PROGRAMMIEREN II

DHBW Stuttgart Campus Horb INF2017

AUFGABE VOM LETZTEN MAL

- Freiwillige vor! :-)

QUIZRUNDE

- Wenn man C Code in C++ verwenden will, sollte man sehr vorsichtig sein, damit es nicht zu Kompilierfehlern kommt. Richtig oder Falsch?

QUIZRUNDE II

- In C++ müssen alle Felder und Methoden in einer Klasse deklariert werden, da es sich um eine objektorientierte Sprache handelt. Richtig oder Falsch?

QUIZRUNDE III

- Unterschied zwischen
  - C++
  - ++C ?

QUIZRUNDE IV

```
    char x[] ="";
    std::cin >> x; // Eingabe: "Gollum"
```

QUIZRUNDE V

Ja oder nein?

```
std::cin >> std::string x;
```

AGENDA FÜR DIE HEUTIGE VORLESUNG

- Erstellen von Zufallszahlen
- Structs im C Sinne
- Pointer

ZUFALLSZAHLEN

- Computer beherrschen in der Regel nur Pseudozufallszahlen
- Als **Pseudozufall** wird bezeichnet, was zufällig erscheint, in Wirklichkeit jedoch berechenbar ist
- Deterministische Algorithmen erzeugen bei gleichen Seed of die gleiche Zahlenfolge
- Seed ist der Startwert für den Generator, häufig wird hier der aktuelle Timestamp genutzt

RANDOM NUMBERS IN C

```c
int main()
{
    // Initialise random engine with seed
    // seed is the current seconds sind 1970
    srand((unsigned)time(NULL));

    // dice
    int max = 6;
    int min = 1;

    for(int i=0;i<10;++i){
        printf("%i. result: %i\n",i,
        rand() % (max - min +1) + min
        /*Getting a random number*/
        );
    }

    return 0;
}
```

RANDOM NUMBERS C++

```cpp
#include <random>
#include <iostream>

int main()
{
    //Will be used to obtain a seed for the random number engine
    std::random_device rd;
    //Standard mersenne_twister_engine seeded with rd()
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(1, 6);
    // role the dice 10 times
    for (int n = 0; n < 10; ++n)
        //Use dis to transform the random unsigned int generated
        //by gen into an int in [1, 6]
        std::cout << dis(gen) << ' ';
    std::cout << '\n';
}
```

ERSTE PROGRAMMIER-AUFGABE

- Die Aufgabe in CppLotto.cpp aus dem bekannten GitHub

STRUCTS IM C SINNE

- Ein struct ist eine zusammengesetzte Datentypdeklaration, die eine physikalisch gruppierte Liste von Variablen definiert

STRUCT EXAMPLE 1

```
struct tag_name {
    type member1;
    type member2;
    /* declare as many members as desired,
but the entire structure size must be known to the compiler. */
};
```

STRUCT EXAMPLE 2

```
typedef struct tag_name {
    type member1;
    type member2;
} struct_alias;
```

STRUCT EXAMPLE 3

```
struct account {
    int account_number;
    char *first_name;
    char *last_name;
    float balance;
};

struct account s;
```

STRUCT EXAMPLE 4

```
typedef struct account {
    int account_number;
    char *first_name;
    char *last_name;
    float balance;
} account_t;

account_t s;
```

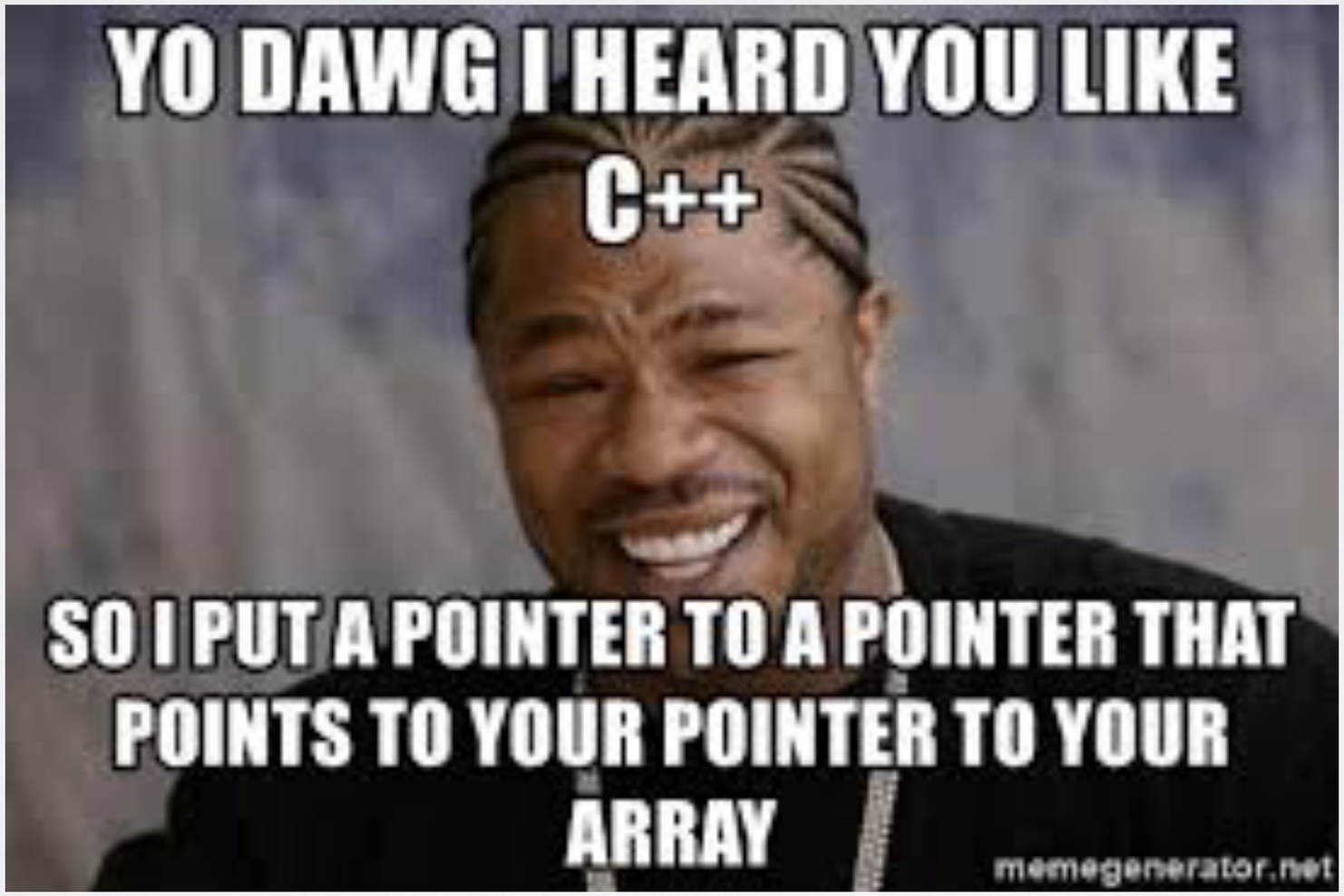
EIN PAAR FAKTEN ZU STRUCTS

- Structs sind aus anderen Datentypen zusammengefasste Datentypen

- Die Größe von Structs muss zur Compilezeit feststehen

- Structs können mit typedef als eigener Typ definiert werden

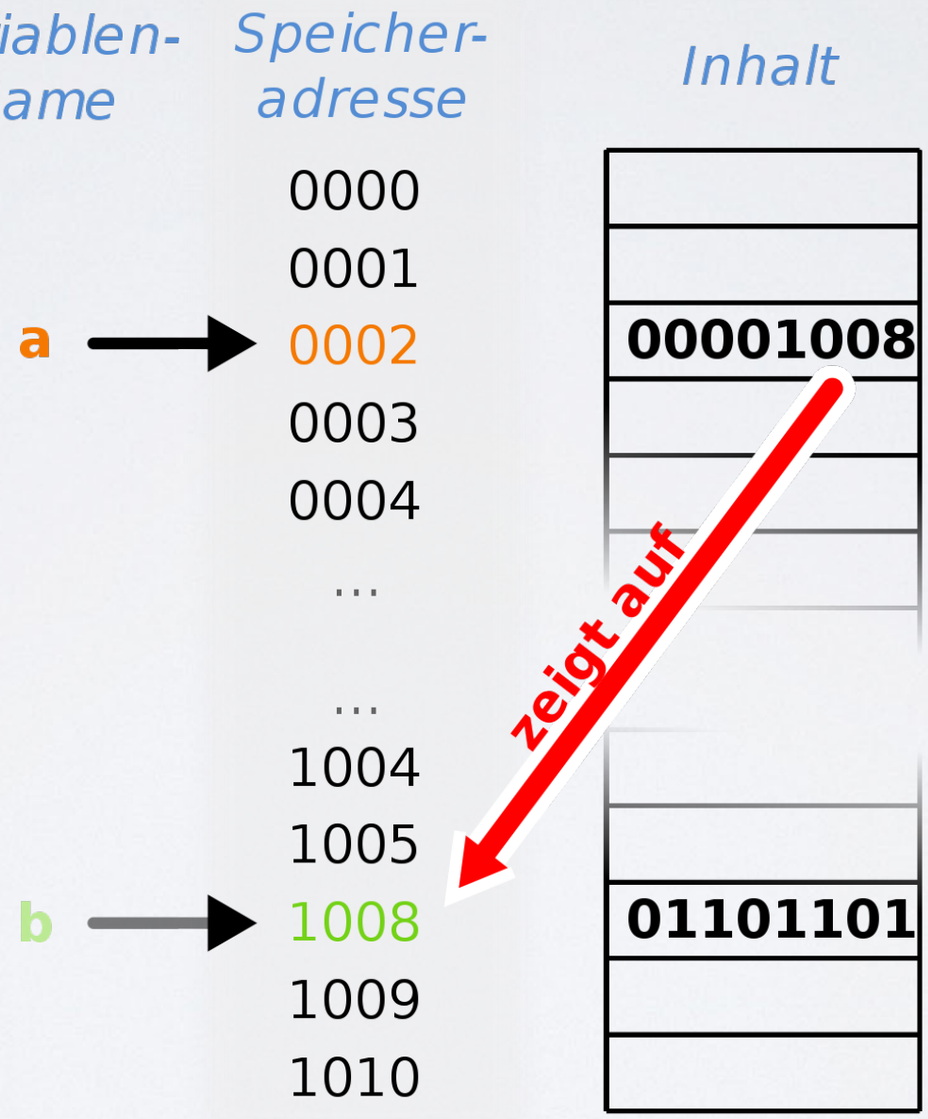
POINTER


YO DAWG I HEARD YOU LIKE
C++
SO I PUT A POINTER TO A POINTER THAT POINTS TO YOUR POINTER TO YOUR ARRAY
memegenerator.net

ALLGEMEINES ZU POINTERN

- Pointer (Zeiger) zeigen auf Speicherstellen
- In C++ kann man Pointer Typisieren, dadurch weiß die Anwendung wie groß der Speicher ist
- Im Gegensatz zur Referenz müssen Zeiger explizit dereferenziert werden
- Besonderer Pointer ist der nullptr oder NULL

ZEIGER SCHAUBILD



ZEIGER BEISPIEL I

```c
#include <stdio.h>
int main()
{
    long x = 9532523;
    long *x_ptr = &x;
    printf("Address: \n\t %p \n contains value:\n\t %d\n original val is:
\n\t%d\n",
     x_ptr, *x_ptr, x);
}
```

```
Torstens-MacBook-Pro-2:day2 torstenhopf$ ./pointers
Address:
        0x7ffee05deb48
 contains value:
        9532523
 original val is:
        9532523
```

POINTERS AND ARRAYS

```
// in c and c++ arrays and pointers are somehow familiar with
    // each other. The pointer links to the first element of
the
    // array
    char buf[50];
    char* string = buf;

    // you can even use array operators on the pointer
    // this uses internally the sizeof(char) to do this
     string[0] = 'a';
     string[1] = 'b';
```

INDIREKTER ZUGRIFF

```
    char letter = 'b';
    char *letterPointer = &letter;
    *letterPointer = '3';
    printf("Omg letter does not contain a letter anymore! but %c\n"
          ,letter);
```

ZEIGERARTITHMETIK

```
    char buf[50];
     char *string = buf;
     // will print all chars of the buf
     for (char *p = buf; *p; ++p)
     {
          printf("%c",*p);
     }
```

ANONYME ZEIGER

```
    void *iCanPointToEverything = 0;
    char *iAmAString = "Vingardium Liviosa";
    // no problem
    iCanPointToEverything = iAmAString;
    //iAmAString = iCanPointToEverything; would need an explicit cast
```

•

BESONDERE ZEIGER

```
    char *c_ptr = NULL;
    char *dont_dereference = nullptr;
    char *i_am_also_null = 0;
```

NULLPOINTER

- Wobei NULL meist void *ptr = 0;
- nullptr in C++11 eingeführtes literal —> immer mit —std=c++11 kompilieren
- 0 erklärt sich selbst
- Wieso ist eine if(ptr) Prüfung nicht die ganze Wahrheit?

REGELN DES POINTER CLUBS

- Schrödingers Pointer
- Traue keinem Pointer den du nicht selbst invalidiert hast
- Setze Pointer wenn du sie noch nicht oder nicht mehr benutzt auf nullptr(!)

STACK UND HEAP

STACK

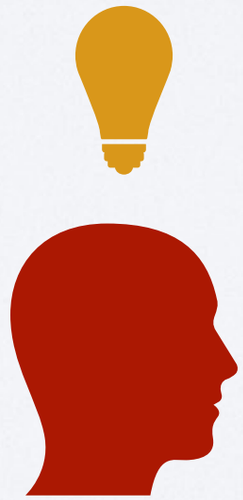
- Begrenzte Größe
- LIFO Datenstruktur (die zuletzt angelegten Daten werden als erstes wieder freigegeben, deshalb auch "Stapel")
- Wächst und schrumpft mit dem Programmverlauf
- Wird verwendet für lokale Variablen und Funktionsparameter
- Kein explizites Freigeben des Speichers nötig
- Das Ablegen und Entfernen von Elementen ist sehr effizient

HEAP

- Der Heap kann innerhalb der Prozessgrenze beliebig groß werden
- Anlegen und freigeben von Objekten ist vergleichsweise langsam
- Auf dem Heap angelegte Objekte können global verfügbar gemacht werden
- In Programmiersprachen ohne Garbage Collector muss der Speicher manuell freigegeben werden, wenn er nicht mehr benötigt wird

WANN LEGE ICH EIN OBJEKT / EINE DATENSTRUKTUR AUF DEM HEAP UND WANN AUF DEM STACK AN?

WANN LEGE ICH EIN OBJEKT / EINE DATENSTRUKTUR AUF DEM HEAP UND WANN AUF DEM STACK AN?



STRUCTS AUF DEM HEAP ERZEUGEN

```
mystruct* s = new mystruct;


void* s2= malloc(sizeof(mystruct));
```

LÖSCHEN VON STRUCTS AUF HEAP

- delete

- delete[ ]

AUFGABE UND HAUSAUFGABE FÜR DAS NÄCHSTE MAL

- Baue ein Telefonbuch
- Verwalte die Telefonnummern mit Vor- und Nachnahme in einem (c)Struct
- Lasse den std::vector Zeiger auf den Eintrag halten
- Erlaube die Suche nach Nachnamen und gib alle Einträge mit passendem Nachnamen aus