7.4: Geschachtelte Funktionen

Falls eine innere Funktion auf die Parameter/lokalen Variablen der *textuell umgebenden Funktionen* zugreifen kann (hier der Fall), braucht diese innere Funktion einen Zeiger auf die Kellerrahmen der äußeren Funktionen.

Im Beispiel: `fuzz()` braucht Zugriff auf `p1` von `foo()` und `p2` von `buzz()`.

Der *dynamische Vorgängerverweis* (DV), also der Zeiger auf den Kellerrahmen des *Aufrufers*, reicht nicht aus, da der Aufrufer i.A. nicht die textuell umgebende Funktion sein muss. Damit ist i.A. nicht statisch bekannt, wie viele von diesen Verweisen zurückverfolgt werden müssen, bis der „richtige“ Kellerrahmen erreicht ist.

Im Beispiel: `bar()` benötigt Zugriff auf `p1` von `foo()`. Wurde `bar()` aus `foo()` heraus aufgerufen, muss der DV nur *einmal* zurückverfolgt werden, um den Kellerrahmen von `foo()` zu finden. Für den Aufruf aus `buzz()` heraus muss die DV-Kette hingegen *zweimal* zurückverfolgt werden.

Wir brauchen deshalb zusätzlich einen sog. *statischen Vorgängerverweis* (SV), also einen Zeiger, der immer auf den Kellerrahmen der textuell umgebenden Funktion zeigt (unabhängig vom Aufrufer!).

Im Beispiel: Der SV für `bar()` zeigt immer auf den Kellerrahmen von `foo()`; unabhängig davon, ob `bar()` von `foo()` selbst oder von `buzz()` aufgerufen wurde. Damit ist der Zugriff auf `p1` einfach.

Lösung 1: Übergabe des SV als zusätzliches implizites Argument.

- Jede innere Funktion bekommt im Kellerrahmen Platz für den SV, also *einen* Zeiger auf den Kellerrahmen der *direkt* textuell umschließenden Prozedur.
- Beim Aufruf einer inneren Funktion muss passender SV-Wert als implizites Argument an aufgerufene Funktion übergeben werden, und zwar in Abhängigkeit von der „Beziehung“ zwischen Aufrufer und aufgerufener Funktion (die Art dieser „Beziehung“ kann statisch bestimmt werden!):
  - Aufruf eines „Kindes“ ($\rightarrow$ größere Schachtelungstiefe; im Beispiel: `foo()` $\rightarrow$ `bar()`): Aufrufer ist gleichzeitig textuell umgebende Funktion und damit statischer Vorgänger $\rightsquigarrow \mathrm{SV}_{\mathrm{callee}} = \mathrm{FP}_{\mathrm{caller}}$
  - Aufruf einer „Schwester“ ($\rightarrow$ gleiche Schachtelungstiefe; im Beispiel: `buzz()` $\rightarrow$ `bar()`): SV des Aufrufers ist gleichzeitig SV der aufgerufenen Funktion $\rightsquigarrow \mathrm{SV}_{\mathrm{callee}} = \mathrm{SV}_{\mathrm{caller}}$
  - Aufruf eines „Onkels“ ($\rightarrow$ kleinere Schachtelungstiefe; im Beispiel: `fuzz()` $\rightarrow$ `bar()`): Differenz der Schachtelungstiefen von Aufrufer und aufgerufener Funktion bestimmen (geht statisch!); *zur Laufzeit* SV-Kette entsprechend oft zurückverfolgen und SV für aufgerufene Funktion bestimmen
- Beim Zugriff auf eine Variable einer textuell umgebenden Funktion: Differenz der Schachtelungstiefen bestimmen (geht statisch!); zur Laufzeit SV-Kette bis zum richtigen Kellerrahmen zurückverfolgen, eigentlicher Zugriff dann wie bekannt.

Lösung 2: Verwendung eines sog. *Displays*.

- Display ist globales Array pro Kontrollfaden (und ggf. pro Funktion)
- `display[i]` zeigt immer auf Kellerrahmen der zuletzt aufgerufenen Funktion mit Schachtelungstiefe `i`

  $\rightsquigarrow$ Zugriff auf Kellerrahmen *jeder* Schachtelungstiefe mit nur einer Indirektion über das Array mögl.
- Verwaltung des Displays:

- bei *Eintritt* in Funktion mit Schachtelungstiefe `i`: alten Wert von `display[i]` auf den Stapel sichern und Zeiger auf eigenen Kellerrahmen in `display[i]` schreiben
- bei *Austritt* aus Funktion mit Schachtelungstiefe `i`: beim Eintritt auf dem Stapel gesicherten Wert von `display[i]` restaurieren

Im Vergleich zur Lösung ohne Displays *geringerer Laufzeitaufwand*, insbesondere bei vielen Zugriffen auf Variablen textuell umgebender Funktionen.