

F.16 Typedef

- Typedef erlaubt die Definition neuer Typen
 - neuer Typ kann danach wie die Standardtypen (int, char, ...) genutzt werden

- Beispiele:

```
typedef int Laenge;  
Laenge l = 5;  
Laenge *p1; Laenge f1[20];
```

```
typedef struct student Student;  
Student s; Student *ps1;  
s.matrnr = 1234567; ps1 = &s; ps1->best = 'n';
```

```
typedef struct student *Studptr;  
Studptr ps2;  
ps2 = &s; ps2->best = 'n';
```

F.17 Ergänzungen zum Präprozessor

1 Parametrisierte Makros

- Präprozessor-Makros können mit Parametern versehen werden, die in den Ersatztext eingebaut werden

- entspricht "optisch" einem Funktionsaufruf
- Unterschied: Makro wird zur Übersetzungszeit expandiert, Funktionsaufruf erfolgt zur Laufzeit
- parametrisierte Makros sind damit ähnlich zu inline-Funktionen (wie z. B. in C++)
- ein Makro wird durch die **#define**-Anweisung definiert

- Syntax:

```
#define Makroname(Par1, Par2, ...) Ersatztext
```

- Vorkommen von *Par1*, *Par2*, etc. im Ersatztext werden entsprechend eingesetzt

F.17 Ergänzungen zum Präprozessor (2)

■ Beispiele:

```
#define max(a, b) ((a) < (b) ? (b) : (a))
...
x = max (n+m, y+z);
```

```
#define numeric(c) ((c) >= '0' && (c) <= '9')
...
c = getchar();
if numeric(c) { ...
```

- im Ersatztext Parameter unbedingt in () klammern
(sonst evtl. Probleme mit Operator-Vorrang im expandierten Text!)
- einige Sonderregeln im Umgang mit Zeichenketten
(Kernighan und Ritchie schreiben dazu:
*The details ... are described more precisely in the ANSI standard ...
Some of the new rules ... are bizarre.)*

F.17 Ergänzungen zum Präprozessor (3)

2 Deaktivieren von Makros

- Makros wirken ab der Zeile in der sie definiert wurden bis zum Ende der Datei - können aber auch wieder deaktiviert werden
- Syntax:

```
#undef Makroname
```

- ab dieser Zeile unterbleibt Expansion des Makros