



Avdeling for informatikk og e-læring, Høgskolen i Sør-Trøndelag

C++ versus Java

Tore Berg Hansen

29.8.2006

Lærestoffet er utviklet for faget LV194D C++ for Javaprogrammerere

1. C++ versus Java

Resymé: I denne leksjonen introduserer vi C++ ved å se på forskjellene og likhetene mellom C++ og Java. Som ramme for sammenligningen tar vi et språkteoretisk utgangspunkt.

Innhold

| | | |
|--------|--|----|
| 1.1. | INNLEDNING..... | 2 |
| 1.2. | HVORFOR LÆRE C++? | 2 |
| 1.3. | HVILKE FORSKJELLER KAN MAN VENTE? | 3 |
| 1.4. | ET SPRÅKTEORETISK UTGANGSPUNKT..... | 3 |
| 1.4.1. | Hva karakteriserer programmeringsspråk?..... | 3 |
| 1.4.2. | Strukturen i et program..... | 4 |
| 1.5. | LIKHETER OG FORSKJELLER..... | 5 |
| 1.5.1. | Nøkkelord og operatorer..... | 6 |
| 1.5.2. | Slik er programmer bygget opp | 12 |
| 1.5.3. | Datatyper..... | 13 |
| 1.6. | LITTERATUR..... | 16 |

1.1. Innledning

Faget heter C++ for Java-programmerere. Det betyr at det er ment å tas av personer med kunnskap om programmering i sin allminnelighet og som dessuten kjenner til objektorientert programmering. Derfor forutsetter vi at vi kan hoppe raskt over de grunnleggende ting om programmering og se på det man spesielt må vite for å kunne skrive programmer i C++.

En Java-programmerer vil ikke ha store problemer med å lære de grunnleggende tingene i C++. Språkene tilhører den samme familie av programmeringsspråk (på fagspråket det samme paradigme). De som utviklet Java har hentet mange ting fra C++ og tok bl.a sikte på å lage en bedre C++ uten de tingene som gjør C++ vanskelig. C++ er et mer omfattende språk enn Java. En årsak til det er at C++ er bakoverkompatibelt med C. C++ er et generelt programmeringsspråk som kan brukes både til å skrive objektorienterte programmer og "tradisjonelle" C programmer. Java på den annen side er utelukkende objektorientert.

1.2. Hvorfor lære C++?

Ja, hvorfor skal man lære C++ hvis Java er en bedre C++? Egentlig er det ikke noe enkelt svar på det spørsmålet. Og du får forskjellig svar avhengig av hvem du spør. Ettersom vi har utarbeidet dette faget mener vi selvsagt at det er flere gode grunner til å lære C++. Her er noen hovedgrunner:

- ✓ En undersøkelse har vist at det finnes rundt 4000 programmeringsspråk i dag. Og antallet er økende. Det dette bl.a forteller er at det ikke finnes et enkelt språk som er best for alle typer programmeringsoppgaver. Svært mange av disse språkene er laget for å optimalisere helt spesielle typer av programmeringsoppgaver. Altså bør du kjenne til flere språk for å kunne gjøre begrunnede valg av hvilket programmeringsspråk som passer best for gitte oppgaver.
- ✓ Når vi mener at C++ er et av de språkene det er nyttig å kjenne til, skyldes det ikke minst historien. I de siste 20 årene har C++ og C vært det mest brukte språk til utvikling av viktige typer kommersiell programvare. Det vil si at det finnes massevis av programmer som fremdeles er i bruk som er skrevet i disse språkene. Skal du jobbe som profesjonell programmerer vil du med stor sannsynlighet på en eller annen måte måtte forholde deg til C++ eller C programmer. Du kan komme til å delta i videreutvikling eller få jobben med å konvertere slike programmer. Dessuten er det slik at spillprogrammerere ser ut til å foretrekke C/C++.
- ✓ C++ er bedre enn Java når kravet til eksekveringshastighet er kritisk. Likedan åpner C++ for tett samspill mellom programmet og maskinvaren/operativsystemet. Årsaken ligger i det som er noe av hovedforskjellen i filosofi mellom C++ og Java. Java er utviklet for å være et utelukkende objektorientert språk. Programmer skrevet i språket skal kunne kjøre på all type maskinvare og være uavhengig av operativsystem. Det er lagt vekt på sikkerhet, spesielt med tanke på at programmer skal brukes i applikasjoner på Internett. Java appletter kan ikke skrive til disk eller inn/ut-porter. Programmer skrevet i Java er avhengig av en virtuell maskin mellom programmene og maskinvaren/operativsystemet. Denne virtuelle maskinen (som ofte er skrevet i C) tolker en spesiell byte-kode som Java-kompilatoren lager. Byte-kode som må tolkes er mindre effektiv enn optimalisert maskinkode laget av en

(C/C++)kompilator for en bestemt plattform. Java er et viktig språk, men vil ikke være det eneste språket. Det er kun enda et språk. Trenger man en applett for Internett eller virkelig plattformuavhengighet velger man Java. Hvis man trenger effektivitet og fleksibilitet velger man C++. Et eksempel kan være spill.

1.3. Hvilke forskjeller kan man vente?

Du har altså bakgrunn i Java og skal lære et nytt programmeringsspråk. En tilnærming er å se på likheter og forskjeller. Det man gjerne lurer på er slike ting som hvordan strukturen på programmene er, hvilke datatyper språket tilbyr, hvilke navnekonvensjoner gjelder for variabler og konstanter, hvilke reserverte ord finnes og hvilke løkke og valgstrukturer har man. Når det gjelder mange av disse tingene er forskjellene mellom C++ og Java små fordi språkene tilhører det samme paradigme. For å få litt orden på sammenligningen skal vi legge en kort språkteoretisk ramme rundt sammenligningen.

1.4. Et språkteoretisk utgangspunkt

1.4.1. Hva karakteriserer programmeringsspråk?

For å nærme oss dette kan vi først se på en måte å inndele programmeringsspråk på. I terminologien rundt programmeringsspråk gjøres denne inndelingen gjerne i form av såkalte paradigmer. Hvis man skal uttrykke det litt populært og praktisk, så kan man si at et paradigme er en anerkjent måte å løse et problem på. Det vil også si at det til paradigmet er et sett med spesielle metoder og teknikker.

Programmeringsspråk klassifiseres gjerne innen fire hovedparadigmer:

- ✓ det imperative
- ✓ det objektorienterte
- ✓ det funksjonelle
- ✓ det logiske

Det som karakteriserer språk som følger det *imperative paradigme* er at navngitte data manipuleres på en trinnvis måte. Programmene er på en måte en detaljert oppskrift som skal følges trinn for trinn – gjør dette, gjør så deretter dette. Det gjør overgangen til maskinkode enkel fordi også datamaskiner tradisjonelt jobber på den måten. Det er det som ligger i ordet imperativ – strengt kommanderende. Ulempen er at programmererne må slite med mange detaljer og med å tilpasse problemet til programmeringsspråket. Assembler er det mest ekstreme imperative språk. Andre kjente er Fortran, Pascal, Cobol, C, Ada, Basic og Algol. Flere av disse finnes i forskjellige varianter (også kalt dialekter).

Det objektorienterte paradigme omfatter mer enn programmeringsspråk. Det må ses i sammenheng med objektorientert analyse og design. I objektorientert analyse prøver man å forstå et problem i form av objekter i problemdomenet, under objektorientert design bygger man opp en løsning i form av samarbeidende objekter som så kan implementeres i et objektorientert programmeringsspråk. I et objektorientert programmeringsspråk innkapsles og skjules dataene i objekter. Dette er det essensielle. Dataene kan kun manipuleres gjennom

operasjoner på objektene, aldri direkte. Operasjonene er objektets ”interface”. Objektorienterte språk er også karakterisert ved *dynamisk binding* mens imperative språk bare har *statisk binding*.

Ved statisk binding blir det bestemt ved kompilering hvilken kode som skal kjøres. Dynamisk binding innebærer at det kan det utsettes til kjøretid å bestemme hvilken kode som skal kjøres. Det objektorienterte paradigme har også elementer av det imperative. Operasjoner kodes etter det imperative paradigme. For dere er Java det mest kjente objektorienterte språk. C++ er også objektorientert, men i tillegg imperativt fordi det også inkluderer C. Norske Simula var det første språk med objekter. Andre kjente objektorienterte språk er Eiffel, SmallTalk, Objective C og Delfi. De to siste er på samme måte som C++, utvidelser av eldre imperative språk (C og Pascal). Slike objektorienterte språk kalles *hybride* når de tillater objektorientering uten å påtvinge programmereren det paradigme. Objektorienterte purister stiller for øvrig meget strenge krav til hva som må til for at et programmeringsspråk skal kunne kalles *rent* objektorientert. Men det vil vi la ligge i denne sammenheng.

I det imperative paradigme spesifiseres i detalj i form av algoritmer hva som skal gjøres for å løse et programmeringsproblem. Språk som følger det *funksjonelle* og *logiske paradigme* lar programmereren konsentrere seg om hvilket problem som skal løses, dvs hva som skal gjøres fremfor hvordan. Språkene sies å være deklarativer. Ideen er at programmerere dermed skal bli mer produktive. I praksis kan grensen mellom imperative og deklarativer språk være flytende. Ulempen med deklarativer språk har hittil vært at de er vanskeligere å implementere effektivt. Det betyr at programmer i denne typen språk vil kjøre langsommere enn programmer som er skrevet i et imperativt språk. De mest kjente funksjonelle språk er antagelig Lisp, ML og Miranda. Det viktigste logiske programmeringsspråk er Prolog.

1.4.2. Strukturen i et program

Setninger i et programmeringsspråk analyseres på samme måte som naturlige språk på fire nivåer:

- ✓ det leksikale nivå
- ✓ syntaktiske nivå
- ✓ det kontekstuelle nivå
- ✓ det semantiske nivå

Det leksikale nivå dreier seg om ord som er tillatte i språket. De er språkets byggeklosser. I programmeringsspråk kalles disse byggeklossene leksikale symboler, leksikale enheter eller token. De leksikale symbolene klassifiseres gjerne som identifikatorer, nøkkelord, operatører, skilletegn, literaler og kommentarer.

Det syntaktiske har med regler for hvordan ord settes sammen til programmer å gjøre. Dvs det beskriver hvordan setninger i programmet bygges opp.

Syntaksen beskrives gjerne i en såkalt kontekstfri grammatikk. Det vanlige er å bruke Backus Naur (BNF) formen eller Extended BNF (EBNF). Her er et lite eksempel i EBNF hentet fra [1].

```

<program> ::= program <setning>* end
<setning> ::= <tilordning> | <løkke>
<tilordning> ::= <identifikator> := <uttrykk>;
<løkke> ::= while <uttrykk> do <setning>+ done
<uttrykk> ::= <verdi> | <verdi> + <verdi> | <verdi <= <verdi>
<verdi> ::= <identifikator> | <tall>

```

Her betyr merkelappene * null eller flere forekomster og + en eller flere forekomster. | representerer alternativer og uthevede ord er nøkkelord. Ord i klammeparenteser er byggeklossene. ::= betyr er definert som. Ellers finner vi tre operatore og et skilletegn. Det er en operator for tilordning (:=), en aritmetisk (+) og en logisk (<=). Skilletegnet er ;. Her er et eksempel på et program som følger denne grammatikken.

Program

```

n := 1;
while n <= 10 do
    n := n + 1;
done
end

```

På det semantiske nivå beskrives hvordan setninger skal forstås og kommandoer utføres. Det kontekstuelle nivå som ligger mellom det syntaktiske og semantiske nivå, har regler som videre detaljerer hva som er gyldige setninger. Eksempelvis kan det i et språk være tillatt ut fra syntaktiske regler å skrive et uttrykk som dette

```
2 + 't'
```

Man legger her sammen to verdier representert henholdsvis med et heltall og en character. Dette kan være ulovlig etter reglene på det konseptuelle nivå (konseptuell betyr å se noe i sammenheng med noe annet). Regelen kan være at det bare er tillatt å legge sammen verdier som tilhører samme type. Andre regler på det konseptuelle nivå kan være at det ikke er lov å benytte variable i uttrykk før variablene er gitt verdier.

1.5. Likheter og forskjeller

Hva er så likhetene mellom C++ og Java? Man finner mange likheter fordi som vi skrev i innledningen, tok de som laget laget Java utgangspunkt i C++. På det leksikale nivå er de

temmelig like. Identifikatorer som brukes til å identifisere data og objekter bygges opp på samme måte med de samme tillatte kombinasjoner av tegn. Lovlige tegn er store og små bokstaver fra det engelske alfabet, tallsiffer og spesialtegn. Konkret betyr det bokstavene A til Z og a – z, sifrene 0 –9 og _ (understrekingstegnet). Første tegn kan ikke være et siffer. Det skilles strengt mellom små og store bokstaver. Det er i utgangspunktet ingen begrensninger på hvor mange tegn som kan brukes, men enkelte C++ kompilatorer kan sette begrensninger på antallet signifikante tegn. I Java kan man også benytte bokstavene Æ, æ, Ø, ø, Å samt spesialtegnet \$.

1.5.1. Nøkkelord og operatører

Man finner mange av de samme nøkkelord, men det er også noen som er spesielle for det ene eller andre språket. Se tabellen som følger. (Vi har også tatt med nøkkelord i C++ som begynner med understrekingstegn (_)). De er Microsoftspesifikke).

| Nøkkelord i Java | Nøkkelord i C og C++ | Kommentar |
|------------------|-------------------------|-----------|
| abstract | | |
| | <code>__asm</code> | |
| | <code>__assume</code> | |
| | <code>auto</code> | |
| | <code>__based</code> | |
| boolean | <code>bool</code> | |
| break | <code>break</code> | |
| byte | | |
| case | <code>case</code> | |
| catch | <code>catch</code> | |
| | <code>__cdecl</code> | |
| char | <code>char</code> | |
| class | <code>class</code> | |
| const | <code>const</code> | |
| | <code>const_cast</code> | |
| continue | <code>continue</code> | |
| | <code>__declspec</code> | |
| default | <code>default</code> | |
| | <code>delete</code> | |
| | <code>dllexport</code> | |
| | <code>dllimport</code> | |

| Nøkkelord i Java | Nøkkelord i C og C++ | Kommentar |
|------------------|----------------------|-----------|
| do | do | |
| double | double | |
| | dynamic_cast | |
| else | else | |
| enum | enum | |
| | __except | |
| | explicit | |
| extends | | |
| | extern | |
| | false | |
| | __fastcall | |
| final | | |
| finally | __finally | |
| float | float | |
| for | for | |
| | friend | |
| goto | goto | |
| if | if | |
| implements | | |
| import | | |
| | inline | |
| | __inline | |
| instanceof | | |
| int | int | |
| | _int8 | |
| | __int16 | |
| | __int32 | |
| | __int64 | |
| interface | | |
| | __leave | |
| long | long | |
| | main | |

| Nøkkelord i Java | Nøkkelord i C og C++ | Kommentar |
|---------------------------|-------------------------------------|-----------|
| | <code>__multiple_inheritance</code> | |
| | <code>_single_inheritance</code> | |
| | <code>__virtual_inheritance</code> | |
| | <code>mutable</code> | |
| | <code>naked</code> | |
| | <code>namespace</code> | |
| <code>native</code> | | |
| <code>new</code> | <code>new</code> | |
| | <code>noreturn</code> | |
| | <code>operator</code> | |
| <code>package</code> | | |
| <code>private</code> | <code>private</code> | |
| <code>protected</code> | <code>protected</code> | |
| <code>public</code> | <code>public</code> | |
| | <code>register</code> | |
| | <code>reinterpret_cast</code> | |
| <code>return</code> | <code>return</code> | |
| <code>short</code> | <code>short</code> | |
| | <code>signed</code> | |
| | <code>sizeof</code> | |
| <code>static</code> | <code>static</code> | |
| | <code>static_cast</code> | |
| | <code>__stdcall</code> | |
| | <code>struct</code> | |
| <code>super</code> | | |
| <code>switch</code> | <code>switch</code> | |
| <code>synchronized</code> | | |
| | <code>template</code> | |
| <code>this</code> | <code>this</code> | |
| | <code>thread</code> | |
| <code>throw</code> | <code>throw</code> | |
| <code>throws</code> | | |

| Nøkkelord i Java | Nøkkelord i C og C++ | Kommentar |
|------------------|----------------------------------|-----------|
| | true | |
| transient | | |
| try | try | |
| | __try | |
| | typedef | |
| | typeid | |
| | typename | |
| | union | |
| | unsigned | |
| | using declaration eller direktiv | |
| | uuid | |
| | __uuidof | |
| | virtual | |
| void | void | |
| volatile | volatile | |
| while | while | |
| | wmain | |

Som man legger merke til er det mange flere nøkkelord i C/C++ enn i Java. Det er også en god del nøkkelord som er Microsoft spesifikke.

Operatorene er stort sett de samme. Neste tabell viser operatorene i Java og C/C++.

| Operatører i Java | Operatører i C/C++ | Kommentar |
|-------------------------|---------------------------|-----------|
| | :: rekkevidde | |
| | :: global | |
| | [] indeksering av tabell | |
| | | |
| . kvalifisering av navn | . valg av medlem (objekt) | |
| () metodekall | -> valg av medlem (peker) | |

| Operatorer i Java | Operatorer i C/C++ | Kommentar |
|---|--|---------------------------------------|
| ! ikke + unær pluss - unær minus ++ inkrement -- dekrement (type) casting new | ++ postfiks inkrement -- postfix dekrement new delete delete[] ++ prefiks inkrement -- prefiks dekrement * dereferering & adressen til | |
| | + unær pluss - unær minus (negering) ! ikke ~ bitvis komplement | |
| | sizeof størrelse av objekt sizeof() størrelse av type typeid() navn av type (type) casting const_cast dynamic_cast reinterpret_cast static_cast | |
| | .* peker til klassemedlem ->* dereferer peker til klassemedlem | |
| * / % | * / % | multiplikasjon divisjon modulus |
| + - | + - | addisjon subtraksjon |
| | << venstre skift >> høyre skift | |

| Operatører i Java | Operatører i C/C++ | Kommentar |
|-------------------|---|--|
| < | < | mindre enn |
| <= | <= | mindre enn eller lik |
| > | > | større enn |
| >= | >= | større enn eller lik |
| == | == | lik |
| != | != | ikke lik |
| | & bitvis og ^ bitvis eksklusiv eller bitvis eller | |
| && | && | logisk og |
| | | logisk eller |
| | e1?e2:e3 betinget utførelse | Denne er spesiell. e1, e2 og e3 er uttrykk |
| = | = | tilordning |
| += | += | sammensatte tilordninger |
| -= | -= | |
| %= | %= | |
| *= | *= | |
| /= | /= | |
| | <<= | venstre skift og tilordning |
| | >>= | høyre skift og tilordning |
| | &= | bitvis og og tilordning |
| | = | bitvis eller og tilordning |
| | ^= | bitvis eks eller og tilord |
| | , | komma |

Som vi ser har C++ mange flere operatører enn Java. Det er også et bidrag til at C++ er et mer komplisert språk enn Java. Ellers er det slik at det brukes de samme operatører for aritmetikk, logikk og tilordning.

Literaler (også navnløse konstanter) angis på samme måte de to språkene. Det samme gjelder kommentarer, bortsett fra at C++ ikke har /**.

1.5.2. Slik er programmer bygget opp

Den syntaktiske struktur er temmelig lik i programmer skrevet i Java og C++. Det samme kan vi si om det kontekstuelle og semantiske nivå. La oss se på noen programeksempler.

| Java | C/C++ |
|--|--|
| <pre>class Omregning { public static void main(String [] args) { final double faktor = 4.2; double antallKalorier = 500; double antallJoule = antallKalorier * faktor; System.out.println("Antall kJ blir " + antallJoule); } }</pre> | <pre>#include <stdio.h> const double faktor = 4.2; void main(int argc, char* argv[]) { double antallKalorier = 500; double antallJoule = antallKalorier * faktor; printf("Antall kJ blir %4.2f", antallJoule); }</pre> |

Her er det som man ser ganske likt når det gjelder oppbygningen av programmene. Man her de samme datatypene, de samme operatørene, de samme skilletegn, den samme oppbygging i blokker mellom krøllparenteser { }. Navngivningen skjer på samme måte. Noen vesentlige forskjeller er det likevel.

- ✓ I Java må alt skje i objekter. Dvs at et program må definere minst en klasse. Klassen må ha samme navn som kildekodefilen. Denne klassen (Omregning i dette eksemplet) må ha en operasjon (metode i Java) som heter `main` og som er `static`. Metoden har en tabell av type `String` som argument.
- ✓ I C/C++ programmer er det funksjonen `main` som er en slags ramme rundt programmet. Alle programmer starter eksekvering med denne funksjonen.
- ✓ Utskrift håndteres i Java gjennom objekter. I C++ er det flere muligheter. I dette eksemplet bruker vi "C måten". (Andre måter som er mer "objektorienterte" skal vi ta for oss etter hvert). Programmet kan med andre ord gjerne gå for å være skrevet i C. Det er nemlig slik at i C kan `main()` returnere `void` (eller ingenting), mens i C++ programmer krever standarden at `main()` returnere en `int`. (Ikke alle C++ kompilatorer er like nøye med dette). I dette eksemplet skjer utskrift ved kall til en biblioteksfunksjon, `printf`. Prototypen til funksjonen ligger i filen `stdio.h`. En prototype er det samme som signaturen til en metode. Kompilatoren må kjenne til denne for å gjøre jobben med å compilere riktig. Vi forteller kompilatoren hvor den kan finne prototypen med kompilatordirektivet `#include`. Dette sier at kompilatoren skal inkludere filen med det angitte navn, i dette tilfellet `stdio.h`. At filnavnet står mellom klammeparenteser, sier at filen finnes i en standard katalog for biblioteker. Det finnes flere kompilatordirektiver. Alle begynner med `#`. Dette med å dele programmer opp i filer med spesielle etternavn (`.h` i dette eksemplet) er vanlig i C og C++. Dette skal vi se nærmere på i senere leksjoner.
- ✓ I C++ har funksjonen `main` to argumenter. Et er et heltall, mens det andre er en tabell med pekere til `char`. (Det er slik tekststrenger håndteres. Mer om det senere). I Java håndteres tekst i klassen `String`. Derfor er det bare et argument i metoden `main` der.

Begge programmene viser typisk kode for programmer skrevet i et imperativt programmeringsspråk. Verken programmet skrevet i Java eller det som er skrevet i C++ er særlig objektorienterte. C++-programmet er det definitivt ikke. Java-programmet har bare en

klasse som for så vidt ikke har annet formål enn å innkapsle et ellers imperativt program. Det betyr at man i Java heller ikke trenger å skrive spesielt objektorienterte programmer. (Det er en av grunnene til at purister ikke anser Java for å være et rent objektorientert språk).

Her følger en variant av C++-programmet.

```
#include <iostream>

const double faktor = 4.2;

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

    double antallKalorier = 500;
    double antallJoule    = antallKalorier * faktor;
    std::cout << "Antall kJ blir " << antallJoule;
    return 0;

}
```

Forskjellen er utskriften og retur fra main(). Her vises ”C++ måten” å gjøre det på. Man sender utskriften til en standard utenheten representert ved klassen *cout*. Den finner vi i biblioteket *iostream* og ettersom *cout* tilhører *namespace std*, må den kvalifiseres med *std::cout*. *namespac* er nytt i C++. Vi kommer tilbake til flere detaljer rundt dette i en senere leksjon.

Den våkne leser la kanskje merke til at i tabellen over operatører så betyr operatoren << venstre skift. Men i dette eksemplet betyr den dirigering av utskrift. Altså er det i C++ slik at det er sammenhengen en operator brukes i som bestemmer hva den står for. Vi har med regler på det kontekstuelle nivå å gjøre. I C++ er det mulig også for programmererne å omdefinere operatører slik at de gjør spesielle ting i spesielle sammenhenger. Det kalles operator overloading. En mekanisme som gjør C++ fleksibelt, men også mer komplisert. Mer om det senere.

1.5.3. Datatyper

Java og C++ har stort sett de samme datatyper. Naturlig nok er det også noen forskjeller. Se tabellen som følger her. Her har vi listet datatypene i C++ og der hvor det finnes tilsvarende i Java er det kommentert. C++ har tre kategorier datatyper. Det er fundamentale (fundamental), avledede (derived) og klasse (class).

| Kategori | Type | Innhold | Tilsvarende Java |
|-------------|--|--|--------------------------|
| Fundamental | char signed char unsigned char | En type som vanligvis inneholder et tegnsett. I Microsoft C++ er det ASCII. Størrelse i MS C++ 1 byte. | char, størrelse 2 byte |
| | short short int unsigned short signed short | En type som har en størrelse lik eller større enn char og mindre eller lik størrelsen til int. Størrelse i MS C++ 2 byte. | short, størrelse 2 byte |
| | int signed int unsigned int | Heltall med fortegn. Størrelse større eller lik short int og mindre eller lik størrelsen til long. Størrelse i MS C++ 4 byte. | int, størrelse 4 byte |
| | <code>__intn</code> | En int hvor størrelsen i bit er lik n . Verdien på n kan være 8, 16, 32, 64. | |
| | long long int unsigned long | Et heltall hvor størrelsen er lik eller større enn størrelsen til int. Størrelse MS C++ 4 byte | long, størrelse 8 byte |
| | float | float er den minste type flyttall. Størrelse i MS C++ 4 byte. | float, størrelse 4 byte |
| | double | En flyttallstype som er større enn eller lik float. Størrelse i MS C++ 8 byte | double, størrelse 8 byte |

| Kategori | Type | Innhold | Tilsvarende Java |
|--|----------------------------|---|------------------|
| | long double | long double og double er like i størrelse, men betraktes som forskjellige typer. Størrelse i MS C++ 8 byte. | |
| Direkte avledede typer er typer som er direkte avledet av eksisterende typer | Tabeller (arrays) | | Samme type |
| | Funksjoner | Funksjoner er datatype. | |
| | Pekere | | |
| | Referanser til objekter | | |
| | Konstanter | Samme som literal | |
| | Pekere til klassemedlemmer | | |
| Sammensatte avledede typer (composed derivatived types) | class | Samling av data og funksjoner. Innhold default private. | class |
| | struct | Samme som class, men innhold public ved default | Finnes ikke. |
| | union | Type som kan inneholde forskjellige variable i samme minneområde. | Finnes ikke. |

I C++ kan man deklareere egne typer med nøkkelordet `enum`. Enum er forkortelse for enumeration på engelsk. Det oversettes til norsk med oppramsing. En enumerated type er en brukerdefinert type som består av et sett med navngitte konstanter. Ved default har første enumerator verdien 0 og hver etterfølgende verdi en større enn foregående. Men dette kan

endres eksplisitt. En enum kan gjøres om til heltall. I de første versjonene av Java var enum ikke med. Nå er den det¹.

Eksempel

```
enum Dager          // Erklærer enum type Dager
{
    lørdag,          // lørdag = 0 ved default
    søndag = 0,      // søndag = 0 likedan
    mandag,          // mandag = 1
    tirsdag,         // torsdag = 2
    onsdag,          // etc.
    torsdag,
    fredag
} idag;             // Variabel idag har type Dager

int tirsdag;        // Error, redefinisjon av tirsdag

enum Dager igår;    // Lovlig i C og C++
Dager imorgen;     // Lovlig bare i C++
```

Vi kommer etter hvert til å se bruk av de fleste datatyper i C++ gjennom dette kurset.

1.6. Litteratur

1. Bal & Grune, "Programming Language Essensial", Addison-Wesley 1994, ISBN 0-201-63179-2

¹ Det er også andre ting i C/C++ som ikke var med i Java til å begynne med, men som etter hvert har kommet til. Man må altså stadig holde seg oppdatert. En annen kommentar er at det Javautviklerne mente var uheldige ting i C/C++ nå har blitt akseptert som nyttig. Templates er også kommet med. Det betegnes som *generics* i Java.