Informatik - Exercise Session
Recursion and Custom Data Types

Concise Pre- and Postconditions

Consider this function from your exercises:

```
bool f(const int n) {
  if (n == 0) return false;
  return !f(n - 1);
}
```

What would be the appropriate pre- and postconditions as short as possible?

Concise Pre- and Postconditions

Consider this function from your exercises:

```
bool f(const int n) {
  if (n == 0) return false;
  return !f(n - 1);
}
```

What would be the appropriate pre- and postconditions as short as possible?
One example (pre: constraints for arguments, post: return value and side effects):

```
// PRE:   n >= 0
// POST1: returns true if n is even, false otherwise
// POST2: returns if n is even // careful with this one
```

Concise Pre- and Postconditions

Consider this function from your exercises:

```
bool f(const int n) {
  if (n == 0) return false;
  return !f(n - 1);
}
```

What would be the appropriate pre- and postconditions as short as possible?
One example (pre: constraints for arguments, post: return value and side effects):

```
// PRE:   n >= 0
// POST1: returns true if n is even, false otherwise
// POST2: returns if n is even // careful with this one
```

Try to keep your pre- and postconditions as short as possible, but still include all relevant information without leaving room for wrong interpretation:
`returns only if n is even` vs. `returns true if n is even`

Tip: Rewriting for-loops

In certain conditions (to be precise: when *iterators* are implemented correctly for the container, which you will see later), we can use a "shorter version" of the `for`-loop to iterate over a container. We can rewrite this snippet:

```
for (int i = 0; i < c.size(); i++) {
    c[i] = do_something(c[i]);
}
```

Tip: Rewriting for-loops

In certain conditions (to be precise: when *iterators* are implemented correctly for the container, which you will see later), we can use a "shorter version" of the `for`-loop to iterate over a container. We can rewrite this snippet:

```
for (int i = 0; i < c.size(); i++) {
    c[i] = do_something(c[i]);
}
```

And without using indices, this becomes:

```
for (int& elem : c) {
    elem = do_something(elem);
}
```

We read the colon as "in": **for `elem` in `c`, do something**

And yes, this works with references as expected!

If you get errors that no 'begin' function is available (or anything with 'iterator'), revert to normal `for`-loops.

Structs

Structs are custom data types ("containers") for variables (and functions/"methods", as we will see later):

Structs

Structs are custom data types ("containers") for variables (and functions/"methods", as we will see later):

```
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;
}
```

Recursion Example: Power Set Explanation

The power set $P(S)$ of a set S is the set of all its subsets $Y \subseteq S$.

Recursion Example: Power Set Explanation

The power set $P(S)$ of a set S is the set of all its subsets $Y \subseteq S$.

$$P(\emptyset) =$$

Recursion Example: Power Set Explanation

The power set $P(S)$ of a set S is the set of all its subsets $Y \subseteq S$.

$$P(\emptyset) = \{\emptyset\} = \{\{\}\}$$

Recursion Example: Power Set Explanation

The power set $P(S)$ of a set S is the set of all its subsets $Y \subseteq S$.

$$P(\emptyset) = \{\emptyset\} = \{\{\}\}$$

$$P(\{a\}) =$$

Recursion Example: Power Set Explanation

The power set $P(S)$ of a set S is the set of all its subsets $Y \subseteq S$.

$$P(\emptyset) = \{\emptyset\} = \{\{\}\}$$

$$P(\{a\}) = \{\{\}, \{a\}\}$$

Recursion Example: Power Set Explanation

The power set $P(S)$ of a set S is the set of all its subsets $Y \subseteq S$.

$$P(\emptyset) = \{\emptyset\} = \{\{\}\}$$

$$P(\{a\}) = \{\{\}, \{a\}\}$$

$$P(\{a, b\}) =$$

Recursion Example: Power Set Explanation

The power set $P(S)$ of a set S is the set of all its subsets $Y \subseteq S$.

$$P(\emptyset) = \{\emptyset\} = \{\{\}\}$$

$$P(\{a\}) = \{\{\}, \{a\}\}$$

$$P(\{a, b\}) = \{\{\}, \{a\}, \{b\}, \{a, b\}\}$$

Recursion Example: Power Set Explanation

The power set $P(S)$ of a set S is the set of all its subsets $Y \subseteq S$.

$$P(\emptyset) = \{\emptyset\} = \{\{\}\}$$

$$P(\{a\}) = \{\{\}, \{a\}\}$$

$$P(\{a, b\}) = \{\{\}, \{a\}, \{b\}, \{a, b\}\}$$

$$S_1 \subset S_2 \implies$$

Recursion Example: Power Set Explanation

The power set $P(S)$ of a set S is the set of all its subsets $Y \subseteq S$.

$$P(\emptyset) = \{\emptyset\} = \{\{\}\}$$

$$P(\{a\}) = \{\{\}, \{a\}\}$$

$$P(\{a, b\}) = \{\{\}, \{a\}, \{b\}, \{a, b\}\}$$

$$S_1 \subset S_2 \implies P(S_1) \subset P(S_2)$$

Recursion Example: Power Set Implementation

General algorithm:

Recursion Example: Power Set Implementation

General algorithm:

1. Select any $x \in S$

Recursion Example: Power Set Implementation

General algorithm:

1. Select any $x \in S$
2. Build the reduced set $S' := S \setminus \{x\}$

Recursion Example: Power Set Implementation

General algorithm:

1. Select any $x \in S$
2. Build the reduced set $S' := S \setminus \{x\}$
3. Compute $P(S')$

Recursion Example: Power Set Implementation

General algorithm:

1. Select any $x \in S$
2. Build the reduced set $S' := S \setminus \{x\}$
3. Compute $P(S')$
4. Return $P(S) = P(S') \cup \{Y \cup \{x\} \mid Y \in P(S')\}$

Recursion Example: Power Set Implementation

General algorithm:

1. Select any $x \in S$
2. Build the reduced set $S' := S \setminus \{x\}$
3. Compute $P(S')$
4. Return $P(S) = P(S') \cup \{Y \cup \{x\} \mid Y \in P(S')\}$

Will this terminate?

Recursion Example: Power Set Implementation

General algorithm:

1. Select any $x \in S$
2. Build the reduced set $S' := S \setminus \{x\}$
3. Compute $P(S')$
4. Return $P(S) = P(S') \cup \{Y \cup \{x\} \mid Y \in P(S')\}$

Will this terminate?
No, we need a base case:

Recursion Example: Power Set Implementation

General algorithm:

1. Select any $x \in S$
2. Build the reduced set $S' := S \setminus \{x\}$
3. Compute $P(S')$
4. Return $P(S) = P(S') \cup \{ Y \cup \{x\} \mid Y \in P(S')\}$

Will this terminate?
No, we need a base case: If $S = \emptyset$, then return $P(S) = \{\emptyset\} = \{\{\}\}$.