ILP 32 | Sring-Methoden | void strcpy(char\* s1, const char\* s2) { | char\* strcat(char\* s1, const char\* s2){ | Datentypen

int=4 | | int i = 0; | size\_t i, j; |

long=4 | size\_t strlen (const char\* s){ | while ((s1[i] = s2[i]) != '\0') i++; | for(i = 0; s1[i] != '\0'; i++); | int a; a = 1; d.Zimmer = 109; struct res

float=4 | size\_t i; | } | for(j = 0; s2[j] != '\0'; j++) | int \*b; b = &a; d.Gebaeude = 'c'; {

double=8 | for(i = 0; s[i] != '\0'; i++) {} | | s1[i+j] = s2[j]; | res d; d.Art = „Hoersaal“; int Zimmer;

char=1 | return i; |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | s1[i+j] = '\0'; | res \*e; char Gebaeude;

char\*=4 | } | strcmp(const char\* s1; const char\* s2){ | return s1; | float c[2] = {2.5, 3.6} e = (res\*) malloc (sizeof(res)) char\* Art;

LP64 | | while(\*s1 != '\0' && (\*s1==\*s2)) | } | int \*\*f; f = &b; e → Zimmer = 149; };

long=8 | | s1++ , s2++; | | e → Gebaeude = 'G' typedef struct res res;

char\*=8 | | return \*(const unsigned char\*)s1 - \*(const unsigned char\*)s2; | e → Art = „Buero“;

| | } | |

Übersetzungseinheiten | /\*...Wuerfel.c...\*\ | /\*...Quadrat.c...\*\

/\*.....C-Main hat keine Header...\*\ | #include <stdlib.h> | double flaeche(const double seintenlaenge)

WuerfelTest.c | #include „Quadrat.h“ | {

| double oberflaeche(const double kantenlaenge) | return zumquadrat(seitenlaenge);

#include <stdio.h> | { | {

#include <stdlib.h> | return flaeche(kantenlaenge) \* 6); | double zumquadrat(const double d)

#include „Wuerfel.h“ | } | {

#include „Quadrat.h“ | double volumen(const double kantenlaenge) | return d \* d;

int main(int argc, char \*argv[]) | { | }

{ | return flaeche(kantenlaenge) \* kantenlaenge; |

double k; | } | /\*...Quadrat.h...\*\

double c; | | #ifndef QUADRAT\_H

double f; | /\*...Wuerfel.h...\*\ | #define QUADRAT\_H

/\* double\* a; \*/ | /\*zugehörige Header\*\ | double flaeche(const double seintenlaenge);

k = atof(argv[1]); /\*parse String->double\*/ | #ifndef WUERFEL\_H | double zumquadrat(const double d);

/\* a = (double\*) malloc(sizeof(double)); | #define WUERFEL\_H | #endif

if ( a == NULL) | double oberflaeche(const double kantenlaenge); |

{ | double volumen(const double kantenlaenge); |

printf(„Speicherreservierung | #endif |

fehlgeschlagen!\n); |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

return 1; | argv → FeldvonStringpointern | #define → Name(Para.Liste) Ausdruck

}; \*/ | argc+1 → Feldgröße | bsp: #define max(a, b) ((a)>(b) ? (a) : (b))

f = oberflaeche(k) | argv[0] → Programmname |

c = volumen(k); | argv[1] → erstes Kommandozeilenargument ddd → Obduzieren von Datenleichen

printf(„Kantenlaenge %f, Oberflaeche %f, | argv[argc-1] → letztes Kommandozeilenargument Schrittweises Durchlaufen des

Volumen %f\n“, k,f,c); | argv[argc] → NullPointer Programms möglich

free(a); |

return 0; | Valgrind eigenet sich, um Speicherzugriffsfehler zu ermitteln bzw.

} | alles was mit Speicherverwaltung zu tun hat

| Dynamisch: |

Shell-Skripts | #!/bin/sh | bsp statisch:

Statisch: | gcc -c -I. Irgendwas.c -fPIC“ | #!/bin/sh

#!/bin/sh | “ “ | gcc -c -I. Liste.c

/\*..einzelne Übersetzung..\*\ | gcc -shared irgendwas.o “ “ -o Ziel.so | gcc -c -I. Fachnote.c

gcc -c -I. Irgenwas.c | gcc sieheLetzerBeiStatisch | gcc -c -I. Notenspiegel.c

“ “ “ “ “ “ “ “ “ “ “ “ | | ar rs libaufgabe5.a liste.o fachnote.o

| | gcc notenspiegel.o -L. -laufgabe5 -o notenspiegel

ar rs ziel.a irgendwas.o “ “ “ | |

gcc ausführbareDatei.o -L. -lZiel(ohneLib) -o ausführbareDatei |Quadrat.c

Makefiles:

# Makefile

# Kommando-Variablen | Malloc → reserviert Speicher für Wert eines Typen

CC = gcc → | | Typ \***Zeigername = (Typ\***) malloc(sizeof(Typ))

CFLAGS = -W -Wall -ansi -pedantic → | Variablen für die vordefinierten C-Übersetzungsregeln | Calloc → reserviert Speicher für Feld von Wert eines Typen

CPPFLAGS = -I. → Include-Dateien im aktuellem Verzeichnis suchen | Typ \***Zeigername = (Typ\***) calloc(Feldgröße, sizeof(Typ))

RM = rm -f \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#Hilfsvariablen

TARGET = hallo

OBJECTS = gruss.o

SOURCES = $(TARGET).c $(OBJECTS:.o=.c) → .o=.c o Endung o wird durch Endung c ersetzt

HEADERS = $(OBJECTS: .o=.h)

#Musterregel

%.o: %.c → Datei mit c wird in Datei mit o-Endung erstellt, hier kann auch z.B %.pdf: %.tex sein

$(CC) $(CPPFLAGS) $(CFLAGS) -c $< $@ → &@ Ziel auf das die Regel angewendet wird, %< die Erste Quelle zum aktuellem Ziel, $^ alle Quellen zum Ziel

#Standardziele | Arten von Laufzeitfehlern:

.Phony: all clean | Absturz

all: $(TARGET) | → unerwartetes Ende, z.B. Speicherzugriffsfehler → ddd und valgrind

clean: | Endlosschleife

$(RM) $(TARGET) $(TARGET).o $(OBJECTS) depend | → Programm scheint zu hängen, aber läuft und läuft → ddd über Coredateierstellung

depend: $(SOURCES) $(HEADERS) | Speicherüberlauf

$(CC) $(CPPFALGS) -MM $(SOURCES) > $@ | → der ganze Rechner wird langsamer, weil das Programm sämtlichen Speicher belegt hat

| → valgrind, damit prüfen, ob nach einer Speicherallokierung die Freigabe vergessen wurde

#Ziel zur Programmerstellung | Fehlverhalten

$(TARGET) : $(TARGET).o $(OBJECTS) | → das Programm macht nicht was es soll → ddd, über Breakpoints Fehlerverhalten ermitteln

$(CC) $(CPPFLAGS) $(CPFLAGS) $^ -o $@# |

|

#Abhängigkeiten |

include depend |

Parameter für gcc:

-o | bestimmt den namen der ausgabedatei(Standard: „a.out“)

-c | der quellcode wird nur kompiliert, aber noch nicht gelinkt(neu Endung oft „.o“

-g | fügt debugging-symbole in die binärdatei ein

-pedantic | weist auf nicht ISO konforme mit einer Warnung hin

-Wall | aktiviert sinnvolle warnungen, die vom gcc-compiler unterstützt werden

-ansi | erlaubt nur ANSI C-konforme konstrukte

-L | verzeichnisauswahl des Linkers(suchpfad für bibliothek)

-l | bibliothek die eingebunden werden soll

-I | verzeichnissauswahl der preprozessor(suchpfad für header)

Empfohlene option : -W -Wall -ansi -pedantic

Strings:  
const char \*s = „Halli“;

const char \*t = „Hallo“;

char \*st = NULL;

st = (char\*) malloc (strlen(s) + strlen(t));

st = strcat(strcpy(st, s), t);

free(st);

Makefile zu Wuerfel

#Makefile

#Hilfsvariablen

CC = gcc

CPPFLAGS = -I.

RM = -rm -f

TARGET = WuerfelTest

OBJECTS = Wuerfel.o Quadrat.o $(TARGET).o

SOURCE = $(OBJECTS:.o=.c)

HEADERS = Wuerfel.h Quadrat.h

.Phony: all clean

all: $(TARGET)

clean:

$(RM) $(TARGET) $(OBJECTS)

%.o:.c

$(CC) $(CPPFLAGS) -c $< $@

$(TARGET): $(OBJECTS)

$(CC) $(OBJECTS) -o $(TARGET)

Unterschiede: JAVA, C, C++

|  |  |  |  |
| --- | --- | --- | --- |
| FEATURES  POSIX (Portal operating System Interface):  Ist ein Standard für die Programmierschnittstelle von Betriebssystemen.  Bei Fehlermeldungen durch das Betriebssystem werden mittels POSIX Fehlernummern(errno) zurückgegeben(Wenn die errno != 0, dann liegt ein Fehler vor). Mittels If-Anweisung Fragen wir den möglichen Fehler ab und geben die errno zurück. | C | C++ | Java |
| Paradigmen | Prozedural | Prozedural/OOP | OOP |
| Produkt | Ausführbarer Masschienencode | Ausführbarer Masschienencode | Java Bytecode |
| Speicherverwaltung | Manuell | Manuell | Garbagecollector |
| Pointer | Ja | Ja | Nein |
| Stringtype | Char Array | CharArray/Objekt | Objekt |
| Datentypen | Struct, union | Struct, union, class | Class |
| Vererbung | - | Mehrfachvererbung | Einfachvererbung |
| Generische Datentypen | - | Ja | Ja |
| Ausnahmebehandlung | - | Ja | Ja |

#include <stdlib.h>

#include <stdio.h>

int main(int argc, char \*argv[])

{

int i;

int sum;

for(i = 1; i < argc; i++)

{

sum += atoi(argv[i]);

}

printf("Summe = %d\n", sum);

return 0;

}

Kleines Programm mit mehreren Kommandozeilenargumenten