Hello.cpp*

(Global Scope)

```cpp
#include <iostream>
using namespace std;

int main()
{
    //say hello
    cout << "Hello C++" << endl;

    system("PAUSE");
    return 0;
}
```

Tomas Sandnes – santom@westerdals.no

2015/16 Vår

PG4400 C++ programmering

FORELESNING 8 (AV 12)

Dagens temaer

- SDL repetisjon & tips.
- Prosjektoppsett generelt.
- Strømmer (fil Input/Output).
- Bit operasjoner (binære tall).
- Innleveringsoppgaven.

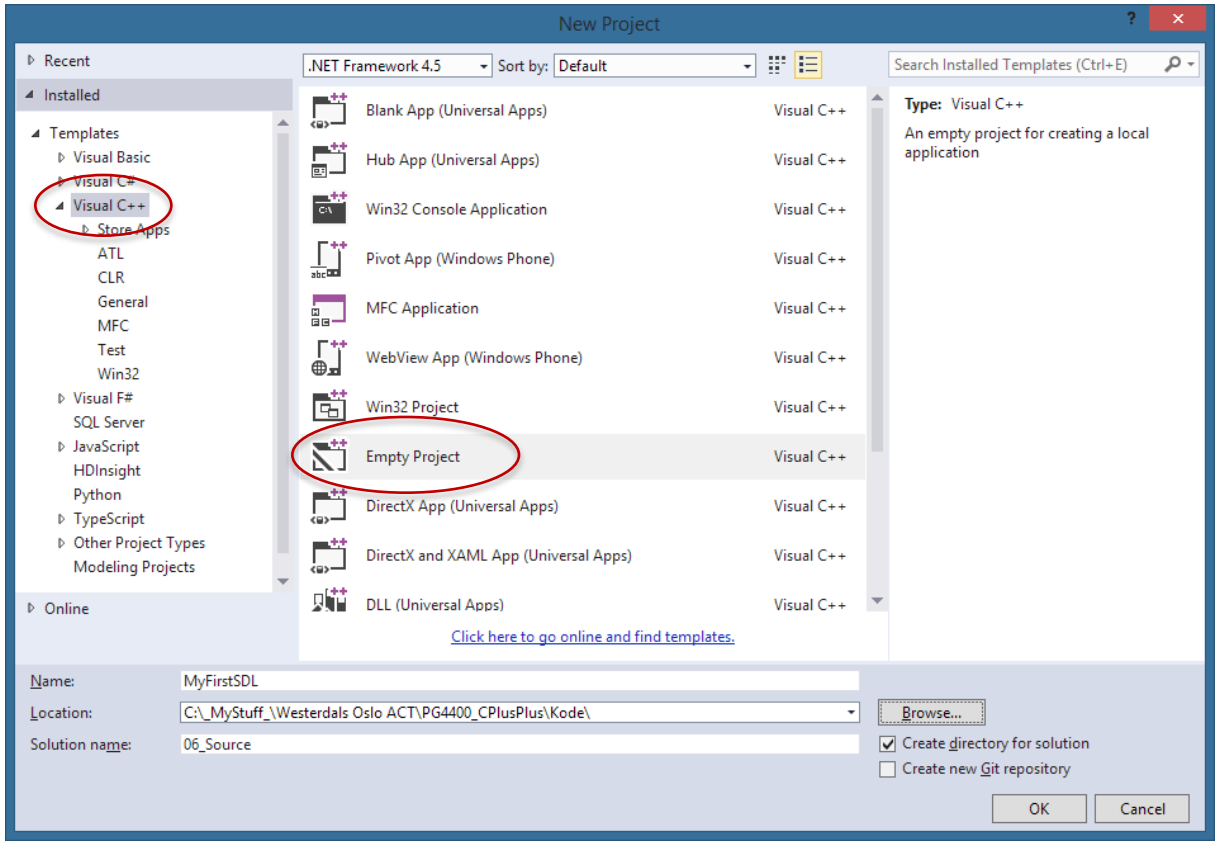
SDL – repetisjon

- SDL: Simple
  Directmedia
  Layer

- Nyttige lenker:
  - SDL homepage
  - SDL wiki
  - Lazy Foo SDL tutorials
  - (Lenkene finnes også på its learning, PG4400.)

Vårt første SDL prosjekt – rep.

- Opprett et "Visual C++" -> "Empty Project":



SDL via NuGet – rep.

- I stedet for å laste ned manuelt, lage macroer, sette paths, kopiere lib'er ...
  - Drop alt det styret der!

- VELDIG enkelt:
  - Høyreklikk prosjektet ditt.
  - Velg "Manage NuGet packages..."
  - Søk ("Search") etter: "sdl2".
  - Finn og installér: "sdl2.v140"

- FERDIG! :-D

SDL extension libraries – rep.

- Det finnes en del nyttige "extension libraries" til sdl, disse er:
- sdl2_image.v140 for å kunne benytte en rekke bildeformater, bl.a.: bmp, gif, jpg, png, tiff.
- sdl2_ttf.v140 for å kunne benytte TrueType fonts.
- sdl_mixer 2.0 for flerkanals lyd.
  - Har ikke klart å finne denne i NuGet, installer etter "gamle måten"
  - Link: http://www.libsdl.org/projects/SDL_mixer/
- sdl_net 2.0 for nettverksstøtte.
  - Har ikke klart å finne denne i NuGet, installer etter "gamle måten"
  - Link: http://www.libsdl.org/projects/SDL_net/

SDL og dynamisk minne – rep.

- SDL oppretter gjerne objekter dynamisk (altså med `malloc`) internt i sine funksjoner/metoder.
  - Dette kommer ikke alltid tydelig fram, og vi må derfor lese dokumentasjonen/kodekommentarene til funksjonene vi bruker.

- Eksempel: Alt som har med grafikk/blit'ing å gjøre i SDL benytter surfaces.
  - Når vi kaller SDL funksjoner som gjør noe med surfaces, returnerer disse som oftest pekere til nye surfaces (laget med `malloc`).
  - For å frigjøre dette minnet igjen må vi kalle: `SDL_FreeSurface(SDL_Surface*)`

Få opp et SDL vindu – rep.

```
#include <iostream>
#include <SDL.h>
// NB: Denne koden tar seg IKKE av feilhåndtering ved init.
int main(int argc, char* argv[]) {
    SDL_Init(SDL_INIT_VIDEO); // Init. SDL2
    SDL_Window* window = NULL; // Pointer to Window
    // Lag et vindu med gitte settings
    // For alle mulige konfigurasjonsalternativer, se: http://goo.gl/8vDJN
    window = SDL_CreateWindow(
        "Et awesome SDL2-vindu!",            //    window title
        SDL_WINDOWPOS_UNDEFINED,             //    initial x position
        SDL_WINDOWPOS_UNDEFINED,             //    initial y position
        550,                                 //    width, in pixels
        400,                                 //    height, in pixels
        SDL_WINDOW_SHOWN | SDL_WINDOW_OPENGL //    flags
        );
    // Sjekk om noe gikk galt
    if (window == NULL) {
        std::cerr << "Failed to create window: "
            << SDL_GetError() << std::endl;
        SDL_Quit(); // Rydd opp!
        return EXIT_FAILURE;
    }
    SDL_Delay(3000); // Pause i 3 sekunder, lukk vinduet
    SDL_DestroyWindow(window);
    SDL_Quit(); // Be SDL om å rydde opp
    return EXIT_SUCCESS;
}
```

Laste inn et bilde – rep.

```
// NB: denne koden "bygger på" forrige eksempel!
SDL_Renderer* renderer; // Pointer to window's renderer

// Lag en renderer til det spesifikke vinduet. Setter Hardware accelerated flag.
renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
// Sjekk om noe gikk galt
if (renderer == NULL)
{
    std::cerr << "Failed to create renderer: "
        << SDL_GetError() << std::endl;
    SDL_DestroyWindow(window); SDL_Quit(); // rydd opp
    return EXIT_FAILURE;
}

// Set renderens bakgrunnsfarge til svart
SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
SDL_RenderClear(renderer); // Nullstiller renderen (til svart)

// Last inn et bilde fra disk (NB: støtter KUN BMP!)
SDL_Surface* surface = SDL_LoadBMP("my_picture.bmp");
// Sjekk for feil
if (surface == NULL)
{
    std::cerr << "Failed to load image: "
        << SDL_GetError() << std::endl;
    SDL_DestroyRenderer(renderer); SDL_DestroyWindow(window); SDL_Quit(); // rydd opp
    return EXIT_FAILURE;
}
```

Laste inn et bilde – rep. forts.

```
// … fortsettelse fra forrige slide

// Konverter surface om til et HW Accelerated Texture, laster objektet opp på skjermkortet
SDL_Texture* drawable = SDL_CreateTextureFromSurface(renderer, surface);

// Sett opp et "koordinatsystem" for bildet
SDL_Rect coords;
coords.h = surface->h; // Samme bredde og høyde som surface
coords.w = surface->w;
coords.x = 0; // Endre disse for å "flytte" bildet i vinduet/renderer
coords.y = 0;

SDL_FreeSurface(surface); // Frigjør surface fra CPU-minnet

/* GAME LOOP START */
// Endre koordinatene ved brukerinput (coords.x, coords.y

// Legg til objektet i vinduets renderer
SDL_RenderCopy(renderer, drawable, NULL, &coords);

// BLIT/rendre bildet
SDL_RenderPresent(renderer);
SDL_RenderClear(renderer); // Tøm renderen for innhold

/* GAME LOOP SLUTT */
```

SDL: fjerne console for release

- TIPS: Om du vil fjerne console vinduet fra release build versjonen: (det gjør seg ikke så godt når spillet skal ”lanseres” ...)
  1. Gå til Project->Properties, deretter:
  2. Configuration Properties -> Linker -> System.
  3. Gjør om første rad fra ”Console” til ”Windows”.

- NB: Gjør dette for release build da!

Prosjektoppsett, generelt

- TIPS, for prosjekter med eksterne ressurser: (grafikk, m.m.)
  - Ressursene bør ligge i en mappe plassert i solution mappa.
  - Koden bør ligge i egen project mappe under solution mappa.
  - Path til ressursene bør spesifiseres relativt når de refereres i koden.

- Eksempel: Om jeg har grafikk i en Graphics mappe, bør denne mappa plasseres i solution mappa, og elementer i Graphics mappa bør hentes ved å spesifisere pathen relativt, på denne måten:
  - ”`..//Resources//gameover.bmp`” (← Eller annet filnavn.)

- Struktur som dette gjør ressursene tilgjengelige for både:
  - Debugging i fra Visual Studio, og:
  - exe filer i både debug & release mappene.

Output ... vs Working Directory

- Å referere til filer i et prosjekt i Visual Studio kan være litt tricky:
  - Visual Studio har forskjellige "startmapper" avhengig av om man kjører i debug mode eller kjører via exe fila.

- Dette kan endres i prosjektoppsettet:
  - Project -> Properties -> Configuration Properties -> General -> Output Directory.
  - Project -> Properties -> Configuration Properties -> Debugging -> Working Directory.

- Sett f.eks. Working Directory lik Output Directory:
  - `$(SolutionDir)$(Configuration)\`

Forskjellige filtyper – rep.

- .cpp filer (source files / kildefiler)
  - Inneholder *definisjoner* (implementering) av funksjoner og klasser.
- .h(eader) filer (eller .hpp filer)
  - Inneholder *deklarasjoner* (prototyper) av funksjoner og klasser.
- Lib'er (objektfiler)
  - Inneholder ferdig kompilert versjon av innholdet i cpp filer. (Obs: Disse er ikke synlige i prosjektet.)
- Resource filer
  - Inneholder elementer i prosjektet som ikke er kode. Versjonsinformasjon, icon-fil, bildefiler, ...

Strømmer (data til/fra fil)

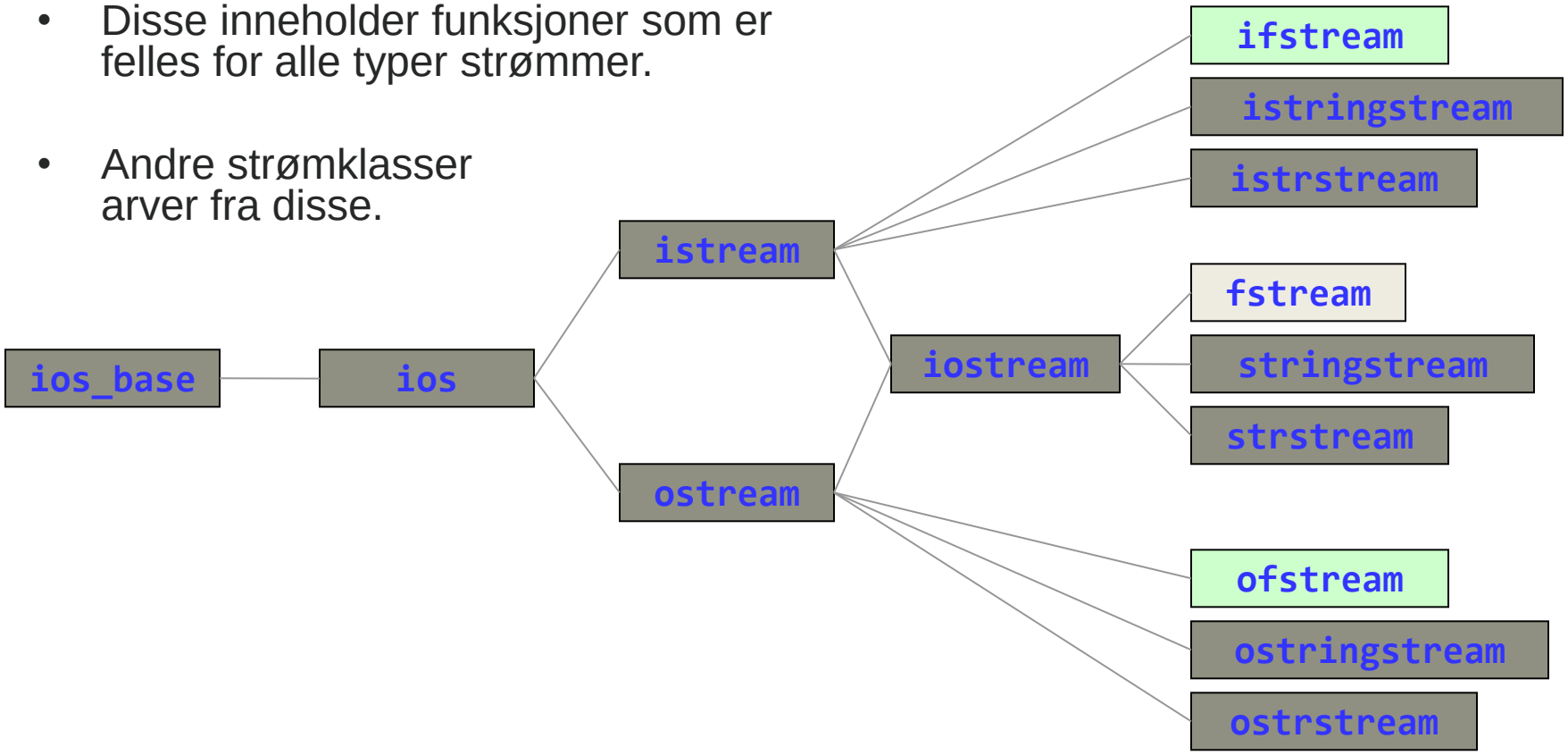
- En strøm (stream) er en sekvens av bytes.

- Bruken av strømmer for lesing og skriving til ulike media er grunnleggende i C++.
  - Vi kjenner alt til `cin` og `cout`.
  - (Trivia: 'c' står her for "character" – ref. Stroustrup.)

- En strøm kobles til forskjellige typer enheter:
  - Disker/minne
  - Keyboard
  - …

Strømmer – forts.

- Grunnleggende klasser for alle strømmer er `ios_base` og `ios`.
- Disse inneholder funksjoner som er felles for alle typer strømmer.
- Andre strømklasser arver fra disse.



Strømmer – forts.

- Det finnes noen predefinerte strøm-objekter i disse klassene:
  - `cin` standard input, type `istream`.
  - `cout` standard output, type `ostream`.
  - `cerr` standard error, type `ostream`.
  - `clog` standard logg, type `ostream`.

- Deklarert i `<iostream>`. (Som arver både `<ostream>` og `<istream>`.)

Koble strømmer til filer

- Vi kan definere strømmer som kobles til filer for lesing eller skriving.

- Slike strømobjekter må være en av typene:
  - `ifstream` (lesing)
  - `ofstream` (skriving)
  - `fstream` (lesing og/eller skriving)

- Disse klassene er deklarert i headerfilen `<fstream>`.

- Strømmene kobles til en navngitt fil med et kall til funksjonen `open()`, eller med et implisitt kall i konstruktøren.
  - (Se eksempel neste slide.)

Koble strømmer til filer – forts.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
        ofstream fileOut;
        fileOut.open("highScores.txt");
        for (int i = 0; i < 5; ++i) {
                fileOut << "Highscore: " << i * 3 << endl;
        }
        fileOut.close();

        ifstream fileIn("highScores.txt");
        string score;
        while (!fileIn.eof()) {
                getline(fileIn, score);
                cout << score << endl;
        }

        system("Pause");
        return 0;
}
```

Inkluderer fstream.

Åpne fil v1: Eksplisitt kall til **`open(...)`**.

Overlastet **`<<`** for ofstream-objekter. (Brukes som for **`cout`**.)

Eksplisitt kall til **`close()`**. Buffer flush'es.

Åpne fil v2: Implisitt kall til i konstruktøren.

`eof()`: lese til vi når enden på fila.

`getline(...)` er også definert for ifstreams.

Koble strømmer til filer – forts.

- Merk kallet til `close()`. Et eksplisitt kall til `close()` stenger forbindelsen til strømmen.

- Destruktøren i [io]fstream-objekter kaller også på `close()`.

- Dermed blir `close()` utført også når et [io]fstream objekt opphører å eksistere.
  - (Enten delete eller ut av scope.)

Funksjoner relatert fstream

open(<filnavn>)	Åpner en navngitt fil. Filnavn MÅ være cstring/char array.
close()	Lukker forbindelsen til en strøm
bool is_open()	Sjekker om en forbindelse kan brukes.
clear()	Reset strøm
get()	Henter enkelttegn fra strøm
getline()	Henter linje fra strøm

- For mer om stream/fil io, se msdn, cplusplus.com, e.l.

Sjekke åpne filer

- Et ifstream-objekt er ikke garantert å være koblet til en strøm, selv om objektet er opprettet.

- Vi må sjekke eksplisitt om den ønskede filen faktisk er åpen.
  - Kan kalle på medlemsfunksjonen `is_open` for å oppnå dette.
  - Kan også teste på objektet for seg selv.

```
ifstream fileIn("enfil.txt");
if (!fileIn.is_open())  // Eller: if (!fileIn)
{
    cout << "Klarte ikke åpne filen " << endl;
}
```

Operatoren `!` er overlastet for stream objekter.

Fil-modi

- En fil kan åpnes i ulike modi. Dette bestemmes vha. parameter når man kaller `open(...)`. De ulike fil-modi er definert i ios_base:
  - `ios_base::in` Åpne for lesing
  - `ios_base::out` Åpne for skriving
  - `ios_base::ate` Gå til end-of-file
  - `ios_base::app` Legg til innhold ved end-of-file
  - `ios_base::trunc` Skriv over innhold (trunkér)
  - `ios_base::binary` Binær fil
- Kan kombinere modi med | (bitwise OR).
- Default modi:
  - for en ifstream: `ios_base::in`
  - for en ofstream: `ios_base::out | ios_base::trunc`

Binære tall – rep. 1. klasse

- Binære tall / totallsystemet representerer tall ved hjelp av sifrene 1 og 0.

- Det finnes operasjoner som er spesifikke for binære tall:
  - NOT (komplement/invers)
  - AND
  - OR
  - XOR (exclusive OR)

Operasjoner på binære tall

- NOT (komplement/invers)

- NOT er en unær operator, og betyr at vi tar den inverse verdien av hver bit, slik at 0 -> 1 og 1 -> 0.

NOT

1	1	1	0	0	0	1	0

0	0	0	1	1	1	0	1

Operasjoner på binære tall

- AND (alle = ‘1’ gir ‘1’ til resultat)

	1	0	1	1	0	1	0	1
AND	1	1	1	0	0	0	1	0
	1	0	1	0	0	0	0	0

- OR (en eller flere ‘1’ gir ‘1’ til resultat)

	1	0	1	1	0	1	0	1
OR	1	1	1	0	0	0	1	0
	1	1	1	1	0	1	1	1

Operasjoner på binære tall

- XOR (‘ulike verdier’ gir ‘1’)

	1	0	1	1	0	1	0	1
XOR	1	1	1	0	0	0	1	0
	0	1	0	1	0	1	1	1

- Hver av disse operasjonene kan i C+ utføres med en egen operator:
  - NOT ~
  - AND &
  - OR |
  - XOR ^

||, &&, | og &

- || og && fungerer som i Java. (Logisk ”or” og ”and” i forbindelse med `if...else` eller liknende.
  - *MERK:* Stopper så fort en test garantert er `true` eller `false`!

```
if (myIntPtr && (0 < *myIntPtr))
{
        ...
}
```

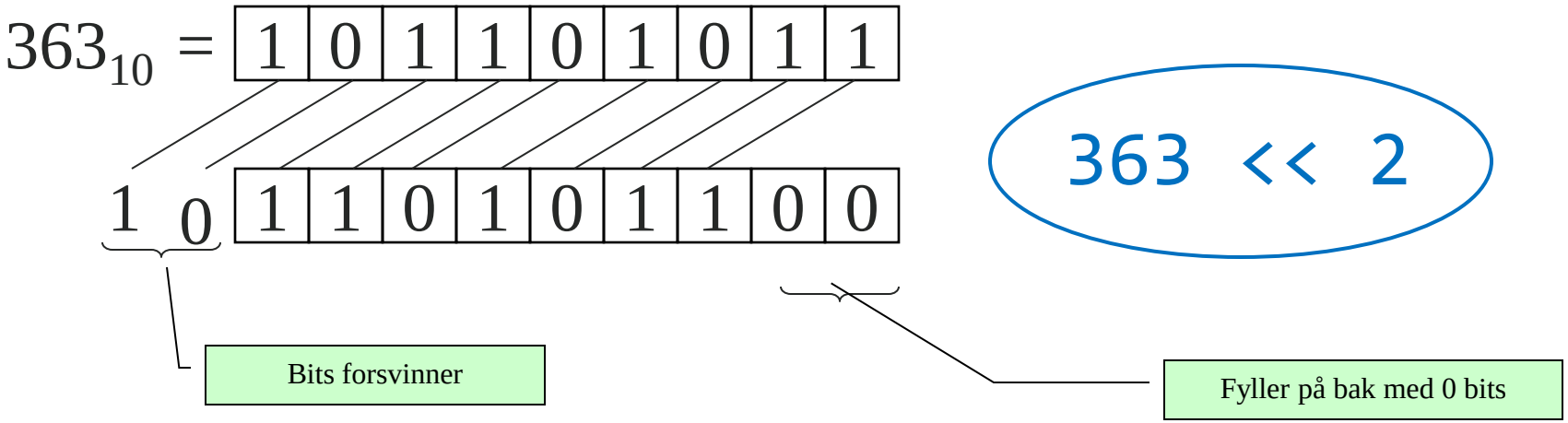
- I tillegg har vi ”bitwise or” og ”bitwise and”. Disse representeres med | og &, og brukes for å summere bits:

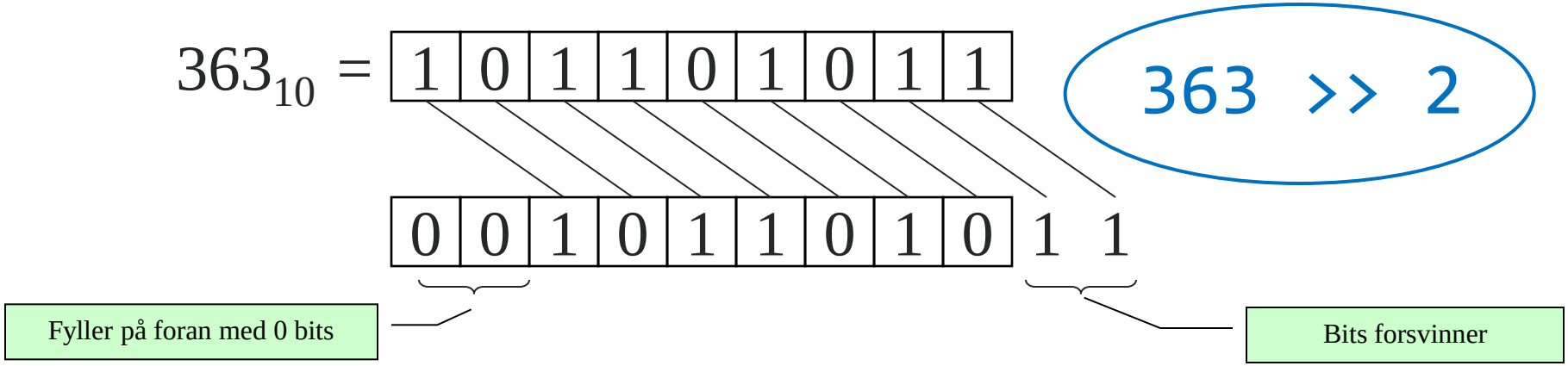
```
short bothBits = bitValueOne | bitValueTwo;
```

Bit-shift operatorene << og >>

- I tillegg har C++ noen spesielle operatorer for å manipulere bit-strenger: Shift-operatorene `<<` og `>>`. (Må ikke forveksles med input- og output-operatorene!)
- .Syntaks: *`verdi << shift`*.
- `<<` betyr at alle bit i *`verdi`* skal skiftes *`shift`* antall ganger til venstre, og de ledige plassene fylles med `0` (tallet blir som oftest større).
- `>>` betyr at alle bit i *`verdi`* skal skiftes *`shift`* antall ganger til høyre, og de ledige plassene fylles med `0` (tallet blir mindre).

Bit-shift operatorene << og >>





Bit-operatorer og tilordning

- Alle operatorer vi har sett så langt kan også brukes i tilordningsuttrykk:

~	→	~=
&	→	&=
|	→	|=
^	→	^=
<<	→	<<=
>>	→	>>=

```
int a = 23, b = 7;
a <<= 2;  // a = a << 2;
cout << a << endl;  // 92
a &= b;   // a = a & b;
cout << a << endl;  // 4
a ^= b;   // a = a ^ b;
cout << a << endl;  // 3
```

Bit-flagg

- I noen sammenhenger har vi bruk for å slå av eller på en eller flere enkelt-bit (for eksempel til bruk som “flagg”).

- La bitstreng være en rekke av bits, og flagg 1 og 2 være enkeltbits. Slå på bits med |.

Flagg 1	0	0	0	1	0	0	0	0
Flagg 2	0	0	0	0	0	0	1	0

Flagg 1 | Flagg 2

Bitstreng	0	0	0	1	0	0	1	0

Bit-flagg, eksempel

- Fra opprettelse av windu, SDL 2.0:

```
// Flags for SDL: We're using graphic card memory
// (opposed to system memory), and double-buffering.
Uint32 windowFlags = SDL_WINDOW_SHOWN | SDL_WINDOW_OPENGL;
```

- Hvis vi sjekker verdiene: (medlemmer i en enum)

```
SDL_WINDOW_OPENGL = 0x00000002,
SDL_WINDOW_SHOWN = 0x00000004,
```

- Ergo inneholder `windowFlags` det hexadesimale tallet `00000006`.
- *Spm: Hvor mange flagg kan et slikt hexadesimalt tall inneholde?*
  - 32 flagg: 4 flagg per siffer, ganger 8 siffer.

Innleveringsoppgaven

- Breakout-clone!
- Skal være ferdig om ca 1 måned:
  - Søndag 17. april, 23:59.
    (Rett før neste prosjektuke i PJ3100.)
- Teller 40% av karakteren.
- Gjøres i grupper på 2-3 personer.
  - Jeg anbefaler å benytte parprogrammering: Altså at man sitter sammen og koder, ikke koder hver sin del.

Vis innleveringsdokument.

Neste gang og øvingstimer

- Neste gang: *(om 2 uker, først påskeferie.)*
  - Gjesteforelesning ved Olve Maudal: "Modern C++".
  - Veldig dyktig foreleser, dette vil dere ikke gå glipp av! :-)

- Dagens oppgaver:
  - Begynne på innleveringen:
    Sette opp prosjekt, inkludere SDL pakker, m.m.

The

... på teoridelen. Men vi fortsetter frem til 14:00 med lab/øvinger!

(Her, i *dette* rommet! ;-)