int (*arr)[8] – Pointer to an Int-Array
int* arr[8] – Array of Int*

va_start(ptr, länge): initialisiert varargs ptr
va_arg(ptr, größe): Liest Argument und erhoht va_list ptr um 1.

arr[**outer**][**inner**]
arr[**1**][**2**] = *(arr[**1**] + **2**) = *(*(arr + **1**) + **2**)) = (*(arr + **1**))[**2**] *(Achtung Klammern!)*
Array-Initialisierung padded immer mit 0en!

sizeof (***pointer***) = systemarchitektur größe
sizeof (**array**) = anzahl_elemente * sizeof (elementtyp)
sizeof (**char**) = 1
sizeof (**short**) = 2
sizeof (**int**) = 4
sizeof (**long**) = 4 / 8
sizeof (**long long**) = 8
sizeof (**float**) = 4
sizeof (**double**) = 8

Zugriff auf das erste Element

Zeiger-Variante	Array-Variante
`*ptr`	`ptr[0]`
`*array`	`array[0]`

Zugriff auf das n-te Element

Zeiger-Variante	Array-Variante
`*(ptr+n)`	`ptr[n]`
`*(array+n)`	`array[n]`

Zugriff auf die Anfangsadresse

Ohne Adressoperator	Mit Adressoperator
`ptr`	`&ptr[0]`
`array`	`&array[0]`

Zugriff auf die Speicheradresse des n-ten Elements

Ohne Adressoperator	Mit Adressoperator
`ptr+n`	`&ptr[n]`
`array+n`	`&array[n]`

Operator (höchste bis niedrigste Priorität)	Assoziativität
`(), [], {}, ->, ., ++, --`	links
`!, ~, ++, --, +, -, *, &, sizeof`	rechts
`(type)`	rechts
`*, /, %`	links
`+, -`	links
`<<, >>`	links
`<, <=, >, >=`	links
`==, !=`	links
`&`	links
`^`	links
`|`	links
`&&`	links
`||`	links
`?:`	rechts
`=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=`	rechts
`,`	links

Strukturen
Gesamtgröße ergibt sich durch Addition der einzelnen Komponenten. Sollte die Größe kleiner als 4 Bytes sein, dann soll man padden und auf 4 Bytes aufrunden.
typedef ist ähnlich wie *data* in Haskell.

Zeiger auf Strukturen:
`struct person *p1; Zugriff: (*p1).name oder p1 -> name`

Unions
Unions werden bei kleinen Datentypen set **nicht** mit 0 gepadded!

Speicherklassen

Codesegment:
    Maschinencode des Programms
Datensegment
    Globale (externe) Variablen
Stack
    Lokale Variablen
    Rücksprungadresse
    Parameter einer Funktion
Heap
    Dynamische Variablen

Speicherverwaltung
malloc wird immer auf void gecastet
lieber die Antwort wo vor malloc gecasted wird!

Makros
#v → mit "" einsetzen ##v → as - is einsetzen
__LINE__ **__FILE__** **__func__** *(Kein Makro)*

va_list	*va_list argPtr;*	Abstrakter Datentyp (wird auch als Argumentzeiger bezeichnet), mit dem die Liste der Parameter definiert wird und mit dem der Zugriff auf die optionalen Argumente realisiert wird.
va_start	**void va_start(va_list argPtr, lastarg);**	Argumentliste initialisiert den Argumentzeiger **argPtr** mit der Position des ersten optionalen Arguments. An **lastarg** muss der letzte fixe Parameter in der Liste übergeben werden.
va_arg	**type va_arg(va_list argPtr, type);**	Gibt das optionale Argument zurück, auf das **argPtr** im Augenblick verweist, und setzt den Argumentzeiger auf das nächste Argument. Mit **type** gibt man den Typ des zu lesenden Arguments an.
va_end	**void va_end(va_list argPtr);**	Hiermit kann man den Argumentzeiger **argPtr** beenden, wenn man diesen nicht mehr benötigt.