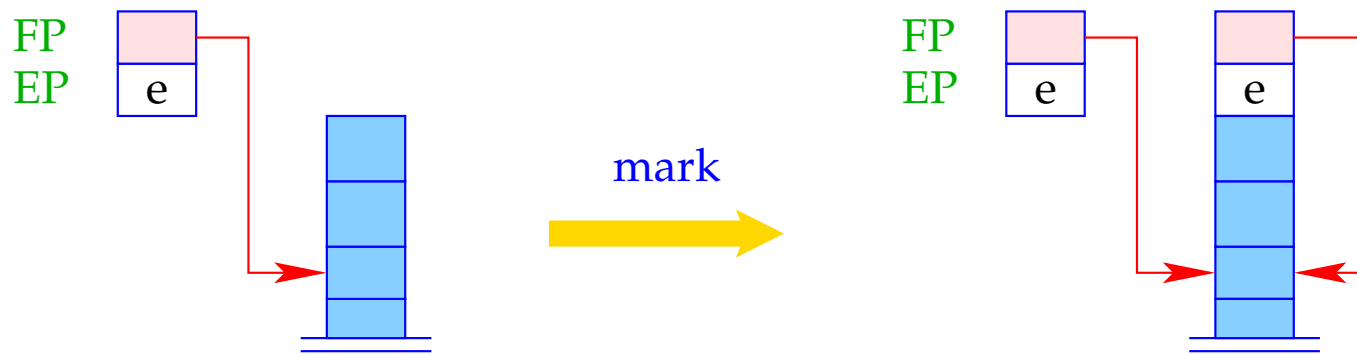


Der Befehl `mark` legt Platz für Rückgabewert und organisatorische Zellen an und rettet `FP` und `EP`.

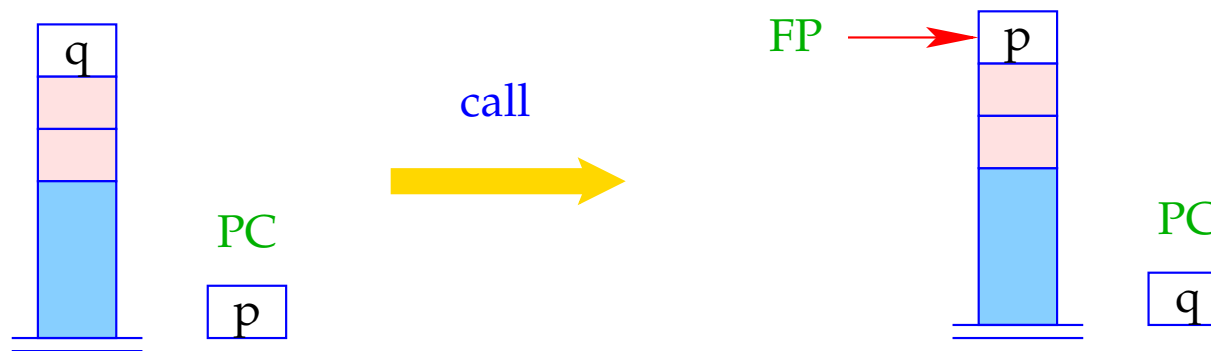


$S[SP+1] = EP;$

$S[SP+2] = FP;$

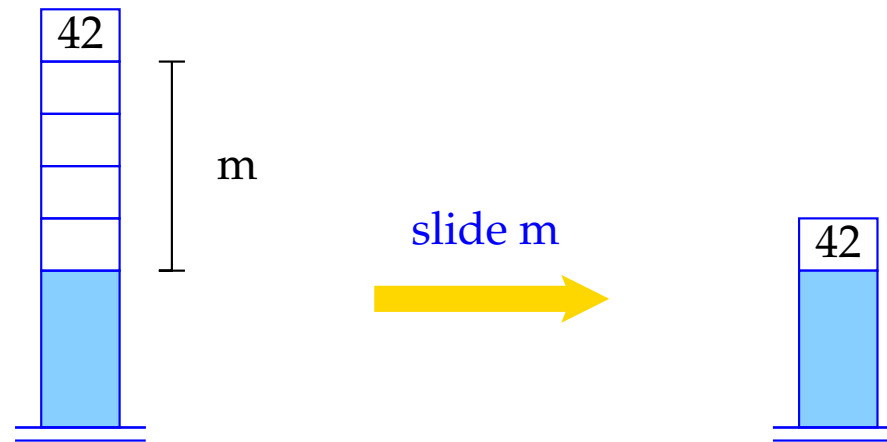
$SP = SP + 2;$

Der Befehl `call` rettet die Fortsetzungs-Adresse und setzt `FP` und `PC` auf die aktuellen Werte.



```
tmp = S[SP];  
S[SP] = PC;  
FP = SP;  
PC = tmp;
```

Der Befehl `slide` kopiert den Rückgabewert an die korrekte Stelle:



```
tmp = S[SP];
```

```
SP = SP-m;
```

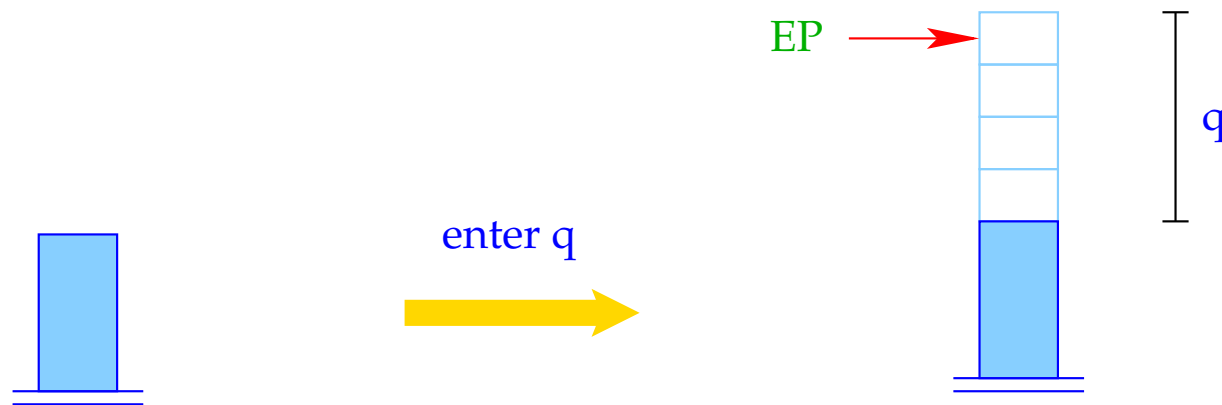
```
S[SP] = tmp;
```

Entsprechend übersetzen wir eine Funktions-Definition:

```
code t f (specs) { V_defs ss } ρ =  
    _f:  enter q      // setzen des EP  
        alloc k      // Anlegen der lokalen Variablen  
        code ss ρf  
        return      // Verlassen der Funktion
```

wobei  $q$  =  $max + k$  wobei  
 $max$  = maximale Länge des lokalen Kellers  
 $k$  = Platz für die lokalen Variablen  
 $\rho_f$  = Adress-Umgebung für  $f$   
// berücksichtigt *specs*, *V\_defs* und  $\rho$

Der Befehl `enter q` setzt den EP auf den neuen Wert. Steht nicht mehr genügend Platz zur Verfügung, wird die Programm-Ausführung abgebrochen.

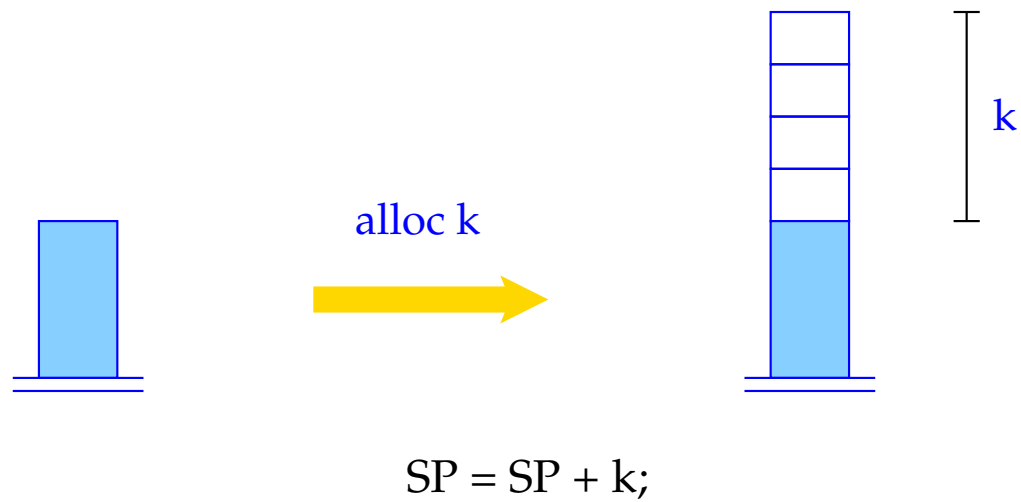


```
EP = SP + q;
```

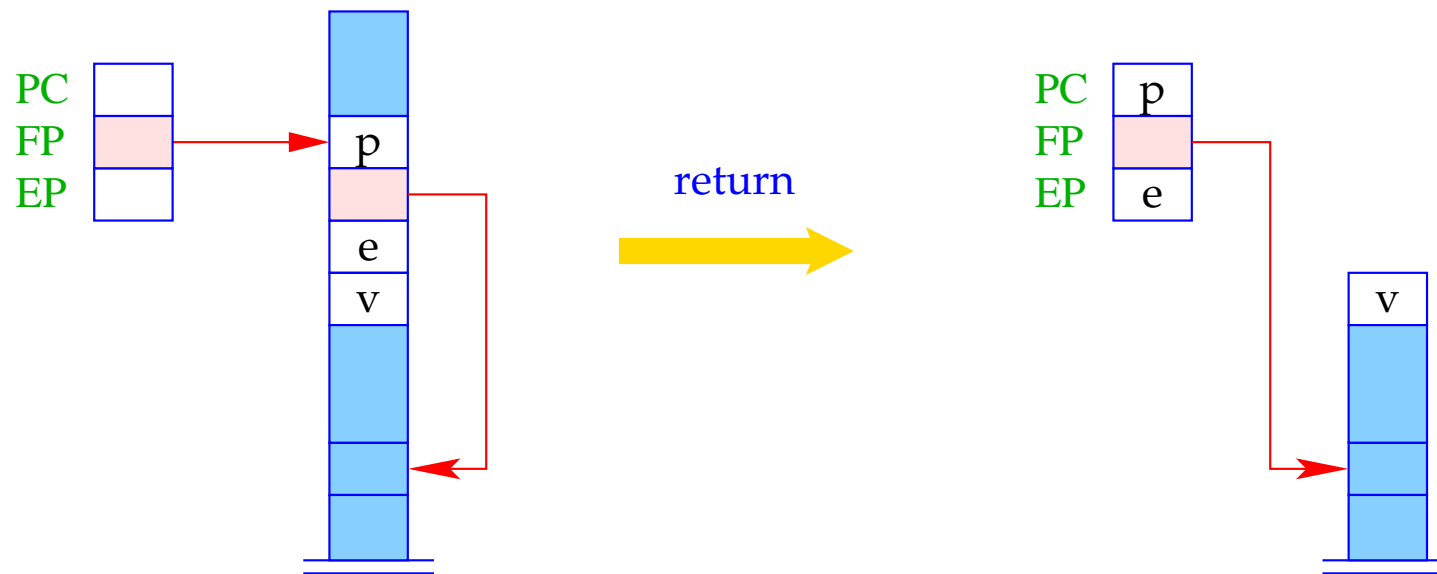
```
if (EP ≥ NP)
```

```
    Error ("Stack Overflow");
```

Der Befehl `alloc k` reserviert auf dem Keller Platz für die lokalen Variablen.



Der Befehl `return` gibt den aktuellen Keller-Rahmen auf. D.h. er restauriert die Register `PC`, `EP` und `FP`:



$PC = S[FP]; EP = S[FP-2];$

if ( $EP \geq NP$ ) Error ("Stack Overflow");

$SP = FP-3; FP = S[SP+2];$

## 9.4 Zugriff auf Variablen, formale Parameter und Rückgabe von Werten

Zugriffe auf lokale Variablen oder formale Parameter erfolgen relativ zum aktuellen FP.

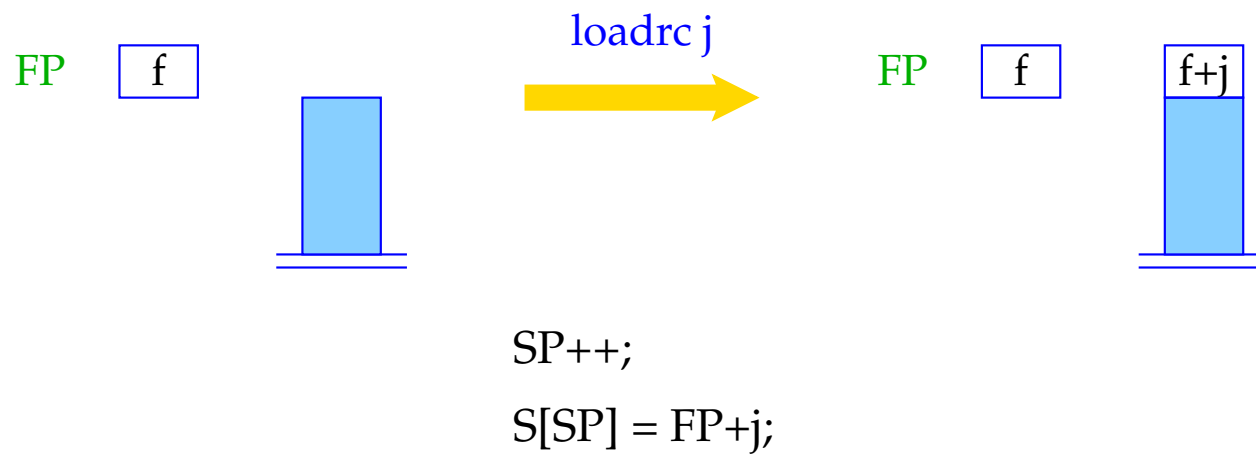
Darum modifizieren wir `codeL` für Variablen-Namen.

Für  $\rho x = (tag, j)$  definieren wir

$$\text{code}_L x \rho = \begin{cases} \text{loadc } j & tag = G \\ \text{loadrc } j & tag = L \end{cases}$$



Der Befehl `loadrc j` berechnet die Summe von `FP` und `j`.



Als Optimierung führt man analog zu `loada j` und `storea j` die Befehle `loadr j` und `storer j` ein:

`loadr j` = `loadrc j`  
`load`

`bla; storer j` = `loadrc j; bla`  
`store`

Der Code für `return e;` entspricht einer Zuweisung an eine Variable mit Relativadresse  $-3$ .

$$\text{code } \text{return } e; \rho = \text{code}_R e \rho$$

`storer -3`  
`return`

**Beispiel:** Für die Funktion

```
int fac (int x) {  
    if (x ≤ 0) return 1;  
    else return x * fac (x - 1);  
}
```

erzeugen wir:

<code>_fac:</code>	<code>enter q</code>	<code>loadc 1</code>	<code>A:</code>	<code>loadr -3</code>	<code>mul</code>
	<code>alloc 0</code>	<code>storer -3</code>		<code>loadr -3</code>	<code>storer -3</code>
	<code>loadr -3</code>	<code>return</code>		<code>loadc 1</code>	<code>return</code>
	<code>loadc 0</code>	<code>jump B</code>		<code>sub</code>	<code>B:</code> <code>return</code>
	<code>leq</code>			<code>mark</code>	
	<code>jumpz A</code>			<code>loadc _fac</code>	
				<code>call</code>	

Dabei ist  $\rho_{\text{fac}} : x \mapsto (L, -3)$  und  $q = 1 + 5 = 6$ .

## 10 Übersetzung ganzer Programme

Vor der Programmausführung gilt:

$$SP = -1 \quad FP = EP = 0 \quad PC = 0 \quad NP = \text{MAX}$$

Sei  $p \equiv V\_defs \ F\_def_1 \dots F\_def_n$ , ein Programm, wobei  $F\_def_i$  eine Funktion  $f_i$  definiert, von denen eine `main` heißt.

Der Code für das Programm  $p$  enthält:

- Code für die Funktions-Definitionen  $F\_def_i$ ;
- Code zum Anlegen der globalen Variablen;
- Code für den Aufruf von `main()`;
- die Instruktion `halt`.

Dann definieren wir:

```
code  $p \ \emptyset$    =   enter (k + 6)
                   alloc (k + 1)
                   loadc 0
                   mark
                   loadc _main
                   call
                   slide (k + 1)
                   halt
                   _f1: code  $F_{def_1} \ \rho$ 
                   ⋮
                   _fn: code  $F_{def_n} \ \rho$ 
```

wobei  $\emptyset \hat{=}$  leere Adress-Umgebung;  
 $\rho \hat{=}$  globale Adress-Umgebung;  
 $k \hat{=}$  Platz für globale Variablen