

Laboration 5

Väljare (extrauppgift)

Inledning

Detta är en betyghöjande extrauppgift. Du får inte redovisa den förrän du är godkänd på alla obligatoriska laborationer.

Syftet med laborationen är att visa hur det objektorienterande tänkandet kan användas för att förenkla alla slags problem, även sådan som av tradition nästan alltid hanteras av procedurorienterad C-kod. Du kommer att få tillfälle att använda fler standardbehållare ur STL och att skriva undantagshantering (exceptions).

Förberedelser

Lägg dina filer i en separat katalog och se till att den är lässkyddad för alla utom dina två laborationskamrater. Om du vill att en assistent ska kunna titta på dina filer, ge AFS-gruppen `oopk_e00:assistenter` läsrättigheter i katalogen.

Ladda alltid kursmodulen innan du börjar arbeta:

```
module load oopk_e00
```

Kopiera en makefile från kurskatalogen:

```
cp $OOPKHOME/Makefile .
```

och redigera filen.

Läs avsnitt 7.8 i kursboken (C++ Primer, 3:e upplagan) om vad argumenten `argv` och `argc` till `main` innehåller och hur de används.

Uppgiften

Många program startas genom att man i ett terminalfönster skriver programmets namn följt av ett antal *väljare* (eng: *switches* eller *options*). En väljare förändrar på något sätt programmets beteende eller ger programmet mer information. Exempel är väljarna `-l` och `-F` till programmet `ls` som påverkar vilken information som skrivs ut:

```
ls -l -F
```

Ofta tar väljarna också argument, t ex

```
lp -d orange  
head -n 10
```

Dessa väljare och deras eventuella argument skickas med som textsträngar till `main`. Funktionen `main` har ju som bekant två argument:

```
int main(int argc, char* argv[])
```

Det första argumentet anger hur många ord (egentligen tokens) som angavs på kommandoraden inklusive programnamnet. Dessa ord återfinns i arrayen `argv`. Om kommandot var `lp -d orange` så är `argc==3`, `argv[0]=="lp"`, `argv[1]=="-d"` och `argv[2]=="orange"`.

I kursboken beskriver Lippman hur man traditionellt avkodar `argv` och `argc` och visar hur krånglig koden blir. Sedan gör han ett försök att objektifiera lösningen, men egentligen flyttar han bara undan koden från `main` och den blir inte lättare att skriva eller förstå. Därför ska du implementera en genuint objektorienterad lösning på problemet att tolka väljare och deras argument.

Varje väljare ska representeras av ett objekt. Objektet ska veta vad väljaren heter, om väljaren tar något argument och i så fall av vilken typ (textsträng, heltal etc). Objektet ska titta i `argv` om väljaren finns där och i så fall omvandla argumentet till rätt C++-typ. Detta värde ska efter avkodningen av `argv` enkelt kunna hämtas av programmet. Om väljaren inte har angivits ska objektet istället lämna ifrån sig ett standardvärde (default). En väljare som inte tar argument kan anses associerad med ett logiskt värde som är `true` om väljaren angavs, annars `false`.

Vi tar ett exempel. Programmet `lp` tar (bl a) en väljare `-c` (copy) utan argument, `-d` (destination) som har ett strängargument, och `-q` (priority) som tar ett heltalsargument. Dessa ska kunna avkodas på följande sätt:

```
int main(int argc, char* argv[])
{
    OptionParser parser;
    BoolOption copy_option(parser, "-c");
    StringOption destination_option(parser, "-d", "local");
    IntOption priority_option(parser, "-q", 30);

    parser.parse(argc, argv);

    // Testutskrift
    if (copy_option.value())
        cerr << "-c angavs" << endl;
}
cerr << "destination: " << destination_option.value() << endl;
cerr << "priority: " << priority_option.value() << endl;
```

där `copy_option.value()` returnerar en `bool`, `destination_option.value()` en `string`, och `priority_option.value()` en `int`. Om man startar programmet med

```
lp -c -q 17
```

får man följande utskrift:

```
-c angavs
destination: local
priority: 17
```

Själva avkodning av `argv` sker i anropet av `OptionParser::parse`. För varje väljare i `argv` går parsern igenom de option-objekt som har associerats med parsern och ser om de känner igen väljaren.

Dina objekt ska också kunna hantera långa namn, t ex

```
StringOption destination_option(parser, "-printer", "local");
IntOption priority_option(parser, "-priority", 30);
```

Det ska vara tillåtet att förkorta väljarnamnet så länge det är otvetydigt:

```
lp -prio 17    # OK, betyder lp -priority 17
lp -pri 17    # Fel, -priority eller -print?
```

All felhantering ska ske med undantagshantering (exceptions). Om man t ex anger en väljare som inte finns eller anger ett argument som inte är av rätt typ ska klasserna du har implementerat *inte* skriva ut felmeddelanden direkt på skärmen. De ska istället skicka iväg ett exception av lämplig typ som kan fångas upp av `main` som i sin tur kan skriva ut ett lämpligt meddelande. Skriv kod i `main` som testat att felhanteringen fungerar.