



Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

Hvis du ikke vil være med på opptaket:

	La være å delta med webkameraet ditt.
	La være å delta med mikrofonen din.
To: <span>Marianne Sundby</span> (Privately) Type message here...	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.



Høyskolen  
Kristiania

# PG3401 Programmering i C for Linux

Bengt Østby

# **PG3400 Programming i C for Linux**

## **Lecture #9 : More on structs, lists, trees & 3<sup>rd</sup> party libraries + Windows**

# Repetition...

- Preprocessor
- Structs and unions
- Pointers to structs
  
- And then...
  - Lists
  - Trees

# #include <stdio.h> vs. #include "list.h"

- Instructs the preprocessor to include some other file
- It simply reads the given file before continuing in the current file.

< > implies the system include folders

- /usr/include
- /usr/local/include

" " implies a given path, by default the cwd

- #include "list.h"
- #include "/home/hka/list.h"
- #include "../include/list.h"

# Other preprocessor

#define - #undef

#if - #elif - #else - #endif

#ifdef - #else - #endif

#pragma – mostly **#pragma once** or #pragma pack(*n*)

# #ifdef and macros

```
#ifdef DEBUG
    #define DEBUG_PRINT(x) printf(x)
#else
    #define DEBUG_PRINT(x)
#endif
```



# #ifdef vs #if defined()

```
#if defined(_MSC_VER)
```

```
    // Microsoft C spesific defs...
```

```
#elif defined(__GNUC__)
```

```
    // Gnu C spesific defs...
```

```
#else
```

```
    #pragma message «Unsupported compiler!»
```

```
#endif
```

# Structures

A heterogeneous collection of data members

Much like classes but :

- All its members are *public*
- Just data members – no methods

*Unions* are also possible but different from structs.

# struct

Structures must be declared before using them.

```
struct employee{  
    int id;  
    char name[20];  
    int salary;  
};
```

Then the declaration of a variable of the employee type.

```
struct employee e1,e2,e3;
```

Notice the **struct** keyword again!

# struct

You can use typedef

```
typedef struct _EMPLOYEE {  
    int id;  
    char *name;  
    int salary;  
} EMPLOYEE;
```

Now you can use

```
EMPLOYEE e1,e2,e3;
```

- or -

```
struct _EMPLOYEE e4, *pe;
```

# struct

You can skip the name completely!

- Not really useful, but supported

You can use the same name for the struct and typedef.

- Handy to keep code readable

Accessing elements using '.'

```
e1.id = 10;  
strcpy(e1.name, "penguin");  
e1.salary = "42";
```

Accessing from pointers using ->

# More about structs

## Nested structs:

- struct inside struct inside struct
- Accessing by multiple “.” accesses
- Easy stuff for java-ers

## Recursive definitions are a bit complicated!

- You need to have the size of struct in compile time!
- Solution : Pointers

Remember to allocate memory!

# Size of structs

**sizeof()** struct is a bit weird!

Example :

```
typedef struct{  
    int id;  
    char name[38];  
    int salary;  
} employee;  
employee e1,e2,e3
```

Padding...

Force non-padding - **#pragma pack(1) .. #pragma pack()**

# Assignment & comparison

‘=’ will do a bitwise copy:

- shallow copy
- take care when you have pointers!
- Easy to create memory leaks.

Comparison is not trivial

‘==’ is pretty much illegal on structs

You have to just compare the members individuals



# Pointers to struct

Example declaration:

```
typedef struct employee{  
    int id;  
    char name[38];  
    int salary;  
} employee;  
employee *e1,*e2;
```

- Now, they are just pointers like any other pointers
- It is a programmers problem to assign memory and properly initialize them.
- How do we use them?

# Memory allocation

Allocate N structs.

```
struct employee *pStruct;  
pStruct = malloc(N*sizeof(struct employee)) ;
```

Accessing them:

```
pStruct[i].id // will work  
(pStruct+i)->id // will also work
```

# Initialization

Example:

```
typedef struct{
    double x;
    double y;
} coord;

typedef struct {
    char name[20];
    coord vertices[4];
} rect;

rect r1 = {"first",
    {
        {0,1},{0,0},{1,0},{1,1}
    }
};

rect r2 = { "second", 0,1,0,0,1,0,1,1};
```

They can be fewer in number - rest are set to 0

# Welcome to the rabbit hole

# Vanlig feil fordi dere lærer høynivå først...

```
pElement->psValue = psValue
```

Hva er pekere, og hvorfor virker ikke det alltid hvis psValue er en streng peker?

# Vanlig feil fordi dere lærer høynivå først...

```
char buffer[256];
while (true){ // Følgende er en del av student kode fra i fjor:

    memset(buffer, 0, sizeof(buffer));
    printf("Add an element\n");
    printf("Type the value of the element:\n");
    if(fgets(buffer, sizeof(buffer)-1, stdin) != NULL){
        //Check if the input contains only
        if(strcmp(buffer, "\n") != 0){
            strtok(buffer, "\n");
            //Add element to the database
            if(addElement(&db, buffer) == OK){
                printf("\nElement <%s> was added\n«
                "Size of the database is now: %d\n\n",
                buffer, getSizeDatabase(&db));
            }
        }else{
            printf("ERROR: not valid input\n");
        }
    }
}
```

# Vanlig feil fordi dere lærer høynivå først...

```
int addElement(DATABASE **ppDatabase, char *psValue){
    int iCheck = OK;

    //Create the element and set the value
    QUEUE *pElement = (QUEUE*) malloc(sizeof(QUEUE));
    memset(pElement, 0, sizeof(QUEUE));

    if(pElement == NULL){
        printf("ERROR: not able to create element\n");
        iCheck = NOT_OK;
    }
    pElement->psValue = psValue;

    //Add the element in the end of the list
    if((*ppDatabase)->pHead == NULL && (*ppDatabase)->pTail == NULL){

        //The database is empty and the element will be the first element
        (*ppDatabase)->pHead = pElement;
        (*ppDatabase)->pTail = pElement;
    }else{

        (*ppDatabase)->pTail->pNext = pElement;
        pElement->pPrev = (*ppDatabase)->pTail;
        (*ppDatabase)->pTail = pElement;
    }

    (*ppDatabase)->iSize++;

    return iCheck;
}
```

```
typedef struct _QUEUE{
    struct _QUEUE *pNext;
    struct _QUEUE *pPrev;
    char *psValue;
}QUEUE;
```

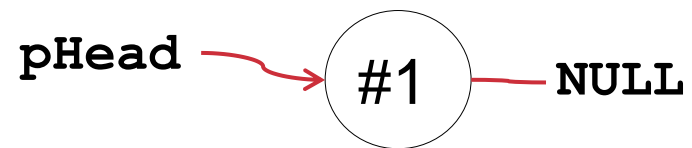
# Lists and trees...

```
typedef struct _LIST {  
    struct _LIST *pNext;  
    char szName[80];  
} LIST;
```

```
LIST *pHead = NULL;
```

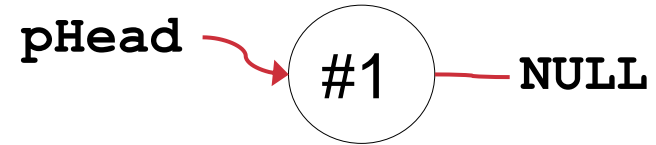
```
LIST *pThis = calloc(1, sizeof(LIST));
```

```
pHead = pThis;
```

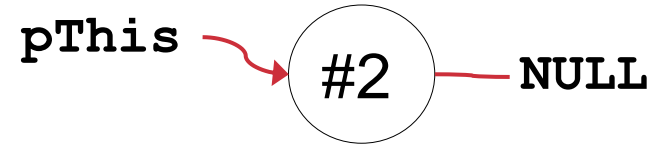




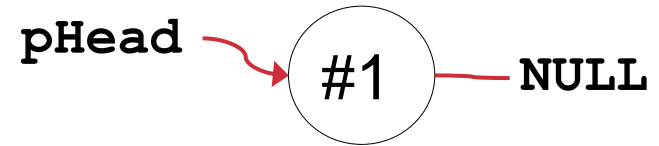
# Lists and trees...



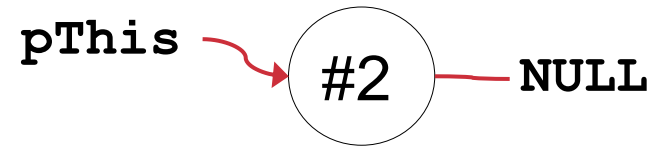
```
LIST *pThis = calloc(sizeof(LIST)) ;
```



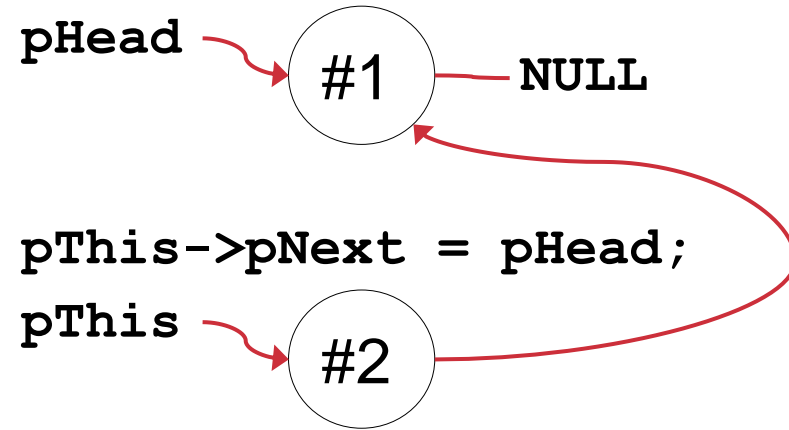
# Lists and trees...



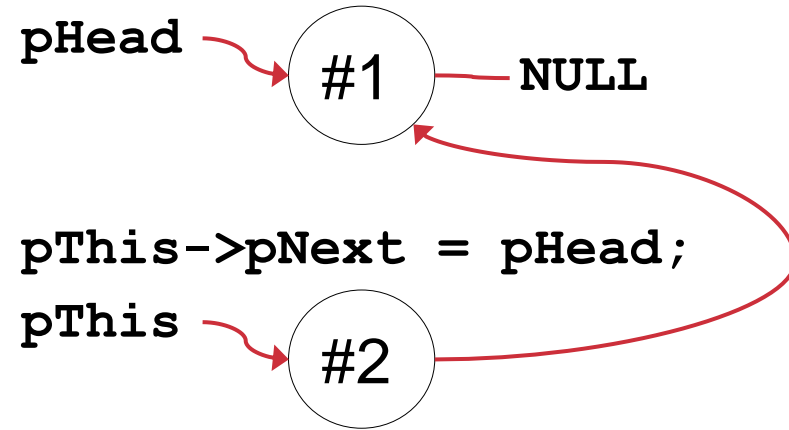
`pThis->pNext = pHead;`



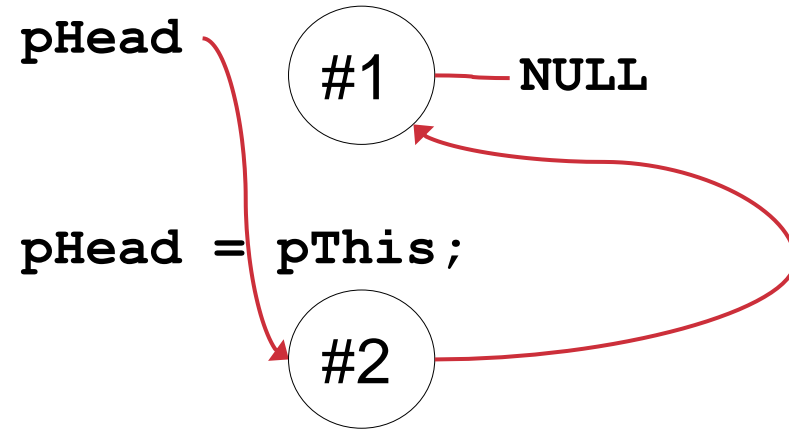
# Lists and trees...



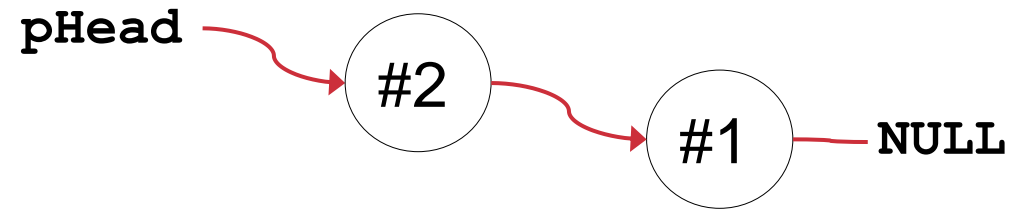
# Lists and trees...



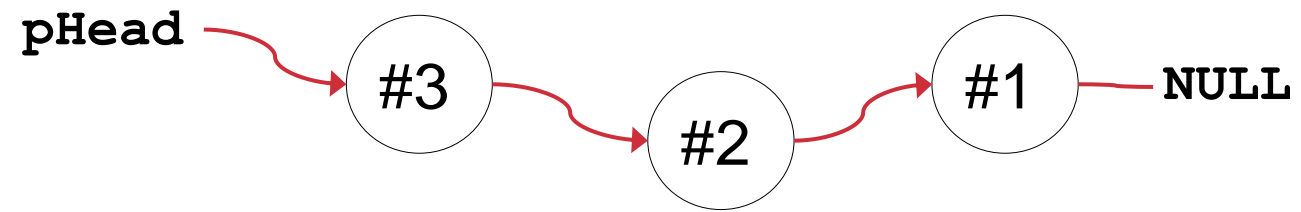
# Lists and trees...



# Lists and trees...

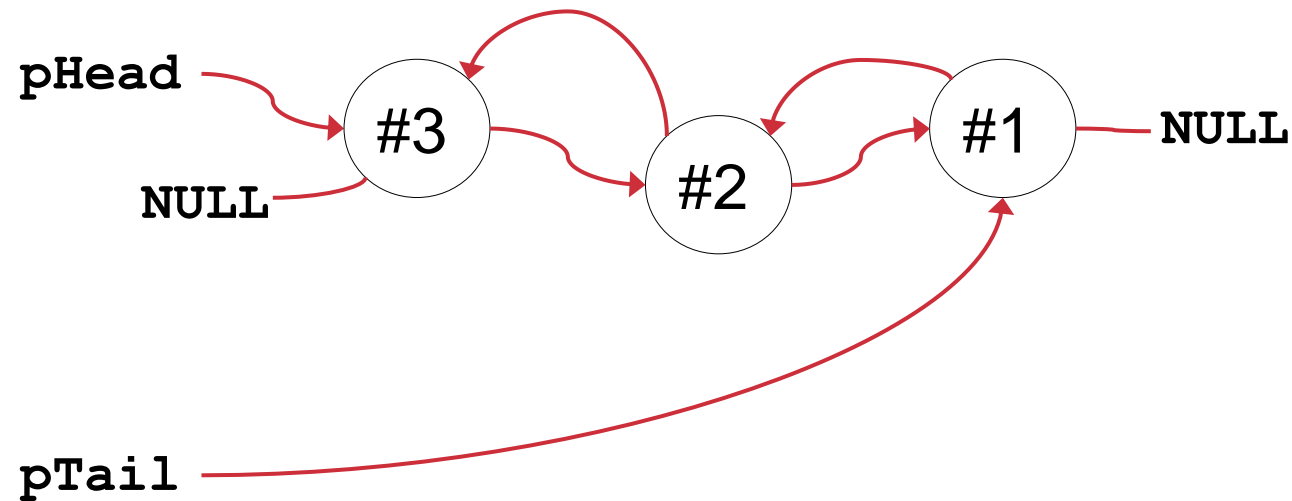


# Lists and trees...



# Lists and trees...

```
typedef struct _LIST {  
    struct _LIST *pNext;  
    struct _LIST *pPrev;  
    char szName[80];  
} LIST;
```





# A key-value database with 3 possible «types»

string = «Headline»  
timeout = 5  
server = «\\gandalf»

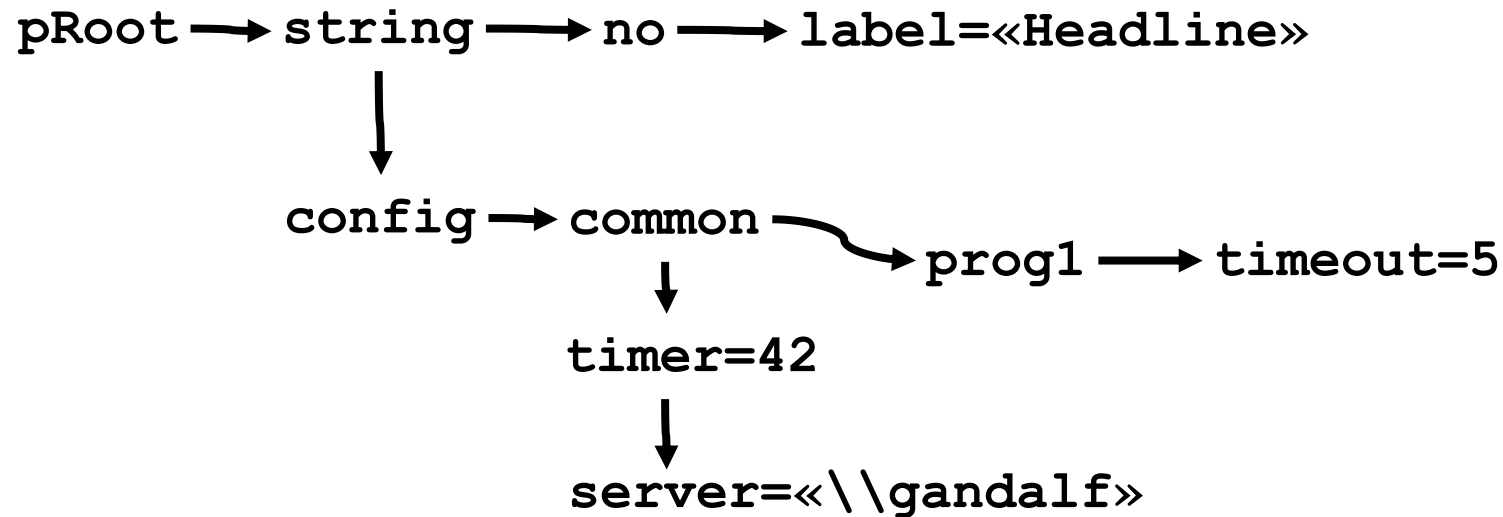
```
typedef struct _LIST {  
    char szKey[32];  
    char *pszValue;  
    int iValue;  
    struct _LIST *pNext;  
} LIST;
```

# Key-value database

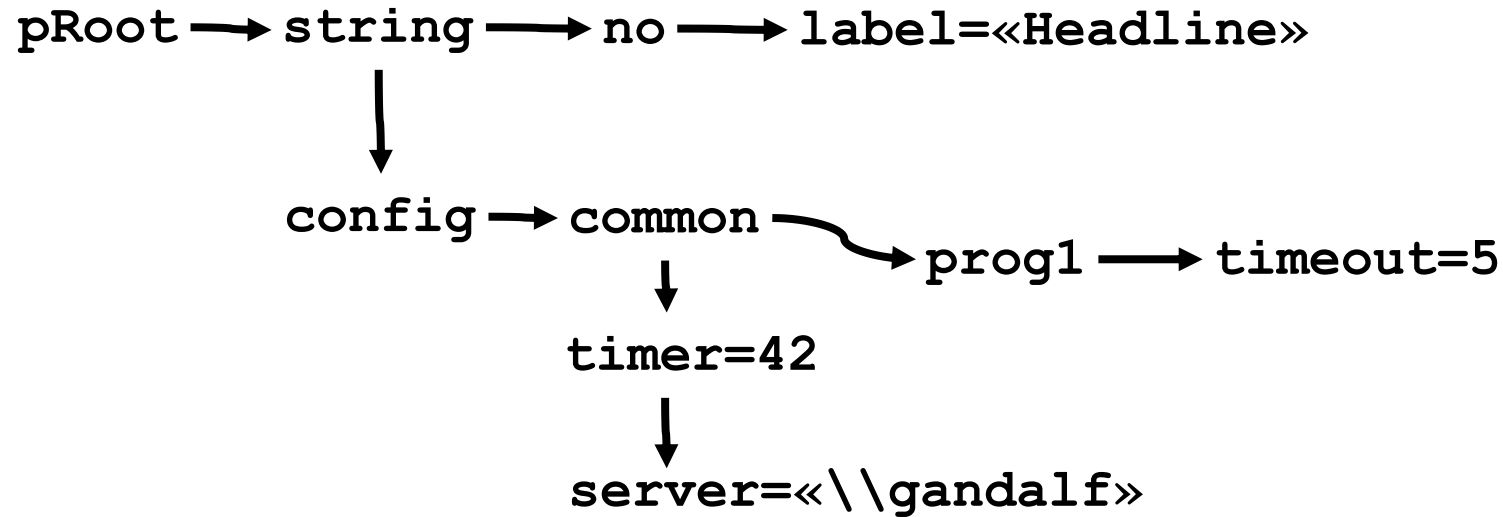
```
string.no.label = «Headline»  
config.common.prog1.timeout = 5  
config.timer = 42  
config.server = «\\gandalf»
```

# Key-value database

```
string.no.label = «Headline»  
config.common.prog1.timeout = 5  
config.timer = 42  
config.server = «\\gandalf»
```



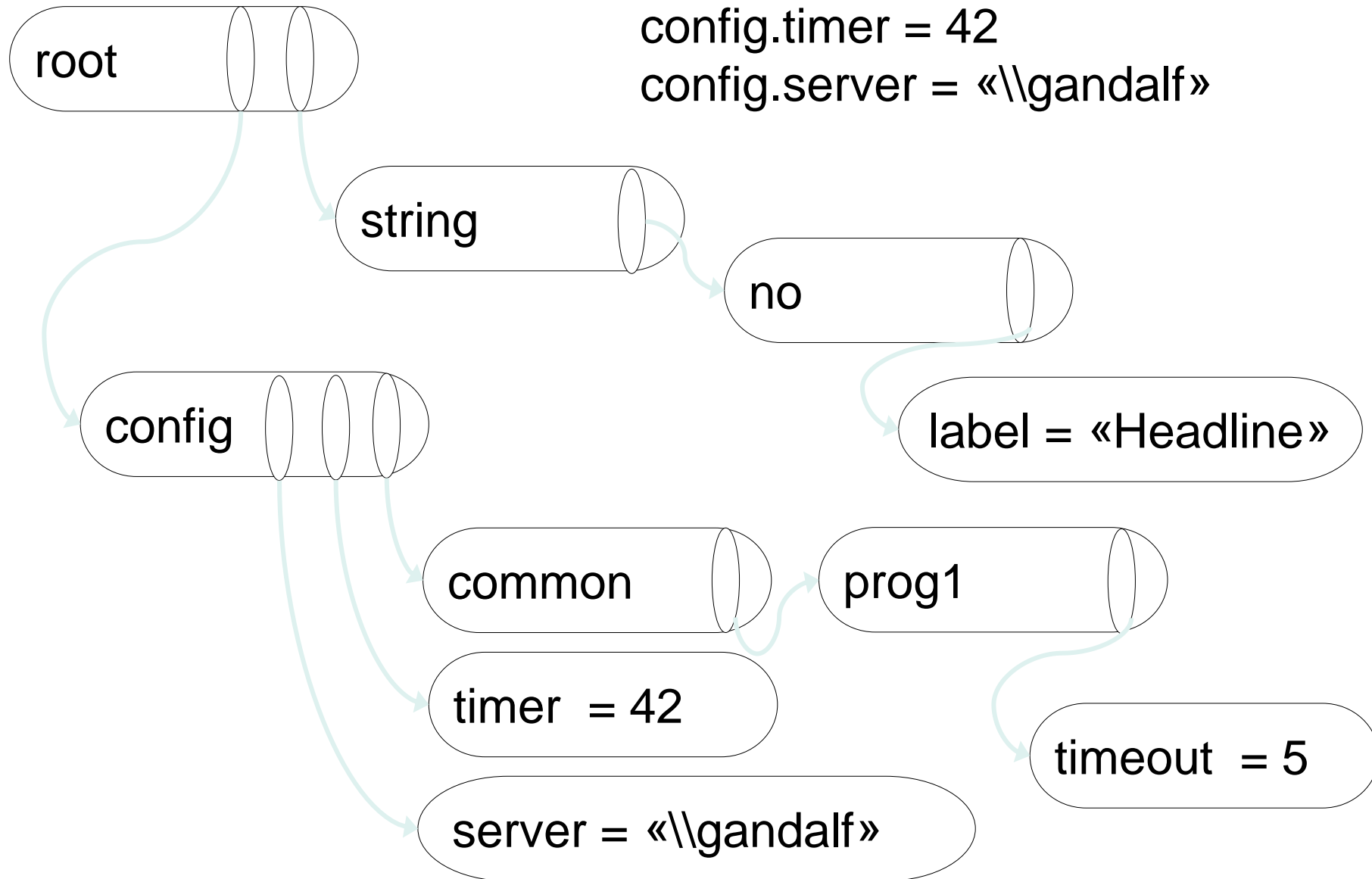
# Key-value database



```
typedef struct _NODE {  
    char szKey[32];  
    char *pszValue;  
    int iValue;  
    struct _NODE *pNext;  
    struct _NODE *pDown;  
} NODE;
```

# Key-value database

```
string.no.label = «Headline»  
config.common.prog1.timeout = 5  
config.timer = 42  
config.server = «\\gandalf»
```



# Key-value database

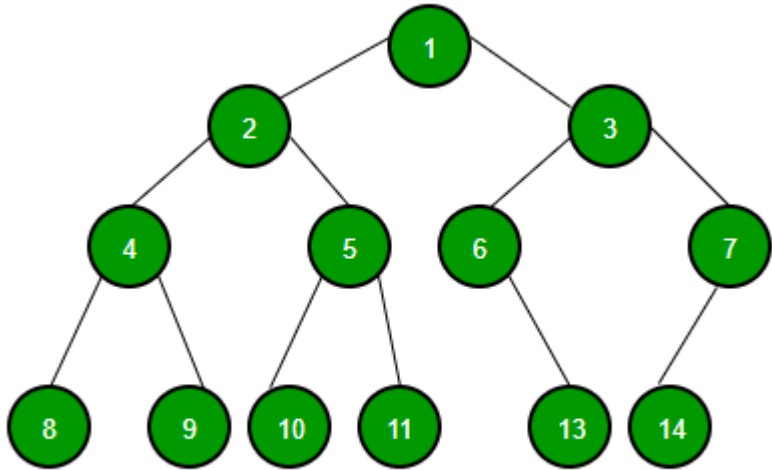
```
#define MAX_NODES 10

typedef struct _BNODE {
    char szKey[32];
    char *pszValue;
    int iValue;
    struct _NODE *pNodes[MAX_NODES] ;
} BNODE;
```

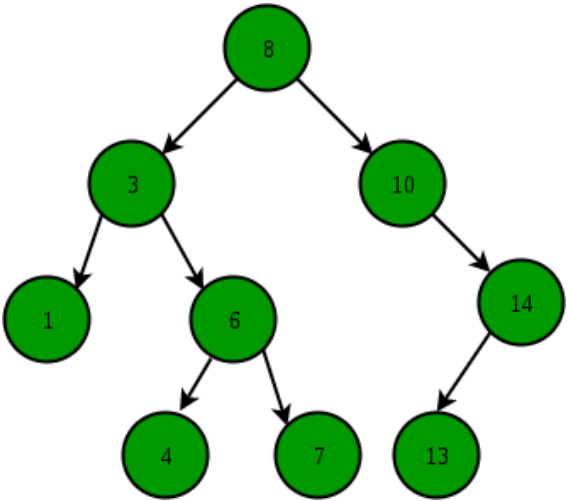
# Key-value database

```
typedef struct _BNODE {  
    char szKey[32];  
    char *pszValue;  
    int iValue;  
    int iNodes;  
    struct _NODE *pNodes[];  
} BNODE;
```

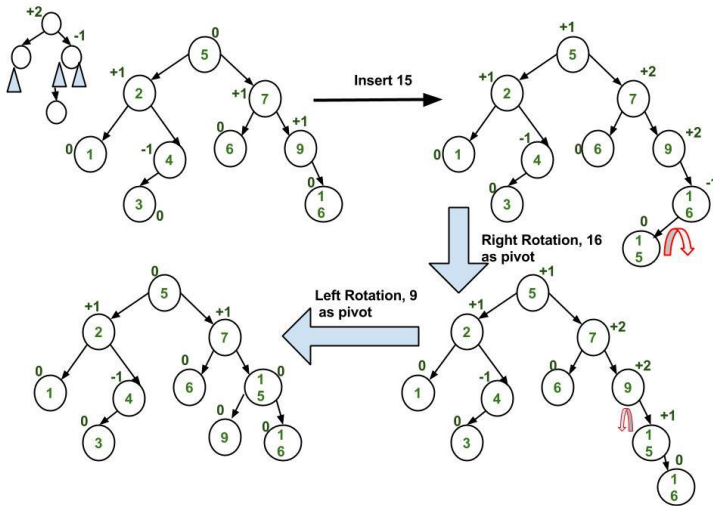
# Different trees



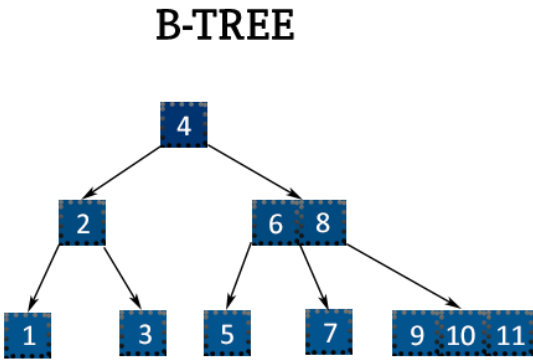
Binary tree



Binary search tree

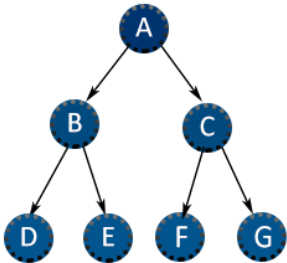


AVL tree



B-TREE

BINARY TREE



B tree (vs Binary tree)



# LIST\_ENTRY (double linked list)

```
typedef struct _LIST_ENTRY
{
    struct _LIST_ENTRY *Flink;
    struct _LIST_ENTRY *Blink;
}LIST_ENTRY,*PLIST_ENTRY;
```

```
typedef struct newstruct
{
    int num2;
    char alpha;
    LIST_ENTRY list_entry;
    float exp;
}MYSTRUCT,*PMYSTRUCT;
```

```
#define CONTAINING_RECORD(address, type, field) (\
    (type *) ((char*)(address) - (unsigned long) (&((type *)0)->field)))
```

# LIST\_ENTRY (double linked list)

Used in Windows kernel (and must be used there because many kernel mode APIs use these as generic lists), but can also be used on Linux.

<https://www.codeproject.com/Articles/800404/Understanding-LIST-ENTRY-Lists-and-Its-Importance>

[https://man7.org/linux/man-pages/man3/LIST\\_ENTRY.3.html](https://man7.org/linux/man-pages/man3/LIST_ENTRY.3.html)

Not taught in this course because using your own linked lists is just as easy (and it is not ANSI-C), but worth mentioning when we learn about lists.

I mention it because LIST\_ENTRY is very slick, if you start using it you will probably never go back... 😊

# Row-major vs column major storage of arrays

[https://en.wikipedia.org/wiki/Row-\\_and\\_column-major\\_order](https://en.wikipedia.org/wiki/Row-_and_column-major_order)

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

The two possible ways are:

Row-major order,  
e.g., C (0-indexed)

Address	Access	Value
0	<code>A[0][0]</code>	$a_{11}$
1	<code>A[0][1]</code>	$a_{12}$
2	<code>A[0][2]</code>	$a_{13}$
3	<code>A[1][0]</code>	$a_{21}$
4	<code>A[1][1]</code>	$a_{22}$
5	<code>A[1][2]</code>	$a_{23}$

Column-major order,  
e.g., Fortran (1-indexed)

Address	Access	Value
1	<code>A(1,1)</code>	$a_{11}$
2	<code>A(2,1)</code>	$a_{21}$
3	<code>A(1,2)</code>	$a_{12}$
4	<code>A(2,2)</code>	$a_{22}$
5	<code>A(1,3)</code>	$a_{13}$
6	<code>A(2,3)</code>	$a_{23}$

# Making and using static libraries:

- Making a static library  
`gcc -c obj1.c obj2.c obj3.c -O0 -g`  
`ar rcs libname.a obj1.o obj2.o obj3.o`
- Linking a static library  
`gcc -o program obja.o objb.o libname.a`
- The code in **libname.a** becomes a part of the **program**.

# Shared library (IKKE PENSUM)

- Shared by multiple programs!
- Linked externally and loaded at runtime:

```
gcc -c -fPIC -O0 -g obj1.c obj2.c obj3.c  
gcc -shared -fPIC -o libsom.so obj1.o obj2.o obj3.o
```

- Link with the library (the simple way):

```
gcc -L/home/hka/shared -Wl,-rpath=/home/hka/shared  
-Wall -o test test.c -lsom
```

See also <https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html> (and other sources)

# How to use 3<sup>rd</sup> party libraries

- OpenSSL
- Curl

We talked about downloading applications from SOURCE and compiling them in a lab-exercise in TK110x (Lecture 0x05 Operatingsystem), but I assume none did...

I recommend doing this now. There are two things you do not want to write yourself; encryption and http...

OPENSSL: Can be used in any application for standalone encryption, but also for SSL and TLS encryption. Guarantees a safe implementation of encryption. I recommend testing standalone functions like those for AES and various hashing, do not try to implement custom TLS protocols – it is quite hard...

CURL: Very simple API for downloading files over HTTP/HTTPS 😊

# How to use 3<sup>rd</sup> party libraries

For ANY open source library follow these steps:

1. Google «Using OpenSSL from C on Linux»
2. Follow instructions given

You can be almost sure it will not be using something as simple as GCC, it will be some other tool you have never heard of, just follow the instructions. If you cannot get it to work someone else has struggled with the same – so google it.

It is not required for the exam, but it will a) help you understand libraries better, and b) help you prepare for working as a programmer.

***Employer: We need to move from XORing the userdata to using AES.***

***You: It is Linux, I know this...***

# Makefiles confusing?

Read all about it at:

<http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>



# **WINDOWS PROGRAMMING**

## **(ikke pensum)**

# Windows programming

- There were some interest last time to learn some of the differences between Linux and Windows system programming in C
- This is not obligatory, and it is not relevant for the exam, those who don't want to learn about this can feel free to log off now 😊

# Compiler

- Default compiler is Microsoft
- Most companies use Microsoft Visual C++ IDE, but you can also use their command line make tools
- Most kernel developers use the command line, but later versions of the IDE also support kernel development and compiling in GUI

# Thread handling

- On Linux we used POSIX threads; pthread
- The POSIX subsystem on Windows was removed in Windows XP, but 3<sup>rd</sup> party libraries exist to be able to use pthreads on Windows – but no one does that...
- In Windows programming you use CreateThread

<https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread>

- A thread function takes void\* argument, but the creating of the thread has a different syntax, which means you can #IFDEF around the create function, but still keep the same thread function

# Synchronization

- Main difference is that LINUX has un-named objects such as mutex and semaphores – this means that objects must be created and then sent to the other thread or process (through fork)
- In WINDOWS objects are named, this means that threads and processes can open their own instance of mutexes and threads with the same NAME – and use that to synchronize
- This has security issues though, as ANYONE can open a mutex and signal it, which might destroy the application...
- Windows has a “multiple wait” function, that can wait on all objects being signaled
- Combining with many objects such as THREADS being signal-able objects a function can wait for all threads to close, which can have a TIMEOUT value so you can do work and then go back into a wait state...

# Synchronization #2

- All though named objects are the most used, and is supported, on Windows it is preferable to use un-named objects as that makes your code easier to port
- At Norman this was the main pain-point; code was developed for Windows and the Linux developers had to rewrite all synchronization primitives on Linux to “simulate” named objects on Windows...
- Windows synchronization seam to be written to be most flexible, Linux synchronization is written to be “cleaner” and possibly result in better code?

# Process synchronization

- Process synchronization on Linux can be tricky (because it lacks named synchronization primitives), it is easy if one application creates the other through FORK, but if not then it is – tricky...
- On Windows named objects means it is as easy to synchronize processes as it is to sync threads
- On Linux you typically use pipes or sockets for passing data
- On Windows shared memory is common for passing data

# Creating a new process

- Very easy, or complex depending on what you need, to create on Windows

<https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa>

```
BOOL CreateProcessA( LPCSTR lpApplicationName,  
LPSTR lpCommandLine,  
LPSECURITY_ATTRIBUTES lpProcessAttributes,  
LPSECURITY_ATTRIBUTES lpThreadAttributes,  
BOOL bInheritHandles,  
DWORD dwCreationFlags,  
LPVOID lpEnvironment,  
LPCSTR lpCurrentDirectory,  
LPSTARTUPINFOA lpStartupInfo,  
LPPROCESS_INFORMATION lpProcessInformation );
```



# Creating a new process #2

But then it becomes hard:

<https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessasusera>

<https://docs.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createprocesswithlogonw>

<https://docs.microsoft.com/en-us/windows/win32/api/winsvc/nf-winsvc-createservicea>

It is possible to exploit this by creating processes for other users, it is difficult because Microsoft try to prevent “privilege escalation”, but if you are admin you can do a lot of tricks with “as user” features 😊

And, what were they thinking?

<https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createremotethreadex>

# System call (file operations)

- You can use FILE\* and fopen/fread etc on Windows
- But you don't, as the Windows API is so much richer...

<https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea>

```
HANDLE CreateFileA( LPCSTR lpFileName,  
DWORD dwDesiredAccess,  
DWORD dwShareMode,  
LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
DWORD dwCreationDisposition,  
DWORD dwFlagsAndAttributes,  
HANDLE hTemplateFile );
```

# Network operations

- Windows has implemented their own sockets, called WINSOCK
- All network operations will be different between Windows and Linux
- We will learn network operations on Linux next week, but I will not go into WINSOCK programming, other than giving you this for future reference:

<https://www.winsocketdotnetworkprogramming.com/winsock2programming/>

# Windows is UCS-2 (Unicode UTF-16)

- ALL Windows specific functions that take a string for interacting with the operating system exist in both a A() and W() version
  - The A() version will take an ANSI string (which technically is Windows 1251), any filename is limited to 256 characters (MAX\_PATH variable)
  - The W() version will take a WIDE string (Unicode string UCS-2 encoded), any filename is limited to 32,767 \_if\_ the string is prepended with "\\?\" to the path
  - If you develop on Windows make a habit to always call the W variant explicit
  - In KERNEL though strings are UNICODE\_STRING, which is \_not\_ zero terminated...
- ```
typedef struct _UNICODE_STRING { USHORT Length;  
    USHORT MaximumLength;  
    PWSTR Buffer;  
} UNICODE_STRING, *PUNICODE_STRING;
```

# Windows is UCS-2 (Unicode UTF-16) #2

- String functions called str\* also have a Unicode variant

**wcscpy()**

**wcscat()**

**wcsicmp()**

If you want the compiler to choose between Unicode or ANSI/ASCII you can use `_tcscpy()` and `T"string"` – but I prefer a developer to choose between the two...

# Deprecated and “unsafe”

- Trying to use strcpy and the other “unsafe” functions in a modern Windows compiler will result in compiler error

<https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/strcpy-wcscpy-mbscpy?view=vs-2019>

- Several “safe” versions have been developed during the last 20 years; Microsoft wants you to use strcpy\_s function

```
errno_t strcpy_s( char *dest, rsize_t dest_size, const char *src );  
errno_t wcscpy_s( wchar_t *dest, rsize_t dest_size, const wchar_t *src );
```

# Snart eksamen

Husk at mengdetrening er alfa-omega når det gjelder programmering!

Nå er det tidspunktet for å begynne å forberede seg til eksamen, har dere jobbet godt i faget blir det ikke vanskelig – men henger du etter i faget så begynn i dag med å komme ajour, du vil få veldig dårlig tid på eksamen hvis du må LÆRE DEG pensum under eksamen...

28. oktober: Network programming (cirka 1 time forelesning, 3 timer øving...)

1. november: EKSAMENSOPPGAVE UTLEVERES

4. november: Safe programming + EXAM prep

11. november: «Repetisjonsforelesning», spørsmål til eksamen

15. november: Innlevering av eksamen

PG3401-H21-Exercises-Lecture10-12\_exam\_preperation.pdf

Applikasjon styrt av en «meny» med brukerinput

Nettverksapplikasjon med klient og server

Multithreadet server (bygger på oppgaven over)

+ Debugger koden fra forelesning 7