AS-0.101 C/C++ -ohjelmoinnin peruskurssi / Tentti 4.5.2004/Hannu Laine

Vastaa neljään kysymykseen! Tentin arvosteluasteikko on 0 – 24 pistettä. Kaikkien kysymysten painoarvo on sama (6 pistettä/tehtävä). **Lähtökohtana on, että kysymyksessä 3 käytetään C++-kieltä ja muissa C:tä.**

Palauta jokainen tehtävä erillisellä konseptilla!

1.

Kun näppäimistöltä halutaan lukea merkkijono, on ongelmana se, että lukufunktion kutsujan vastuulla on tarjota puskuri, jonne luettavat merkit tallennetaan. Pahimillaan tämä ongelma aiheuttaa ohjelman kaatumisen, jos syötetään enemmän merkkejä kuin puskuriin mahtuu. Näin käy esimerkiksi silloin, kun käytetään funktiota gets tai funktiota scanf muotomääreellä %s. Ongelmaa voidaan pienentää käyttämällä funktiota fgets tai muotomäärettä %ns, missä n on kentän leveys. Tällöinkin ongelmaksi jää se, että luettu merkkijono ei sisällä kaikkia näppäimistöltä syötettyjä merkkejä, jos niitä on enemmän kuin puskuriin mahtuu.

Kirjoita funktio, get_string, jossa nämä ongelmat on poistettu siten, että funktio huolehtii itse tilanvarauksesta sen mukaan montako merkkiä käyttäjä syöttää näppäimistöltä. Funktio palauttaa sitten osoittimen luettuun merkkijonoon. Etukäteen ei ole siis olemassa periaatteessa mitään rajaa syötettävien merkkien määrälle.

Huomautus 1. Muodostettava merkkijono on tietysti aito c-kielen stringi, joka sisältää päätösnollan, eikä siinä saa olla varattuna ylimääräistä tilaa.

Huomautus 2. Syöte loppuu rivinvaihtoon ('\n').

2.

Projektissa käsitellään erilaisia "asioita". Asian käsittelyssä tarvittavat asian tiedot sisältyvät tietotyyppiin Thing. Koska asioita ei pystytä käsittelemään siinä tahdissa kuin niitä tulee käsiteltäväksi, ne laitetaan asioiden jonoon. Tätä varten on olemassa valmiina abstrakti tietotyyppi asiajono, joka muodostuu tietotyypistä Tqueue ja sen käsittelyyn tarkoitetuista funktioista:

```
void initializeQueue(Tqueue *pqueue);       //alustaa jonon

void dequeue( Tqueue *pqueue, Thing *pthing); //ottaa jonosta asian

void enqueue(Tqueue *pqueue, Thing thing);   //laittaa asian jonoon

int isEmptyQueue(Tqueue queue);   //testaa onko jono tyhjä (tulos 1)
                                  //vai ei (tulos 0).
```

Nyt kuitenkin halutaan priorisoida asioiden käsittelyä. Tätä varten tarvitaan asioiden prioriteettijono, jossa asialle annetaan prioriteetti silloin, kun se laitetaan jonoon. Mahdolliset prioriteetit ovat 0, 1, 2, ..., n, missä n on prioriteettijonon alustuksen yhteydessä annettava luku. Asioita prioriteettijonosta pois otettaessa ne tulevat prioriteetin mukaan kuten prioriteettijonosta kuuluu (eli korkeamman prioriteetin asia aina ennen matalamman prioriteetin asiaa).

Eräs kätevä tapa toteuttaa prioriteettijono on käyttää n (n on prioriteettien määrä) kappaletta tavallisia jonoja ja laittaa asiat prioriteetin mukaan sisäisesti omiin jonoihinsa. Ulos otettaessa asia otetaan aina korkeimman prioriteetin jonosta, jossa asioita on.

Tehtävänä on **toteuttaa asioiden prioriteettijono** ylläkuvatulla periaatteella eli määritellä tietotyyppi TPriQueue ja toteuttaa sen käyttöfunktiot:

```
void initializePriQueue(TPriQueue *priqueue, int n); //n on prioriteettien
                                                     //määrä
void dequeuePri(TPriQueue *priqueue, Thing *thing);
void enqueuePri(TPriQueue *priqueue, Thing thing, int priority);
int isEmptyPriQueue(const TPriQueue *priqueue);
```

Huomautus. Tässä prioriteetti on sitä korkeampi, mitä suurempi arvo sillä on.

3.

On olemassa luokka Die, joka määrittelee nopan ominaisuudet.. Luokan Die nopalla voidaan generoida satunnaisia kokonaislukuja halutulta alueelta.

Luokalla Die on jäsenfunktiot
- Die (konstruktori)
- roll
- reset

Konstruktorilla asetetaan satunnaislukujen alueen alaraja ja yläraja. Jäsenfunktiolla roll saadaan seuraava satunnaisluku (pseudosatunnaislukujen) sarjasta. Funktio reset resetoi nopan. Tämä tarkoittaa käytännössä sitä, että satunnaislukugeneraattorille annetaan uusi siemenluku. Tällä "perusnopalla" on ominaisuus, että se "ei muista" jo arvottuja lukuja. Tästä seuraa, että heitettäessä noppaa useita kertoja saattaa sama luku tulla uudelleen.

Eräässä sovelluksessa (esimerkiksi lottorivin numeroiden arvonnassa) tarvitaan noppaa, joka useamman kerran peräkkäin heitettäessä (loton tapauksessa 7 kertaa) ei anna samaa numeroa uudelleen. Kehitä tällainen "erikoisnoppa". Erikoisnopan konstruktorissa voidaan kertoa paitsi halutun satunnaislukualueen alaraja ja yläraja myös sen satunnaislukusarjan pituus, jossa ei saa esiintyä duplikaatteja. Erikoisnopan resetointi tarkoittaa paitsi uuden siemenluvun antamista myös muistin nollaamista. Lukualue, samoin

kuin haluttu duplikaatittoman sarjan pituus, säilyy resetoinnissa (tällaista resetointia tarvitaan esim. tilanteessa, kun lähdetään arpomaan itselle uutta seuraavaa veikattavaa lottoriviä).

Kirjoita tällaisen erikoisnopan **luokan DieWithNoDuplicates esittely ja sen kolmen jäsenfunktion toteutukset** hyödyntäen tietysti noppaluokkaa Die.

4.

Toteuta ns. rengaspuskuri käyttäen renkaaksi linkitettyä listaa. Tällaisen rakenteen perusajatus on se, että linkatun listan viimeiseen solmuun ei laiteta NULL osoitetta, vaan linkki takaisin ensimmäiseen solmuun. Tällainen rakenne tarjoaa tehokkaan ja kätevän menetelmän pitää muistissa aina viimeisen n:n tapahtuman tiedot, kun uusia tapahtumia syntyy koko ajan. Tällä kertaa nämä tapahtumat ovat yksinkertaisesti yksi kerrallaan jatkuvasti syötettäviä merkkejä ja halutaan, että tallessa on aina n viimeksi syötettyä merkkiä. Toteuta tätä tarkoitusta varten yllämainitulla periaatteella ADT rengaspuskuri, joka muodostuu tietotyypistä TRingBuffer ja operaatiofunktioista

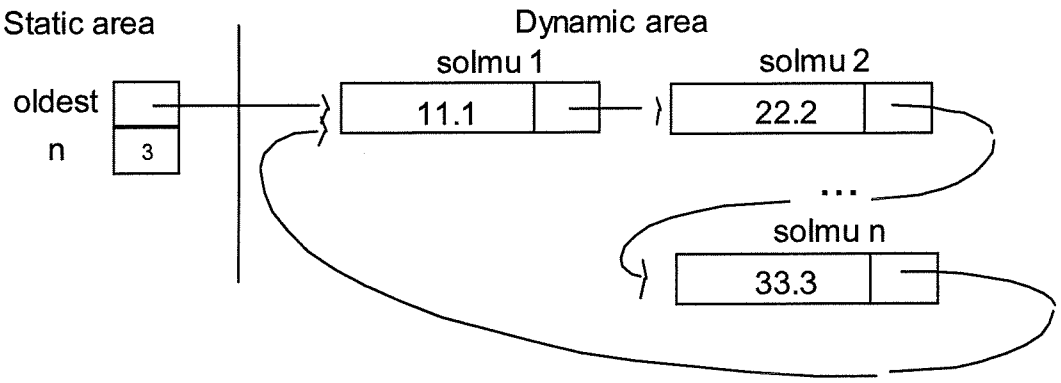
```
void initialize(TRingBuffer *ring, Titem *initialValues, int n);
void add(TRingBuffer *ring, Titem item);
void display(const TRingBuffer *ring);
```

Koska vaaditaan, että rengaspuskurissa olevien solmujen määrä määrätään ajon aikana (kun initialisointifunktiota kutsutaan) määrittele tietotyyppi TRingBuffer siten, että se sisältää solmujen lukumäärän ja tämän lisäksi tietysti osoittimen solmuun, jossa on ko. hetkellä vanhin arvo.

Initialize funktio luo renkaaksi linkitetyn listan varaamalla solmuille tilaa dynaamisesta muistista ja tallentaen niihin datan parametrina (initialValues) välitetystä taulukosta. Funktio add vain korvaa rengaspuskurissa ennestään olleen vanhimman arvon uudella parametrissa item annetulla arvolla. Tämä funktio ei siis enää varaa tilaa uudelle solmulle. Funktio display vain tulostaa ruutuun kaikki rengaspuskurissa olevat arvot (tässä tapauksessa) merkit alkaen vanhimmasta.

Esimerkki ohjelman toiminnasta: Jos initialisointifunktiota kutsutaan kokoarvolla 3 ja taulukolla, jossa on merkit a, b ja c, niin funktiolla display tulostuisi tietysti a, b, c. Jos tämän jälkeen add-funktiolla lisättäisiin d, niin display funktio tulostaisi b, c, d. Jos tämän jälkeen vielä lisättäisiin e, niin tulostuisi c, d, e, jne.

Alla oleva kuva valaisee asiaa.



Huomautus. **Tehtävänä on siis kirjoittaa tarvittavat tietomäärittelyt tietotyypille TRingBuffer ja toteutukset yllä mainituille kolmelle funktiolle**. Mitään sovellusta tai main-funktiota ei tarvitse kirjoittaa.

5.

Abstrakteja tietotyyppejä voidaan toteuttaa C:ssä tietueilla ja funktioilla ja C++:ssa luokilla. Vertaile näitä tapoja keskenään.