

Kommerzielles Programmieren in C

ar	ruv dv xv	replace, update, verbose(=anzeigen) delete, verbose eXtract, verbose
Call by reference		mache(&a); int mache(int *pa); verändert a!
Call by value		mache(a); int mache(int aa); a bleibt unverändert!
cb		cb programm.c > schoen.c
Datei	Typdekl. Öffnen Schreiben Lesen Schließen Dateiende Binär Ende Anfang Lesen	FILE *dat; (systemabhängiger Typ) dat = fopen("name.dat", "r"); mit read, write, append, binary fprintf(dat, "%c ...", ...); wie printf fscanf(dat, "%...", ...); fclose(dat); feof(dat) != 0 Dateiende erreicht fseek(dat, 0, SEEK_END); anstatt 0: -i * sizeof(typ) fseek(dat, i * sizeof(typ), SEEK_SET); fread(buffer, sizeof(typ), anzahl, dat); buffer: (void *) malloc
lint		lint -Aa programm.c
make	Makefile	#Anmerkung a.out: prog.o libname.a cc prog.o -L. -lname libname.a: quelle1.o quelle2.o ar ruv libname.a quelle1.o quelle2.o prog.o: prog.c prog.h cc -c prog.c clean: rm *.o *.lst core
malloc, free	<stdlib.h>	p=malloc(sizeof(typ)); stellt Speicher zur Verfügung free(p); gibt Speicher frei
Operatoren	15. Stufe → 14. Stufe ← 13. Stufe → 12. Stufe → 11. Stufe → 10. Stufe → 9. Stufe → 8. Stufe → 7. Stufe → 6. Stufe → 5. Stufe → 4. Stufe → 3. Stufe ← 2. Stufe ← 1. Stufe →	(), [], ->, . !, ~ (not), ++, -- (in-/decrement), * (ptr), & (Adresse), (Typ) *(mal), /(div), %(mod) +, - (Addition) <<, >> (Bitverschiebung) <, <=, =, > (Vergleiche) ==, != (Vergleiche) & (binary and) ^ (binary excl. or) (binary or) && (and) (or) a?b:c ⇔ if (a) {b;}else {c;} =, +=, -=, *=, %=, ^=, =, &=, >>=, <<= (Zuweisung) , (Variablenliste)
Präprozessor	#...	#define ; Vorsicht: direkte Ersetzung, Klammern #ifdef / #ifndef NAME #if ... #elif ... #endif
RCS		ci prog.c co -l/-u prog.c (Lock, Unlock)

		<pre>rcc -u/-l/-o prog.c (Lock, Unlock, Obsolete(=löschen)) rlog prog.c (Änderungsgeschichte) rsdiff prog.c (Differenz Ausgeben) \$Author\$, \$Id\$, \$Date\$, \$Revision\$ ident a.out (gibt \$Id\$ aus)</pre>
Strings	<string.h>	<p><u>Länge</u>: strlen(char *s) - liefert Länge ohne "\0"</p> <p><u>Vergleich</u>: strcmp(char *s1, *s2) liefert $\begin{cases} 0 \text{ für } s1 = s2 \\ -1 \text{ für } s1 < s2 \\ 1 \text{ für } s1 > s2 \end{cases}$</p> <p><u>Kopieren</u>: strcpy(char *s1, char *s2)</p>
Strukturen		<pre>typedef struct knot { int info; struct knot *next; }Knoten; Schachtelung möglich</pre>
Umsetzungs- zeichen	int char, string float, ... sonstiges	<pre>%d, %i, %o oktal, %x hex %c, %s %f, %e (Mantisse) %[\n] liest bis zum <return> ein (z.B. für Strings) %-9f für linksbündige Ausgabe</pre>
Union		wie struct, aber Komponenten nur wahlweise verwendbar
Variable	int char float,double (normal) static register	<pre>[-32768, 32768]; 032: oktal; 0x1af: hexadezimal [-128 ... 127] unsigned: [0, 255] mind. [-1e37,-1e-37] und [1e-37,1e37] interne Var. verdeckt gleiche interne Variable in Blöcken: bleibt erhalten außerhalb: nur in dieser Datei bekannt schneller Speicher (wenn möglich), & Operator nicht zulässig</pre>
variable Arg.- zahl	<stdarg.h>	<pre>int summe(int n, ...) { va_list a; /* Typedeclaration */ int sum = 0; va_start(a,n); /* Initialisierung */ while (n--) {sum += va_arg(a, int); } /* Elemente Einlesen */ va_end(a); /* Makro Beenden */ return sum; } </pre>
Vektoren	mehrdim.	row major, 1. Index langsam (Zeile), 2. schnell (Spalte)
XDB	xdb a.out	<pre>l: list, t: trace, v 10: gehezu 10. Zeile, v main: gehezu main breakpoints: b 10: erstellen, lb: list, db: delete c continue, s step, s 5 step * 5, r run, q quit td: toggle display, down/up: Schritt zurück/vor p (3*i+b): berechnen des Ausdrucks, assertion: a if (z==20){p z}{x}</pre>
Zeiger	Anweisung Typen Funktionen	<pre>p++; erhöht Adresse um sizeof (Typ) p+= i; erhöht Adresse um i* sizeof (Typ); i ist int int *p[3]; Vektor von Zeigern auf int int (*p)[3]; Zeiger auf int-Vektor int *(*p)[3][2]; Zeiger von 3*2 Vektor von Ptr. auf int const int *p=&ci; Zeiger auf konst. int int *const cp=&i; konst. Zeiger auf int. const int *const cp=&ci; konst. Zeiger auf konst. int void *p; Zeiger auf void, Typumwandlung nötig, +, -, ++, ... verboten int (*f) (int i); Zeiger auf Fkt. mit int Arg, die int liefert int *f (int i); Fkt. mit int Arg, die Zeiger auf int liefert</pre>