Notes til eksamen i DAT154

By Sondre Lindaas Gjesdal

Pensumliste DAT154 Vår 2020

C# 6 for programmers / Paul Deitel, Harvey Deitel.

Hele boken er i utgangspunktet pensum, MEN:

- Mange konsepter skal være kjent fra tidligere programmeringskurs, slik som klasser, objekter, kontrollstrukturer, metoder, arv, unntakshåndtering. Kapitlene som omhandler dette kan leses vært verfladisk eller hoppes over om dere allerede føler dere behersker dette. Det vil ikke være naturlig å ha utdypende spørsmål om dette på eksamen, MEN dere kan bli bedt om å forlate kode eller krive kode som inneholder disse tingene, så det er viktig dere har forståelse av det.
- Boken bruker en del sider på å utdype spesifikke kontroller i grafiske elementer. Det er ikke spesielt relevant å pugge disse, en oversikt over den overordnende virkemåten til slike kontroller generelt er nok.
- Hold fokus på det vi har snakket om i timene

Windowsprogrammering med Windows SDK/C++

- SDK-delen av boken er pensum (til og med side 72)
- Hold fokus på det vi har snakket om i timene

Microsoft Application Architecture Guide 2nd Edition

- Del 1 (Kapittel 1 t.o.m. 4) er pensum

Forlesningsnotater

Forelesningsnotater/slides og tilhørende kodeksempler som lagt ut på Canvas er pensum

Laboppgaver

Laboppgavene er pensum. Det er her viktig at dere skjønner hva dere har gjort og hvorfor/hvordan det virker. Det forutsettes at dere har implementert alt som oppgavebeskrivelsen ba om.

Stikkord

(Listen inneholder sentrale elementer, men det kan selvsagt dukke opp temaer fra pensum som ikke står på denne kortlisten under eksamen)

C++

What is C

C is a functional programming language form the 70s. Used to make complex systems like drivers and OS

What is C++

- C++ is a high-level programming language developed by Bjarne Stroustrup at Bell Labs. Based on C
- C++ adds OO features to it's predecessor, C. C from 1970s, and C++ from 1980s.

Where is C/C++ used

- OS like Linux, Windows etc
- Drivers for different hardware units
- Advanced logic
- A lot of programs written between 1970 and now.

Differences from Java

1. C++ makes machine code directly
2. Has Pointers
3. Handles memory manually - No garbage collection

Similarities

Object oriented

C++

Typical C/C++ Development Environment usually goes through five phases:

1. Edit
2. preprocess
3. compile
4. link
5. execute.

- Edit: Make a text file
- Preprocess: 1 part compiling - include header files and resolve constant symbols
- Compile: After symbols and include files have been resolved and included - transfers text to machine code
- Link: Bind your machine code modules together with libraries and make a complete program
- Execute: Run the program

Hello World

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World!\n";
    cin.get();
}
```

Header files in C++

The C/C++ Standard Library gives a variety of functions, which signature/prototype is in a header files.

In C normally all the header files begin with the .h extension, but in C++ not always like `<iostream>` A header file in C/C++ contains:

- Function definitions
- Data type definitions
- Macros

Header files offer of a preprocessor directive called `#include`. These preprocessor directives are responsible for instructing the C/C++ compiler that these files need to be processed before compilation.

C++ program normally contains the header file `<istream>` which has input and output functions/objects

- cin for input
- cout for output

The Preprocessor and `#include`

Before the source code is compiled, it gets automatically processed due to the presence of preprocessor directives. All preprocessor instructions typically start with a # sign. The most common is `#include` and `#define`.

File, compiling and linking

In C++ the code files are first made into pure machine code by the COMPILER(cl). These are called object files.

Then the linker binds and links the object files together and makes a real program

Namespace C++

Namespaces allow us to group named entities that otherwise have global scope into narrower scopes, giving them namespace scope.

- Namespace is a feature added in C++ and is not present in C
- A namespace is a declaretive region that provides a scope to the identifiers (names of the types, function, variables etc.) inside it.
- Multiple namespace blocks with the same name are allowed. All declarations within those blocks are declared in the named scope.

Klasser i C++

Nothing special to note compared to C# and Java. Maybe that one specifies what is public and private with the `private:` and `public:` modifiers(?).

Konstuktører/Destruktører

Constructors are as normal constructors. Can also be defined like this

```
class A
{
  public:
  int x;
  //constructor
  A(); // constructor declared
};

// Constructor definition
A::A()
{
  i = 1;
}
```

Destructor

```
class A
{
  public:
  // defining destructor for class
  ~A()
  {
    // statement
  }
}
```

Example to see how the Constructor and Destructor are calles

```
class A
{
  // Constructor
  A()
  {
    cout << "Constructor called";
  }

  // Destructor

  ~A()
  {
    cout << "destructor called";
  }
};

int main()
{
  A obj1; // constructor called
  int x = 1;
```

```
  if(x)
  {
    A obj2; // constructor called
  }   // destructor called for obj2
}; // constructor called for obj1
```

Minnehåndtering

It's important to delete objects after use, especially in graphics, to prevent memory leaks.

Pekere/addresser

A pointer in C is simply a variable that stores an adress. All tables in C are simply(constant) pointers to a place in memory.

It takes some time to get used to pointers, they are the main difference between C/C++ and Java/C#.

"WIKI: A **Pointer** is a variable that holds a memory address where a value lives. A pointer is declared using the * operator before an identifier. As **C++** is a statically typed language, the type is required to declare a pointer".

A C/C++ pointer normally has a TYPE which indicates which TYPE of variable it points to. Very important to understand this

`char *pch; double *pd; int *pi;`

- Is used to declare a pointer, ex: `int *p;`
- & the address operator is used to extract the address of a variable - typically to initiate a pointer
    - `int x = 10;`
    - `int *p = &x;`

`*` is used to dereference(get the contents) of WHAT a pointer points to. ex: `cout << *p;`

```
int main()
{
  int i = 3;
  double x = 3.14;

  int* pi = &i;
  double* px = &x;

  cout << "i = " << *pi << "x = " << *px;
}
```

Another example:

```
int x = 10, y = 30;

int* p = &x; //10
```

```
cout << "x=" << x << endl;

*p = 20;

cout << "x=" << x << endl; //x= 20

cout << "*p= "<< *p << endl; //*p= 20

p = &y;

cout <<"*p= " << *p << endl;// *p=30
```

- Struktur

SDK

Struktur

- Main method(WinMain)
- Code to register a windows class
- Code to create a main window of that class
- A message that loops **IMPORTANT**
- A window procedure attached to the main window. **IMPORTANT**, Where all the `WM_`commands is located
- An SDK project contains several important files/filetypes:
    - *.h: Regular C++ header files
    - *.rc: Visual Studio resource files. Contains definitions for graphical resources, like dialog boxes and menus.
    - Other resources such as bitmaps.
    - Resource.h: Defines constants for all resources used in the project
    - *.cpp: Regular C++ source files
- Vindusmeldinger
    - Meldingsløkke
    - Meldingshåndtering
- Resurser
    - Editing resource files
        - Can be edited by hand(normally not)
        - Use the Visual Studio Resource Editor
- Dialoger

    - A dialog is used for input of data to the program or for just giving information to the user.
        1. Create the dialog in the resource editor
        2. Write a window procedure for the dialog
        3. Show the dialog:

```
LRESULT DlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam){}
DialogBox(hInst, MAKEINTRESOURCE (IDD_DIALOG1), hWnd, DlgProc);
```

- Dialog Windows Messages
    - `WM_INITDIALOG`
        - Sent when the dialog is created. All dialog initialization code should be called from here(ex: Setting default values in controls)
    - `WM_COMMAND`
        - Sent whenever an event occurs on controls in the dialog, like clicking a button, selecting a menu item, etc.
        - Get the triggering control with `LOWORD(wParam)`

Grafikk

- Device Contexts
- GDI-Objekter
- Virtuelle device contexts/kopiering
- Animasjon

Funksjonspekere

Tråder

.Net

- Runtime platform and development frameworkd for cross-platform applications.
- This means you may mix C#, C++, F#, VB.NET or whatever language
- It is composed of several components

Struktur

CLR

- Common Language Runtime
    - The CLR is a kernel that reside on top of the OS, and can execute code in a language called Common Intermediate Language(CIL)
    - All .NET applications are compiled to CIL instead of native machine code
        - Similar concept as Java bytecode

Managed Code

- Managed Code is code that targets the CLR

- Code that does not target the CLR(old DLL's, API calls, etc) is unmanaged code
- Managed code is handled by the CLR, that provides
    - Security checks/constraints
    - Memory Management/Garbage Collection
    - Access to managed data
- Typer

Assemblies/metadata

- A .NET assembly is an executable (.exe) or library (.netmodule) that requires the CLR to run
- Such a file contains code compiled MSIL in addition to code, it also contains metadata(ildasm.exe)
- Similar to a Java .jar file
- Contains metadata which describes the interface. These data may be read by a browser, for example the class browser in Visual Studio, or more important the compiler during the build process when multiple assemblies are involved.
- Must be run under a CLR
- Is portable(To any system with a CLR)

Navnerom/referanser

- A class is identified by both its namespace and actual name
- When we use a class, we need to tell the compiler both the namespace and the name of the class

```
System.Console.Writeline("Hello World");
Library.Book book = new Library.Book("Title");
```

- For Commonly used namespaces, we can import the namespace by using the using directive

```
using System;
class ChelloWorld{
  public static void Main(){
    Console.WriteLine("Chello World");
  }
}
```

Some important namespaces

- **System** - Fundamental and base classes, interfaces and exceptions
- **System.Data** - Classes for accessing and managing data (from external sources)
- **System.Drawing** - Basic GDI+ support
- **System.Text** - String handling
- **System.Windows** - GUI components

Creating a namespace

- A namespace is created by wrapping our class(es) in a namespace block.

```
namespace NS
{
  public class X
  {
    public static String HW()
    {
      return "Hello World NS";
    }
  }
}
```

Nesting namespaces

- Namespaces can also be nested

```
namespace NS1
{
  namespace NS2
  {
    public class X
    {
      public static void F()
      {
        System.Console.WriteLine("F's in the chat for guy, bois");
      }

    }
  }
}
```

- The `using` keyword does not provide access to namespaces nested below the specified level

Correct:

```
NS1.NS2.X.F();

using NS1.NS2;
X.F();
```

Incorrect:

```
using NS1;
NS2.X.F();
```

```
using NS1;

X.F();
```

Aliases in namespaces

- One can use aliases to provide a simplified name for a class or namespace

```
using s = System; // namespace alias
using c = System.Console; // Class
```

References

- References points to external libraries.
- By default, not everything is referenced from a project
- Might need to set up a reference to access a given namespace
- References can be browsed in the object browser in VS

Reference types

- Reference types are stored on the heap
- A pointer to the memory location occupied by the type is stored on the stack
- Assigning a reference type to a new variable creates a new pointer on the stack pointing to the same object on the heap
- Implicit inheritance from System.Object

Value types

- A value type does not point to the heap, it keeps its value on the stack
- When assigned to a new variable, the content is copied
- Still behaves as an Object, and has methods
- Implicit inheritance from System.ValueType

FCL (Framwork Class Library)

- The FCL is a comprehensive collection of reusable types, including classes, interfaces and data types included in the .NET framework to provide access to system functionality.
- FCL acts as a standard library.
- The reusable types of FCL provide a simple interface to developers due to:
    - Their self-documenting nature
    - Lesser learning curve to understand the framework, which expedites and optimized the development process
    - Seamless integration of third-party components with classes in FCL

Biblioteker

- .Net Library assemblies are almost identical to executable assemblies
- They do not need a Main() method
- To create a library, compile with `/target:library`
- To reference a library in an executable, compile with `/r:libraryname.dll`

Grafiske grensesnitt

- Windows Forms
- WPF
- Universal Apps
- Delegater

Hendelser (Events)

- Håndtering
- Typer

Grafikk

- Tegneverktøy
- Animasjon

Linq

- Databaser
- XML

Web

- ASP.NET
- MVC
- WebAPI

Databaser

- ADO.NET
- Linq

Lambda

Arkitektur

- Modulær oppbygning
- Bruk av multiple programmeringsspråk
- Programvarearkitektur
- Arkitekturmønstre
- Universell utforming