Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2017-01-16

Arv och polymorfi

Mål

I denna laboration ska du skapa ett objektorienterat program som använder arv, polymorfi och flera delar av det du tidigare lärt dig.

Läsanvisningar

- Objektorientering
  - Arv (inheritance)
  - Polymorfi (polymorphism)
- Klasser
  - Referensmedlemsvariabler (&)
  - Konstanta medlemsvariabler (const)
  - Konstanta medlemsfunktioner (const)
  - (Statiska medlemsvariabler (static))
- Argument till main (kommandoradsargument)
  - C-strängar
  - Inbyggda arrayer (C-arrayer)
- Typkonverteringar (to_string, stod, stoi)
  - Fånga undantag från dessa (try, catch, exception)
- Repetition
  - Minneshantering, pekare
  - "Allt" om klasser
  - Filseparering

Uppgift: Förenklad kretssimulator

Du ska skriva ett program som utför en förenklad simulering av en elektrisk krets. En (icke förenklad) elektrisk krets skulle kunna bestå av exempelvis Resistorer, Kondensatorer, Spolar, Dioder, Transistorer, Spänningskällor och Strömkällor.
I våra förenklade kretsar förekommer Resistorer (som har en viss resistans), Kondensatorer (som har en viss kapacitans och en nuvarande laddning) och Batterier (som har en outtömlig laddning). Dessa komponenter är sammankopplade via kopplingspunkter. Varje komponent måste vara kopplad till två (olika) kopplingspunkter. Varje kopplingspunkt representeras av en viss potential och kan vara kopplad till oändligt många komponenter. Observera att det räcker att komponenterna vet hur de är kopplade. Kopplingspunkterna behöver bara veta "sin" potential, inte vad som är anslutet.
Du ska skapa klasser för att representera de olika komponenterna och göra det möjligt att bygga upp olika "kretskort" i huvudprogrammet. Målet är att det ska vara relativt enkelt att bygga upp en ny krets med dina klasser. Därefter ska alla komponenter i kretsen simuleras i små tidsintervall.

Simulering av komponenter

Simuleringen går till så att varje komponent utför sitt arbete under ett litet tidsintervall, och detta upprepas under ett stort antal iterationer. Varje komponent har ett speciellt beteende i hur den överför laddningspartiklar (förändrar potentialen) från kopplingspunkt till kopplingspunkt.
OBS! I detta program frångår vi för enkelhets skull hur det fungerar i verkligheten på flera punkter. Istället följer vi följande specifikation:

Ett batteri ser till att kopplingspunkten på plus-sidan får samma potential som batteriets spänning och kopplingen på minussidan får potential noll.

En resistor förflyttar laddningspartiklar från sin mest positiva kopplingspunkt till sin minst positiva kopplingspunkt. Mängden flyttade partiklar beror på skillnaden i potential, hur stor resistansen är, samt hur lång tid som förflyter.

Exempel: Antag att resistansen är 2.0Ω och vi simulerar i steg om 0.1 tidsenheter. Om potentialen på sida a är $5.0V$ och potentialen på sida b är $9.0V$ blir spänningen över resistorn $9.0V - 5.0V = 4.0V$. Nu flyttas $4.0/2.0 * 0.1$ laddningspartiklar från sida b (som ju har högst potential) till sida a.

En kondensator "suger upp" positiva laddningspartiklar från sin mest positiva sida och lagrar dessa internt. Samtidigt suger den upp lika många negativa laddningspartiklar från sin minst positiva sida (vilket i praktiken innebär att den lägger till positiva laddningspartiklar där). Mängden laddningspartiklar som sugs upp beror på hur stor potentialskillnad det är mellan sidorna, hur mycket som tidigare sugits upp, och kapacitansen.

Exempel: Antag att kapacitancen är 0.8 Farad och vi simulerar i steg om 0.1 tidsenheter. Om potentialen på sida a är $5.0V$ och potentialen på sida b är $9.0V$ så är skillnaden $9.0 - 5.0 = 4.0V$. Om vi har $3.0V$ lagrat sedan tidigare kommer vi nu lagra ytterligare $0.8 * (4.0 - 3.0) * 0.1$. Dessa tas från sidan med högst potential och läggs till både internt och på andra sidan.

Spänningen över en komponent mäts genom att ta skillnaden i potential på vardera sida.

Strömmen genom en resistor fås genom att dela spänningen över resistorn med dess resistans. Strömmen "genom" en kondensator approximerar vi med $C(V - L)$ där C är kapacitancen, V är spänningen över kondensatorn och L dess laddning. Ett batteri låtsas vi alltid har strömmen noll.

För kretsar med enbart ett batteri, resistorer och kondensatorer ska spänningen över varje komponent stabiliseras efter många iterationer. Hur många iterationer som krävs beror på storlek av komponenternas värden och hur stort varje tidsintervall är. Det stabila värdet ska stämma med det resultat som kan räknas fram enligt elläran. Beroende på hur man kopplat positiva och negativa sidan av komponenter kan det tänkas skilja på tecken.
Huvudprogrammet ska från kommandoraden ta in:

- hur många iterationer som ska simuleras
- hur många iterationer (rader) som ska skrivas ut

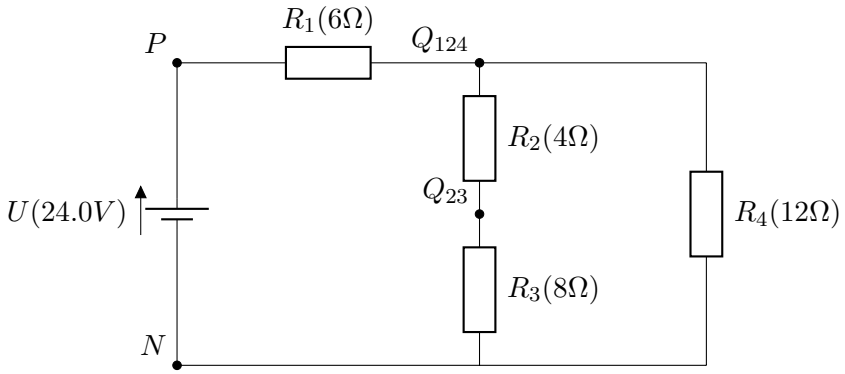
- hur stort tidsintervall som simuleras i varje steg
- spänningen på batteriet

Ett typexempel på indata skulle kunna vara att vi vill simulera 1 000 000 iterationer i steg om 0.01 tidsenheter med spänningen 10 Volt och totalt 10 utskrifter (d.v.s. 100000 iterationer mellan varje utskrift).
KRAV: Om något kommandoradsargument saknas eller är ogiltigt ska programmet avsluta med ett instruktivt felmeddelande. Ogiltiga värden ska detekteras genom att fånga de undantag som slängs av std::stoi, eller fånga egendefinerade undantag som slängs inifrån egna underfunktioner för felkontroll.
I huvudprogrammet ska du bygga upp tre olika kretsar enligt nedan. Klassdesignen ska vara sådan att detta är enkelt. Dessa kretsar ska simuleras och spänningen över varje komponent ska skrivas ut i en tabell med så många rader som efterfrågats. Resultatet ska stämma val överens med exemplen. OBS! Vi bryr oss inte om mycket små avvikelser, rådgör med assistenten om du är osäker.
Ett exempel på hur en krets med två parallellkopplade resistorer kan byggas upp och simuleras ses i kodexempel 1.

```
Connection p, n;
vector<Component*> net;
net.push_back(new Battery("Bat", 24.0, p, n));
net.push_back(new Resistor("R1",  6.0, p, n));
net.push_back(new Resistor("R2",  8.0, p, n));
simulate(net, 10000, 10, 0.1);
```

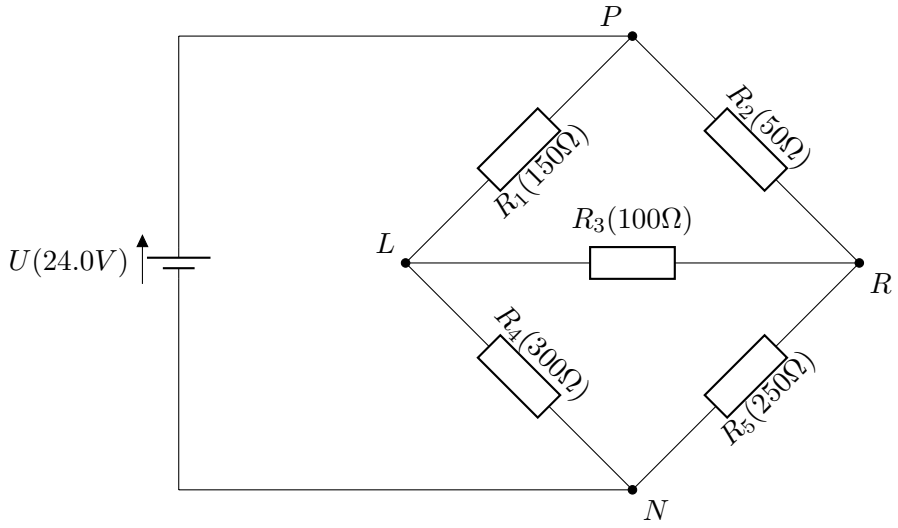
Listing 1: Kretsexempel

Kretsexempel



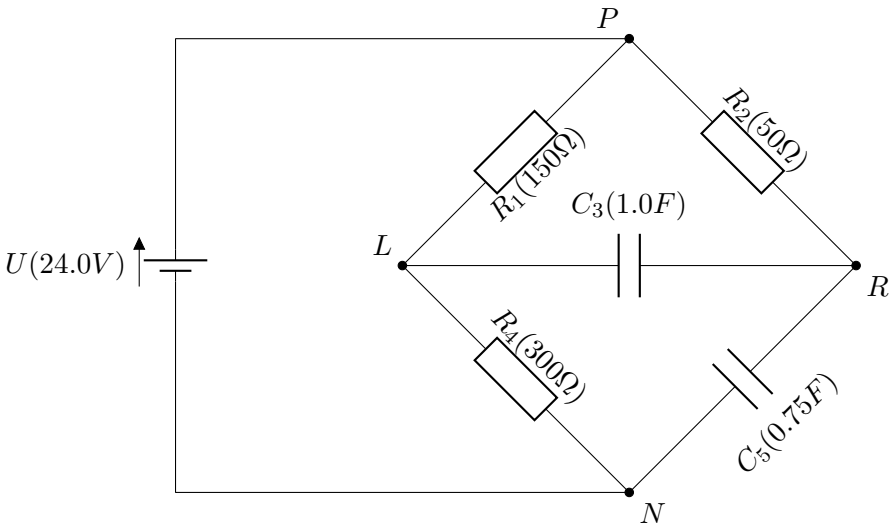
Figur 1: Krets 1

Kretsen i figur 1 har 4 resistorer och ett batteri som är sammankopplade i kopplingspunkterna P, N, Q_{124} och Q_{23}.



Figur 2: Krets 2

Nästa krets (figur 2) har 5 resistorer som är ganska besvärliga att räkna ut spänningen över "för hand", där det kanske snabbare att skriva programmet. Vi kan se att R_1 går mellan P och L, R_3 mellan R och L och så vidare.
Sista kretsen (figur 3) har även några kondensatorer. För denna krets är det intressant hur den beter sig när spänningen ändras, och att rita grafer över detta, dock fungerar inte vår enkla simulering bra för detta och C++ saknar enkla grafritningsverktyg. Att skapa växelströmkällor eller transistorer



Figur 3: Krets 3

lämnas därför till den intresserade. Vi tittar bara att slutvärdena då kondensatorerna är fullt laddade är rimliga.

Körexempel 1

```
./a.out  200000 10 0.01 24

 Krets 1:
         Bat          R1          R2          R3          R4
  Volt  Curr  Volt  Curr  Volt  Curr  Volt  Curr  Volt  Curr
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00
 24.00  0.00 11.99  2.00  3.99  1.00  7.98  1.00 11.97  1.00

 Krets 2:
         Bat          R1          R2          R3          R4          R5
  Volt  Curr  Volt  Curr  Volt  Curr  Volt  Curr  Volt  Curr  Volt  Curr
 24.00  0.00  7.47  0.05  5.28  0.11  2.19  0.02 16.52  0.06 18.72  0.07
 24.00  0.00  6.40  0.04  4.72  0.09  1.68  0.02 17.60  0.06 19.28  0.08
 24.00  0.00  6.35  0.04  4.69  0.09  1.66  0.02 17.65  0.06 19.31  0.08
 24.00  0.00  6.34  0.04  4.69  0.09  1.65  0.02 17.65  0.06 19.31  0.08
 24.00  0.00  6.34  0.04  4.69  0.09  1.65  0.02 17.65  0.06 19.31  0.08
 24.00  0.00  6.34  0.04  4.69  0.09  1.65  0.02 17.65  0.06 19.31  0.08
 24.00  0.00  6.34  0.04  4.69  0.09  1.65  0.02 17.65  0.06 19.31  0.08
 24.00  0.00  6.34  0.04  4.69  0.09  1.65  0.02 17.65  0.06 19.31  0.08
 24.00  0.00  6.34  0.04  4.69  0.09  1.65  0.02 17.65  0.06 19.31  0.08
 24.00  0.00  6.34  0.04  4.69  0.09  1.65  0.02 17.65  0.06 19.31  0.08

 Krets 3:
         Bat          R1          R2          C3          R4          C5
  Volt  Curr  Volt  Curr  Volt  Curr  Volt  Curr  Volt  Curr  Volt  Curr
 24.00  0.00  8.50  0.06  4.10  0.08  4.40  0.01 15.50  0.05 19.90  0.03
 24.00  0.00  7.31  0.05  0.94  0.02  6.38  0.01 16.69  0.06 23.06  0.01
 24.00  0.00  7.57  0.05  0.30  0.01  7.27  0.00 16.43  0.05 23.70  0.00
 24.00  0.00  7.79  0.05  0.12  0.00  7.67  0.00 16.21  0.05 23.88  0.00
 24.00  0.00  7.90  0.05  0.05  0.00  7.85  0.00 16.10  0.05 23.95  0.00
 24.00  0.00  7.96  0.05  0.02  0.00  7.93  0.00 16.04  0.05 23.98  0.00
 24.00  0.00  7.98  0.05  0.01  0.00  7.97  0.00 16.02  0.05 23.99  0.00
 24.00  0.00  7.99  0.05  0.00  0.00  7.99  0.00 16.01  0.05 23.99  0.00
 24.00  0.00  8.00  0.05  0.00  0.00  7.99  0.00 16.00  0.05 24.00  0.00
 24.00  0.00  8.00  0.05  0.00  0.00  8.00  0.00 16.00  0.05 24.00  0.00
```

För att tydligare se vad som händer kommer vi nu studera ett enda steg med krets 1 då vi antar att steget är 1 tidsintervall. Vi börjar med batteriet, som sätter kopplingspunkt `P` till att ha 24V. Därefter går vi till R_1 som har $P = 24$, $Q_{124} = 0$ så potentialen (skillnad i laddning) är 24V. Med resistans 6 och steglängd 1 flyttas $1 * 24/6 = 4$ över till Q_{124} som därefter har laddning 4. På samma sätt har R_2 potential 4V och flyttar därför över $1 * 4/4 = 1$ till Q_{23}. R_3 flyttar $1 * 1/8 = 0.125$ till N. För R_4 blir det lite krånligare. Nu har vi tagit 1 från Q_{124} och N har värdet 1/8. Därför flyttar R_4 över $1 * ((4 - 1) - 1/8)/12 = 0.24$ till N. Stegvis skulle det bli enligt nedan:

1. Bat sätter $P = 24$, $N = 0$.
2. R_1 sätter $P = 20$, $Q_{124} = 4$
3. R_2 sätter $Q_{124} = 3$, $Q_{23} = 1$
4. R_3 sätter $Q_{23} = 0.88$, $N = 0.12$
5. R_4 sätter $Q_{124} = 3 - 0.24 = 2.76$, $N = 0.36$

Skulle vi skriva ut en rad i tabellen efter detta första steg skulle vi få följande resultat:

Bat		R1		R2		R3		R4	
Volt	Curr	Volt	Curr	Volt	Curr	Volt	Curr	Volt	Curr
24.00	0.00	17.24	2.87	1.89	0.47	0.51	0.06	2.40	0.20

Detta för att

$$
\begin{array}{lcll}
P & = & 24 - 4 & = 20 \\
Q_{124} & = & 4 - 1 - 0.24 & = 2.76 \\
Q_{23} & = & 1 - 1/8 & = 7/8 \\
N & = & 1/8 + 0.24 & = 0.36 \\
Bat\ (U, I) & = & (24, 0) & (alltid) \\
R_1\ (U, I) & \Rightarrow & (P - Q_{124} & = 17.24, \quad 17.24/6 = 2.87) \\
R_2\ (U, I) & \Rightarrow & (Q_{124} - Q_{23} & = 1.88, \quad 1.88/4 = 0.47) \\
R_3\ (U, I) & \Rightarrow & (Q_{23} - N & = 0.51, \quad 0.51/8 = 0.06) \\
R_4\ (U, I) & \Rightarrow & (Q_{124} - N & = 2.4, \quad 2.4/12 = 0.2)
\end{array}
$$

Bonusuppgift: Bättre objektorientering

Om vi tittar närmare på den givna koden för att skapa en krets så ser vi att det vore lämpligt att representera hela kretsen som en klass. Det blir då naturligt att anropa `simulate` som en medlemsfunktion i kretsen, och vi får hjälp med minneshanteringen genom att lägga till en destruktor. De problem vi då får är hur kretsen ska byggas upp så att alla anslutningspunkter och komponenter ingår (composition) i kretsklassen. Det man kan tänka sig är att det finns en medlemsfunktion för att lägga till en ny namngiven kopplingspunkt, och en annan för att lägga till en komponent. I denna uppgift ska ni lösa detta. Modifiera ert klassdiagram och diskutera med assistent innan ni börjar koda.

Om något går fel när användaren försöker bygga upp kretsen (exempelvis om hen namnger en kopplingspunkt som inte finns) ska ett undantag av egendefinierad typ kastas. Undantagsklassen ska ärva från en väl vald klass i klassheirarkin från `<stdexcept>`.