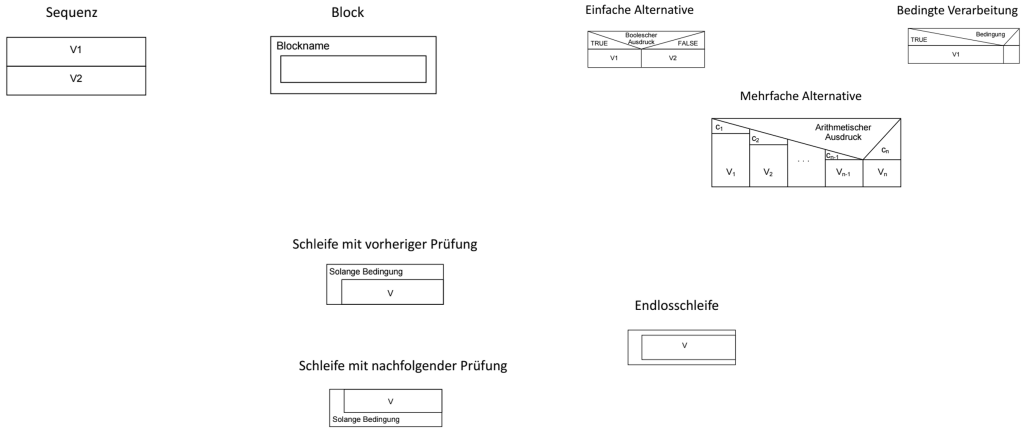
1 Wichtige Befehle

Volatile:
Die Variabel wird durch den Compiler nicht weg optimiert. Die Variable wird nicht in das Prozessregister gelegt.
#define:
Die Variablen welche durch #define definiert wurden werden vom Preprozessor durch ihren vorgegebenen Wert ersetzt. (Der Compiler sieht die #define Varialben nie)
Typedef:
Reine Textersetzung. (Z.B Überall wo ein typendefiniertes Struct aufgerufen wird wird der Code der Struct durch den Compiler ergänzt)
Continue:
In den nächsten Schleifendurchgang springen.
#include
$< Name >$ → Name wird im definierten Include-Verzeichniss gesucht
"Name" → Wird im aktuellen Verzeichniss gesucht
printf → S.465

2 Sonstiges

C ist casesensitive Funktionsprototyp → Deklaration einer Funktion
L- und R- Werte → lvalue braucht immer Schreibzugriff und bei rvalue reicht Lesezugriff
Globale Variablen werden **automatisch auf 0 initialisiert** (wie globale Arrays auch)
Lokale Variablen werden beim **Aufruf des Blockes** angelegt.
Funktionsprototyp → Deklaration einer Funktion.
Stub → Entwurf eines Programm (beinhaltet alle Elemente damit das Programm ausgeführt werden kann)
Bedingungsoperator → A?B:C (if(A) then(B) else(C))
Sequenzpunkte Bestimmter Punkt an dem alle bisherigen Nebeneffekte abgearbeitet wurden. (Ende einer if Bedingung / Zwischen ||&& / Semikolon)
const char* text → text ist ein Pointer auf ein konstanten Char
char* const text → Der Pointer Text ist konstant
Interruptvektortabelle → Tabelle von Funktionspointern
#-Operator → Konvertiert Argument in String
##-Operator → Zeichenfolge links und echts des Operators wird zusammengezogen

3 Nassi-Shneiderman-Diagramme



4 Gültigkeit, Sichtbarkeit und Lebensdauer

Variablen sind immer so lokal wie möglich zu definieren
Gültigkeit:

Eine Variabel ist dann gültig wenn sie an der genannten Stelle dem **Compiler bekannt** ist und **nicht durch eine andere Variable verdeckt** wird.
Sichtbarkeit:
Wird eine lokale Variable mit dem selben Namen wie eine globale Variable erstellt ist nur die lokale sichtbar.
Lebensdauer:
Zeit in welcher der Compiler der Variable ein Speicherplatz zu Verfügung stellt.

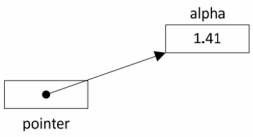
5 Arbeiten mit char-Variablen

Falls zu wenig Speicher auf einem Gerät vorhanden ist kann es vorkommen, dass das zusammenkopieren von verschiedenen Char - Arrays selber programmiert werden muss. Dies kann mit While, for Schleifen am einfachsten realisiert werden.
Falls genügend Speicherplatz vorhanden:
Library $< string.h > \rightarrow strCopy(char * dest, constchar * src)$;
Fügt den String src ans Ende des Strings dest an. Ist allerdings dest nicht genügend gross kann dies zu Problemen führen daher wird **strnCpy() bevorzugt**
Sonst:
while(dest[i] = src[i]) **Weisst zu und ist true bis src[i] = Backslash 0**
i++;
Funktionen von $< string.h >$ die mit **str** beginnen **erkennen das Backslash 0**.Funktionen die mit **mem** beginnen **nicht**.
Wenn **strn** verwendet wird dann kann es keinen Buffer - Overflow geben.

6 Pointer Basics S.163 Advanced S.320

Variable in welcher eine Adresse von einer im Speicher befindlichen Variable oder Funktion gespeichert ist.

```
float alpha;
float* pointer;
alpha = 1.41f;
pointer = &alpha;
```



Der **Speicherbedarf eines Pointers** ist **unabhängig vom Typ**. So gross wie die maximale Adresse.
Falls man ein Pointer initialisieren will muss man diesen zu einem **Nullpointer** machen. (Zeigt nicht auf Adresse 0)
Der Adressoperator **&** (Referenzierungsoperator) liefert die **Adresse einer Variable**.
Der Inhaltsoperator ***** (Dereferenzierungsoperator liefert den Wert einer Speicherzelle.

6.1 Array-Pointers

Der Name eines Arrays kann als konstante Adresse des ersten Elementes (Index 0) des Arrays betrachtet werden.
Die Befehle $\&array[0]$ bedeutet das gleiche wie array

6.2 Pointer auf Funktionen

Jede Funktion befindet sich an einer definierten Adresse im Codespeicher.
Ein Funktionspointer wird wie folgt initialisert:
Rückgabetyp der Funktion (*pointer)(Datentyp der Parameter);
pointer = functionsname;
ausgabewert = pointer(parameter);

7 Arrays

sizeof(arr)/sizeof(arr[0]) gibt die Anzahl der Elemente zurück.

Beim Befehl **pointer + n** bewegt sich der Pointer um n * sizeof(Typ) Bytes.

8 Lexikalische Konventionen

Namen dürfen aus Buchstaben, Ziffern und Underscores bestehen, allerdings darf das erste Zeichen keine Ziffer sein.

8.1 Enumerations / Aufzählungstyp

Anonyme Enumerations können dazu verwendet werden ganzzahlige symbolische Konstanten zu definieren. (Falls nicht ganzzahlig werden const gebraucht).

8.2 Auswertungsreihenfolge

***p++** → p wird dereferenziert und danach wird der Pointer inkrementiert
***++p** → p wird zuerst inkrementiert und danach dereferenziert
(*p)++→ Zuerst wird *p dereferenziert und der Wert danach um 1 erhöht.
int* alpha[8] → Array von 8 int-Pointern
int (*alpha)[8] → Pointer auf ein Array mit 8 int Werten.
int * (*alpha)[8] → Pointer auf ein Array aus 8 int Pointern

9 Bit-Operationen

Kann nur mit unsigned Typen durchgeführt werden.
$A\&B$ → Bitweise AND
$A|B$ → Bitweise OR
$\sim A$ → Bitweise NOT
$A \hat{} B$ → Bitweise XOR
$A >> n$ → Rechts-Shift um n Bits
$A << n$ → Links-Shift um n Bits

10 Structs

Structs können **zugewiesen** Werden (a = b)
Grösse ist nur **mit dem sizeof** Operator zu bestimmen. (Nicht zusammenzählen der einzelnen Variablen). Variablen fangen immer im nächsten Byte an dies wird **Alignment** genannt.
Die Adresse einer Strukturvariablen kann mit dem Adressoperator ermittelt werden.

10.1 Zugreifen auf Strukturvariable

Auf Feld von einer Strukturvariablen:
(Name des Structs).(Name des Feldes)
Zugriff über ein Pointer auf eine Strukturvariable:
(Name des Pointers)->(Name des Feldes)
Es ist meistens effizienter einen Pointer auf eine Struct an eine Funktion zu übergeben als der Struct an sich da der Kopieraufwand grösser ist.

10.2 Unions

Unions sind ähnlich wie Struts allerdings ist immer nur eine Variable / Feld aktiv, dies führt dazu das man weniger Speicher braucht. Die grösse einer Union wird immer durch die grösste Variabel bestummen.
Da sich die Variablen den Speicherplatz teilen beeinflussen sie den Inhalt von sich gegenseitig.

11 Iteration und Rekurstion

11.1 Rekursion

Funktion enthält Abschnitt welcher sich **selbst aufruft**. Alle noch auszuführende Befehle werden im Stack gespeichert. (Kann zu einem Stack-Overflow führen)

11.2 Iteration

Funktion enthält Abschnitt welcher mehrfach durchlaufen wird. **Jede Funktion** ist **iterativ** definierbar. In der Praxis wird immer die Iterative Form gebraucht.

12 Speicherklassen

Vier verschiedene Speicherbereiche

- Code
  Zugriff: lesen, ausführen (im Flash EPROM oder im RAM)
  Programm im Maschinencode und eventuell Konstanten (meist grosse Konstante Arrays)
- Datensegment / Stack / Heap
  Zugriff: lesen und schreiben (im RAM (evtl. Register))

  Datensegment
  Globale Variablen, static Variablen

  Stack
  Lokale Variablen, Parameter einer Funktion, Rücksprungadressen

  Heap
  Dynamische Variablen (Speicherplatz erst zu Laufzeit gefordert)

13 Arbeiten in grossen Projekten

Gültigkeit von Variablen o.Ä möglichst klein halten (Information Hiding)

13.1 Programm aus mehreren Dateien

Zuerst müssen alle Sourcedateien compiliert werden und danach werden sie zusammengelinkt und eine einzige auszuführende Datei entsteht. (Wird Buildprozess genannt).
Interne Bindungen → Symbole welche nur in einer Übersetzungseinheit vorkommen
Externe Bindungen → Symbole werde in mehreren Übersetzungeinheiten gebraucht (Werde über Linker untereinander bekannt gemacht).
Speicherklasse "extern":
Eine externe Variable kann nur **in einer Datei definiert werden (ohne extern)**. In den anderen wird sie mit **extern deklariert (bekannt gemacht)**.
Der Speicherplatz wird nur einmal benötigt
Speicherklasse "static":
Der Wert einer Static Variable bleibt erhalten nachdem die Funktion beendet wurde. Nur in definierter Datei sichtbar

14 Preprocessor

Output des Preprocessors wird dem eigentlichen Compiler übergeben < # - Befehle werden durch den Preprocessor ausgeführt.Arbeitet Zeilerorientiert

15 Input / Output Kapitel 16.8

Dateienzugriff über Filepointer:
Mit einem Filepointer kann eine Datei mit dem C- Code verknüpft werden.
FILE * fp = fopen("test.dat", "rw")
Es gibt drei vordefinierte Standardkanäle (Filepointer):

- Standardeingabe **stdin**
  Tastatur
- Standardausgabe **stdout**
  Konsole
- Standardfehlerausgabe **stderr**
  Oft die Konsole

15.1 Umlenkung

Mit einer Umlenkung kann die Standardaus- und eingabe umgeleitet werden, zum Beispiel in ein Textfile.
Dies kann mit den Operatoren < (Eingabe) und > (Ausgabe) gemacht werden.

16 Kommandozeilenparameter

Ziel ist die Parameter für ein Programm gerade beim Aufrufen der Funktion mitzugeben.
Dafür wird **int main** angepasst und zwar wie folgt:
int main(int argc, char* argv[]); **argc** → Argumentcounter **argv** → Argumentvektor (argv[0] beinhaltet immer den Namen des Programms)