

Informatik 2

Assignment Project Exam Help

<https://tutorcs.com>

03 – Arrays

WeChat: cstutorcs

Sommer 2021

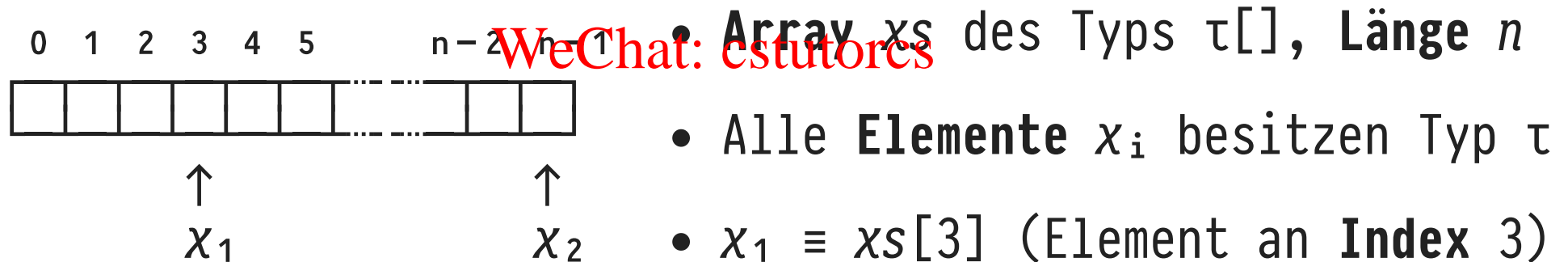
Torsten Grust
Universität Tübingen, Germany

1 | Arrays (Felder)

Arrays sind Container, die mehrere CO-Werte (auch: Elemente) speichern und in Sequenz organisieren:

- $\tau[]$ ist der Typ der **Arrays mit Elementen des Typs τ** .
- Arrays sind **homogen**: alle Elemente besitzen Typ τ .
- Arrays speichern eine **fixe Anzahl $n \in \mathbb{N}_0$** von Elementen:

<https://tutorcs.com>



Damit spiegeln Arrays direkt die lineare Organisation des primären Speichers (RAM) wider.

Array-Allokation

Array-Allokation `alloc_array(·,·)` [Ⓔ] reserviert als Seiteneffekt Speicher für ein Array des Typs `τ[]`, Länge `n`:

```
τ[] xs;
```

```
⋮
```

```
xs = alloc_array(τ, n)
```

Assignment Project Exam Help

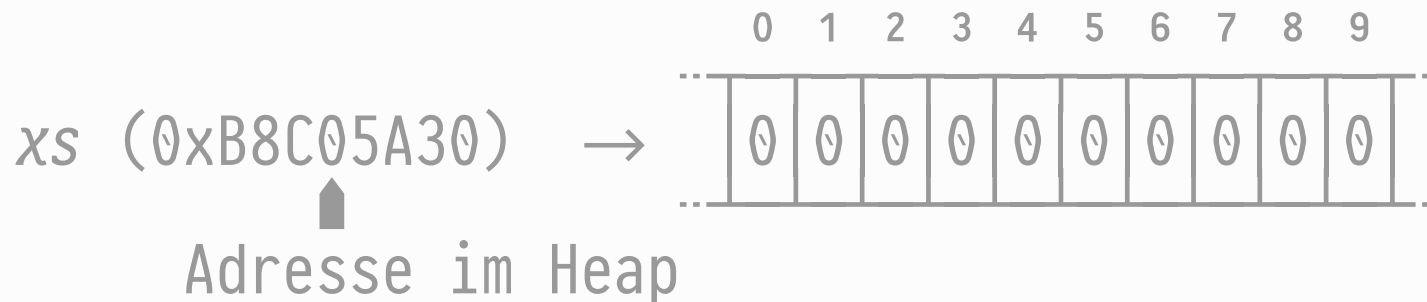
<https://tutorcs.com>

- Array `xs` wird `n × s` Bytes Speicherplatz benötigen, wenn ein Element des Typ `τ` jeweils `s` Bytes belegt.
- Da C0-Variablen maximal Werte der Größe 32 Bit halten, liefert `alloc_array(·,·)` eine **Referenz** auf das Array:

```
--> int[] xs = alloc_array(int, 10);  
xs is 0xB8C05A30 (int[] with 10 elements)
```

Typ τ	Default-Wert
int	0
bool	false
char	'\0' (Zeichen mit ASCII-Code 0)
string	""

```
int[] xs; WeChat: cstutorcs  
xs = alloc_array(int, 10); /* Referenz: 0xB8C05A30 */
```



Array-Indexierung

Array-Indexierung $xs[i]$ ^① (auch: *Indexing*) bezeichnet den Wert des Elementes an Index i (i : Ausdruck des Typs `int`).

- Sei `xs` ein Array der Länge `n`. Indexierung `xs[i]` mit ungültigem Index $i \notin [0, n-1]$ führt zu Laufzeitfehler und Programmabbruch (*index violation*).
<https://tutorcs.com>
- Zuweisung an `xs[i]` ^① überschreibt das Element an Index i in Array `xs` mit Wert des Ausdrucks `e`:

```
xs[i] = e;
```

¹ Da `xs[i]` links von der Zuweisung `=` steht, bezeichnet man `xs[i]` hier als *lvalue* (*left value*). Ein Vorkommen in Ausdruck `e` wird als *rvalue* bezeichnet. (Gleiche Bezeichnungen gelten für Variablen `v`.)

2 | Beispiel: Die Fibonacci-Sequenz

Die **Sequenz der Fibonacci-Zahlen** F_i (für $i \geq 0$) ist wie folgt definiert:

$$F_0 \stackrel{\text{def}}{=} 0$$

$$F_1 \stackrel{\text{def}}{=} 1$$

$$F_i \stackrel{\text{def}}{=} F_{i-1} + F_{i-2}, \text{ falls } i \geq 2$$

- Konstruiere Funktion `fib`, die das Array der ersten n Fibonacci-Zahlen berechnet, so dass `fib(n)[i] == Fi`:

```
int[] fib(int n)
```

-  Siehe File `fib.c0`.

3 | C0: Verträge/*Dynamic Checks* (Vorbedingungen)

Vorbedingungen (auch: *Preconditions*) stellen **Anforderungen an die Parameter** beim Aufruf einer Funktion *f*:

```

τ f(τ1 p1, ..., τn pn)
/*@ requires e1; @*/
⋮
//@ requires ek;
{ ... }

```

Assignment Project Exam Help
<https://tutorcs.com>
 Preconditions (Prädikate)
 Body von f

- Die Vorbedingungen *e_j* (Ausdrücke des Typ *bool*, ohne Seiteneffekte) beziehen sich auf die Parameter *p_i* von *f*.
- Die *e_j* (nur diese!) dürfen Ausdrücke *\length(xs)* (Typ *int*) nutzen, um die Länge eines Arrays *xs* zu überprüfen.

WeChat: cstutorcs

C0: Verträge/*Dynamic Checks* (Nachbedingungen)

Nachbedingungen (auch: *Postconditions*) sichern erwartete **Eigenschaften des Resultats** einer Funktion f zu:

```

 $\tau$   $f(\tau_1 p_1, \dots, \tau_n p_n)$ 
/*@ ensures  $e_1$ ; @*/
  ⋮
//@ ensures  $e_k$ ;
{
  ...
  return  $e$ ;
}

```

Assignment Project Exam Help
<https://tutorcs.com>
 WeChat: cstutorcs

Postconditions (Prädikate)
 Body von f

- Die Nachbedingungen e_j (Typ `bool`) beziehen sich auf
 - die Parameter p_i von f und
 - das Resultat e von f mittels Ausdruck `\result` (Typ τ).

C0: Verträge/*Dynamic Checks*

 **Dynamic checks** (Option **-d**) müssen aktiviert sein


Bei Verletzung (auch nur einer) Vor-/Nachbedingung e_j wird das laufende Programm gestoppt.

Assignment Project Exam Help

- **Vorbedingungen** (**requires** e_j):

- Check direkt vor Eintritt in den Body von f .
- Erfüllung ist Aufgabe des Aufrufers von f .
- Body von f darf unter Annahmen e_j gebaut werden. 

- **Nachbedingungen** (**ensures** e_j):

- Check direkt vor Ausführung von **return**.
- Erfüllung ist Aufgabe des Bodys von f selbst.
- Aufrufer darf sich auf Eigenschaften e_j des Resultats von f verlassen. 

4 | C0: Iteration (**for**-Loop)

<code>init;</code>	← Deklaration/Initialisierung (Counter)
<code>while (e) {</code>	← Block <i>block</i> [nochmals] ausführen?
<code>block</code>	
<code>update;</code>	← Update (Counter)
<code>}</code>	

lässt sich äquivalent durch einen **for-Loop** ausdrücken:

<code>for (init; e; update) {</code>	← Initialisierung/Check/Update des Counters
<code>block</code>	
<code>}</code>	

- *init* und *update* sind optional: **for**(; *e*;) {...}.
- Wenn in *init* eine Variable deklariert wird, umfasst ihr Scope *e*, *block* und *update*.

5 | C0: Zuweisung + Operator = Zuweisungs-Operator

Sei lv ein $lvalue$ ¹, $\oplus \in \{+, -, *, /, \%, \&, |, ^, <<, >>\}$ ein binärer Operator. Dann existiert folgende Abkürzung:

$$lv = lv \oplus e; \quad \equiv \quad lv \oplus = e;$$

Assignment Project Exam Help
Zuweisungs-Operator ⑤

<https://tutorcs.com>

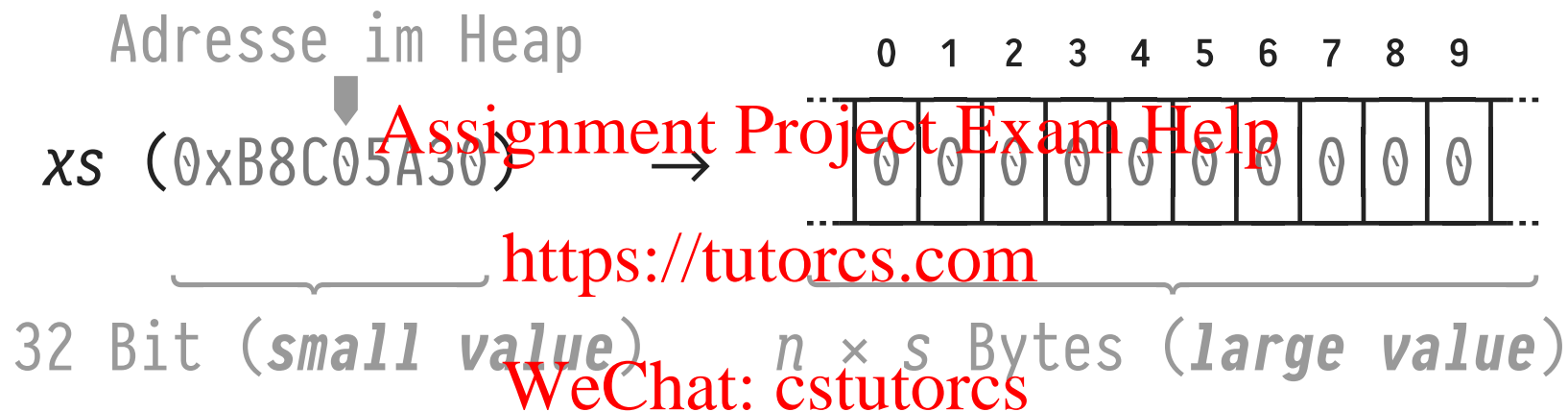
- Weiterhin: $lv = lv + 1; \equiv lv += 1; \equiv lv++;$ (– analog).
- Idiomatischer Einsatz: $\text{for (int } i = a; i \leq w; i++) \{ \dots \}.$

 Im $lvalue$ v können sich Seiteneffekte “verstecken”. 
Nach Abkürzung werden diese *nur noch einmal* ausgelöst.

¹ Ein *left value* ist ein C0-Konstrukt, auf das zugewiesen werden kann — beispielsweise eine Variable v oder ein Array-Element $xs[i]$ — und damit *links* von einer Zuweisung ($=$) stehen darf.

6 : C0 (Arrays): Variablen des Typs $\tau[]$ sind Referenzen

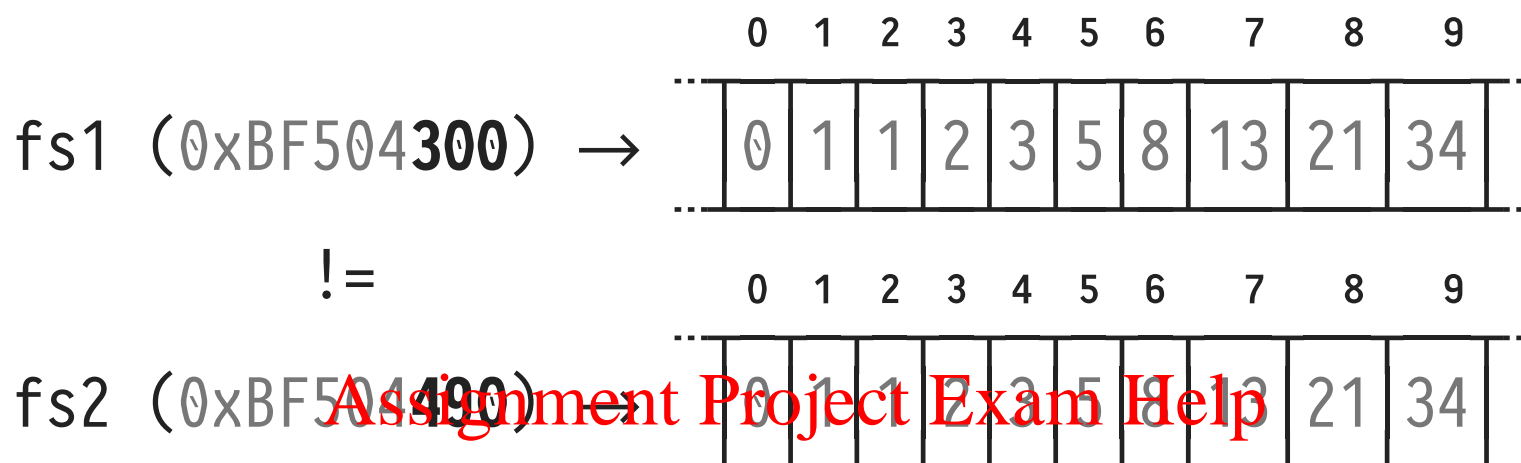
Slide 04: Variablen xs des Typs $\tau[]$ speichern **Referenzen** auf das eigentliche Array (*nicht* das Array selbst):



- C0 unterscheidet **small values** (`int`, `bool`, `char`: Größe ≤ 32 Bit) und **large values** (`string`, $\tau[]$: Größe flexibel).
- Das hat Konsequenzen für die Operationen Gleichheit (`==`) und Zuweisung (`=`) auf Arrays.

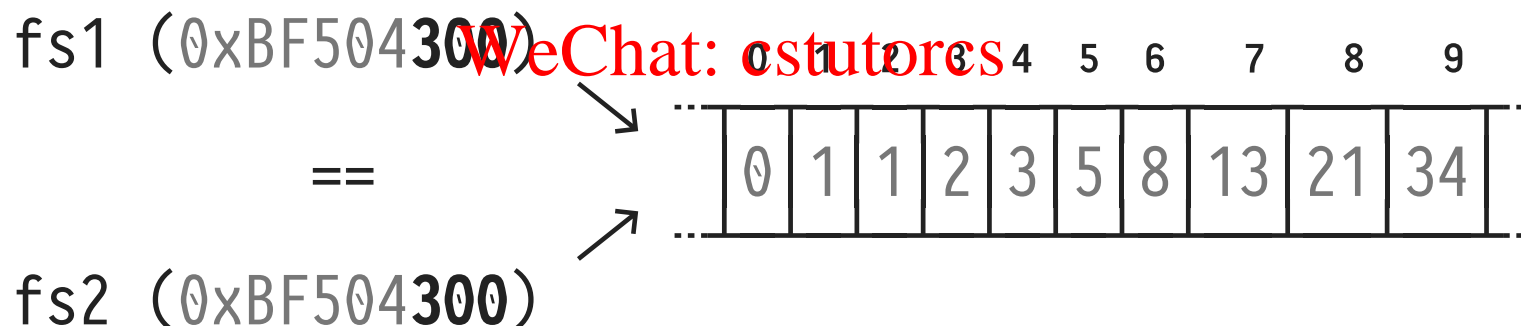
C0 (Arrays): Variablen des Typs $\tau[]$ sind Referenzen


1



<https://tutorcs.com>

2



In C0 agieren die Operatoren `==`, `!=` und `=` ausschliesslich auf *small values* ( `alias.c0`).

7 | Beispiel: CO-Bibliothek `img`

Die **CO-Bibliothek** (auch: *Library*) `img` ermöglicht die Manipulation der Pixel-Daten von PNG-Bildern:²

1. Neuer Typ `image_t` zur Repräsentation der Pixel-Daten.
2. Operationen auf diesen Pixel-Daten:

<https://tutorcs.com>
`image_t image_load(string);` // PNG aus File laden

[WeChat: cstutorcs](https://tutorcs.com)
`int image_width(image_t);` // Breite/Höhe (*w/h*) des
`int image_height(image_t);` // Bildes in Pixeln

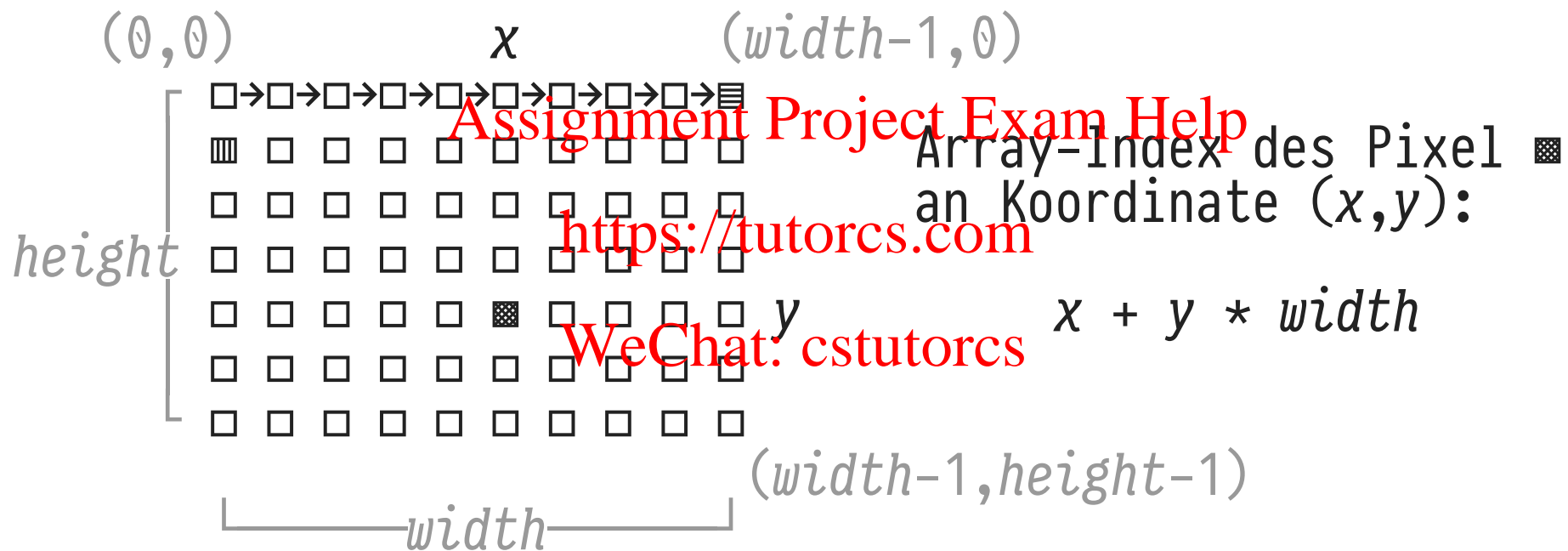
`int[] image_data(image_t);` // Array aller Pixel

`image_t image_create(int, int);` // Neues Bild, *w×h* Pixel
`void image_save(image_t, string);` // PNG in File speichern

² *Portable Network Graphics*, ein Standard für die verlustfreie und portable Speicherung von Pixel-Bildern (<http://www.libpng.org/pub/png/spec/1.2/PNG-Contents.html>).

CO-Bibliothek `img`

Funktion `image_data` liefert ein Pixel-Array in *row-major order* (siehe →). Array-Indizes der einzelnen Pixel:



- Zwei Pixel $\blacksquare (width-1, y)$ und $\blacksquare (0, y+1)$ sind im Array benachbart.

Codierung von **ARGB**-Pixeln (mit Transparenz)

32 Bits codieren die Transparenz (α), Rot-, Grün- und Blau-Anteile (r/g/b) der Farbe jedes Pixels:



- Der α -Wert (8 Bit) bestimmt die Transparenz des Pixels (0 \equiv vollkommen transparent, $2^8-1 \equiv$ voll sichtbar).

Einschub: Hexadezimal-Notation

Lange Bitsequenzen lassen sich kompakt im **hexadezimalen** Zahlensystem (16 Ziffern $0, \dots, 9, A, \dots, F$) notieren:

x_{16}	x_2	x_{10}
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

- Hexadezimale Literale des Typs `int` in C0 (der Präfix `0x` verhindert Mehrdeutigkeiten wie `13`):

$0x\langle hex\ digits \rangle$	$\langle hex\ digits \rangle \in \{0\dots 9, A\dots F, a\dots f\}^+$
---------------------------------	--

8 | Beispiel: *Bluescreen* (Filmtechnik)

Die **Bluescreen**-Technik³ ersetzt alle bläulichen Pixel eines Bildes durch Pixel eines neuen Hintergrundes (  .

- Funktion **blueish**: Ist Pixel **pixel** bläulich?

Assignment Project Exam Help

```
/* Is pixel blueish (does blue dominate red/green)? */
bool blueish(int pixel) {
    int red    = (pixel & 0x00FF0000) >> 16;
    int green  = (pixel & 0x0000FF00) >> 8;
    int blue   = (pixel & 0x000000FF);

    /* pixel is blueish if its blue component dominates
       the average of all colors */
    return blue > ((red + green + blue) / 3);
}
```

³ <https://de.wikipedia.org/wiki/Bluescreen-Technik>

9 | C0: Bedingter Ausdruck (*Conditional Expression*)

Sei p ein Prädikat und e_1, e_2 Ausdrücke des gleichen Typs τ .
 Der **bedingte Ausdruck** \textcircled{E} (auch: *conditional expression*)

$$p ? e_1 : e_2$$

Assignment Project Exam Help

hat den Typ τ und den Wert $\begin{cases} e_1, & \text{falls } p == \text{true} \\ e_2, & \text{sonst.} \end{cases}$

<https://tutorcs.com>
 WeChat: cstutorcs

- $p ? e_1 : e_2$ entspricht direkt Schemes $(\text{if } p \ e_1 \ e_2)$. Die (illegale!) Form $p ? e_1$ wäre semantisch fragwürdig.
- Mixfix-Operator $\square ? \square : \square$ assoziiert nach rechts. Damit gilt $p_1 ? e_1 : p_2 ? e_2 : e_3 \equiv p_1 ? e_1 : (p_2 ? e_2 : e_3)$ [vgl. das Scheme-Idiom $(\text{if } p_1 \ e_1 \ (\text{if } p_2 \ e_2 \ e_3))$].

C0: Update der Operator-Tabelle

Priorität	Operatoren	Assoziativität	
13	\neg , \sim	rechts	Negation/Invertierung
	$\square++$, $\square--$	—	Inkrement/Dekrement
12	$*$, $/$, $\%$	links	
11	$+$, $-$	links	
10	$<<$, $>>$	links	
9	$<$, $<=$, $>$, $>=$	links	
8	$==$, $!=$	links	
7	$\&$	links	
6	\wedge	links	
5	$ $	links	
2	$\square ? \square : \square$	rechts	bedingter Ausdruck
1	$=$, $+=$, $-=$, $*=$, $/=$	—	Zuweisung und
	$\%=$, $\&=$, $\wedge=$, $<<=$, $>>=$	—	Zuweisungs-Operatoren

C0-Operator-Tabelle (Ausschnitt)

- \square steht für die Platzierung der Argumente (die meisten Operatoren sind binäre Infix-Operatoren: $\square * \square$).