Laboration 2.1: Länkad lista

Läs varje deluppgift noggrant då det är viktigt att ni följer instruktionerna för att ni ska få ihop allt rätt. Om ni undrar över något eller tycker att något är otydligt, hör av er till labbassistenten eller till stefan.bygde@mdh.se.

Rekommenderad förberedande läsning: https://en.wikipedia.org/wiki/Linked_list

Steg 1: Ladda ned och kompilera labbskelettet

Ladda ner `linkedList.zip` från Blackboard. Öppna ett nytt **tomt** projekt i er utvecklingsmiljö. Lägg till alla filer i zip-filen till ert projekt. Observera att detta projekt inte kommer att gå att kompilera just nu.

Kommentarerna i labbskelettet saknar bokstäverna å,ä,ö för att undvika konstigheter på vissa system.

Steg 2: Välj vilken typ av lista ni vill göra

Tänk igenom hur ni vill implementera er länkade lista. Enkellänkad eller dubbellänkad? Vill ni representera er lista som en nodpekare eller som en struct? Se dokumentet “Tips för lab 2” på Canvas för för- och nackdelar med olika typer.

När ni bestämt er så öppna filen `list.h` som ligger i ert projekt:

1. Gå till avsnittet “1. Struct-definitioner” i filen och kommentera bort den listnod definition ni tänkt använda.
2. Gå till avsnittet “2. Typdefinition” i filen och kommentera bort den typ ni tänkt använda (t.ex `struct node*` eller `struct list`). Det är viktigt att er typ heter precis `List`, så ändra inte detta. Det är tillåtet att lägga till fler medlemmar ifall ni väljer en struct-representation.

Notera att, beroende på vad ni valt, så kanske det inte går att kompilera labbskelettet ännu! Detta beror på att vissa funktioner i `list.c` ännu inte har korrekt typ på sina returvärden.

Steg 3: Implementera interfacet

I `list.h` finns redan ett färdigt interface till den länkade listan. I filen finns även kommentarer som förklarar eventuella returvärden samt vad funktionerna ska göra. Öppna `list.c` och fyll i kod för de olika funktionerna.

Under tiden ni lägger till funktioner kan ni testa er lista genom menyn som finns i `test_list.c` eller testerna som också finns där (notera dock att testerna inte kommer att gå igenom förrän ni skrivit korrekt kod till samtliga funktioner, men menyn kommer att fungera för färdiga funktioner). För att köra testerna eller menyn så avkommentera en av funktionsanropen i `main()` som ligger i `test_list.c`.

Tips: för att så snabbt som möjligt få programmet att kompilera utan fel kan du börja med funktionen `createEmptyList()`.

Utförande och rekommendationer:

- Det är tillåtet och *starkt* rekommenderat att dela upp era funktioner i mindre "hjälpfunktioner" där det behövs. Dessa ska i så fall deklareras som **`static`** och läggas längst upp i `list.c`-filen.
- Det är rekommenderat (men inte ett krav) att implementera så många av dessa funktioner som möjligt rekursivt, för att öva på rekursion inför kommande labbar.
- Där det står pre/postcondition i kommentarerna måste ni använda asserts för att verifiera dessa. Det är också fullt tillåtet att lägga till egna pre/post-conditions.
- I steg 3 ska ni alltså enbart behöva göra ändringar i filen `list.c`, ingen annanstans.

Övrigt Att tänka på:

- Nyckelordet **`const`** framför ett argument betyder att funktionen inte kan ändra innehållet i argumentet. Detta är användbart i samband med pekare för att se till att funktionen inte ändrar på datat som pekaren pekar på. Anledningen att typen **`const`** `Data` ska skickas in är att funktionen inte ska ändra på datat som ska ligga i listan , även om det råkar vara en pekare (detta kommer att tas upp på föreläsning 4).

För att uppgiften ska räknas som färdig behöver alla tester i funktionen `testFunction()` i `list_text.c` gå igenom (funktionen får naturligtvis inte ändras).