constness & const correctness

- In Theorie: Alles was const sein kann, soll auch const sein.
- Heisst: Falls eine Variable/ein Objekt nicht verändert werden muss, dann soll const verwendet werden.
- Insbesondere auch wenn eine Funktion ein Objekt nur lesen möchte, es aber schon klar ist, dass die Funktion das Objekt nicht verändern möchte.

`https://stackoverflow.com/questions/136880/sell-me-const-correctness`

`https://isocpp.org/wiki/faq/const-correctness`

In der Praxis/für unseren Kurs: Es hilft euch bugs zu vermeiden, z.B. in längeren Codes. Aber: Ihr müsst keine extra Energie verwenden um alles perfekt const correct zu machen (wenn es nicht steht).

Referenzen

int u int

```
int foo1( int& i) { return ++i; }
```

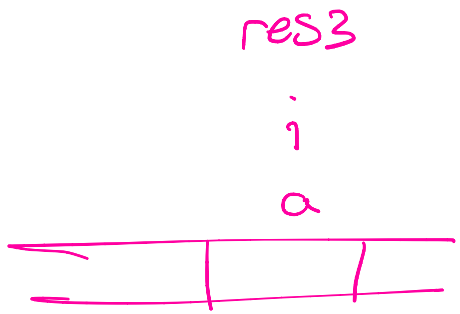
local i

```
int& foo2( int i) { return ++i; }

const int& foo3( int& i) {

    return ++i;
}

// e.g. in main
int a = 5;
int& b = a; // ref auf a, wie bisher
int res1 = foo1(a); // a nacher 6, res1 = 6
const int& res3 = foo3(a);
```



Rekursion

```
void function(input size){
    // base case
    ...
    function(smaller input size) // recursive call here
    ...
}
int main(){
    ...
    function(input size)
    ...
}
```

Rekursion: Ansätze

- Grosses Problem in kleinere Teilprobleme Aufteilen: Was könnten mögliche Teilprobleme sein?
- Immer kleinere Teilprobleme machen, bis die Aufgabe einfach lösbar ist: Welche Aufteilung hilft tatsächlich und macht Fortschritt in richtung base case?
- Base case: Die Problemgrösse, welche einfach zu lösen ist und die rekursion beendet.
- Wie können die Lösungen der Teilprobleme helfen, das ursprüngliche Problem zu lösen? Oder anders, wie können die Teillösungen zusammengesetzt werden, damit sie zur Gesamtlösung beitragen?

Rekursion: "Checkliste"

- Problemgrösse wird beim nächsten function call kleiner?
- Haben wir einen Base case?
- Wird unsere Rekursion terminieren? Heisst: Nähern wir uns dem base case?
- Werden Lösungen schon gelöster Teilprobleme benutzt/zusammengesetzt um das nächst grössere Problem zu lösen?

Recursion Exercise: Power Set

$\{a, b, c\}$

A set's *power set* is defined as the set of all its subsets $X \subseteq S$. That is, given a set S, its power set 2^S is defined as:
$2^S := \{X \mid X \subseteq S\}$

Example Given the set $A = \{a, b, c\}$, its power set 2^A is the following: $2^A = \{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

Recursion Exercise: Power Set

Q: What is the power set of the empty set $\{\}$?
A:

Q: What is the power set of $\{a\}$?
A:

Q: What is the power set of $\{a, b\}$?
A:

Recursion Exercise: Power Set

Q: What is the power set of the empty set $\{\}$?
A: A set containing a single element – the empty set. $2^{\{\}} = \{\{\}\}$

Q: What is the power set of $\{a\}$?
A: $2^{\{a\}} = \{\{\}, \{a\}\} = \{\{\}\} \cup \{\{a\}\}$

Q: What is the power set of $\{a, b\}$?
A: $2^{\{a,b\}} = \{\{\}, \{a\}, \{b\}, \{a, b\}\} = \{\{\}, \{a\}\} \cup \{\{b\}, \{a, b\}\}$
Note: $2^{\{a\}}$ is a subset of $2^{\{a,b\}}$.

Recursion Exercise: Power Set

Generally we see: Given two sets S_1 and S_2, if $S_1 \subset S_2$, it follows that $2^{S_1} \subset 2^{S_2}$.

Q: Can we use this to propose a general pattern for computing a power set? E.g. given an "old" set $\{a, b\}$ and a "new" set $\{a, b, c\}$?

Recursion Exercise: Power Set

Q: What is the general pattern for computing a power set?
A: We can notice, each time we add a new character, the power set doubles and half of the "new" set is the power set of the "old" set.

$\cup \{c\}$

$$\underbrace{\underbrace{\underbrace{\underbrace{\{\}}_{2^{\{\}}} \quad \{a\}}_{2^{\{a\}}} \quad \{b\} \quad \{a,b\}}_{2^{\{a,b\}}} \quad \{c\} \quad \{a,c\} \quad \{b,c\} \quad \{a,b,c\}}_{2^{\{a,b,c\}}}$$

"result"

Recursion Exercise: Power Set

Formulated a bit more mathematically:

$$
\begin{aligned}
2^{\{a,b,c\}} &= \{\{\}, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\} \\
&= \{\{\}, \{a\}, \{b\}, \{a,b\}\} \\
&\qquad\qquad \cup \{\{c\}, \{a,c\}, \{b,c\}, \{a,b,c\}\} \\
&= \{\{\}, \{a\}, \{b\}, \{a,b\}\} \\
&\qquad\qquad \cup \{Y \cup \{c\} \mid Y \in \{\{\}, \{a\}, \{b\}, \{a,b\}\}\} \\
&= 2^{\{a,b\}} \cup \{Y \cup \{c\} \mid Y \in 2^{\{a,b\}}\}
\end{aligned}
$$

Can we use this to propose a recursive algorithm?

Recursion Exercise: Power Set

Given a set of characters (e.g. $S = \{a, b, c\}$, we want the power set of S.
Proposal for a recursive algorithm:

Recursion Exercise: Power Set

Given a set of characters (e.g. $S = \{a, b, c\}$), we want the power set of S.
Proposal for a recursive algorithm:

$S = \{a, b, c\}$

$S' = \{a, b\}$

1. Select some character $x \in S$.
2. Build a new set $S' := S \setminus \{x\}$.
3. Compute $2^{S'}$.
4. Return $2^S = 2^{S'} \cup \{Y \cup \{x\} \mid Y \in 2^{S'}\}$.

$2^{S'}$

Rekursion: "Checkliste"

- Problemgrösse wird beim nächsten function call kleiner?
- Haben wir einen Base case?
- Wird unsere Rekursion terminieren? Heisst: Nähern wir uns dem base case?
- Werden Lösungen schon gelöster Teilprobleme benutzt/zusammengesetzt um das nächst grössere Problem zu lösen?

Recursion Exercise: Power Set

Given a set of characters (e.g. $S = \{a, b, c\}$, we want the power set of S.
Proposal for a recursive algorithm:

1. Select some character $x \in S$.
2. Build a new set $S' := S \setminus \{x\}$.
3. Compute $2^{S'}$.
4. Return $2^S = 2^{S'} \cup \{Y \cup \{x\} \mid Y \in 2^{S'}\}$.

Recursion Exercise: Power Set

1. If $S = \{\}$, base case:
   1.1 Return $\{\{\}\}$.
2. Otherwise, general case:
   2.1 Select some character $x \in S$.
   2.2 Build a new set $S' := S \setminus \{x\}$. → new set
   2.3 Compute $2^{S'}$ recursively.
   2.4 Return $2^S = 2^{S'} \cup \{Y \cup \{x\} \mid Y \in 2^{S'}\}$.

(template)

Power Set Possible Solution

using CharSet = Set<char>
using SOCS = Set<Set<char>>

S

```
SetOfCharSets power_set(const CharSet& set) {
  // base case: empty set
  if (set.size() == 0) {
    return SetOfCharSets(CharSet());
  }

  // set has at least 1 element
  // split set into two sets.
  // (1) a set containing only the first element
  // (2) a set containing all elements except the first element
  CharSet first_element_subset = CharSet(set.at(0));
  CharSet remaining_subset = set - first_element_subset;

  // get power set for remaining subset
  SetOfCharSets remaining_subset_power_set = power_set(remaining_subset);

  // init result with power set of remaining subset
  SetOfCharSets result = remaining_subset_power_set;

  // add first element to every set in the powerset
  for (unsigned int i = 0; i < remaining_subset_power_set.size(); ++i) {
    result.insert(first_element_subset + remaining_subset_power_set.at(i));
  }

  return result;
}
```

$\{\{\}\}$

conquer

$x \in S$

$S \setminus \{x\}$

"S'"

divide

compute $2^{S'}$

"combine"

x

∪

"OOP"

Structs

```
struct strange {
    int n;
    bool b;
    std::vector<int> a = std::vector<int> (0);
};
int main () {
    strange x = {1, true, {1,2,3}};
    strange y = x; // all elements are copied
    std::cout << y.n << "  " << y.a[2] << "\n";
    // outputs: 1  3
    return 0;
}
```

= 0;

= 0;

n b a

{ }

y.a.size()

obj.

Geometry Exercise

```
struct vec {
    double x;
    double y;
    double z;
};
```

FYI: List/uniform initialization (since C++11)

int a;
int b = 5;

(Kommt z.B. in "Geometry Exercise", ihr könntet aber auch alles ohne lösen.)

Idee: Alles nach dem gleichen Schema initialisieren:

```
type var_name{arg1, arg2, ....arg n}
```

{ }

Hier eine schöne Übersicht:

https://www.geeksforgeeks.org/uniform-initialization-in-c/

Hier einige gute Gründe für diese Neuerung: https://stackoverflow.com/questions/39487065/what-does-return-statement-mean-in-c11

FYI: List/uniform initialization (since C++11)

(Kommt z.B. in "Geometry Exercise", ihr könntet aber auch alles ohne lösen.)

Für Euch:

Mit {} wird euch der richtige Objekt-Typ kreiert, je nach dem was in den Klammern steht auch richtig mit den Werten initialisiert. Hier im struct Beispiel:

```
// return empty/default initialized object
return {}
// initialize new object
vec a{1,1,0};
vec a = {1,1,1}
```