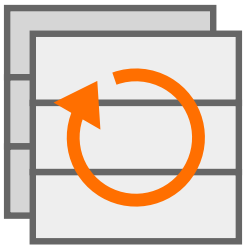


Operating System Group

TUHH

Technische Universität Hamburg

BSB – Übung 0: Speicher und C++

Yannick Loeck

2022-04-19

Inhalt

TUHH
Technische Universität Hamburg

BSB - HÜ0

Ihr werdet C/C++ schreiben.

Worum geht es in der Übung heute:

- Disassembly
- Eigenschaften von C++
- Konzepte zu Speicher

Disassembly

Disassembly

TUHH
Technische Universität Hamburg

- Lesbarere Darstellung von Maschinencode
  - Compiler/Assembler/Linker machen aus Hochsprache Maschinenbefehle
  - Dazu korrespondieren Assembly-Instruktionen wie mov, add
  - Disassembler übersetzt zurück

- Syntaxunterschiede
  - AT&T (GNU) wie bei cp: Befehl <Quelle> <Ziel>
    ```
    movl $5, %eax
    ```
  - Intel: Befehl <Ziel> <Quelle>
    ```
    mov eax, 5
    ```

Beispielprogramm

TUHH
Technische Universität Hamburg

```
int i, j;

int main() {
    i = 5;
    j = 3;
    i -= j;
    return 0;
}
```

- `g++ -m32 -fno-PIE simple.c -o simple`

- `nm -numeric-sort -S simple`

```
0804915d 00000030 T main
...
0804c01c 00000004 B i
0804c020 00000004 B j
```

Beispielprogramm

Technische Universität Hamburg

```
int i, j;

int main() {
    i = 5;
    j = 3;
    i -= j;
    return 0;
}
```

- `g++ -m32 -fno-PIE simple.c -o simple`

- `nm -numeric-sort -S simple`

```
0804915d 00000030 T main
...
0804c01c 00000004 B i
0804c020 00000004 B j
```

Disassembly

TUHH
Technische Universität Hamburg

- `objdump -d simple`

```
0804915d <main>:
  804915d:   55                                    push   %ebp
  804915e:   89 e5                                 mov    %esp,%ebp
  8049160:   c7 05 1c c0 04 08 05                  movl   $0x5,0x804c01c
  804916a:   c7 05 20 c0 04 08 03                  movl   $0x3,0x804c020
  8049174:   a1 1c c0 04 08                        mov    0x804c01c,%eax
  8049179:   8b 15 20 c0 04 08                     mov    0x804c020,%edx
  804917f:   29 d0                                 sub    %edx,%eax
  8049181:   a3 1c c0 04 08                        mov    %eax,0x804c01c
  8049186:   b8 00 00 00 00                        mov    $0x0,%eax
  804918b:   5d                                    pop    %ebp
  804918c:   c3                                    ret
```

Disassembly

```
0804915d <main>:
  804915d:   55                                  push   %ebp
  804915e:   89 e5                               mov    %esp,%ebp
  ...
  804918b:   5d                                  pop    %ebp
  804918c:   c3                                  ret
```

- Vorherigen Base Pointer sichern: Return-Adresse
- Base Pointer auf Stack Pointer setzen: Variablen und Parameter

Register

`ESP` — Stack Pointer
`EBP` — Base Pointer ('Pointer to data on the stack')
`EAX` — Operands and results data

R: 64 bit, E: 32 bit

Disassembly

```
8049160:   c7 05 1c c0 04 08 05    movl   $0x5,0x804c01c
804916a:   c7 05 20 c0 04 08 03    movl   $0x3,0x804c020
8049174:   a1 1c c0 04 08          mov    0x804c01c,%eax
8049179:   8b 15 20 c0 04 08       mov    0x804c020,%edx
804917f:   29 d0                   sub    %edx,%eax
8049181:   a3 1c c0 04 08          mov    %eax,0x804c01c
8049186:   b8 00 00 00 00          mov    $0x0,%eax
```

- Code:

```
i = 5;
j = 3;
i -= j;
return 0;
```

Register

ESP — Stack Pointer
EBP — Base Pointer ('Pointer to data on the stack')
EAX — Operands and results data

R: 64 bit, E: 32 bit

Disassembly - Stack

TUHH
Technische Universität Hamburg

```
int do_something(int val) {
  int other = 3;
  return val - other;
}
```

- Rücksprungadresse: Stack-Pointer bzw. ebp +0x4
- Funktionsparameter: +0x8
- 1. Lokale Variable: -0x4 (2. bei -0x8)
- Stack wächst nach unten

```
0804915d <_Z12do_somethingi>:
 804915d:   55                          push   %ebp
 804915e:   89 e5                       mov    %esp,%ebp
 8049160:   83 ec 10                    sub    $0x10,%esp
 8049163:   c7 45 fc 03 00 00 00        movl   $0x3,-0x4(%ebp)
 804916a:   8b 45 08                    mov    0x8(%ebp),%eax
 804916d:   2b 45 fc                    sub    -0x4(%ebp),%eax
 8049170:   c9                          leave
 8049171:   c3                          ret
```

Compileroptimierung

Technische Universität Hamburg

```
int main() {
    int j;
    for (int i = 0; i < 100; i++) {
        j++;
    }
    return j;
}
```

- Kompiliert mit -O3

```
08049040 <main>:
 8049040:    b8 64 00 00 00      mov   $0x64,%eax
 8049045:    c3                  ret
```

C++

Structs und Classes

TUHH
Technische Universität Hamburg

C with Classes

- Verbund von Daten und Methoden
- In C++ quasi gleich
  - Unterschied: struct member default `public`, class default `private`
- Innerhalb: `this` zeigt auf eigene Instanz
- `virtual` (Methode): Kann von Kindern überschrieben werden
- `static` (Member/Methode): Zwischen Instanzen geteilt

Inheritance

TUHH
Technische Universität Hamburg

```
class A {
public:
   int x;
};

class B : A {
   int y;
};


int main(void) {
   A a;
   a.x = 4;
   B b;
   b.x = 3;
}
```

Inheritance

TUHH
Technische Universität Hamburg

```
class A {
public:
  int x;
};

class B : A {
  int y;
};


int main(void) {
  A a;
  a.x = 4;
  B b;

  b.x = 3;
}
```

- error: 'int A::x' is inaccessible within this context

Inheritance

TUHH
Technische Universität Hamburg

```
class A {
public:
   int x;
};

class B : public A {
   int y;
};


int main(void) {
   A a;
   a.x = 4;
   B b;
   b.x = 3;
}
```

Initializer Lists


```
class Countdown {
    int counter;
  public:
    void down() { counter--; }
    Countdown(int initial) { counter = initial;}
};
```

```
class Foo {
    Countdown c;
  public:
    Foo(int count) {}
};
```

- error: Constructor for 'Foo' must explicitly initialize the member 'c' which does not have a default constructor

Initializer Lists

TUHH
Technische Universität Hamburg

```
class Countdown {
    int counter;
  public:
    void down() { counter--; }
    Countdown(int initial) { counter = initial;}
};
```

```
class Foo {
    Countdown c;
    const int initial;
  public:
    Foo(int count) : c(count), initial(count) {}
};
```

- Auch für Initialisierung von const membern

Casts in C++

TUHH
Technische Universität Hamburg

C-Style Casts

- gehen auch in C++, aber lieber nicht benutzen (gerade bei Klassen)

```
double x = 2.0;
int y = (int)x;
```

```
class Foo {
    int a;
    double b;
};

class Bar {
    std::string s;
};

Foo f;
Bar* b = (Bar*)&f; // kein warning/error!
```

Casts in C++

TUHH
Technische Universität Hamburg

`reinterpret_cast`

- Cast zwischen beliebigen Pointertypen, keine Garantien

```
class Foo {
    int a;
    double b;
};

class Bar {
    std::string s;
};

Foo f;
Bar* b = reinterpret_cast<Bar*>(&f); // kein warning/error!
```

Casts in C++

TUHH
Technische Universität Hamburg

`static_cast`

- Umwandlung von Basistypen (z.B. double auf int)
- Umwandlung von Pointern
- Keine vollen Type Safety Checks wie bei `dynamic_cast`

```
double x = 2.0;
int y = static_cast<int>(x);

Foo f;
Bar* b = static_cast<Bar*>(&f);
```

- error: invalid ‘static_cast’ from type ‘Foo*’ to type ‘Bar*’

Casts in C++

TUHH
Technische Universität Hamburg

`dynamic_cast`

- Umwandlung von Pointern und Referenzen auf Objekte
- Ergebnis der Typumwandlung muss valides vollständiges Objekt sein

```
Bar* b = dynamic_cast<Bar*>(&f);
```

- error: cannot ‘dynamic_cast’ ‘& f’ (of type ‘class Foo*’) to type ‘class Bar*’ (source type is not polymorphic)

Casts in C++

TUHH
Technische Universität Hamburg

`const_cast`

- hinzufügen oder entfernen von `const/volatile`

```
void output(int* i) {
    std::cout << *i << std::endl;
}

const int x = 5;
output(&x);

output(const_cast<int*>(&x));
```

Funktionsüberladung

TUHH
Technische Universität Hamburg

- Selber Funktionsname, verschiedene Argumente

```
void do_magic(int i) {
    i *= 2;
}

void do_magic(double d) {
    d -= 42.0;
}
```

Funktionsüberladung

TUHH
Technische Universität Hamburg

- Selber Funktionsname, verschiedene Argumente
- Auch möglich: Default-Argumente

```
void do_magic(int i, bool alt = false) {
    if (alt)
        i++;
    else
        i *= 2;
}

int do_magic(int i) {
  // Könnte man definieren, dann aber do_magic(int)
  // nicht mehr aufrufen:
  // error: call ... is ambiguous
}

void do_magic(double d) {
    d -= 42.0;
}
```

Funktionsüberladung

- Selber Funktionsname, verschiedene Argumente
- Auch möglich: Default-Argumente
- Operatorüberladung ist auch Funktionsüberladung

```
int x, y;
x += y;


struct Vec_2D {
    int x, y;
    void operator+=(Vec_2D& other) {
        this->x += other.x;
        this->y += other.y;
    }
};

Vec_2D a, b;
a += b;
```

Speicher

Arten von Speicherobjekten

TUHH
Technische Universität Hamburg

1. Globale Objekte: globale Variable
   - in DATA oder BSS segment der object file
2. Lokale Objekte: (Funktions-)lokale Variable
   - auf dem Stack
   - Scopes / begrenzte Gültigkeit
   - **Stackgröße in StuBS: 4096 Bytes**

Arten von Speicherobjekten

TUHH
Technische Universität Hamburg

1. Globale Objekte: globale Variable
   - in DATA oder BSS segment der object file
2. Lokale Objekte: (Funktions-)lokale Variable
   - auf dem Stack
   - Scopes / begrenzte Gültigkeit
   - **Stackgröße in StuBS: 4096 Bytes**
3. Objekte im Heap
   - malloc (C), new (C++)
   - **von Linux/libc geliefert, ⇒ gibt es in StuBS nicht**

Arten von Speicherobjekten

TUHH
Technische Universität Hamburg

1. Globale Objekte: globale Variable
   - in DATA oder BSS segment der object file
2. Lokale Objekte: (Funktions-)lokale Variable
   - auf dem Stack
   - Scopes / begrenzte Gültigkeit
   - **Stackgröße in StuBS: 4096 Bytes**
3. Objekte im Heap
   - malloc (C), new (C++)
   - **von Linux/libc geliefert, $\Rightarrow$ gibt es in StuBS nicht**

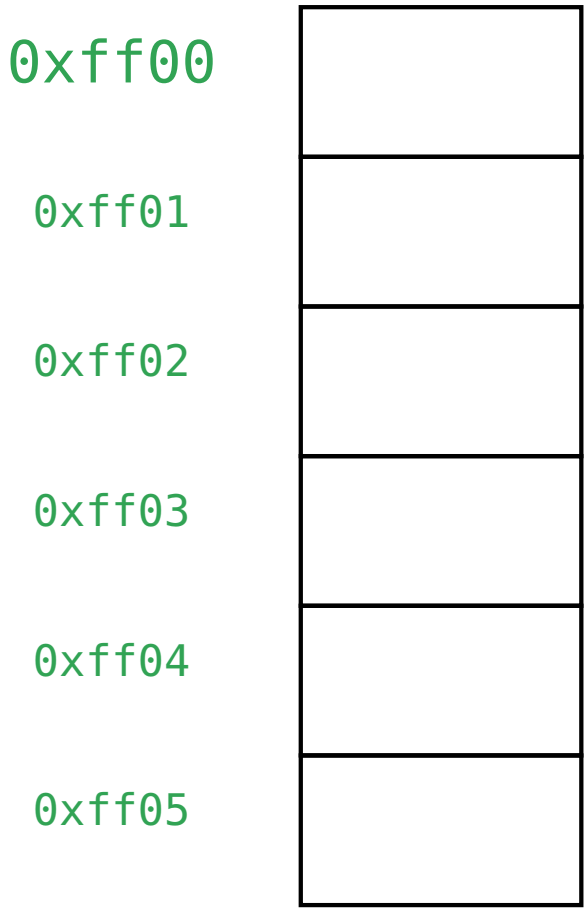
Pointer vs Reference

- Objekt/Variable liegt an Speicherplatz
- Pointer: Zeiger auf Speicher, kann `NULL` (nicht valide) sein
  - Dereferenzieren: Wert der an Speicherstelle liegt
  - `NULL`-Pointer dereferenzieren: **Exception?**
- Reference(C++): Alias für Objekt, immer gültig (nie `NULL`)
  - Muss nicht dereferenziert werden

```
int foo = 23;
int& bar = foo;
std::cout << bar << std::endl; // 23
bar = 42;
std::cout << foo << std::endl; // 42
```

- Warum überhaupt Referenzen? Kein Kopieren

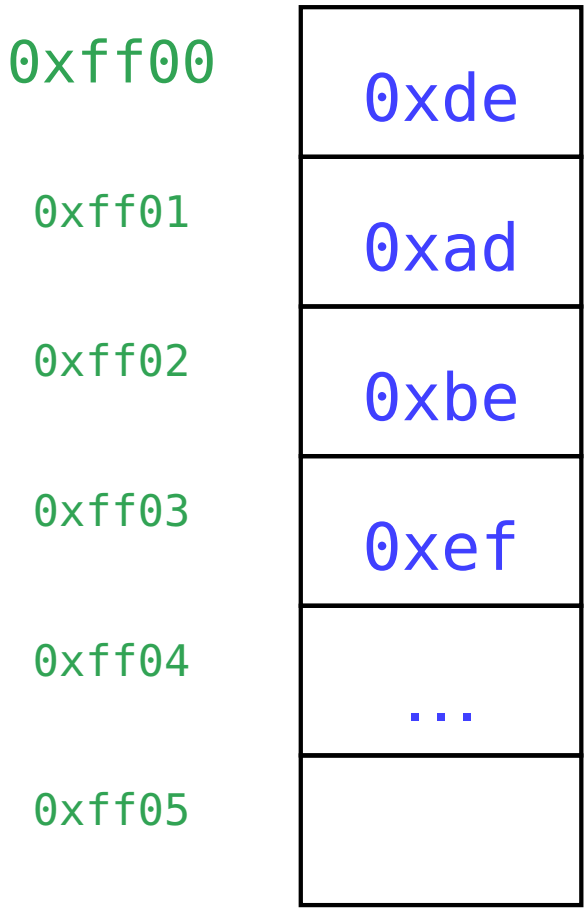
Speicher und Pointer



- Pointer: Adresse und Datentyp
- Ausnahme: `void*`

```
void* addr = 0xff00;
```

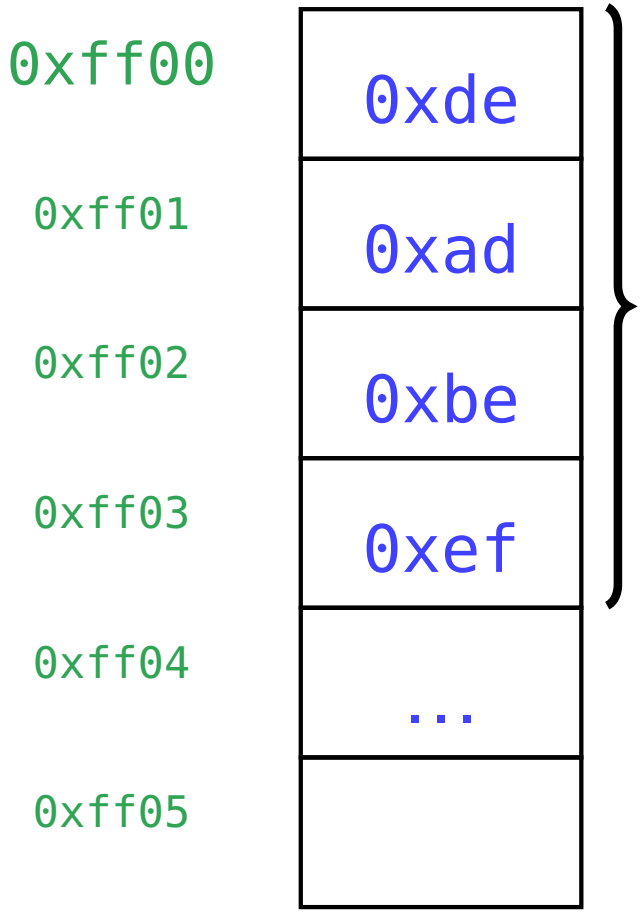
Speicher und Pointer



- Pointer-Typ $\Leftrightarrow$ Länge im Speicher
- `int`: 4 Byte
- *Dereferenzieren* mit `*` vor Pointer

```
int* iptr = 0xff00;
*iptr =
```

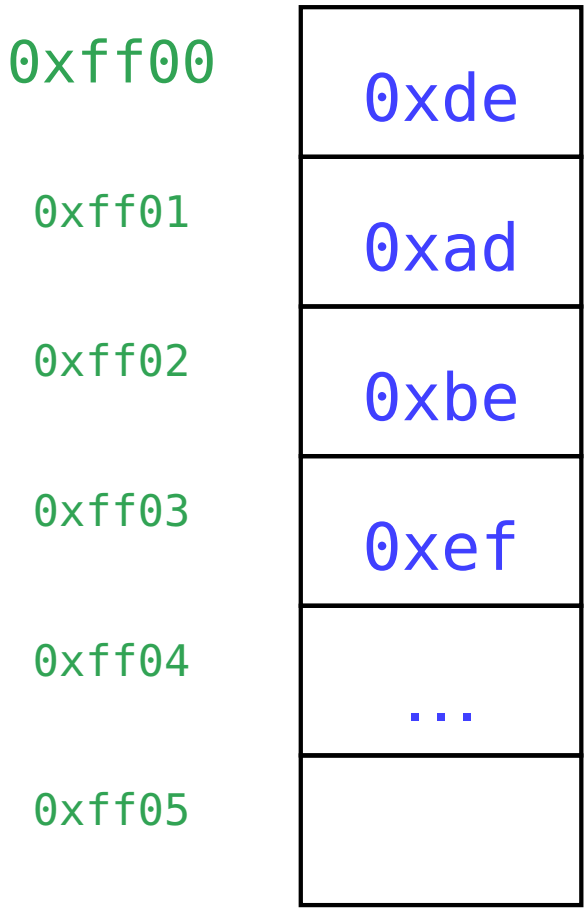
Speicher und Pointer



- Pointer-Typ $\Leftrightarrow$ Länge im Speicher
- `int`: 4 Byte
- *Dereferenzieren* mit `*` vor Pointer

```
int* iptr = 0xff00;
*iptr = 0xdeadbeef;
```

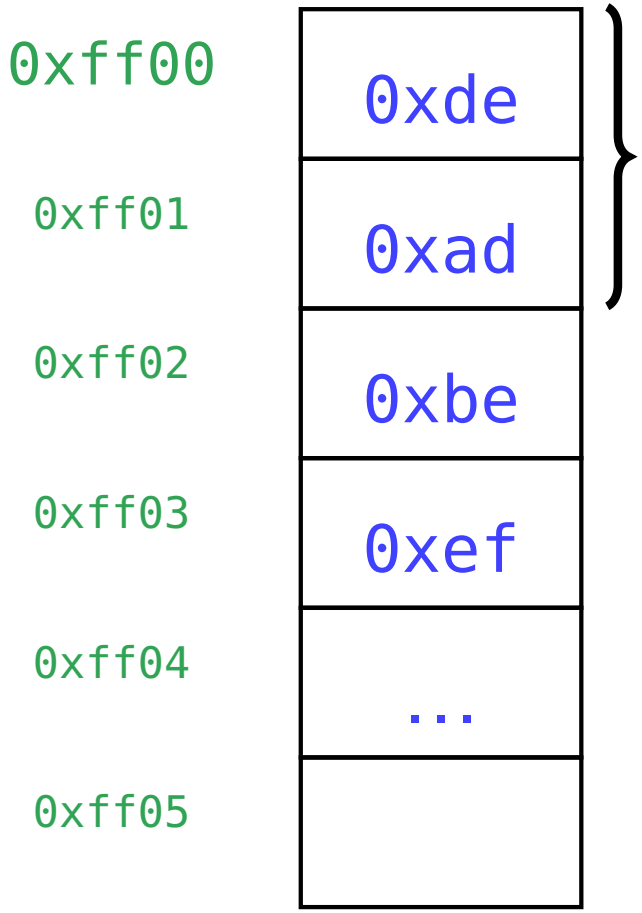
Speicher und Pointer



- Pointer-Typ $\Leftrightarrow$ Länge im Speicher
- `short`: 2 Byte
- *Dereferenzieren* mit `*` vor Pointer

```
short* sptr = 0xff00;
*sptr =
```

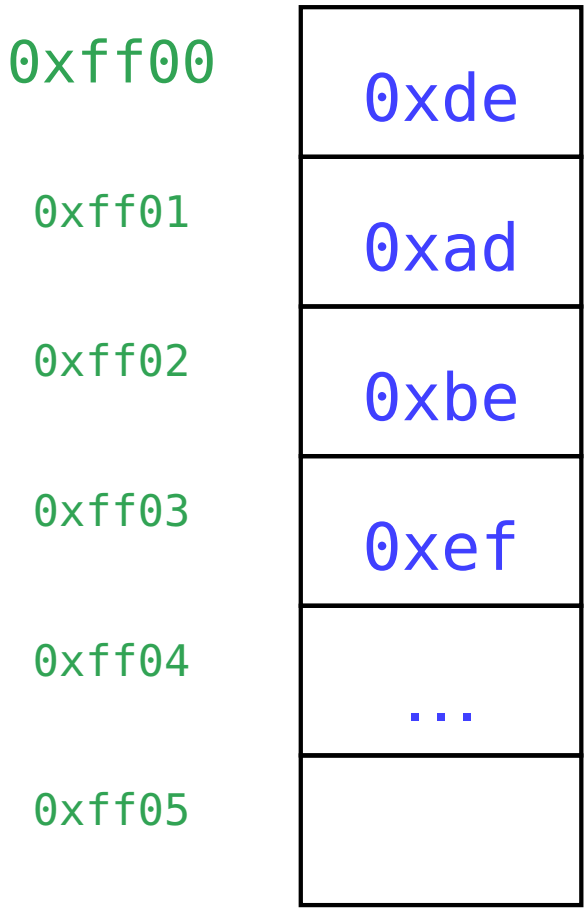
Speicher und Pointer



- Pointer-Typ $\Leftrightarrow$ Länge im Speicher
- `short`: 2 Byte
- *Dereferenzieren* mit `*` vor Pointer

```
short* sptr = 0xff00;
*sptr = 0xdead;
```

Speicher und Pointer

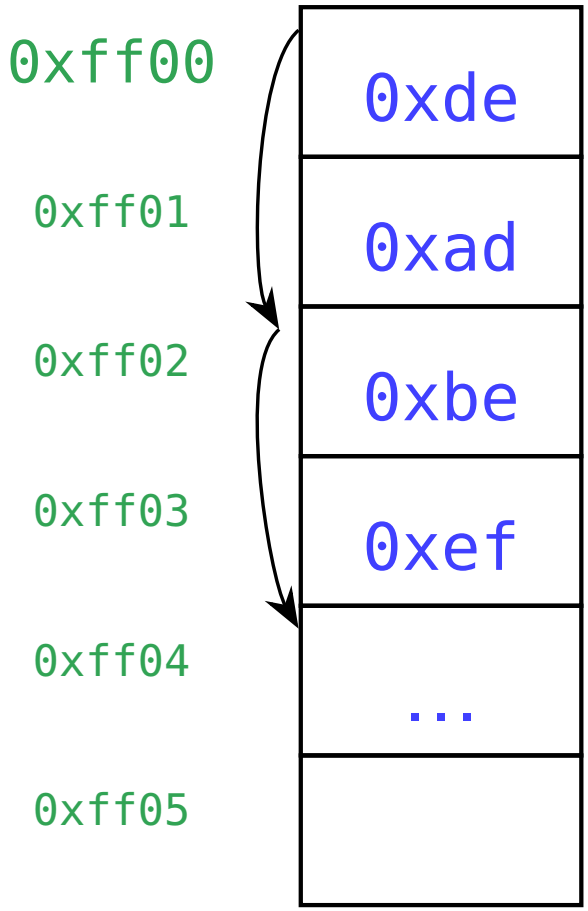


- Rechnen mit Pointer-Arithmetik

```
void* ptr = 0xff00;
ptr + 2 = 0xff02;

short* sptr = 0xff00;
sptr + 2 =
```

Speicher und Pointer

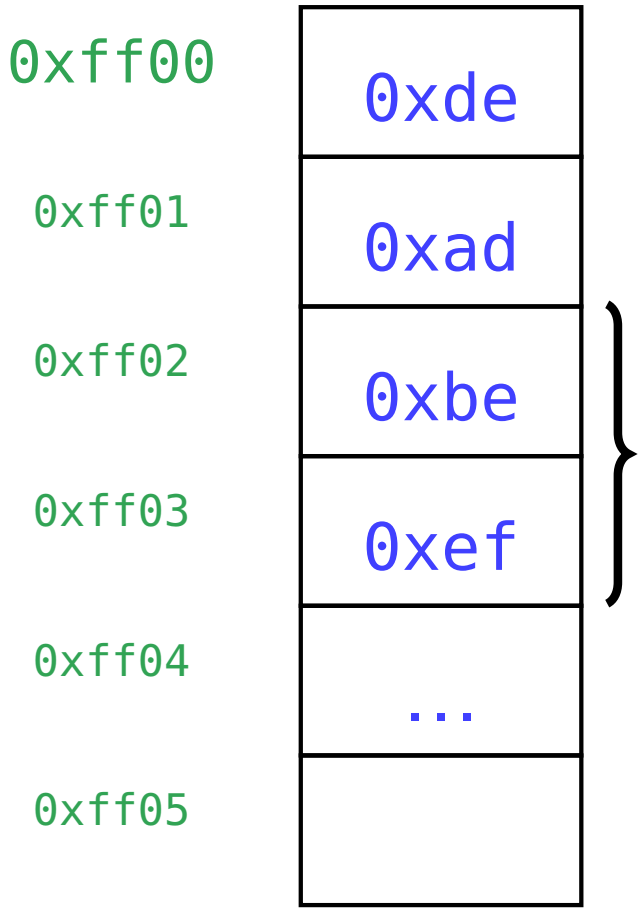


- Rechnen mit Pointer-Arithmetik
- **Achtung!**

```
void* ptr = 0xff00;
ptr + 2 = 0xff02;

short* sptr = 0xff00;
sptr + 2 * sizeof(short) = 0xff04;
```

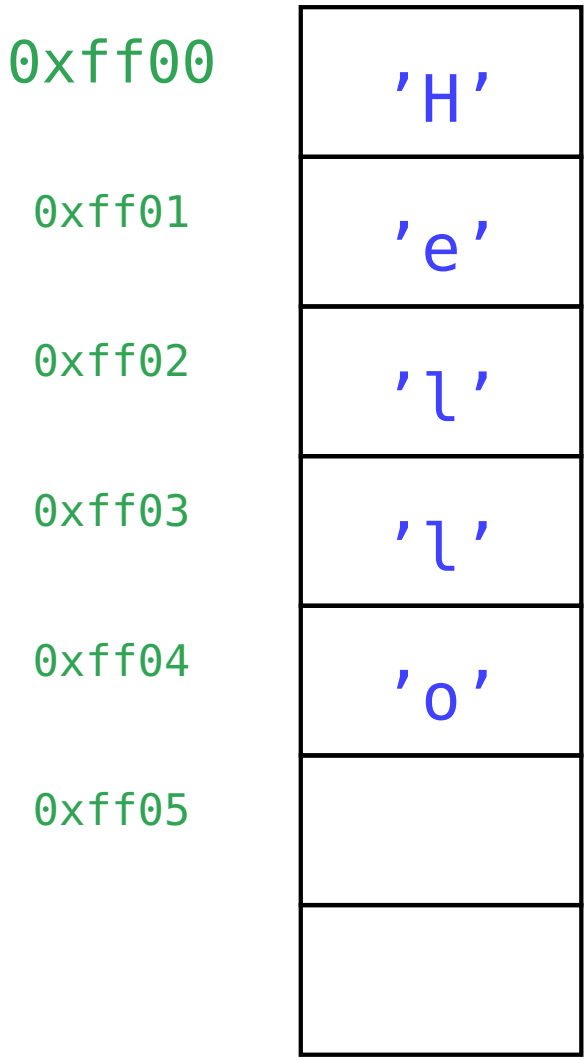
Speicher und Pointer



- Rechnen mit Pointer-Arithmetik

```
short* sptr = 0xff00;
*(sptr + 1) = 0xbeef;
```

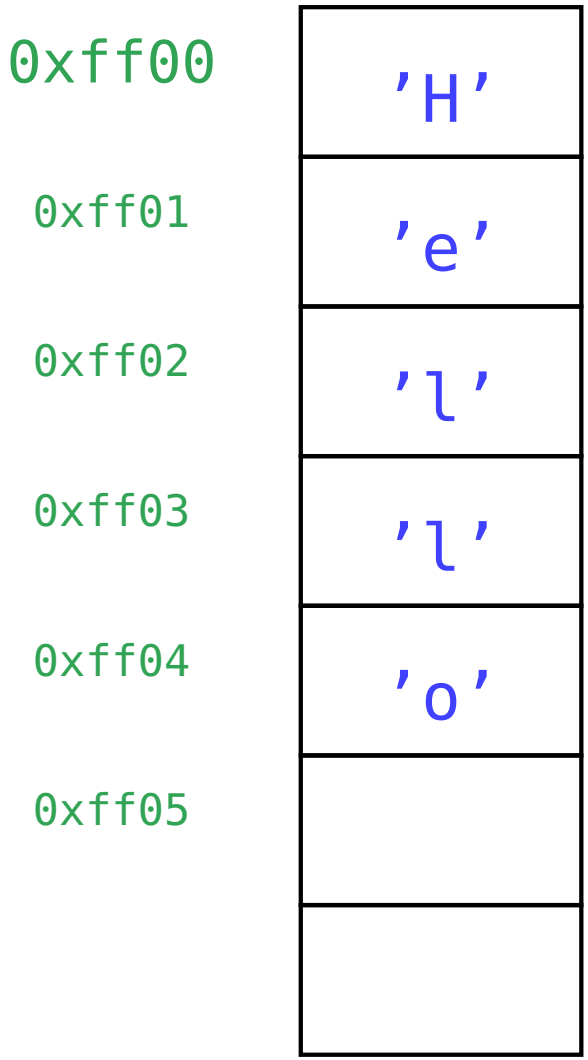
Speicher und Pointer



- C Strings

```
char* str = 0xff00;
*str = 'H';
str++;
*str = 'e';
```

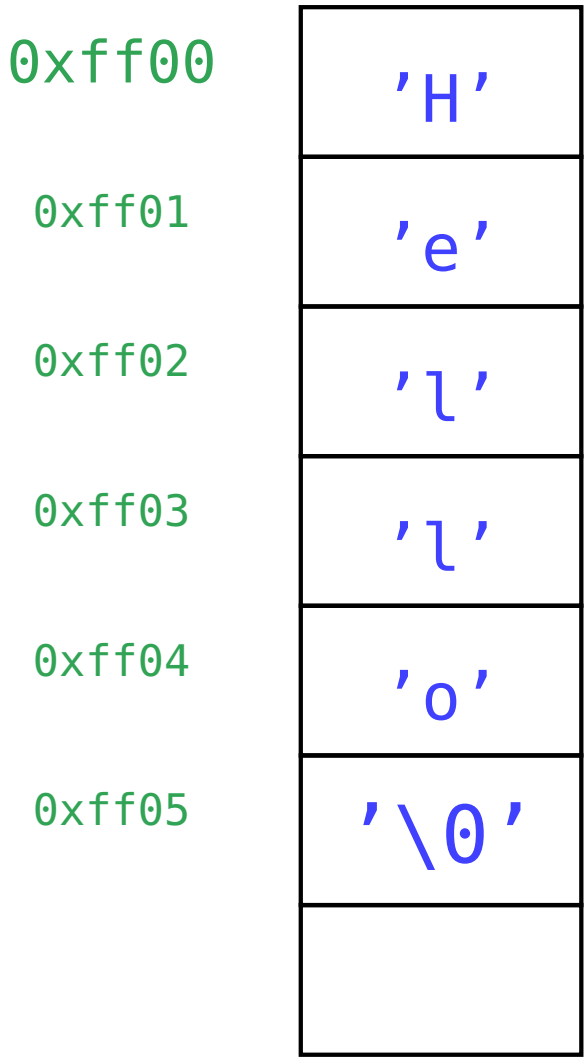
Speicher und Pointer



- Wollen über den ganzen String iterieren
- *Keine Länge gegeben*

```
char* str = 0xff00;
while (                ){
    print(*str);
    str++;
}
```

Speicher und Pointer



- Wollen über den ganzen String iterieren
- *Keine Länge gegeben*
- Nullzeichen/NULL-termination
- **Achtung: Stringlänge**

```
char* str = 0xff00;
while (*str != '\0'){
    print(*str);
    str++;
}
```

Größe von Structs

TUHH
Technische Universität Hamburg

```
struct Something {
  int value;
  char c;
  short foo;
};

std::cout << sizeof(Something) << std::endl;
```

- Wie groß ist das `struct` in Bytes?

Größe von Structs

TUHH
Technische Universität Hamburg

```
struct Something {
  int value;
  char c;
  short foo;
};

std::cout << sizeof(Something) << std::endl; // 8
```

- 8 Byte!
- Compiler fügt für memory alignment Padding Byte(s) hinzu

Größe von Structs


```
struct Something {
  int value;
  char c;
  short foo;
};
static_assert(sizeof(Something) == 7, "Wrong size");
```

- Beim Kompilieren:

```
error: static assertion failed: Wrong size
```

Größe von Structs

TUHH
Technische Universität Hamburg

```
struct Something {
  int value;
  char c;
  short foo;
} __attribute__((packed));
static_assert(sizeof(Something) == 7, "Wrong size");
```

- `__attribute__((packed))` verhindert hinzufügen von padding
- `static_assert` sichert Bedingung zur Compile-Zeit

Sonstiges

Logische vs Bitoperationen - Beispiele

TUHH
Technische Universität Hamburg

$3 \;\&\; 1 =$

$3 \;\&\&\; 1 = 1$

$4 \;\&\; 1 =$

$4 \;\&\&\; 1 = 1$

Logische vs Bitoperationen - Beispiele

```
3 & 1 = 1  = 0 b 0 1 1
             & & &
           0 b 0 0 1
```

```
3 && 1 = 1
```

```
4 & 1 = 0  = 0 b 1 0 0
             & & &
           0 b 0 0 1
```

```
4 && 1 = 1
```

```
int mask_do_a = 1; // 0b001
int mask_do_b = 2; // 0b010
int mask_do_c = 3; // 0b011
int mask_do_d = 4; // 0b100

if (value & mask_do_a) { do_a() } // ok
if (value & mask_do_c) { do_c() } // Vorsicht!
                                  // Nicht eindeutig
```

Logische vs Bitoperationen - Beispiele


```
3 & 1 = 1  = 0 b 0 1 1
             & & &
           0 b 0 0 1
```

```
3 && 1 = 1
```

```
4 & 1 = 0  = 0 b 1 0 0
             & & &
           0 b 0 0 1
```

```
4 && 1 = 1
```

```
int mask_do_a = 1; // 0b001
int mask_do_b = 2; // 0b010
int mask_do_c = 3; // 0b011
int mask_do_d = 4; // 0b100

if (value & mask_do_a) { do_a() } // ok
if (value & mask_do_c) { do_c() } // Vorsicht!
                                  // Nicht eindeutig
```

Volatile

```
int i = 50;
while (i > 3) {
   ...
}
```

- Compiler: "Wert ändert sich nicht", ersetzt durch `while (true)`

```
int i = 5;
i += i * 3;
i++;
i /= 2;
```

- Register sind schnell, RAM vergleichsweise langsam
- Optimierung durch Compiler: Möglichst viel in Registern arbeiten
- Wert ist dann zwischendurch in Speicher falsch

`volatile` verhindert diese Optimierungen

Sonstiges

TUHH
Technische Universität Hamburg

- Integer-Division

```
printf("%d, %d\n", 5/2, 3/4); // 2, 0
```

- Ausgabe von ASCII-Zeichen

```
printf("%c, %c, %d\n", 65, 'A', 'A'); // A, A, 65
```

Bonus: gdb

TUHH
Technische Universität Hamburg

- Programm starten: **r**un
- Call Stack bis hierhin: **b**ack**t**race
- Call Frame wechseln: **f**rame [nummer]
- Breakpoint: **b**reak, info [b]
- Weitermachen (Stoppt bei Breakpoint) **c**ontinue [anzahl]
- Arten von Breakpoints: Funktion, Zeilennummer, Adresse , ...
- Conditional break: b if <condition>
- Watchpoint: Wenn Ausdruck Wert verändert
- Ausgabe: **p**rint, p/x, p/t
- Nächster Ausführungsschritt: **s**tep, si
- Bei jedem Stopp Befehl ausführen: command [breakpoint-id]
- Immer aktuelle Instruktion ausgeben: display /i *$pc

Bonus: Git

TUHH
Technische Universität Hamburg

Manpages

`man git-<command>`, also z.B. `man git-commit`

- remote: Repository (zum pushen oder pullen)
- clone: Lokale Kopie von Repository erstellen
- fetch: Nach updates im remote checken
- pull: Updates aus dem remote laden
- add: Lokale changes stagen, -i für interaktiv
- restore: Datei aus stage entfernen, Changes einer Datei zurücksetzen
- commit: Staged changes mit Titel/Beschreibung zu commit bündeln
- push: Lokale commits ins remote bringen
- .gitignore: Datei für von git zu ignorierende Dateinamen(smuster)