



Übungsstunde W11

Informatik (RW & CBB & Statistik) – HS 23



rwko.ch/lily

Heutiges Programm

Follow-up

Feedback zu **code expert**

Ziele

& vs *

Referenzen vs Pointer

`this->`

Dynamische Datenstrukturen & Iteratoren

Outro

1. Follow-up

Follow-up aus vorherigen Übungsstunden

Bezüglich PVK

- Ihr solltet mittlerweile alle eine Mail vom VMP erhalten haben, die euch auf die PVKs hinweist

2. Feedback zu **code** expert

Allgemeines zu **code expert**

- In Woche09 gab es viel weniger Abgaben. Ist es wegen der Bonusaufgabe oder hat es (auch noch) andere Gründe?

E8:T1: "Vector and Matrix Operations"

- Achtet euch auf die "constness" der Funktionsargumente
- Die Vektoren und Matrizen sollen nicht verändert werden → als **const** Referenzen übergeben

E8:T4: "Trapezoid Printing"

- Lest die Aufgabenstellungen genau durch :)
- Achtung bei **print_diamond** und **print_hourglass**: Es braucht Spezialfälle, falls die Breiten 0 sind, sonst gibt es unendlichen Output

Fragen/Unklarheiten?

3. Ziele

Ziele

- Unterschiede zwischen Pointern und Referenzen verstehen
- Programme mit Pointern tracen und schreiben können
- Programme mit dynamischem Speicher schreiben können
- simple Container implementieren können

4. & VS *

Bedeutungen von &

Das Symbol & hat in C++ viele Bedeutungen.

Bedeutung von &

1. als AND-operator

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

2. um eine Variable als Alias zu *deklarieren*

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

3. um die *Adresse* einer Variable zu erhalten (address-operator)

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

Bedeutungen von *

Dito mit dem Symbol *.

Bedeutung von *

1. als (arithmetischer) Multiplikation-operator

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

2. um eine Pointer-Variable zu deklarieren

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

3. um auf eine Variable via ihrem Pointer zuzugreifen
(dereference-operator)

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

Fragen/Unklarheiten?

5. Referenzen vs Pointer

References

```
void references(){  
    int a = 1;  
    int b = 2;  
    int& x = a;  
    int& y = x;  
    y = b;  
  
    std::cout  
    << a << " "  
    << b << " "  
    << x << " "  
    << y << std::endl;  
}
```

Trace das Programm und schreibe den erwarteten Output hin, wenn die Funktion aufgerufen wurde

References

```
void references(){  
    int a = 1;  
    int b = 2;  
    int& x = a;  
    int& y = x;  
    y = b;  
  
    std::cout  
    << a << " "  
    << b << " "  
    << x << " "  
    << y << std::endl;  
}
```

Trace das Programm und schreibe den erwarteten Output hin, wenn die Funktion aufgerufen wurde

2 2 2 2

Pointers

```
void pointers(){  
    int a = 1;  
    int b = 2;  
    int* x = &a;  
    int* y = x;  
  
    std::cout  
    << a << " "  
    << b << " "  
    << x << " "  
    << y << std::endl;  
}
```

Trace das Programm und schreibe den erwarteten Output hin, wenn die Funktion aufgerufen wurde

Pointers

```
void pointers(){  
    int a = 1;  
    int b = 2;  
    int* x = &a;  
    int* y = x;  
  
    std::cout  
    << a << " "  
    << b << " "  
    << x << " "  
    << y << std::endl;  
}
```

Trace das Programm und schreibe den erwarteten Output hin, wenn die Funktion aufgerufen wurde

1 2 0x7ffe4d1fb904 0x7ffe4d1fb904

(Die Adressen könnten bei jedem Aufruf anders sein!)

Pointers and Addresses

```
void ptrs_and_addresses(){  
    int a = 5;  
    int b = 7;  
  
    int* x = nullptr;  
    x = &a;  
  
    std::cout << a << "\n";  
    std::cout << *x << "\n";  
  
    std::cout << x << "\n";  
    std::cout << &a << "\n";  
}
```

Trace das Programm und schreibe den erwarteten Output hin, wenn die Funktion aufgerufen wurde

Pointers and Addresses

```
void ptrs_and_addresses(){
    int a = 5;
    int b = 7;

    int* x = nullptr;
    x = &a;

    std::cout << a << "\n";
    std::cout << *x << "\n";

    std::cout << x << "\n";
    std::cout << &a << "\n";
}
```

Trace das Programm und schreibe den erwarteten Output hin, wenn die Funktion aufgerufen wurde

5

5

0x7ffe4d1fb914

0x7ffe4d1fb914

(Die Adressen könnten bei jedem Aufruf anders sein!)

Fragen/Unklarheiten?

6. this->

WTF ist `this->`?

Bedeutung von `this->`

`this->` hat zwei Teile

- `this`

- ist ein Pointer zum *aktuellen* Objekt (Class oder Struct)
- also vom Typ `T*`

- `->`

- ist ein sehr cool aussehender Operator
- `this->member_element` ist äquivalent zu `*(this).member_element`
- Der Pfeil-Operator dereferenziert einen Pointer zu einem Objekt, um auf einen seiner Members zuzugreifen (Funktionen oder Variablen)

7. Dynamische Datenstrukturen & Iteratoren

"Our-List" Primer I

Wir implementieren unsere eigene Linked-List (zumindest Teile davon)



- Eine Liste besteht aus "Blöcken" von **lnodes**, wobei eine **lnode** immer auf die nächste zeigt
- Aber was ist überhaupt eine **lnode**?
- Antwort: ein Struct, das aus einem **int value** und einem **lnode pointer** besteht

"Our-List" Primer I

Erste Aufgabe: Implementiere einen Constructor, der eine neue Liste mit Iteratoren initialisiert

- Wir wollen schreiben können: `our_list my_list(begin, end);`
- Idee: Benutze die Iteratoren, um neue `lnodes` in die Liste hinzuzufügen
- Wie können wir auf die verschiedenen Elemente zugreifen?

- Zugriff auf Wert der `lnode`, auf die der Iterator zeigt:

`*it`

- Nächste `lnode` in der Folge:

`node->next`

- Pointer zu neuer `lnode` erstellen::

`new lnode{value, pointer}`

Denkt daran: `new T` gibt einen `T*` zurück

Aufgabe "our_list::init"

- Öffnet "our_list::init" auf **code expert**
- Überlegt euch, wie ihr das Problem mit Stift und Papier angehen würdet
- Programmiert eine Lösung (optional in Gruppen)

Aufgabe "our_list::init" (Lösung)

```
our_list::our_list(our_list::const_iterator begin,
                  our_list::const_iterator end) {
    this->head = nullptr;           // Init head (safely)

    if (begin == end) {return;}    // Case: empty list

    our_list::const_iterator it = begin; // Adding first element
    this->head = new lnode { *it, nullptr };
    ++it;
    lnode *node = this->head;

    for (; it != end; ++it) {      // Adding remainig elements
        node->next = new lnode { *it, nullptr };
        node = node->next;
    }
}
```

Fragen/Unklarheiten?

Zweite Aufgabe: Implementiere eine Funktion der Class "our_list", die eine Node mit der nächsten tauscht

- Ihr könnt eine recht ähnliche Herangehensweise wie bei anderen swap Funktionen benutzen (also mit einer temporären Variable `tmp`)
- Aber:
 - Benutzt Pointer
 - Was passiert im Fall 0 (wenn der Head Pointer getauscht werden soll)?
 - Wie könnt ihr vermeiden, dass nicht plötzlich auf unerlaubten Speicher zugegriffen wird?

Aufgabe "our_list::swap"

- Öffnet "our_list::swap" auf **code expert**
- Überlegt euch, wie ihr das Problem mit Stift und Papier angehen würdet
- Programmiert eine Lösung (optional in Gruppen)

Aufgabe "our_list::swap" (Lösung)

```
void our_list::swap(unsigned int index) {  
  
    if (index == 0) {  
  
        assert(this->head != nullptr);  
        assert(this->head->next != nullptr);  
  
        lnode* tmp = this->head->next;  
        this->head->next = this->head->next->next;  
        tmp->next = this->head;  
        this->head = tmp;  
  
    }  
}
```

Aufgabe "our_list::swap" (Lösung)

```
else { lnode* prev = nullptr;
       lnode* curr = this->head;

       while (index > 0) {                               // Find the element
           prev = curr;
           curr = curr->next;
           --index;
       }

       assert(curr != nullptr);
       assert(curr->next != nullptr);

       lnode* tmp = curr->next;                           // Swap with the next one
       curr->next = curr->next->next;
       tmp->next = curr;
       prev->next = tmp;                                  }}// two '}' to close function
```

Fragen/Unklarheiten?

8. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schöne Woche!