# PROGRAMMIEREN II

DHBW Stuttgart Campus Horb INF2017

# AUFGABE VOM LETZTEN MAL

- Nehme die Aufgabe von vorhin und erzeuge mit gen_bin.cpp mehrere solcher Binär-Dateien
- Schreibe ein Programm, dass eine Liste von Dateien als Start-Parameter bekommt
- Jede dieser Dateien soll in einem eigenen Thread entschlüsseln
- Am Ende soll auf der Kommandozeile ausgegeben werden, welche Datei mit welcher Größe und welcher Anzahl Nachrichten entschlüsselt wurden sowie wie viele Sekunden dies benötigt hat

# AGENDA

- Randwissen
- Programmieraufgaben
- Freies Arbeiten an der Projektaufgabe

# WAS SAGT DAS AUS?

```
x = (y < 0) ? 10 : 20;
```

# WAS SAGT DAS AUS?

```
(a == 0 ? a : b) = 1;
```

# LVALUE :-)

```
if (a == 0)
        a = 1;
else
        b = 1;
```

# URLS SIND VOLL OK IN CPP - RICHTIG ZITIEREN UNDSO

```
void foo() {
    http://stackoverflow.com/
    int bar = 4;

    ...
}
```

# WIE FUNKTIONIERT DAS HIER?

```
template<typename From, typename To>
union union_cast {
    From from;
    To   to;

    union_cast(From from)
        :from(from) { }

    To getTo() const { return to; }
};
```

# KANNTET IHR DIESES FEATURE?

```
map<int, string> m;
string& s = m[42]; // no need for map::find()
if (s.empty()) { // assuming we never store empty values in m
  s.assign(...);
}
cout << s;
```

# WAS MACHT DIESER CODE WOHL?

```
// Must allocate our own memory
  Test *ptr = (Test *)malloc(sizeof(Test));

  // Use placement new
  new (ptr) Test;

  // Must call the destructor ourselves
  ptr->~Test();

  // Must release the memory ourselves
  free(ptr);
```

# EXTERN KEYWORD

- Extern erlaubt die Deklaration einer globalen Variable
- Diese muss irgendwo definiert werden
- Kann dann aber überall verwendet werden
- Im File-Scope —> Außerhalb von Funktionen (und structs)
- Niemals static (!)

# EXTERN EXAMPLE HEADER.H

```h
#ifndef HEADER_H
#define HEADER_H

// any source file that includes this will be able to use "global_x"
extern int global_x;

void print_global_x();

#endif
```

# EXTERN EXAMPLE SOURCE 1

```c
#include "header.h"

// it needs to be defined somewhere
int global_x;

int main()
{
    //set global_x here:
    global_x = 5;

    print_global_x();
}
```

# EXTERN EXAMPLE SOURCE 2

```
#include <iostream>
#include "header.h"

void print_global_x()
{
    //print global_x here:
    std::cout << global_x << std::endl;
}
```

# EXTERN ZUSAMMENFASSUNG

- Linker Hilfe für globale Variablen

- Dennoch: Globale Variablen vermeiden

# INLINE KEYWORD

- In ’98 war dies eine Empfehlung (!) an den Compiler eine Funktion beim Kompilieren „auszuwälzen“

- Heutige Compiler ignorieren diese Empfehlung i.d.R. weil sie besser wissen, was Inline sein muss

- Inline heute: Kennzeichnet dass diese Funktion in unterschiedlichen Übersetzungseinheiten unterschiedliche Definitionen haben kann

# WANN INLINE VERWENDEN

- Nur wenn die Funktion im Header definiert werden soll und in mehren Übersetzungseinheiten vorkommt

- Besonders kleine One Liner können im Header definiert und mit inline versehen werden, da es dem Compiler mehr Optimierung erlaubt (aber auch mehr Zeit kostet)

- Niemals hinzufügen, weil man denkt, man ist schlauer als der Compiler :-)

# NOEXCEPT KEYWORD

```
void f() noexcept; // the function f() does not throw
```

```
void (*fp)() noexcept(false); // fp points to a function that may throw
```

```
void g(void pfa() noexcept);  // g takes a pointer to function that doesn't throw
```

# NOEXEPT KEYWORD ERKLÄRUNG

- Jede Funktion in CPP kann einen von zwei Stati haben:
  - a) Does not throw
  - b) May throw
- NOEXCEPT(expression) gibt an, ob eine Funktion Status a) oder b) hat
- Wenn expression = true, dann wirft der Aufruf der Funktion garantiert keine Exception, ansonsten wirft sie eventuell eine

# WENN'S DANN DOCH KNALLT

```
// whether foo is declared noexcept depends on if the expression
// T() will throw any exceptions
template <class T>
  void foo() noexcept(noexcept(T())) {}

void bar() noexcept(true) {}
void baz() noexcept { throw 42; }  // noexcept is the same as noexcept(true)

int main()
{
    foo<int>();  // noexcept(noexcept(int())) => noexcept(true), so this is fine

    bar();  // fine
    baz();  // compiles, but at runtime this calls std::terminate
}
```

# NOEXCEPT ALS OPERATOR

```
noexcept(may_throw()) —> returns false
noexcept(no_throw()) —> returns true
```

# STD::TERMINATE

- Ruft den registrierten std::terminate_handler auf

- Standardmäßig std::abort

# STD::ABORT

- Beendet das Programm mit SIGABRT
- Wichtig für euch: Cleanup Funktionen werden nicht unbedingt aufgerufen
- Also wenn noexcept euch um die Ohren fliegt, habt ihr ein richtiges Problem

# SIGNALE

**Constant** Explanation

**SIGTERM** termination request, sent to the program

**SIGSEGV** invalid memory access (segmentation fault)

**SIGINT** external interrupt, usually initiated by the user

**SIGILL** invalid program image, such as invalid instruction

**SIGABRT** abnormal termination condition, as is e.g. initiated by std::abort()

**SIGFPE** erroneous arithmetic operation such as divide by zero

# SIGNALE BEISPIELE

```cpp
#include <csignal>
#include <iostream>

namespace
{
  volatile std::sig_atomic_t gSignalStatus;
}

void signal_handler(int signal)
{
  gSignalStatus = signal;
}

int main()
{
  // Install a signal handler
  std::signal(SIGINT, signal_handler);

  std::cout << "SignalValue: " << gSignalStatus << '\n';
  std::cout << "Sending signal " << SIGINT << '\n';
  std::raise(SIGINT);
  std::cout << "SignalValue: " << gSignalStatus << '\n';
}
```

# MUCHO UNDEFINED BEHAVIOR

- In einem Signal Handler darf man fast nur noch ein paar letzte Aufräum- oder Loggingaktionen machen
- Das Verhalten ist undefiniert für
  - Allokationen
  - Casts
  - Library calls
  - …
  - Volle Liste: https://en.cppreference.com/w/cpp/utility/program/signal
- Noch dazu: Eher C Style

# KLEINE PROGRAMMIERAUFGABE

- Schreibe ein Programm das (ausnahmsweise) absichtlich SIGSEGV auslöst und im Handler den Programmierer beleidigt (ca. 15 Minuten)

# DECLTYPE

```
int main()
{
    int i = 33;
    decltype(i) j = i * 2;

    std::cout << "i = " << i << ", "
              << "j = " << j << '\n';

    auto f = [](int a, int b) -> int
    {
        return a * b;
    };

    decltype(f) g = f; // the type of a lambda function is unique and unnamed
    i = f(2, 2);
    j = g(3, 3);

    std::cout << "i = " << i << ", "
              << "j = " << j << '\n';
}
```

# ANWENDUNGSGEBIETE DECLTYPE

- Templateprogrammierung