Enkeltlenket liste

1.Mars. 2017

Sturla Gunnerød
Høgskolen i sør-øst Norge

Innhold

Innledning

En enkeltlenket liste er et kraftig hjelpemiddel for å lagre en liste med informasjon. Listen er dynamisk og består av objekter som er «sydd» sammen. Det vil si at objektene har en peker som referer til neste objekt i kronologisk rekkefølge. I denne oppgaven har vi pekeren «head» som alltid vil peke til første posisjon i listen. Det siste objektet i listen vil peke på «null» som markerer slutten av listen.

Ved hjelp av ulike metoder som vi implementer i denne oppgaven kan vi legge til objekter og endre på objekter. Denne oppgaven lagrer tall, men kan også skrives om til å lagre tekststrenger eller tegn. I det større bilde vil man kanskje ønske å heller lagre objekter. Da vil det som vi kaller «head.value» i oppgaven peke til dette objektet.

Oppgavetekst

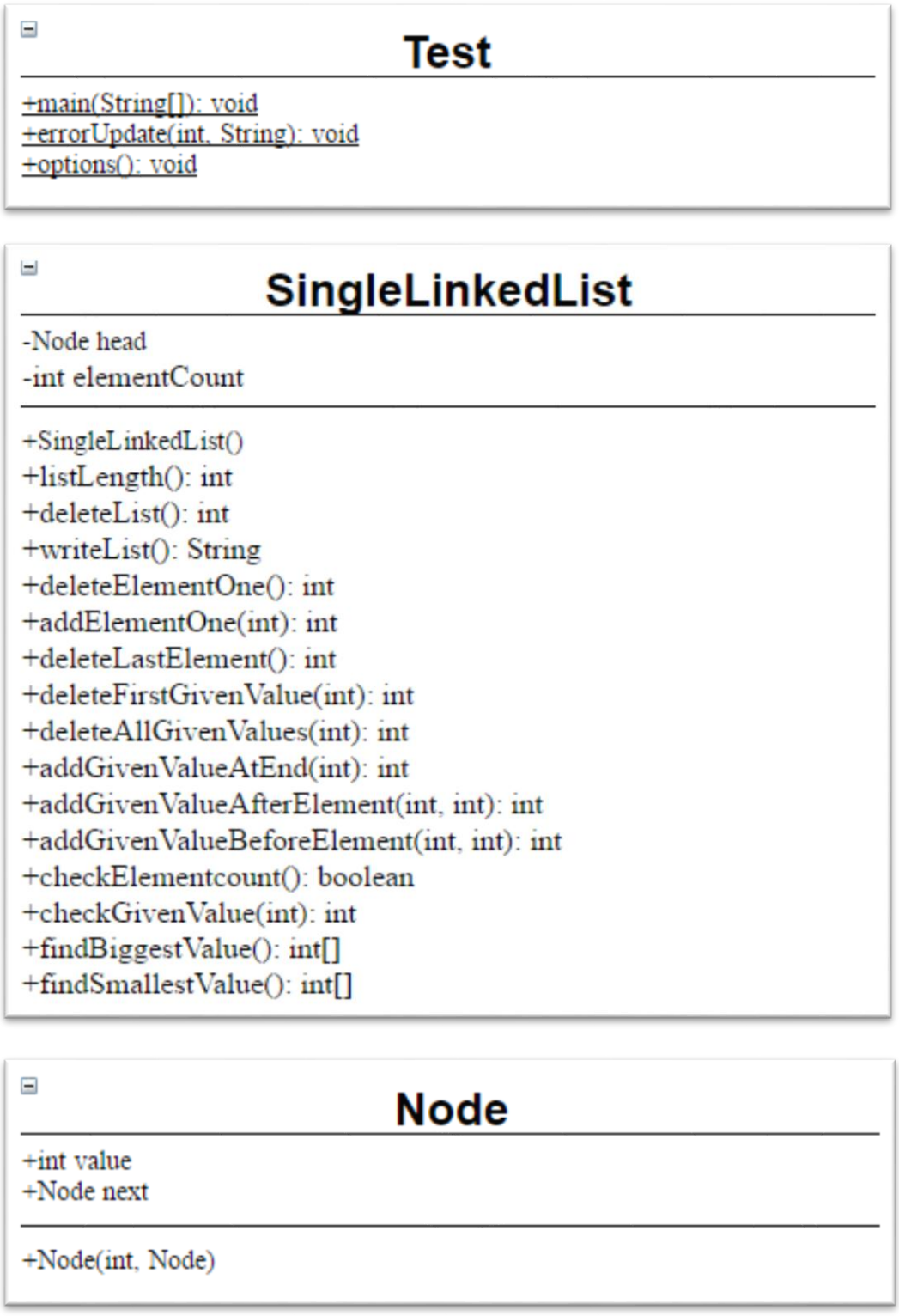
Skriv et program der du tar utgangspunkt i klassene Node og SingleLinkedList (forelesning nr. 4) og benytter disse til programmeringen. Du skal ha et hovedprogram (main), som ligger i egen klasse. Dette skal inneholde en meny (liste av tilgjengelige kommandoer) med tanke på å legge inn data, fjerne data, skrive ut og andre funksjoner for en lenket liste. I menyen listes alle tilgjengelige kommandoer opp. Programmet skal avsluttes ved eget valg. Listen skal ikke være sortert og det kan forekomme like tall. Det er ikke nødvendig med en GUI-løsning. Følgende funksjoner skal implementeres og legges inn som metoder i klassen SingleLinkedList:

a. Slett det første elementet i listen.
b. Slett det siste elementet i listen.
c. Slett et element med oppgitt verdi fra listen (første forekomst).
d. Slett alle element med oppgitt verdi fra listen.
e. Legg til et element med oppgitt verdi i starten av listen.
f. Legg til et element med oppgitt verdi i slutten av listen.
g. Legg til et element etter et element med oppgitt verdi (én gang).
h. Legg til et element foran et element med oppgitt verdi (én gang).
i. Skriv ut lengden på listen.
j. Sjekk om listen listen 'stemmer' (elementCount er riktig). Returner true dersom antall stemmer, false ellers.
k. Tell opp antall forekomster av element med gitt verdi i lista, dette antallet skrives ut.
l. Skriv ut hele listen. 5 elementer pr. linje.
   (Denne metoden kan benytte println-metoden.)
m. Slett hele listen. Hvor mange elementer som ble slettet, skrives ut.
n. Finn og skriv ut verdi av største element i listen.
o. Finn og skriv ut verdi av minste element i listen.

Det skal ikke gjøres endringer i datafeltene for klassene Node og SingleLinkedList. Alle metode-navn og variabler skal være på engelsk.

Klassene skal heller ikke gjøre utskrift eller lese noe inn fra bruker (bortsett fra metode 12).

UML-skjema


Test
+main(String[]): void
+errorUpdate(int, String): void
+options(): void
SingleLinkedList
-Node head
-int elementCount
+SingleLinkedList()
+listLength(): int
+deleteList(): int
+writeList(): String
+deleteElementOne(): int
+addElementOne(int): int
+deleteLastElement(): int
+deleteFirstGivenValue(int): int
+deleteAllGivenValues(int): int
+addGivenValueAtEnd(int): int
+addGivenValueAfterElement(int, int): int
+addGivenValueBeforeElement(int, int): int
+checkElementcount(): boolean
+checkGivenValue(int): int
+findBiggestValue(): int[]
+findSmallestValue(): int[]
Node
+int value
+Node next
+Node(int, Node)

Generell tankegang

For mange av de metodene som skal implementeres kreves det å komme i riktig posisjon i listen. Som sagt i innledningen har vi «head», som peker på starten av listen, og «null», som peker på slutten av listen. Disse to pekerne er essensielle for at vi i det hele tatt skal kunne traversere igjennom listen. Vi vil aldri endre på «head» sin posisjon fordi vi alltid må ha et utgangspunkt.

Der hvor metodene ønsker at vi skal legge til eller endre objekter som er enten i midten av listen eller på sutten av listen, så må vi traversere igjennom den slik at vi får stilt oss i riktig posisjon. Vi lager vår egen peker, «cPtr» (currentPointer), som alltid kopierer «head» sin peker. På den måten har vi lagd to like pekere, og vi trenger ikke å flytte på «head».

Ved hjelp av en while-loop eller en for-loop gjør det mulig å traverser igjennom listen. Loopene er så å si like gode, men for at en for-loop skal kunne fungere, må vi vite for langt vi skal bevege oss. I denne oppgaven har vi «int elementCount» som holder styr på hvor lang listen er. Dermed kan vi ved hjelp av «elementCount» vite hvor langt vi skal gå uten å bevege oss ut av listen. Ved en while-loop trenger vi kun «null» som referansepunkt, siden «null» peker på slutten av listen. Inne i både for- og while-loopen vil vi flytte på «cPtr». Det gjøres ved å sette cPtr sin peker lik cPtr.next. Det vil si at vi flytter «cPtr'en» et element framover for hver gang loopen kjøres.

Det vil også oppstå enkelttilfeller i hver metode. Man burde være på vakt hvor tilfelle er at:

1. Listen er tom
2. Objektet skal legges på starten av lista
3. Objektet skal legges på slutten av lista
4. Objektet skal legges i midten av lista
5. Elementet man søker etter ble ikke funnet

Med dette i tankene skal vi nå gå løs på fremgangsmåten.

Fremgangsmåte

1.Slett det første elementet i listen

Her må vi sjekke at listen ikke er tom fordi vi ikke kan slette et element som ikke er der. Dette kan gjøres ved å sjekke om «head» peker til «null». Det vil si at head, som peker til første posisjon i listen, vil i dette tilfelle peke til slutten av listen. Vi kan også ved hjelp av «elementCount» sjekke om antallet objekter i listen er lik 0. Hvis listen inneholder et element vil vi gjøre slik at head peker til posisjon nummer to i listen. Gjøres ved at pekeren til «head» blir satt til «head.next».

2.Slett det siste elementet i listen

Her kan også tilfelle med at listen er tom forekomme. Forklart i 1. fremgangsmåte. Hvis det kun er et element i listen. Vil vi kunne bruke forrige metode. Hvis det er flere enn et element i listen må vi flytte oss i riktig posisjon ved hjelp av «cPtr». Som sagt i tankegang kan vi enten bruke while- eller for-loop for å oppnå dette. Her er det viktig å huske på at vi vil stoppe et element foran det siste. Slik at vi kan sette det nestesiste elementet til å peke til «null», altså slik at nestsiste element blir siste element.

3.Slett et element med oppgitt verdi fra listen (Første forekomst)

I denne metoden har vi unike tilfeller hvor listen er tom eller kun har ett element, forklart i 1. og 2 fremgangsmåte. Hvis vi skal slette et element når listen er større enn én må vi ha en while-loop som sjekker at pekeren våres ikke peker på «null», men også at oppgitt verdi ikke har blitt funnet. Hvis noen av disse tilfellene forekommer stopper while-loopen. Så må vi ha en if-test som sjekker om vi fant verdien vi lette etter, eller om vi gikk igjennom hele listen uten å finne verdien.
I denne metoden har vi flyttet på to pekere. Vi har flyttet på en peker «pPtr» (previousPointer) og «cPtr», hvor «pPtr» ligger et element bak «cPtr». Ved å sette «pPtr» sin peker til «cPtr.next» vil vi slette elementet i mellom, og få ønsket resultat.

4.Slett alle element med oppgitt verdi fra listen

Her har vi alle de samme tilfellene som forrige metode. Det vi rett og slett trenger her er en for-loop utenfor for forrige metode. En for-loop som traverserer igjennom listen baklengs. Dette er fordi hvis listen er for lang vil den ikke slette alle oppgitte elementer. Grunnen til det er at «int i» - verdien våres i for-loop argumentet vil stige, mens lengden på listen vil synke, noe som gjør at de vil kun gå igjennom listen halvveis. Dette løses ved å sette denne «int i»-verdien til listLength()

metoden. Dette tallet fra listLength() vil oppdatere seg hver gang vi går igjennom for-loopen.

5.Legg til et element med oppgitt verdi i starten av listen

Her har vi ingen unike tilfeller. Vi går rett på sak og lager en ny node «aNode». «aNode» sin peker blir satt til «head». Så blir «head» sin peker satt til «aNode». Det er viktig å gjøre det i nevnt rekkefølge, eller så vil vi miste «head» sin peker og vi mister resten av lista.

6.Legg til et element med oppgitt verdi i slutten av listen

I denne metoden har vi også tilfelle hvis listen er tom. Da bruker vi forrige metode. Hvis listen er større enn null så må vi traversere igjennom listen slik at vi havner på siste element. Vi lager en ny node «aNode». Når vi lager «aNode» lar vi dens peker bli satt til «null». «cPtr» står på siste element, og vi kan dermed sette «cPtr» sin peker til den nye «aNode'en».

7.Legg til et element etter et element med oppgitt verdi (en gang)

De unike tilfellene som kan forekomme i denne metoden er at listen er tom, vi fant ikke søkeverdien og søkeverdien er siste element. Listen er tom er forklart i metode 1, og vi fant ikke søkeverdien er forklart i metode 3. Hvis søkeverdien er siste element bruker vi forrige metode. Vi traverserer igjennom listen ved hjelp av en while-loop, som sjekker om vi har funnet søkeverdien eller vi har nådd slutten av listen. Dermed bruker vi en if-test som sjekker om vi fant søkeverdien før vi nådde slutten, deretter en else-if som sjekker om søkeverdien er siste verdien i lista. Ved sistnevnte forekomst vil vi bruke forrige metode. Ved forekomst av søkeverdi i midten av listen vil den nye «aNode'en» sin peker bli satt til «cPtr.next» sin peker, og «cPtr.next» sin peker bli satt til «aNode».

8.Legg til et element før et element med oppgitt verdi (en gang)

De unike tilfellene som kan forekomme i denne metoden er at listen er tom, vi fant ikke søkeverdien og søkeverdien er første element. Listen er tom er forklart i metode 1, og vi fant ikke søkeverdien er forklart i metode 3. Hvis søkeverdien er første element bruker vi metode 5. I denne metoden vil vi også trenge en «pPtr» (previousPointer) node. Vi traverserer igjennom listen ved hjelp av en while-loop, som sjekker om vi har funnet søkeverdien eller vi har nådd slutten av listen. Dermed bruker vi en if-test som sjekker om vi nåde slutten av listen eller om vi fant verdien. Her vil «pPtr» ligge et element bak «cPtr». Dette oppnås ved å sette «pPtr» sin peker til «cPtr», og «cPtr» sin peker til «cPtr.next» sin peker inne i

while-loopen. Vi oppretter en ny «aNode». «aNode.next» sin peker blir satt til «cPtr», og «pPtr.next» sin peker blir satt til «aNoide». Vi vil også trenge en else-if-test for å sjekke om første element er søkeverdien. Da vil metode 5. bli brukt.

9.Skriv ut lengden på listen

Her trenger vi kun å returnere «elementCount». Vi kunne også traversert igjennom hele listen og telt opp antall elementer.

10.Sjekk om listen stemmer

I denne metoden traverser vi igjennom listen med en while-loop. For hver gang while-loop kjøres, altså at vi flytter «cPtr» et element framover, så teller vi antall ganger vi har flyttet oss. Dermed sjekker vi denne verdien opp mot «elementCount'en» våres.

11.Tell opp antall forekomster av element med gitt verdi i lista

Vi traverserer igjennom listen med en while-loop. Inne i while-loopen har vi en if-test som sjekker om det elementet vi står på har verdien vi søker etter. Har den verdien vi søker etter plusser vi på en teller. Dermed returnerer vi antall vi fant.

12.Skriv ut hele listen. Kun 5 elementer pr. linje

Her har vi tilfelle ved at listen er tom. Da returnerer vi kun en «String» med «listen er tom». Hvis ikke listen er tom så traverserer vi igjennom listen ved hjelp av en while-loop og legger verdien på det elementet vi er på i en «String». Vi oppnår linjeskift hvor hvert 5 element ved å ha en teller som teller antall ganger vi har gått igjennom while-loopen. Så har vi en if-test inne i while-loopen som tar telleren og sjekker om den gir null i rest når telleren blir delt på 5. Dette gjøres ved en operator som heter modulo. Hvis argumentet i parametere blir tilfredsstilt så har vi et linjeskift i if-testen. Til slutt blir «String'en» returnert.

13 Slett hele listen og skriv ut antall som ble slettet

I denne metoden oppretter jeg en ny variabel av «int» som tar vare på «elementCount» sin nåværende verdi. Så sletter jeg listen ved å sette «head» sin peker til «null», og deretter «elementCount» til 0. Så returnerer jeg verdien jeg tok vare på, og det var så mange elementer som ble slettet. «head» peker til starten av lista, og «null» peker til slutten av lista. Ved å sette disse lik hverandre forsvinner alt i mellom, og vi har en tom liste.

14. Finn og skriv ut verdi av største element i listen

I denne metoden kan et unikt tilfelle være at listen er tom og vi ikke finner noen verdier. Da må vi først lage en if-test som sjekker dette. If-testen sjekker at

«elementCount» ikke er lik null. Deretter traverserer vi igjennom listen med en while-loop (dette er selvfølgelig hvis det finnes elementer i listen), og lager oss en «temp» (temporary) verdi som settes lik den første verdien i lista. Deretter for hver gang vi flytter oss til neste verdi så sjekker vi om denne verdien er større en «temp» verdien. Hvis den er større blir «temp» satt til denne verdien, hvis ikke går vi videre i traverseringen. Når vi har traversert igjennom hele listen, returnerer vi denne verdien som den største verdien funnet.

15 Finn og skriv ut verdi av minste element i listen

Denne er veldig lik forrige metode. Man må passe på at listen ikke er tom. Deretter setter vi en «temp»-verdi lik den første verdien i lista. Vi begynner å traversere igjennom lista og hvis det neste elementet i lista er lavere en «temp» verdien så byttes de om. Til slutt returneres denne verdien.

Kommentar

Jeg har valgt å lage en variabel «error» som enten blir satt til -1, 0 eller 1. Dette er får å få et mer responsivt hovedprogram slik at jeg vet hva metoden utførte. Ved -1 så er listen tom, ved 0 ble ikke ønsket verdi funnet, og ved 1 så utførte metoden det som skulle skje. Det er derfor de fleste av metodene mine returnerer en «int verdi». Der hvor jeg måtte returnerer mer enn én verdi så valgte jeg å returnere et array.

Big-O notasjon

Big-O forteller oss noe om hvor mange operasjoner en metode må utføre i verste tilfelle. Dette kan være greit å kunne for å optimalisere koden sin. Jo færre operasjoner du trenger å utføre for at metoden skal utføre det den skal, jo raskere vil programmet gå

Metoder; Klasse: singleLinkedList	**Big O notasjon**
listLength()	O(1)
deleteList()	O(1)
writeList()	O(n)
deleteElementOne()	O(1)
addElementOne()	O(1)
deleteLastElement()	O(n)
deleteFirstGivenValue()	O(n)
deleteAllGivenValues()	$O(n^2)$
addGivenValueAtEnd()	O(n)
addGivenValueAfterElement()	O(n)
addGivenValueBeforeElement()	O(n)
checkElementcount()	O(n)
checkGivenValue()	O(n)
findBiggestValue()	O(n)
findSmallestValue()	O(n)

Bibliografi

Programmering wiki. (2017, Mars 1). Hentet fra http://programmering.wiki.ifi.uio.no/Lenket_liste

UML-skjema. (2017, Mars 1). Hentet fra https://www.draw.io/