Exercise 3

Indledning

Første del af denne øvelse har til formål at introducere dig til brugen af Visual Studios debugger. Debuggeren er en stor hjælp til fejlfinding. Især runtime fejl kan være svære at lokalisere og da er debuggeren uundværlig hjælp.

Når du debugger et program, stepper du igennem programmet linie for linie. Du har for hver enkelt linie mulighed for at se værdien af variable i progammet (og meget andet godt). For at debugging giver værdi, er det meget vigtigt at du, for hvert step du tager, gør dig klart hvilke variable du forventer skifter værdi når du stepper, og hvad de skifter til. Det er jo netop når det forventede ikke stemmer overens med det faktiske, at du kan ske at identificere en fejl i programmet – eller din forventning til det.

Vigtigt: Læs og forstå dette! I den første del af opgaven vil der blandt andet være en del udleveret kode – kode, der er fejlbehæftet, og som du skal bruge debuggeren for at finde fejlen i. Det er vigtigt at du forstår at formålet med øvelsen ***ikke*** er at finde og rette fejlen i det udleverede kode. Det vil du i mange tilfælde nok kunne gøre uden debuggeren. Formålet er derimod at du bruger debuggeren til at gøre det – at du vænner dig til at bruge debuggeren og de redskaber den stiller til rådighed. På den måde ved du hvad du skal gøre den dag, du pludselig *ikke* kan gennemskue et problem blot ved ”code staring”!

Anden del af øvelsen omhandler anvendelse af `const` med klasser og med pointere, samt initialisering af pointere.

---

1. del – Debugging

Exercise 3.1:

Hvis du har en tidligere opgave, som du aldrig fik til at virke (fordi den fejler når du afvikler programmet), så find den frem og debug den, til du finder ud af, hvad fejlen er.

Ellers find en tidligere øvelse som virker, men som du måske ikke helt forstår forløbet i. Debug den og følg med i forløbet og variablenes værdier, og se om ikke det hjælper med forståelsen.

Hvis du ikke har en sådan opgave, så find en alligevel!!! Du skal jo ***øve*** debugging – OG debugging kan også anvendes til blot at følge med i hvordan et program

afvikles – rækkefølge af kode, funktionskald, returværdier osv. ☺. Øv dig i at bruge tastaturgenveje til at sætte breakpoints (F9), starte debugging (F5), samt at steppe ind i (F11) og over (F10) funktionskald.

Exercise 3.2:

Sammen med denne øvelse ligger filen `program1.cpp`. Opret et nyt Visual Studio projekt og kopier filen til projektets mappe.

a) Kør programmet og se hvad der sker.

b) Betragt programmets kildekode. Afgør, hvilke 2 værdier du vil indtaste. Sæt et breakpoint et passende sted i koden og evaluér værdien af programmets variable.

c) Debug nu programmet og find fejlen. Brug den fremgangsmåde der er beskrevet i starten af denne opgave.

d) Ret fejlen i programmet. Brug debuggeren til at eftervise at fejlen er rettet.

Exercise 3.3:

Sammen med denne øvelse ligger filerne `program2.cpp, ClassA.h` og `ClassA.cpp`. Kopier filerne til projektets mappe (husk at fjerne `program1.cpp`).

a) Kør programmet og se hvad der sker.

b) Brug samme fremgangsmåde som i Exercise 3.2 spm. b-d til at debugge programmet, finde fejlen, rette fejlen og eftervise at fejlen er rettet

Exercise 3.4:

Sammen med denne øvelse ligger filerne `program3.cpp, ClassB.h, ClassB.cpp, ClassC.h` og `ClassC.cpp`. Fjern de ”gamle” filer og kopier de ”nye” filerne til projektets mappe.

a) Kør programmet og se hvad der sker.

b) Brug samme fremgangsmåde som i Exercise 3.2 spm. b-d til at debugge programmet, finde fejlen, rette fejlen og eftervise at fejlen er rettet

c) Tilføj nogle flere objekter af klasserne `ClassB` og `ClassC` og test princippet memberwise assignment ved at kopiere (assigne) objekter til hinanden. Kan du fremprovokere nogle fejl (run-time)? (Hint: Leg med scopes). Prøv også med const objekter.

Exercise 3.5:

Sammen med denne øvelse ligger filen `ClassD.h`. Kopier filen til projektmappen (slet igen de ”gamle” filer).

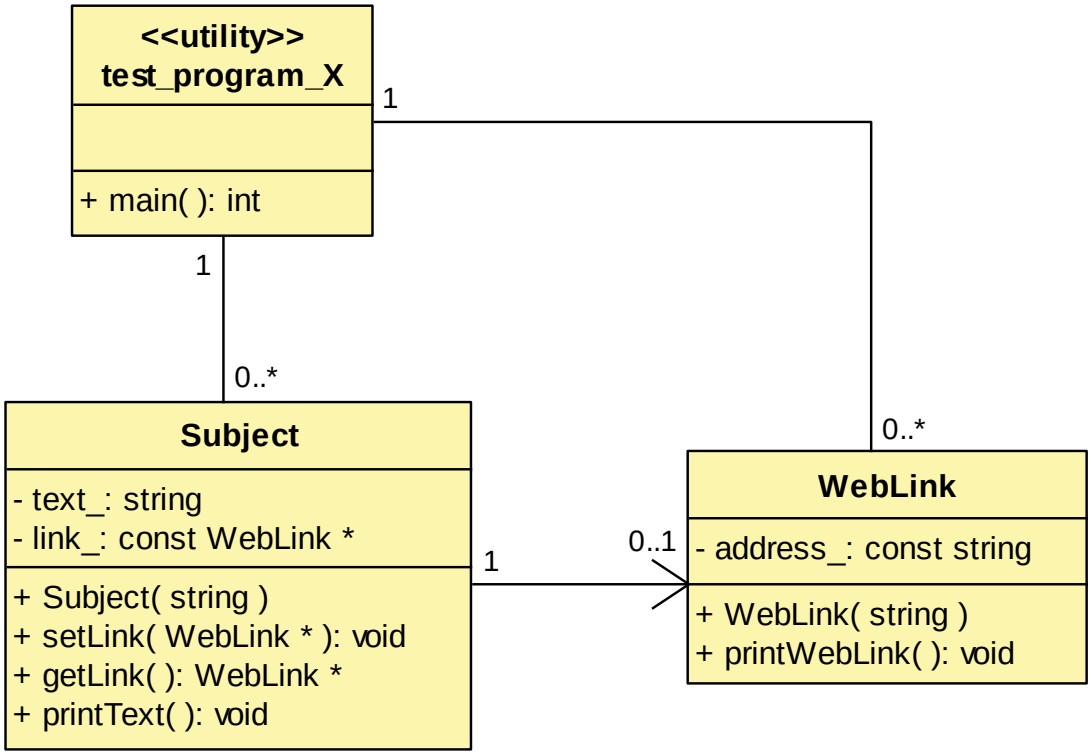
a) Diskutér med en medstuderende hvad der er specielt ved denne klasse? (bortset fra at det hele er skrevet i header-filen ☺).

b) Skriv et lille testprogram, som tester klassen.

2. del - const med klasser og pointere, initialisering af pointere

Exercise 3.6:

Denne øvelse omhandler klasserne i klassediagrammet herunder.



a) Diskutér med en medstuderende hvorfor constructoren i klassen `Subject` ikke har en `WebLink` pointer som parameter OG hvilken betydning det har for initialisering af pointeren `link_`.

Koden til de to print-metoder kan du hente i filen ”print.cpp”, som ligger sammen med øvelsen. Der mangler dog lidt kode, som du selv skal tilføje.

Koden til de øvrige metoder i klasserne skal du selv skrive (og ja…de ER simple så du kan sikkert godt regne ud de skal kunne ☺).

b) Skriv koden til de to klasser `WebLink` og `Subject`, som du kan se i diagrammet herover.

c) Når du mener, at dine klasser er perfekte, så test dem med test programmet ”test_program_X.cpp”, som også ligger sammen med øvelsen.

   Hvis der er fejl når du kompilerer, så er det **IKKE** test programmet der er forkert!!!!!! ☺ Ret din kode indtil det virker sammen med test programmet.

   NB! Du skal selvfølgelig sørge for at forstå, ***hvorfor*** du evt. bliver nødt til at lave ændringer i din kode.