Informatik I – Übung 10 (Spoilerfree)

Pascal Schärli

pascscha@student.ethz.ch

10.05.2019

ETH zürich

Was gibts heute?

- Best-Of Vorlesung
  - Klassen
  - Headerfiles
  - Pointers
  - New / Delete
- Vorbesprechung
  - Quicksort

Best of Vorlesung

Klassen

- Klassen sind wie Structs, nur dass ihre Members standardmässig *private* sind.
- Sie können sowohl Variablen als auch Funktionen beinhalten.
- Klassennamen sind aus Konvention gross geschrieben.

Klassen

```cpp
#include <iostream>

class Wallet{
    unsigned int money;
    std::string owner;

public:
    Wallet(std::string o, unsigned int m){
        money = m;
        owner = o;
    }

    unsigned int money_left(){
        return money;
    }

    bool pay(unsigned int amount){
        if(money >= amount){
            money -= amount;
            return true;
        }
        else{
            return false;
        }
    }

    std::string get_owner(){
        return owner;
    }
};
```

Findet die 10 Fehler

```cpp
main(){
    Wallet w(100, "Pascal");

    std::cout << w.get_money() << " ";
    w.money += 100;
    std::cout << w.pay(150)

    std::cout << w.owner << "/n";
    w.get_owner() = "Eve";
    std::cout << w.owner << "/n";

}
```

Klassen

```cpp
#include <iostream>

class Wallet{
    unsigned int money;
    std::string owner;

public:
    Wallet(std::string o, unsigned int m){
        money = m;
        owner = o;
    }

    unsigned int money_left(){
        return money;
    }

    bool pay(unsigned int amount){
        if(money >= amount){
            money -= amount;
            return true;
        }
        else{
            return false;
        }
    }

    std::string get_owner(){
        return owner;
    }
};
```

```cpp
int main(){
    Wallet w("Pascal",100);
    std::cout << w.money_left() << " "
              << w.get_owner() << "\n";

    // Pay 50.-
    bool success1 = w.pay(50);
    std::cout << success1 << " "
              << w.money_left() << " "
              << w.get_owner() << "\n";

    // Pay 100.-
    bool success2 = w.pay(75);
    std::cout << success2 << " "
              << w.money_left() << " "
              << w.get_owner() << "\n";
}
```

SPOILER

Klassen

```
Wallet(std::string o, unsigned int m){
    money = m;
    owner = o;
}
```

```
Wallet::Wallet(std::string o, unsigned int m): owner(o), money(m){}
```

Headerfiles (.h)

- Um grössere Projekte besser strukturieren zu können werden "Headerfiles" verwendet.
- Sie definieren *was* eine C++ Klasse macht, aber noch nicht *wie*.
- Zu den Header Files kreiert man dann ein *.cpp* File, welches die Klasse implementiert.

Headerfiles (.h)

wallet.h

```
1  #ifndef WALLET_H
2  #define WALLET_H
3
4  #include <iostream>
5
6  class Wallet{
7      unsigned int money;
8      std::string owner;
9
10 public:
11     // POST: initializes new wallet
12     Wallet(std::string, unsigned int);
13
14     // POST: returns how much money
15     //        is left in wallet
16     unsigned int money_left();
17
18     // POST: Pays an amount of money if
19     //        there's enough available
20     bool pay(unsigned int amount);
21
22     // POST: returns the owner of
23               the wallet
24     std::string get_owner();
25 };
26
27 #endif
```

Include Guard stellt sicher, dass Funktionen nicht mehrmals deklariert werden

Funktionen werden deklariert aber nicht implementiert

Dokumentation (Pre/Post Conditions) schreibt man schon im Header file.

Ende Include Guard

Header Files (.h)

wallet.cpp

```
1  #include "wallet.h"
2
3  Wallet::Wallet(std::string o, unsigned int m){
4      money = m;
5      owner = o;
6  }
7
8  unsigned int Wallet::money_left(){
9      return money;
10 }
11
12 bool Wallet::pay(unsigned int amount){
13     if(money >= amount){
14         money -= amount;
15     return true;
16     }
17     else{
18         return false;
19     }
20 }
21
22 std::string Wallet::get_owner(){
23     return owner;
24 }
```

Headerfile wird included

Keine Pre/Post Conditions mehr nötig

Funktionen werden deklariert

Deklaration nach dem Schema *ClassName::functionName*

Header Files (.h)

main.cpp

```cpp
#include <iostream>
#include "wallet.h"

int main(){
    Wallet w("Pascal",100);
    std::cout << w.money_left() << " "
              << w.get_owner() << "\n";

    // Pay 50.-
    bool success1 = w.pay(50);
    std::cout << success1 << " "
              << w.money_left() << " "
              << w.get_owner() << "\n";

    // Pay 100.-
    bool success2 = w.pay(75);
    std::cout << success2 << " "
              << w.money_left() << " "
              << w.get_owner() << "\n";
}
```

Header File wird included

Falls ein .cpp File mitkompiliert wurde, welches:

1. das Headerfile ebenfalls importiert und
2. die Klasse implementiert,

kann die Klasse hier verwendet werden.

References Recap

```
int a = 1;
int b = 2;
int& x = a;
int& y = x;
y = b;
std::cout << a << " " << b << " " << x << " " << y << std::endl;
```

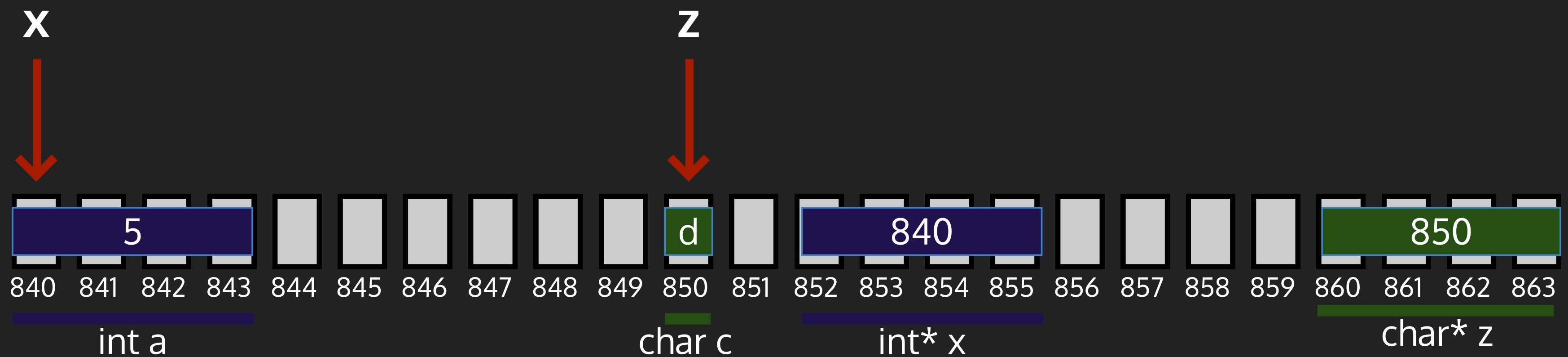
SPOILER

Pointers

- Der Speicher in Computern ist in Bytes (8 Bit) aufgeteilt.
- Jedes Byte hat eine Adresse
- Ein Pointer speichert nicht einen Wert sondern die Adresse von einem Wert.

Beispiel

```
1  int a = 5;
2  int* x = &a;
3
4  char c = 'd';
5  char* z = &c;
```



Operatoren

&

1. Bitwise AND

```
z = x & y;
```

2. Variabel als Referenz deklarieren

```
int& y = x;
```

3. Adresse einer Variabel herausfinden

```
int* ptr_a = &a;
```

*

1. Multiplikation

```
z = x * y;
```

2. Variabel als Pointer deklarieren

```
int* ptr_a = &a;
```

3. Inhalt von einem Pointer herausfinden

```
int a = *ptr_a;
```

New & Delete

- Mit dem Keyword *new* kann man Pointer erstellen, welche auf Daten zeigen die niemals out of scope gehen.
- Im Gegenzug muss man selber die Daten mit dem Keyword *delete* löschen falls man diese nicht mehr braucht.

New & Delete

```cpp
#include <iostream>

int* create_value(){
    int x = 5;
    return &x;
}

int main(){
    int* ptr = create_value();

    std::cout << *ptr << "\n";
    return 0;
}
```

```
warning: address of local variable
'x' returned

Segmentation fault (core dumped)
```

```cpp
#include <iostream>

int* create_value(){
    int* x = new int(5);
    return x;
}

int main(){
    int* ptr = create_value();

    std::cout << *ptr << "\n";

    delete ptr;

    return 0;
}
```

```
5
```

New & Delete (Arrays)

```
1  #include <iostream>
2
3  int* create_array(){
4      int* a = new int[5];
5
6      for(unsigned int i = 0; i < 5; i++){
7          *(a + i) = i;
8      }
9
10     return a;
11 }
12
13 int main(){
14     int* a = create_array();
15
16     std::cout << *a << " ";
17     std::cout << *(a + 1) << " ";
18     std::cout << *(a + 2) << " ";
19     std::cout << *(a + 3) << " ";
20     std::cout << *(a + 4) << " ";
21
22     delete[] a;
23
24     return 0;
25 }
```

Stellt speicher für 5 Integer bereit

Pointer dereferenzieren um den dort gespiecherten Wert zu ändern

Pointer dereferenzieren um den dort gespiecherten Wert zu lesen

Output:

```
0 1 2 3 4
```

Arrays – Syntaktischer Zucker

`*(a + 1)`

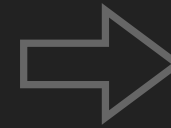
`a[1]`

```
#include <iostream>

int* create_array(){
    int* a = new int[5];

    for(unsigned int i = 0; i < 5; i++){
        *(a + i) = i;
    }

    return a;
}

int main(){
    int* a = create_array();

    std::cout << *a << " ";
    std::cout << *(a + 1) << " ";
    std::cout << *(a + 2) << " ";
    std::cout << *(a + 3) << " ";
    std::cout << *(a + 4) << " ";

    delete[] a;

    return 0;
}
```

```
#include <iostream>

int* create_array(){
    int* a = new int[5];

    for(unsigned int i = 0; i < 5; i++){
        a[i] = i;
    }

    return a;
}

int main(){
    int* a = create_array();

    std::cout << a[0] << " ";
    std::cout << a[1] << " ";
    std::cout << a[2] << " ";
    std::cout << a[3] << " ";
    std::cout << a[4] << " ";

    delete[] a;

    return 0;
}
```

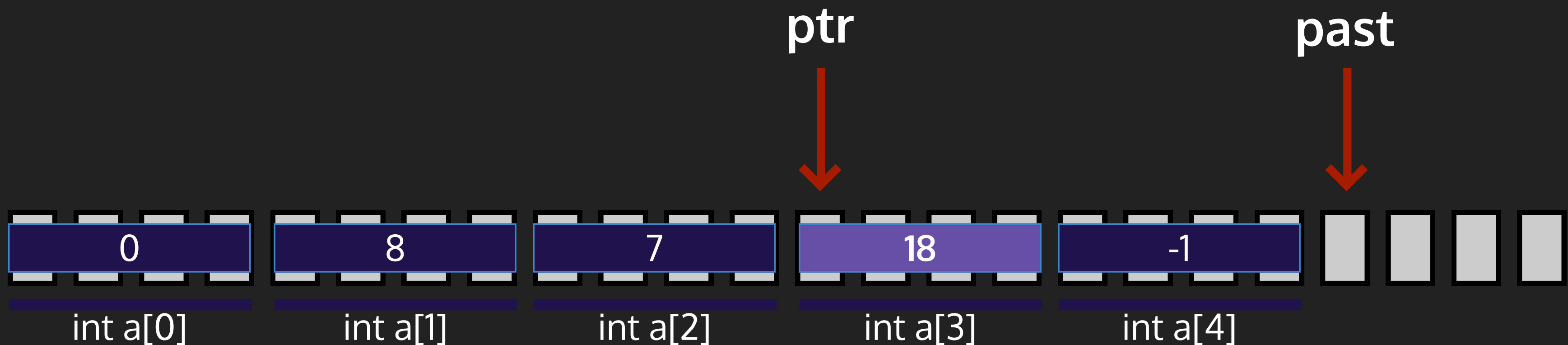
Beispiel

```cpp
1 int* a = new int[5]{0, 8, 7, 2, -1};
2 int* ptr = a;            // pointer assignment
3 ++ptr;                   // Shift pointer to right
4 int my_int = *ptr;       // read target
5 ptr += 2;                // shift by 2 elements
6 *ptr = 18;               // overwrite target
7 int* past = a+5;
8 std::cout << my_int << " " << (ptr < past) << "\n";
9 delete[] a;
```



Output:

SPOILER



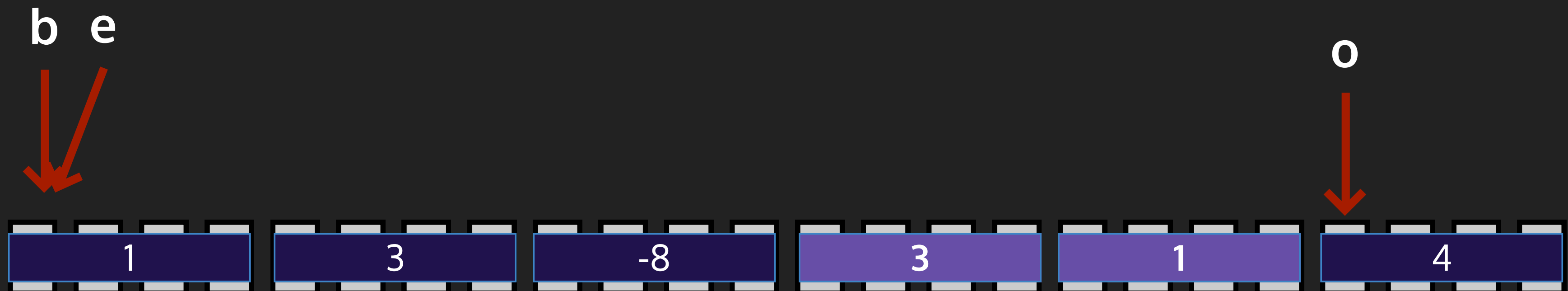
my_int = 8

Findet die 3 Fehler

```cpp
#include <iostream>
int main(){
    int* a = new int[7]{0, 6, 5, 3, 2, 4, 1};
    int* b = new int[7];
    int* c = b;

    // copy a into b using pointers
    for(int* p = a; p <= a + 7; ++p){
        *c++ = *p;
    }

    // cross-check
    for(int i = 0; i <= 7; ++i){
        if(a[i] != c[i]){
            std::cout << "Oops, copy error...\n";
        }
    }
    return 0;
}
```

Aufgabe

```
1 // PRE: [b, e) and [o, o+(e-b)) are disjoint and valid ranges
2 void f (int* b, int* e, int* o) {
3     while (b != e) {
4         --e;
5         *o = *e;
6         ++o;
7     }
8 }
```



Aufgabe

```
1  // PRE: [b, e) and [o, o+(e-b)) are disjoint and valid ranges
2  // POST:
3  //
4  void f (int* b, int* e, int* o) {
5      while (b != e) {
6          --e;
7          *o = *e;
8          ++o;
9      }
10 }
```

Aufgabe

```
1 // PRE: [b, e) and [o, o+(e-b)) are disjoint and valid ranges
2 void f (int* b, int* e, int* o) {
3     while (b != e) {
4         --e;
5         *o = *e;
6         ++o;
7     }
8 }
```

```
int* a = new int[5];
```

Welche dieser Funktionsaufrufe ist valid?

```
f(a, a+5, a+5);
```

```
f(a, a+2, a+3);
```

```
f(a, a+3, a+2);
```

Vorbesprechung

SPOILER

4: Quick Sort – input

```
void input(std::istream& is, int*& begin, int*& end)
```

Input:

5 3 2 1 4 5

1. Speicher Allozieren

```
a = new int[size]
```

2. Speicher Füllen

4: Quick Sort – output

```
void output(std::ostream& os,const int* begin,const int* end)
```

Output:

3 2 1 4 5

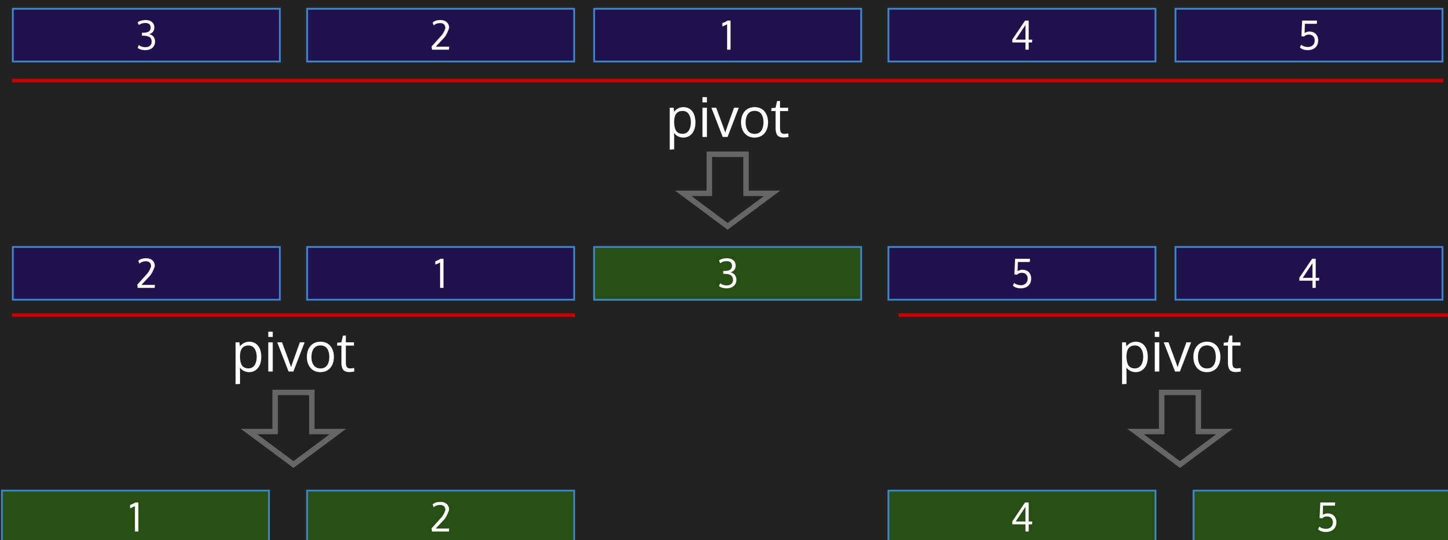
4: Quick Sort - pivot

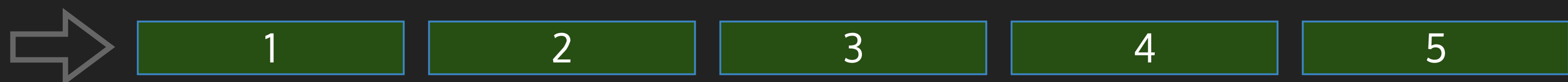
```
int* pivot(int* begin, int* end)
```

```
1 Wähle ein non-pivot Element (b)
2 Swappe entweder mit dem ersten (p) oder
  letzten (e) Element von der Range
3 Schrumpf die Range so dass das neu
  platzierte Element nicht mehr darin
  vorkommt.
```

return p

4: Quick Sort - quicksort





4: Quick Sort - main

```
int main () {
    int* begin;
    int* end;
    input(std::cin,begin,end);
    quicksort(begin,end);
    output(std::cout,begin,end);
    delete[] begin;
    return 0;
}
```

Viel Spass!