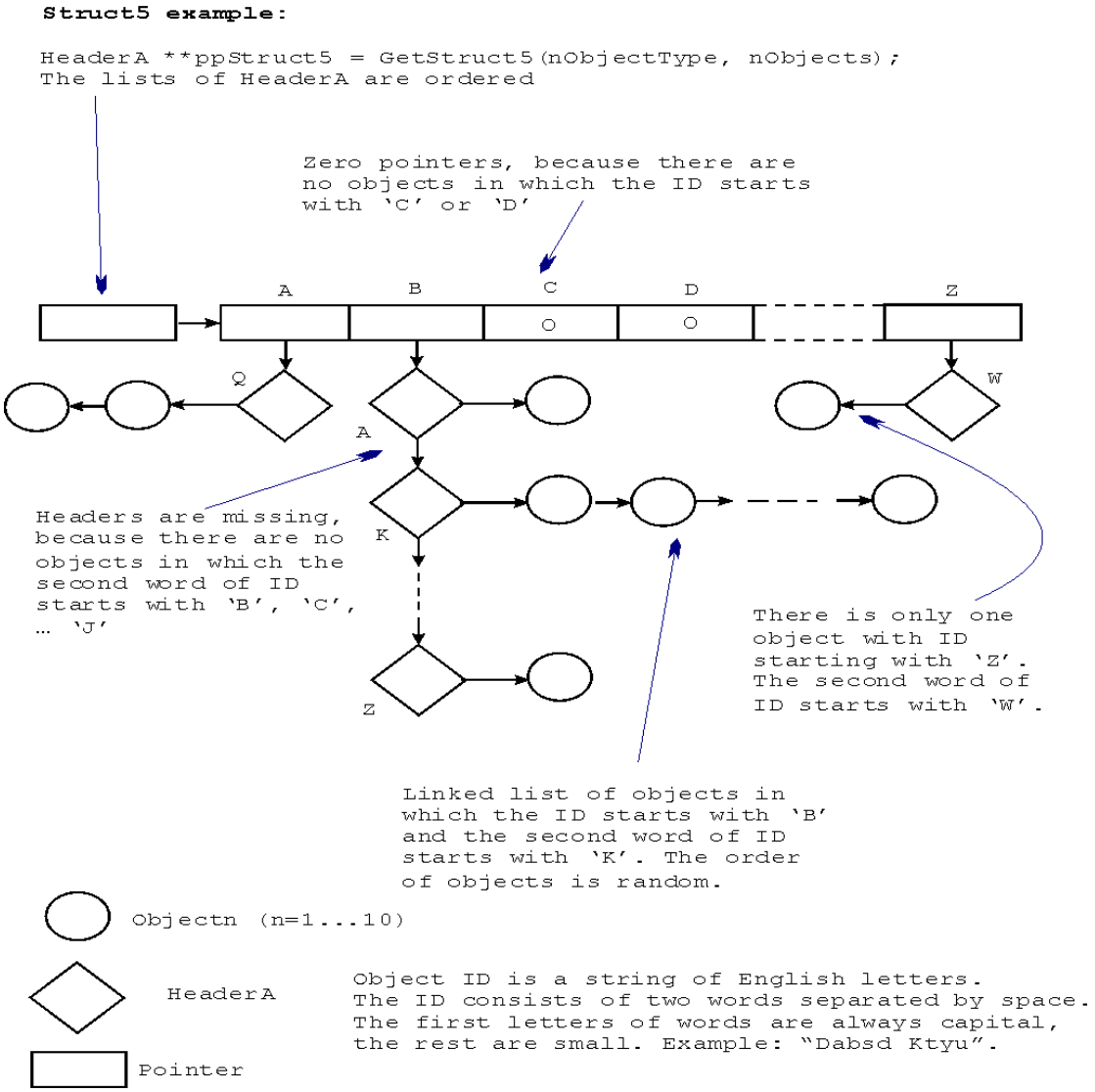
Funktsioon peab olema mitterekursiivne. Tema kirjutamisel lähtuge slaididest *"Kirje eemaldamine kahendpuust (1)"* ja *"Kirje eemaldamine kahendpuust (2)"*.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:
1. Eemaldatav tipp on puu juureks.
2. Eemaldataval tipul ei ole tütartippe.
3. Eemaldataval tipul on ainult parempoolne tütartipp.
4. Eemaldataval tipul on ainult vasakpoolne tütartipp.
5. Eemaldataval tipul on mõlemad tütartipud.
6. Etteantud koodiga kirjet ei olegi.

5. Variant 5

5.1. Lähteandmed

Lähteandmeteks on ahelloenditesse ühendatud objektid, mis on *HeaderA* sidujatest koosnevate ahelloenditega ning viitade vektoriga ühendatud terviklikuks andmestruktuuriks. Objektid võivad olla tüüpidest *Object1*, *Object2*,…*Object10*. Objekti tüübi annab ette juhendaja. Objektide kui C *struct*-ide deklaratsioonid on esitatud failis *Objects.h* ja neid selgitavad skeemid asuvad käesoleva juhendi lisas. *HeaderA* deklaratsiooni võib leida failist *Headers.h*. Lähtestruktuuri näide (rõhutame, et tegemist on üksnes selgitava näitega) on toodud alljärgneval joonisel:



Objekti identifikaatoriks on kõikidel juhtudel C string, mis koosneb kahest sõnast. Kumbki sõna koosneb inglise tähestiku väiketähtedest, kuid sõna esimene sümbol on inglise tähestiku suurtäht. Sõnade vahel paikneb üks tühik. Sõnade pikkusteks võivad olla mistahes nullist suuremad arvud.

Objektide järjekord ahelloendis on juhuslik.

Iga viitade vektori positsioon vastab ühele kindlale identifikaatori esimese sõna algustähele. Kui mingi esimese sõna algustähega seotud objekte ei ole, on viitade vektoris vastaval positsioonil null.

Iga *HeaderA* tüüpi siduja vastab ühele kindlale identifikaatori teise sõna algustähele. Nii näiteks objektid identifikaatoritega *Kask Jaan*, *Kuusk Juhan* ja *Kadakas Johannes* asuvad kõik selles objektide ahelas, mis väljub tähega 'J' seotud *HeaderA* tüüpi sidujast ning antud siduja omakorda asub ahelas, mis väljub tähele 'K' vastavast viitade vektori positsioonist.

HeaderA ahela lülid paiknevad vastavalt tähestiku järjekorrale. Oluline on rõhutada, et kui mingi teise sõna algustähega seotud objekte ei ole, puudub ka vastav siduja.

Lähtestruktuuri genereerib juhendaja poolt antud funktsioon *GetStruct5* (asetseb objektmoodulis *Structs.ob*j, prototüüp on failis *Structs.h*).

5.2. Ülesanne

Laboratoorse töö ülesandeks on kirjutada 6 järgmist funktsiooni:

5.2.1. Esimene funktsioon

*void PrintObjects(HeaderA **ppStruct5);*

Sisendparameetriks on viit lähtestruktuuri viitade vektorile. Funktsiooni ülesandeks on väljastada kõikide struktuuris paiknevate objektide kirjeldused (iga objekt eraldi reas). Väljastavaks funktsiooniks on *printf*. Väljastamise formaatimise string (*printf* esimene parameeter) on toodud failis *Objects.h.*

See funktsioon tuleb kirjutada kõige esimesena, sest ilma temata ei ole võimalik ülejäänud funktsioonide tööd kontrollida.

5.2.2. Teine funktsioon

*int InsertNewObject(HeaderA **ppStruct5, char *pNewID, int NewCode);*

Sisendparameetriteks on viit lähtestruktuuri viitade vektorile; viit uue objekti identifikaatorile ja uue objekti kood. Funktsiooni ülesandeks on luua uus objekt ja lülitada ta andmestruktuuri.

Funktsioon peab esmalt kontrollima, kas uus identifikaator ikka vastab eespool toodud formaadile. Kui see tingimus ei ole täidetud, väljastab funktsioon lihtsalt nulli. Edasi peab funktsioon kontrollima, kas sellise identifikaatoriga objekt ikka tõesti puudub. Kui ta on olemas, väljastab funktsioon samuti lihtsalt nulli. Kui sellise identifikaatoriga objekti ei ole, tuleb ta luua ning lülitada õigesse ahelloendisse. Kõik vajalikud mäluväljad tuleb funktsiooni *malloc* abil tellida. Ka identifikaatori jaoks on vaja omaette mäluvälja, kuhu sisendparameetriks olev string tuleb kopeerida. Kellaaeg või kuupäev tuleb lugeda arvuti kellalt ja teisendada (vastavad funktsioonid on objektmoodulis *DateTime.obj*, prototüübid on failis *DateTime.h*). Uue objekti asukoht oma ahelloendis ei ole oluline. Kui funktsioon täitis oma ülesande, väljastab ta suuruse 1.

Võib juhtuda, et etteantud uue identifikaatori teise sõna algustähega objekte varem ei olnudki ja seetõttu puudub ka vastav siduja. Sellisel juhul tuleb puuduv siduja kõigepealt luua ja paigutada ta sidujate ahelloendis ettenähtud kohale.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Uus objekt tuleb juba olemasolevasse ahelloendisse.
2. Sellise esimese sõna algustähega objekte varem ei olnud, mistõttu *HeaderA* ahelat veel pole.
3. Sellise esimese sõna algustähega objekte oli ka varem, kuid teise sõna algustähega objekte mitte, mistõttu puudub ka vajalik *HeaderA* siduja:
    a. Uus siduja tuleb *HeaderA* ahelloendis kõige esimeseks.
    b. Uus siduja tuleb *HeaderA* ahelloendis kõige viimaseks.
    c. Uus siduja tuleb *HeaderA* ahelloendis kusagile keskele.
4. Uue objekti identifikaator on ebaõiges formaadis.
5. Uue objekti identifikaator on sama mis mõnel juba olemasoleval objektil.

5.2.3. Kolmas funktsioon

Objectn RemoveExistingObject(HeaderA **ppStruct5, char *pExistingID);*

Objectn asemel tuleb kirjutada loomulikult *Object1*, *Object2* jne. Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderC* tüüpi sidujale ja viit stringile, mis peab kokku langema ühe eeldatavalt olemasoleva objekti identifikaatoriga. Funktsiooni ülesandeks on leida see objekt ja eemaldada ta andmestruktuurist. Eemaldatud objekti kustutada ei tohi. Väljundsuuruseks on viit eemaldatud objektile. Kui objekti ei leitud, on väljundsuuruseks null.

Võib juhtuda, et eemaldatav objekt on oma ahelloendis ainuke olemasolev. Sellisel juhul tuleb pärast objekti eemaldamist eemaldada ning kustutada ka tema algustähele vastav *HeaderA* tüüpi siduja. Kui pärast seda ühtegi sidujat alles ei jää, tuleb viitade vektori vastavale positsioonile kirjutada null.

Funktsiooni katsetades proovige kindlasti läbi järgmied olukorrad:

1. Eemaldatav objekt paikneb ahelloendis, kus on rohkem kui üks lüli:
    a. Eemaldatav objekt on oma ahelloendis esimene.
    b. Eemaldatav objekt on oma ahelloendis viimane.
    c. Eemaldatav objekt on oma ahelloendis kusagil keskel.
2. Eemaldatav objekt on oma ahelloendis ainukene, kuid sama esimese sõna esitähega objekte on rohkem, mistõttu *HeaderA* tüüpi sidujate ahel jääb alles.
3. Eemaldatav objekt on oma ahelloendis ainukene ja sama esimese sõna esitähega objekte rohkem ei ole, mistõttu *HeaderA* tüüpi sidujate ahel kaob.
4. Etteantud identifikaatoriga objekti ei ole olemas.

5.2.4. Neljas funktsioon

*Node *CreateBinaryTree(HeaderA **ppStruct5);*

Sisendparameetriks on viit lähtestruktuuri viitade vektorile. Funktsiooni ülesandeks on ehitada kahendotsingu puu, mille tipud viitavad lähtestruktuuris olevatele kirjetele. Võtmeks on seejuures *Objectn* liige *Code*. Väljundparameetriks on viit puu juurtipule. Puu tippu esitav C *struct* on defineeritud failis *Headers.h*.

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Kirje lisamine kahendpuule"*.

5.2.5. Viies funktsioon

*void TreeTraversal(Node *pTree);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule. Funktsiooni ülesandeks on meetodil "vasak-juur-parem" käia läbi kõik puu tipud ja iga tipu puhul trükkida välja tema juurde kuuluva objekti kirjeldus (samuti nagu esimeses funktsioonis).

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Puu läbikäik (3)"*. Magasiniga opereerimiseks kasutage vastavat C *struct*-i failist *Headers.h* ja võtke eeskujuks funktsioonide push ning pop koodid slaididelt "*Näide magasini realisatsioonist (1)"* ja "*Näide magasini realisatsioonist (2)"*

5.2.6. Kuues funktsioon

*Node *DeleteTreeNode(Node *pTree, unsigned long int Code);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule ja võimalikule koodile. Funktsiooni ülesandeks on eemaldada puust tipp, mis viitab etteantud liiget *Code* sisaldavale objektile. Väljundparameetriks on viit pärast eemaldamist saadud puu juurtipule.

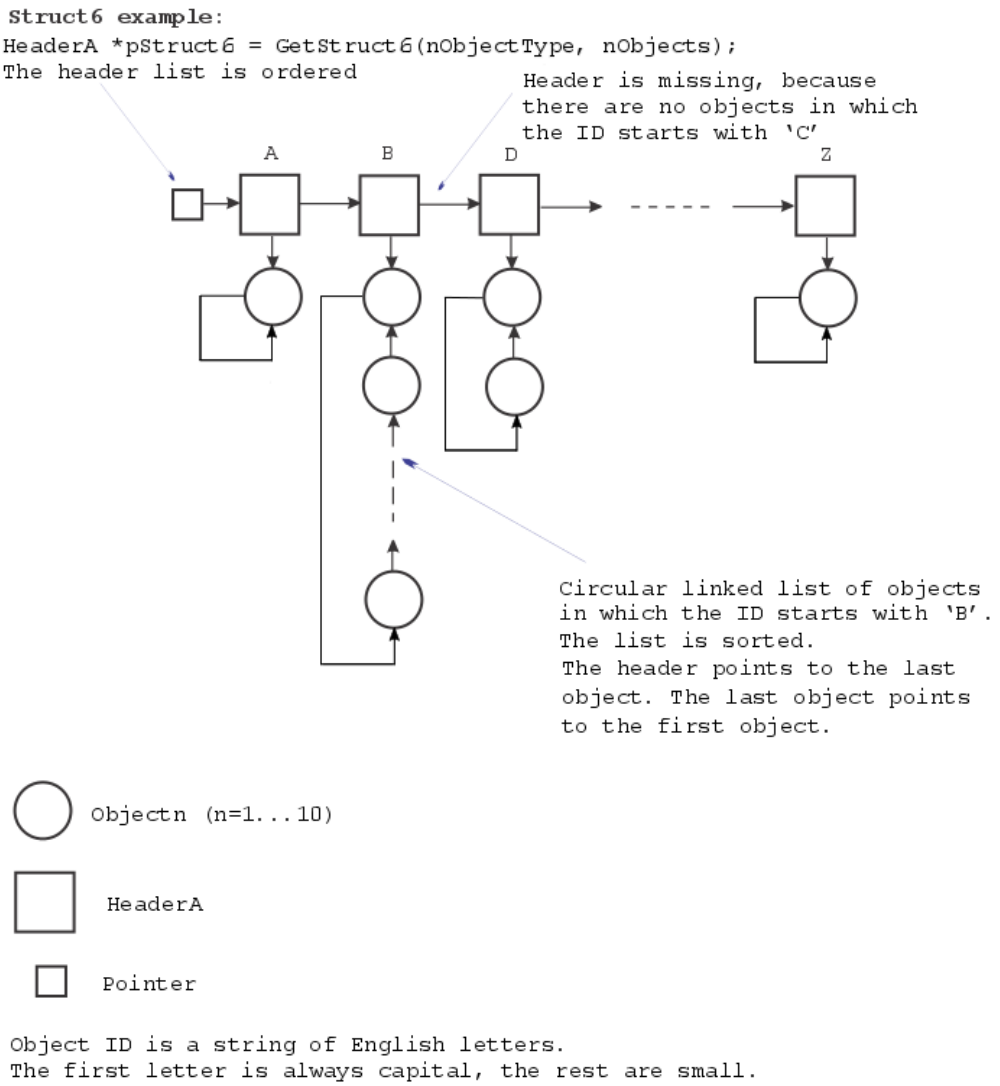
Funktsioon peab olema mitterekursiivne. Tema kirjutamisel lähtuge slaididest *"Kirje eemaldamine kahendpuust (1)"* ja *"Kirje eemaldamine kahendpuust (2)"*.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:
1. Eemaldatav tipp on puu juureks.
2. Eemaldataval tipul ei ole tütartippe.
3. Eemaldataval tipul on ainult parempoolne tütartipp.
4. Eemaldataval tipul on ainult vasakpoolne tütartipp.
5. Eemaldataval tipul on mõlemad tütartipud.
6. Etteantud koodiga kirjet ei olegi.

6. Variant 6

6.1. Lähteandmed

Lähteandmeteks on ahelloenditesse ühendatud objektid, mis on *HeaderA* tüüpi sidujatest koosneva ahelloendiga ühendatud terviklikuks andmestruktuuriks. Objektid võivad olla tüüpidest *Object1*, *Object2*,…*Object10*. Objekti tüübi annab ette juhendaja. Objektide kui C *struct*-ide deklaratsioonid on esitatud failis *Objects.h* ja neid selgitavad skeemid asuvad käesoleva juhendi lisas. *HeaderA* deklaratsiooni võib leida failist *Headers.h*. Lähtestruktuuri näide (rõhutame, et tegemist on üksnes selgitava näitega) on toodud alljärgneval joonisel:



Objekti identifikaatoriks on kõikidel juhtudel C string, mis koosneb inglise tähestiku väiketähtedest. Stringi esimene sümbol on aga inglise tähestiku suurtäht. Stringi pikkuseks võib olla mistahes nullist suurem arv.

Ringahelloend on identifikaatori järgi sorditud (s.t. kõige väiksema identifikaatoriga objekt on kõige esimene). Rõhutame, et siduja näitab ahela kõige viimasele (kõige

suurema identifikaatoriga) liikmele, too omakorda aga esimesele (kõige väiksema identifikaatoriga) liikmele.

Iga *HeaderA* tüüpi siduja vastab ühele kindlale identifikaatori algustähele e. teisiti väljendades – kõik ühe ja sama sidujaga seotud objektid omavad sedasama identifikaatori algustähte. Nii näiteks objektid identifikaatoritega *Jaan*, *Juhan* ja *Johannes* asuvad kõik selles objektide ahelas, mis väljub tähega 'J' seotud sidujast. Sidujad paiknevad *HeaderA* ahelas vastavalt tähestiku järjekorrale. Oluline on rõhutada, et kui mingi algustähega seotud objekte ei ole, puudub ka vastav siduja.

Lähtestruktuuri genereerib juhendaja poolt antud funktsioon *GetStruct6* (asetseb objektmoodulis *Structs.ob*j, prototüüp on failis *Structs.h*).

6.2. Ülesanne

Laboratoorse töö ülesandeks on kirjutada 6 järgmist funktsiooni:

6.2.1. Esimene funktsioon

*void PrintObjects(HeaderA *pStruct6);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on väljastada kõikide struktuuris paiknevate objektide kirjeldused (iga objekt eraldi reas). Väljastavaks funktsiooniks on *printf*. Väljastamise formaatimise string (*printf* esimene parameeter) on toodud failis *Objects.h.*

See funktsioon tuleb kirjutada kõige esimesena, sest ilma temata ei ole võimalik ülejäänud funktsioonide tööd kontrollida.

6.2.2. Teine funktsioon

*int InsertNewObject(HeaderA **pStruct6, char *pNewID, int NewCode);*

Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderA* tüüpi sidujale; viit uue objekti identifikaatorile ja uue objekti kood. Funktsiooni ülesandeks on luua uus objekt ja lülitada ta andmestruktuuri.

Funktsioon peab esmalt kontrollima, kas uus identifikaator ikka vastab eespool toodud formaadile. Kui see tingimus ei ole täidetud, väljastab funktsioon lihtsalt nulli. Edasi peab funktsioon kontrollima, kas sellise identifikaatoriga objekt ikka tõesti puudub. Kui ta on olemas, väljastab funktsioon samuti lihtsalt nulli. Kui sellise identifikaatoriga objekti ei ole, tuleb ta luua ning lülitada õigesse ahelloendisse. Sealjuures ka uute objektidega täiendatud ahelloend peab olema sorditud.

Kõik vajalikud mäluväljad tuleb funktsiooni *malloc* abil tellida. Ka identifikaatori jaoks on vaja omaette mäluvälja, kuhu sisendparameetriks olev string tuleb kopeerida. Kellaaeg või kuupäev tuleb lugeda arvuti kellalt ja teisendada (vastavad funktsioonid on objektmoodulis *DateTime.obj*, prototüübid on failis *DateTime.h*). Kui funktsioon täitis oma ülesande, väljastab ta suuruse 1.

Võib juhtuda, et etteantud uue identifikaatori algustähega objekte varem ei olnudki ja seetõttu puudub ka vastav siduja. Sellisel juhul tuleb kõigepealt luua uus siduja ja paigutada ta sidujate ahelloendis tema jaoks ettenähtud kohale. Kui uus siduja tuleb kõige esimeseks, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Uus objekt tuleb juba olemasolevasse ahelloendisse, kusjuures
    a. Uus objekt tuleb oma ahelloendi esimeseks.
    b. Uus objekt tuleb oma ahelloendi viimaseks (s.t. siduja hakkab osutama temale).
    c. Uus objekt tuleb kusagile oma ahelloendi keskele.
2. Ahelloendit uue objekti jaoks (ja seega ka temale vastavat *HeaderA* tüüpi sidujat) siiamaani ei olnud, kusjuures:
    a. Uus siduja tuleb *HeaderA* ahelloendis kõige esimeseks.
    b. Uus siduja tuleb *HeaderA* ahelloendis kõige viimaseks.
    c. Uus siduja tuleb *HeaderA* ahelloendis kusagile keskele.
3. Uue objekti identifikaator on ebaõiges formaadis.
4. Uue objekti identifikaator on sama mis mõnel juba olemasoleval objektil.

6.2.3. Kolmas funktsioon

Objectn RemoveExistingObject(HeaderA **pStruct2, char *pExistingID);*

Objectn asemel tuleb kirjutada loomulikult *Object1*, *Object2* jne. Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderA* tüüpi sidujale ja viit stringile, mis peab kokku langema ühe eeldatavalt olemasoleva objekti identifikaatoriga. Funktsiooni ülesandeks on leida see objekt ja eemaldada ta andmestruktuurist. Eemaldatud objekti kustutada ei tohi. Väljundsuuruseks on viit eemaldatud objektile. Kui objekti ei leitud, on väljundsuuruseks null.

Võib juhtuda, et eemaldatav objekt on oma ahelloendis ainuke olemasolev. Sellisel juhul tuleb pärast objekti eemaldamist eemaldada ning kustutada ka tema algustähele vastav *HeaderA* tüüpi siduja. Kui kustutatud siduja oli kõige esimene, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Eemaldatav objekt paikneb ahelloendis, kus on rohkem kui üks objekt, kusjuures:
    a. Eemaldatav objekt on oma ahelloendis esimene.
    b. Eemaldatav objekt on oma ahelloendis viimane.
    c. Eemaldatav objekt on oma ahelloendis kusagil keskel.
2. Eemaldatav objekt on oma ahelloendis ainukene, kusjuures:
    a. Kustutatav siduja on *HeaderA* ahelloendis kõige esimene.
    b. Kustutatav siduja on *HeaderA* ahelloendis kõige viimane.
    c. Kustutatav siduja on *HeaderA* ahelloendis kusagil keskel.
3. Etteantud identifikaatoriga objekti ei ole olemas.

6.2.4. Neljas funktsioon

*Node *CreateBinaryTree(HeaderA *pStruct6);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on ehitada kahendotsingu puu, mille tipud viitavad lähtestruktuuris olevatele kirjetele. Võtmeks on seejuures *Objectn* liige *Code*. Väljundparameetriks on viit puu juurtipule. Puu tippu esitav C *struct* on defineeritud failis *Headers.h*.

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Kirje lisamine kahendpuule"*.

6.2.5. Viies funktsioon

*void TreeTraversal(Node *pTree);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule. Funktsiooni ülesandeks on meetodil "vasak-juur-parem" käia läbi kõik puu tipud ja iga tipu puhul trükkida välja tema juurde kuuluva objekti kirjeldus (samuti nagu esimeses funktsioonis).

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Puu läbikäik (3)"*. Magasiniga opereerimiseks kasutage vastavat C *struct*-i failist *Headers.h* ja võtke eeskujuks funktsioonide push ning pop koodid slaididelt "*Näide magasini realisatsioonist (1)"* ja "*Näide magasini realisatsioonist (2)"*

6.2.6. Kuues funktsioon

*Node *DeleteTreeNode(Node *pTree, unsigned long int Code);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule ja võimalikule koodile. Funktsiooni ülesandeks on eemaldada puust tipp, mis viitab etteantud liiget *Code* sisaldavale objektile. Väljundparameetriks on viit pärast eemaldamist saadud puu juurtipule.

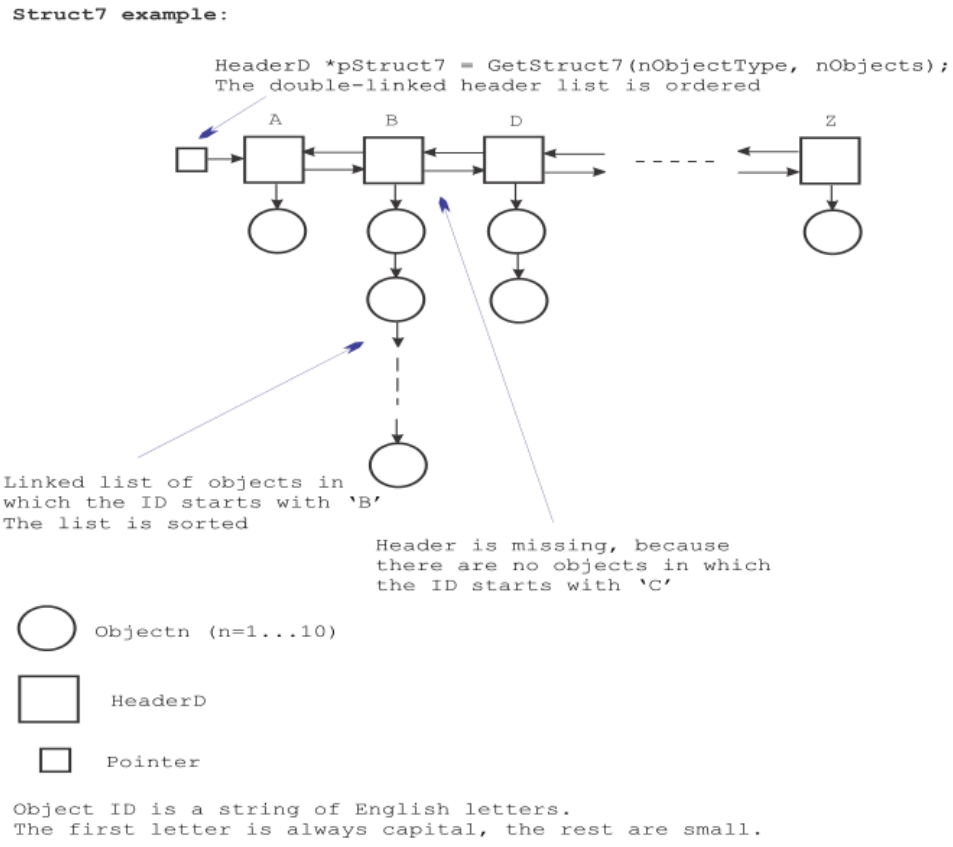
Funktsioon peab olema mitterekursiivne. Tema kirjutamisel lähtuge slaididest *"Kirje eemaldamine kahendpuust (1)"* ja *"Kirje eemaldamine kahendpuust (2)"*.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:
1. Eemaldatav tipp on puu juureks.
2. Eemaldataval tipul ei ole tütartippe.
3. Eemaldataval tipul on ainult parempoolne tütartipp.
4. Eemaldataval tipul on ainult vasakpoolne tütartipp.
5. Eemaldataval tipul on mõlemad tütartipud.
6. Etteantud koodiga kirjet ei olegi.

7. Variant 7

7.1. Lähteandmed

Lähteandmeteks on ahelloenditesse ühendatud objektid, mis on *HeaderD* tüüpi sidujatest koosneva kahekordselt lingitud ahelloendiga ühendatud terviklikuks andmestruktuuriks. Objektid võivad olla tüüpidest *Object1*, *Object2*,…*Object10*. Objekti tüübi annab ette juhendaja. Objektide kui C *struct*-ide deklaratsioonid on esitatud failis *Objects.h* ja neid selgitavad skeemid asuvad käesoleva juhendi lisas. *HeaderD* deklaratsiooni võib leida failist *Headers.h*. Lähtestruktuuri näide (rõhutame, et tegemist on üksnes selgitava näitega) on toodud alljärgneval joonisel:



Objekti identifikaatoriks on kõikidel juhtudel C string, mis koosneb inglise tähestiku väiketähtedest. Stringi esimene sümbol on aga inglise tähestiku suurtäht. Stringi pikkuseks võib olla mistahes nullist suurem arv.

Objektide ahelloend on identifikaatori järgi sorditud (s.t. kõige väiksema identifikaatoriga objekt on see, millele viitab siduja).

Iga *HeaderD* tüüpi siduja vastab ühele kindlale identifikaatori algustähele e. teisiti väljendades – kõik ühe ja sama sidujaga seotud objektid omavad sedasama identifikaatori algustähte. Nii näiteks objektid identifikaatoritega *Jaan*, *Juhan* ja *Johannes* asuvad kõik selles objektide ahelas, mis väljub tähega 'J' seotud sidujast. Sidujad paiknevad *HeaderD* kahekordselt lingitud ahelas vastavalt tähestiku

järjekorrale. Oluline on rõhutada, et kui mingi algustähega seotud objekte ei ole, puudub ka vastav siduja.

Lähtestruktuuri genereerib juhendaja poolt antud funktsioon *GetStruct7* (asetseb objektmoodulis *Structs.obj*, prototüüp on failis *Structs.h*).

7.2. Ülesanne

Laboratoorse töö ülesandeks on kirjutada 6 järgmist funktsiooni:

7.2.1. Esimene funktsioon

*void PrintObjects(HeaderD *pStruct7);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on väljastada kõikide struktuuris paiknevate objektide kirjeldused (iga objekt eraldi reas). Väljastavaks funktsiooniks on *printf*. Väljastamise formaatimise string (*printf* esimene parameeter) on toodud failis *Objects.h.*

See funktsioon tuleb kirjutada kõige esimesena, sest ilma temata ei ole võimalik ülejäänud funktsioonide tööd kontrollida.

7.2.2. Teine funktsioon

*int InsertNewObject(HeaderD **pStruct7, char *pNewID, int NewCode);*

Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderD* tüüpi sidujale; viit uue objekti identifikaatorile ja uue objekti kood. Funktsiooni ülesandeks on luua uus objekt ja lülitada ta andmestruktuuri.

Funktsioon peab esmalt kontrollima, kas uus identifikaator ikka vastab eespool toodud formaadile. Kui see tingimus ei ole täidetud, väljastab funktsioon lihtsalt nulli. Edasi peab funktsioon kontrollima, kas sellise identifikaatoriga objekt ikka tõesti puudub. Kui ta on olemas, väljastab funktsioon samuti lihtsalt nulli. Kui sellise identifikaatoriga objekti ei ole, tuleb ta luua ning lülitada õigesse ahelloendisse. Sealjuures ka uute objektidega täiendatud ahelloend peab olema sorditud.

Kõik vajalikud mäluväljad tuleb funktsiooni *malloc* abil tellida. Ka identifikaatori jaoks on vaja omaette mäluvälja, kuhu sisendparameetriks olev string tuleb kopeerida. Kellaaeg või kuupäev tuleb lugeda arvuti kellalt ja teisendada (vastavad funktsioonid on objektmoodulis *DateTime.obj*, prototüübid on failis *DateTime.h*). Kui funktsioon täitis oma ülesande, väljastab ta suuruse 1.

Võib juhtuda, et etteantud uue identifikaatori algustähega objekte varem ei olnudki ja seetõttu puudub ka vastav siduja. Sellisel juhul tuleb kõigepealt luua uus siduja ja paigutada ta sidujate ahelloendis tema jaoks ettenähtud kohale. Kui uus siduja tuleb kõige esimeseks, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Uus objekt tuleb juba olemasolevasse ahelloendisse, kusjuures:
    a. Uus objekt tuleb oma ahelloendi esimeseks.
    b. Uus objekt tuleb oma ahelloendi viimaseks.
    c. Uus objekt tuleb kusagile oma ahelloendi keskele.
2. Ahelloendit uue objekti jaoks (ja seega ka temale vastavat *HeaderD* tüüpi sidujat) siiamaani ei olnud, kusjuures:

   a. Uus siduja tuleb *HeaderD* ahelloendis kõige esimeseks.
   b. Uus siduja tuleb *HeaderD* ahelloendis kõige viimaseks.
   c. Uus siduja tuleb *HeaderD* ahelloendis kusagile keskele.
3. Uue objekti identifikaator on ebaõiges formaadis.
4. Uue objekti identifikaator on sama mis mõnel juba olemasoleval objektil.

7.2.3. Kolmas funktsioon

Objectn RemoveExistingObject(HeaderD **pStruct7, char *pExistingID);*

Objectn asemel tuleb kirjutada loomulikult *Object1*, *Object2* jne. Sisendparameetriteks on viit viidale, mis omakorda viitab esimesele olemasolevale *HeaderD* tüüpi sidujale ja viit stringile, mis peab kokku langema ühe eeldatavalt olemasoleva objekti identifikaatoriga. Funktsiooni ülesandeks on leida see objekt ja eemaldada ta andmestruktuurist. Eemaldatud objekti kustutada ei tohi. Väljundsuuruseks on viit eemaldatud objektile. Kui objekti ei leitud, on väljundsuuruseks null.

Võib juhtuda, et eemaldatav objekt on oma ahelloendis ainuke olemasolev. Sellisel juhul tuleb pärast objekti eemaldamist eemaldada ning kustutada ka tema algustähele vastav *HeaderD* tüüpi siduja. Kui kustutatud siduja oli kõige esimene, siis muutub ka viit struktuuri algusele. Seetõttu on viit struktuuri algusele nii sisendparameeter kui ka väljundparameeter.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:

1. Eemaldatav objekt paikneb ahelloendis, kus on rohkem kui üks objekt, kusjuures:
   a. Eemaldatav objekt on oma ahelloendis esimene.
   b. Eemaldatav objekt on oma ahelloendis viimane.
   c. Eemaldatav objekt on oma ahelloendis kusagil keskel.
2. Eemaldatav objekt on oma ahelloendis ainukene, kusjuures:
   a. Kustutatav siduja on *HeaderD* ahelloendis kõige esimene.
   b. Kustutatav siduja on *HeaderD* ahelloendis kõige viimane.
   c. Kustutatav siduja on *HeaderD* ahelloendis kusagil keskel.
3. Etteantud identifikaatoriga objekti ei ole olemas.

7.2.4. Neljas funktsioon

*Node *CreateBinaryTree(HeaderD *pStruct7);*

Sisendparameetriks on viit lähtestruktuuri esimesele sidujale. Funktsiooni ülesandeks on ehitada kahendotsingu puu, mille tipud viitavad lähtestruktuuris olevatele kirjetele. Võtmeks on seejuures *Objectn* liige *Code*. Väljundparameetriks on viit puu juurtipule. Puu tippu esitav C *struct* on defineeritud failis *Headers.h.*

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Kirje lisamine kahendpuule"*.

7.2.5. Viies funktsioon

*void TreeTraversal(Node *pTree);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule. Funktsiooni ülesandeks on meetodil "vasak-juur-parem" käia läbi kõik puu tipud ja iga tipu puhul trükkida välja tema juurde kuuluva objekti kirjeldus (samuti nagu esimeses funktsioonis).

Funktsioon peab olema mitterekursiivne, Tema kirjutamisel lähtuge slaidist *"Puu läbikäik (3)"*. Magasiniga opereerimiseks kasutage vastavat C *struct*-i failist *Headers.h* ja võtke eeskujuks funktsioonide push ning pop koodid slaididelt "*Näide magasini realisatsioonist (1)"* ja "*Näide magasini realisatsioonist (2)"*

7.2.6. Kuues funktsioon

*Node *DeleteTreeNode(Node *pTree, unsigned long int Code);*

Sisendparameetriks on viit neljanda funktsiooni poolt ehitatud puule ja võimalikule koodile. Funktsiooni ülesandeks on eemaldada puust tipp, mis viitab etteantud liiget *Code* sisaldavale objektile. Väljundparameetriks on viit pärast eemaldamist saadud puu juurtipule.

Funktsioon peab olema mitterekursiivne. Tema kirjutamisel lähtuge slaididest *"Kirje eemaldamine kahendpuust (1)"* ja *"Kirje eemaldamine kahendpuust (2)"*.

Funktsiooni katsetades proovige kindlasti läbi järgmised olukorrad:
1. Eemaldatav tipp on puu juureks.
2. Eemaldataval tipul ei ole tütartippe.
3. Eemaldataval tipul on ainult parempoolne tütartipp.
4. Eemaldataval tipul on ainult vasakpoolne tütartipp.
5. Eemaldataval tipul on mõlemad tütartipud.
6. Etteantud koodiga kirjet ei olegi.

8. Esimesed sammud

1. Käivitage Visual Studio (alljärgnevalt eeldatakse, et Teil on Visual Studio 2022).
2. Looge uus C++ Windows Console Empty Project. Projekti nimi valige ise. Projekti kataloogi valikul on soovitatav kasutada Visual Studio poolt pakutut.
3. Visual Studio **Solution Explorer** aknas klõpsake hiire parema klahviga projekti nime ikooni. Avanevast menüüst valige **Add ➔ New item**. Avanevast dialoogaknast valige **C++ file** ning andke talle nimeks *Main*.
4. Kirjutage faili *Main.cpp* järgmine tekst[2]:

```
#include "stdio.h"

#pragma warning ( disable : 4996 )

int main()
{
   printf("First run\n");
   return 0;

}
```

5. Viige kursor reale, kuhu on kirjutatud *return* ning vajutage **F9**-le. Sellega panete sinna katkestuspunkti (*breakpoint*). Silumisel peatub programm katkestuspunktis ja Te saate selle hetkeni tehtud töö üle vaadata.
6. Valige menüüst **Build ➔ Build solution**. Visual Studio kompileerib ja lingib Teie programmi. Ärge kasutage **Build ➔ Rebuild solution** - see võib kustutada juhendaja antud failid.
7. Valige menüüst **Debug ➔ Start debugging**. Programm trükib lause *First run* ja peatub katkestuspunktis.
8. Valige menüüst **Debug ➔ Stop debugging**. Programm lõpetab töö.
9. Teie poolt valtud projekti kataloogi peab olema lisandunud õige mitu uut kataloogi, nende nimed sõltuvad ka sellest, milline protsessor (32 või 64 bit) Teie arvutis on. Otsige üles kataloog kus on *Main.cpp* ja samuti kataloog kus on *Main.obj*.
10. Laadige IAS0090 õppematerjalide lehelt töös vajaminevad failid *DateTime.h*, *Headers.h*, *Objects.h*, *Structs.h*, *DateTime.obj*, *Objects.obj* ja *Structs.obj* ning paigutage nad selliselt:

    *.h failid kataloogi kus on *Main.cpp*

    *.obj failid kataloogi kus on *Main.obj*

11. Visual Studio **Solution Explorer** aknas klõpsake hiire parema klahviga ikooni **Header files**. Avanevast menüüst valige **Add ➔ Existing item**. Avanevast dialoogaknast valige kõik 4 ülalnimetatud .h faili.
12. Visual Studio **Solution Explorer** aknas klõpsake hiire parema klahviga projekti nime ikooni. Avanevast menüüst valige **Add ➔ Existing item**. Avanevast dialoogaknast valige kõik 3 ülalnimetatud .obj faili.

---

[2] *#pragma* on vajalik mõningate meie töös mittevajalike kompilaatori hoiatuste mahasurumiseks.

13. Täiendage faili *Main.cpp*:

```
#include "stdio.h"
#include "DateTime.h"
#include "Objects.h"
#include "Headers.h"
#include "Structs.h"

#pragma warning ( disable : 4996 )

int main()
{
   // Kirjutage lähtestruktuuri genereeriv lause. See on:
   // a) Struct1 puhul:
   // ObjectO **pStruct = (ObjectO **)GetStruct1(O, N);
   //
   // b) Struct2 puhul:
   // HeaderA *pStruct = GetStruct2(O, N);
   //
   // c) Struct3 puhul:
   // HeaderB *pStruct = GetStruct3(O, N);
   //
   // d) Struct4 puhul:
   // HeaderC *pStruct = GetStruct4(O, N);
   //
   // e) Struct5 puhul:
   // HeaderA **pStruct = GetStruct5(O, N);
   //
   // f) Struct6 puhul:
   // HeaderA *pStruct = GetStruct6(O, N);
   //
   // g) Struct7 puhul:
   // HeaderD *pStruct = GetStruct7(O, N);
   //
   //    O asemel kirjutage juhendaja poolt antud objekti indeks.
   //    N asemel kirjutage juhendaja poolt antud objektide arv[3].
   //    N ei ole kunagi 0
   //
   return 0;
}
```

14. Valige menüüst **Build ➔ Build solution**. Veenduge, et Visual Studio suudab Teie programmi kompileerida ja linkida.
15. Sellega on projekti ettevalmistused lõppenud ja Te võite hakata tegelema nõutud tarkvara arendamisega.

[3] Kui juhendaja ei ole objektide arvu ette andnud, siis programmi katsetamisel piisab kui 25 < N < 35.

9. Kaitsmine

9.1. Kaitsmise kord

Kaitsmisel tuleb demonstreerida realiseeritud tarkvara tööd ja vastata tööga seotud lisaküsimustele. Eraldi aruannet ei nõuta. Tarkvara töö demonstreerimine toimub Visual Studio töökeskkonnas. Tarkvara loetakse õigesti töötavaks, kui juhendaja antud testid jooksevad korrektselt. Lisaks võib juhendaja paluda üliõpilasel selgitada oma tööd või viia sealsamas 15 kuni 30 minuti jooksul koodi sisse mõningaid väiksemaid muudatusi. Ebaõigesti töötavaid või üldse mittetöötavaid funktsioone vastu ei võeta.

Kaitsmine toimub kahes osas:

1. Esimene, teine ja kolmas funktsioon annavad kokku 15 punkti eeldusel et kõik kolm on edukalt kaitstud enne 11-nda õppenädala algust (s.t. enne 13.11.22). Kaitsmine pärast tähtaja möödumist on võimalik, kuid teenitud punktide arv kahaneb sellisel juhul 10-le.
2. Neljas, viies ja kuues funktsioon annavad kokku 15 punkti eeldusel et kõik kolm on edukalt kaitstud enne 16-nda õppenädala algust..

Need, kellel on esimene ja/või teine osa lõpetamata ning punktid saamata, võivad tulla näitama on kaitsmata pooleliolevat tööd viimasel õppenädalal. Hindamine ja punktide arvestus toimub siis järgmiselt:

- Iga korrektselt töötava funktsiooni eest 3 punkti.
- Iga kirjutatud kuid vigadega töötava funktsiooni (s.t. mõned testid töötavad ja mõned mitte) eest 2 punkti.
- Iga kirjutatud kuid üldse mittetöötava funktsiooni (s.t. ükski test ei tööta kuid fail kompileerub) eest 1 punkti.
- Puuduva või kompileerimisvigadega funktsiooni eest 0 punkti.

Kaitsmiseks tuleb igal üliõpilasel isiklikult kohale tulla. Elektrooniliselt (e-mail, GitHub, jne.) saadetuid töid arvesse ei võeta ja läbi ei vaadata.

9.2. Kaitsmise mõju eksami tulemustele

Kõik, kellel ÕIS lubab ennast eksamile registreerida, võivad ka tulla. Mitte mingeid lisatingimusi ei ole.

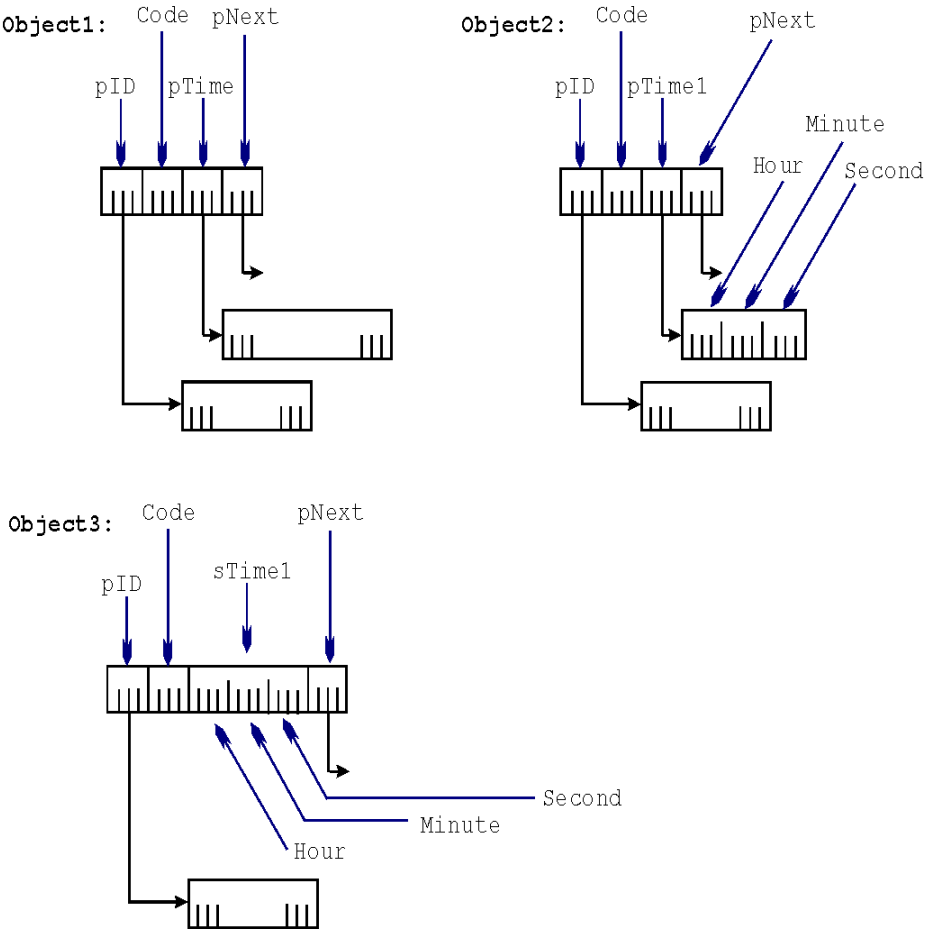
Eksam on kirjalik ning vastata tuleb kümnele küsimusele (need on kõigile antud eksamil viibijatele ühised). Kestvus 1.5 tundi. Kokku saab maksimaalselt 30 punkti. Mitte mingisuguste abimaterjalide kasutamine ei ole lubatud. Vastused võivad olla eesti, vene või inglise keeles.

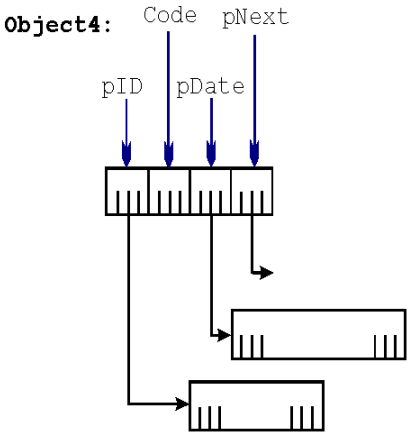
Lõpphinne arvutatakse eksamil ja laboris saadud punktide kogusummast järgmise tabeli alusel:
"0" – alla 30 punkti;
"1" – 30 - 34 punkti;
"2" – 35 - 39 punkti;

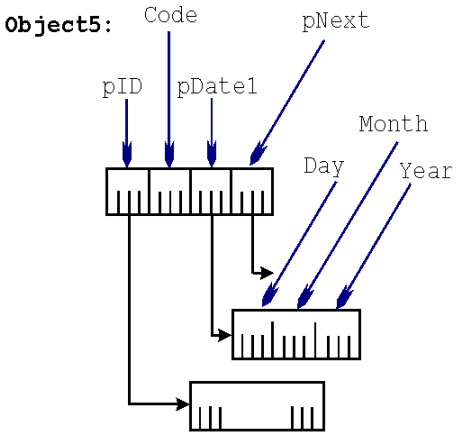
"3" – 40 - 44 punkti;
"4" – 45 - 49 punkti;
"5" – 50 ja enam punkti.

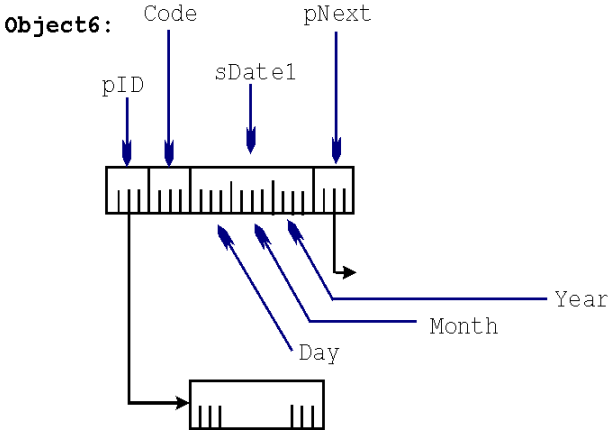
Kuid need, kes said eksamil vähem kui 6 punkti, loetakse läbikukkunuks sõltumata laboris saadud punktide arvust.

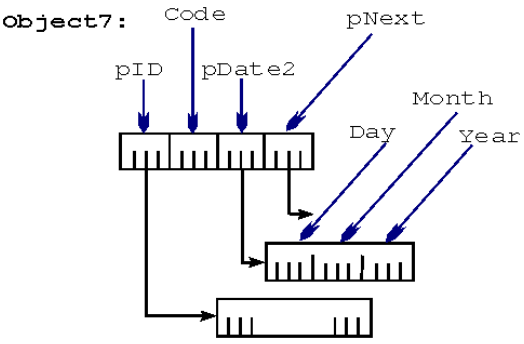
10. Lisa 1: objektide skeemid

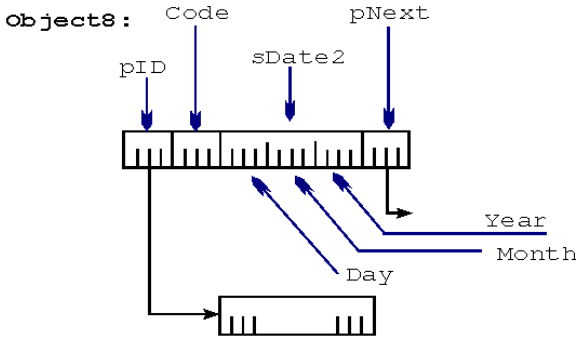


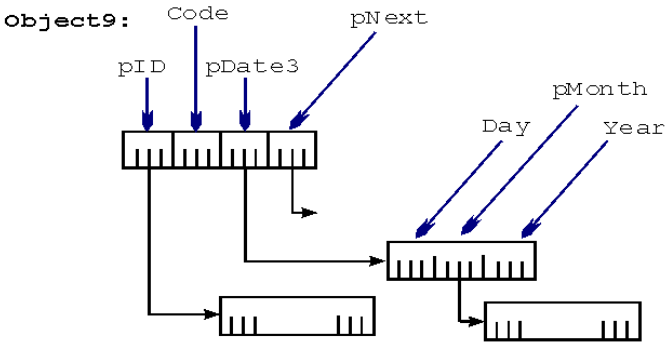

Object4:
Code
pNext
pID
pDate

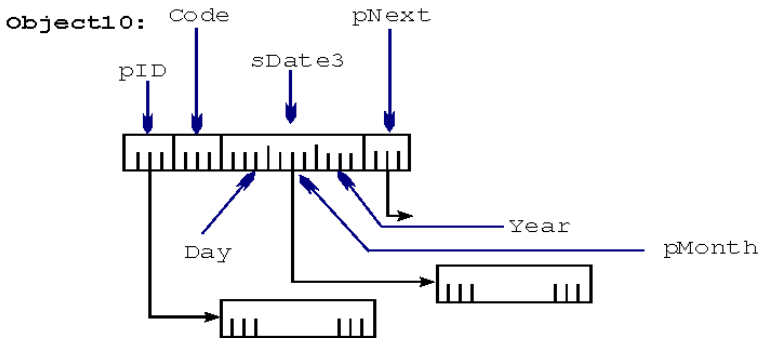

Object5:
Code
pNext
pID
pDate1
Month
Day
Year


Object6:
Code
pNext
pID
sDate1
Year
Month
Day


Object7:
Code
pNext
pID
pDate2
Month
Day
Year


Object8:
Code
pNext
pID
sDate2
Year
Month
Day


Object9:
Code
pNext
pID
pDate3
pMonth
Day
Year


Object10:
Code
pNext
pID
sDate3
Year
Day
pMonth

11. Lisa 2: isikliku arvuti kasutamise kord

Üliõpilasel on õigus teha laboritööd kas osaliselt või täielikult oma isiklikul arvutil. Selleks tuleb aga kõigepealt oma arvutisse installeerida Visual Studio või viimase lihtsustatud versioon Visual Studio Community (nimetatud ka Visual Studio Express Edition). Viimase näol on tegemist maksuta saadava vabavaraga (https://visualstudio.microsoft.com/downloads/). Sobivad on kõik väljalasked alates 2017 aastast.

Üliõpilane, kes tegi laboritöö oma isiklikul arvutil, peab sellesama arvutiga ilmuma ka kaitsmisele.

Tähelepanuks neile, kelle isikliku arvuti operatsioonisüsteemiks ei ole Windows: kui Teil erinevate operatsioonisüsteemidega töötamise alal teadmisi ja kogemusi napib, võite sattuda tõsiste probleemide ette. Allalaaditavad objektmoodulid (vt. *Esimesed sammud*) on mõeldud ainult Windows-ile ja Visual Studio-le. Teistes töökeskkondades nende sobivust katsetatud ei ole.

12. Lisa 3: kontrolltestid

12.1. Esimese osa kontrolltestid

12.1.1. Variandid 1A ja 1B

1. N =35.
2. Väljastada lähtestruktuur.
3. Lisada antud järjekorras objektid identifikaatoritega: Ax, Ab, Az, Ak, Aa, Pa, Pg, Px, Wa, Wb, wk, Wb, WW, W8, W_ ja väljastada muudetud struktuur. Liikme Code väärtuseks võib valida suvalise postiivse täisarvu. Märkus: viie viimase objekti lisamine peab andma veateate.
4. Samas järjekorras eemaldada lisatud objektid ja väljastada muudetud struktuur. Märkus: viie viimase objekti eemaldamise katse peab andma veateate.

12.1.2. Variandid 2A, 2B, 6 ja 7

1. N =35.
2. Väljastada lähtestruktuur.
3. Lisada antud järjekorras objektid identifikaatoritega: Dx, Db, Dz, Dk, Aa, Wu, Wa, Zw, Za, wk, Wa, WW, W8, W_ ja väljastada tulemus. Liikme Code väärtuseks võib valida suvalise postiivse täisarvu. Märkus: viie viimase objekti lisamine peab andma veateate.
4. Samas järjekorras eemaldada lisatud objektid ja väljastada muudetud struktuur. Märkus: viie viimase objekti eemaldamise katse peab andma veateate.

12.1.3. Variandid 3, 4 ja 5

1. N =35.
2. Väljastada lähtestruktuur.
3. Lisada antud järjekorras objektid identifikaatoritega: Dx Gz, Dx Ga, Db Aa, Dk Za, Dr Wa, Aa Aa, Ab Ba, Za Aa, Za Ab, Za Ba, Wx Xa, Wx Aa, zb Kk, Zc ca, Dr Wa, ZB kk, Fa, Fa_Fa ja väljastada tulemus. Liikme Code väärtuseks võib valida suvalise postiivse täisarvu. Märkus: kuue viimase objekti lisamine peab andma veateate.
4. Samas järjekorras eemaldada lisatud objektid ja väljastada muudetud struktuur. Märkus: kuue viimase objekti eemaldamise katse peab andma veateate.

12.2. Teise osa kontrolltestid

1. N=35.
2. Väljastada lähtestruktuur.
3. Moodustada kahendpuu ja käies läbi kõik tema tipud väljastada tippude juurde kuuluvad objektid.

4. Eemaldada puu juurtipp ja käies läbi kõik uue puu tipud väljastada tippude juurde kuuluvad objektid.
5. N=10.
6. Väljastada lähtestruktuur.
7. Moodustada kahendpuu ja käies läbi kõik tema tipud väljastada tippude juurde kuuluvad objektid.
8. Joonistada saadud kahendpuu paberile, märkides tippude juurde üksnes nende võtmed.
9. Eemaldada kahendpuust juhendaja näidatud võtmetega tipud ja käies läbi kõik uue puu tipud väljastada tippude juurde kuuluvad objektid.
10. Teha katse eemaldada tipp, mida ei olegi olemas.