C PROGRAMMIEREN

1. Allgemeine Begriffe

Programm	Vollständige Anweisung an einen Rechner
Spezifikation	Vollständige, detaillierte und unzweideutige Lösung
Algorithmus	Detaillierte und explizite Vorschrift zur schrittweisen Lösung eines Problems (Z.B. 1. Zahl1 einlesen; 2. Zahl2 einlesen; 3. Zahl1 und Zahl2 zusammenzählen; 4. Summe ausgeben). Abfolge von Operationen.
Batchprogrammierung	Stapelverarbeitung, benötigt keine weiteren Angaben, wird nur gestartet
Dialogprogrammierung	Der User kann eingaben machen, die das Programm miteinbezieht.
Interpreter	Setzt in Maschinensprache um (auch nur Teile daraus). Es wird kein vollständiges Programm erzeugt. Arbeitet Zeile für Zeile ab. Langsamer in der Ausführung.
Compiler	Übersetzt Sourcecode in Maschinencode (Objektcode; *.obj). Erzeugt vollständiges Programm, Programm ist sofort verfügbar. Code muss vollständig vorliegen.
Java	Platformunabhängige Programmiersprache. Dank der VirtualMachine sind die Programme auf allen Platformen aufrufbar, aber etwas langsamer.
Assembler	Niedrigste Ebene Programmiersprache. Abhängig von Prozessor.
C	Hardwarenah, Portierbarkeit, schnelle Programme
Kriterien für Gutes Programm	Benutzerfreundlich, Wartbar, Wiederverwendbar (Teile aus Code), Fehlerfrei, gute Performance
Case Tools	Computer Assisted Software Engineering; Umfasst Entwicklungsumgebung (Planung, Analyse, Design, Programmierung, Einführung)
1. Generation	Maschinensprache mit 0 und 1 (Maschinenorientiert)
2. Generation	Assembler Maschinensprache nicht binär sondern mit Symbolik (Maschinenorientiert)
3. Generation	Prozedurale programmiersprachen: Höhere maschinenunabhängige Sprachen. Wendet Algorithmen an. Auch: Quellenprogramm, da Quelltext (=Objektprogramm)
4. Generation	Effizienter, Nicht mehr wie, sondern was.

Headerdatei	Kopfdatei. Wird in das Programm kopiert (#include <headerdatei.h>
Präprozessor	Nimmt temporäre Änderungen am Quellcode vor (include, entfernt Kommentare, ect.)
Laufzeitbibliothek	Sammlung von Funktionen.
#	Spricht den Präprozessor an.
main()	Hauptfunktion des Programmes
{}	Anweisungsblock, fasst Anweisungen zu einem Block zusammen.
\n	Escape Zeichen. Zeilenumbruch
;	Semikolon, ende der Anweisung (nicht bei Anweisungsblöcken)
return0;	Beendet das Programm sauber.
/* Kommentar /*	Setzt kommentare. Werden nicht mitkompilliert
Programmausführung	Erfolgt von Oben nach Unten.
16/32 Bit	Wortgrösse des Prozessors. Anz. Bits, die gleichzeitig über einen Datenbus übertragen werden können.
Speicherbereich	Adresse im Arbeitsspeicher

2. Mit Zahlen und Zeichen Arbeiten

Eine Variable existiert höchstens so lange, wie die Laufzeit eins Programmes.

Datentypen: Bei C beginnt die Variablendeklaration mit einer Angabe des Datetyps, damit der Compiler weiss, welche Art von Daten er speichert.

Regeln: Nicht mit einer Zahl beginnen, keine Sonderzeichen ausser _, keine Syntax-Wörter, Umlaute. Case sensitive!

Ganzzahl: keine Nachkommastelle (z.B. 7,8,-87, -4, ect.)

Wertbereich: Kennt nur Werte, zwischen Anfangswert und Endwert.

short	32768	+ 32767
int	2147483648	+ 2147483647
long	2147483648	+ 2147483647
Char	128	+ 127

32-Bit: kein Unterschied zwischen int und long.

16-Bit: int hat Wertbereich von short.

Kompatibilität: Wenn auf 32 und 16 Bit laufen soll, short und long verwenden

Unsigned

Hier gibt es kein Vorzeichen. Deshalb sind alle Zahlen positiv. Somit hat man die den doppelten Speicherbereich.

```
Unsignet int variable;
Printf ("Hello %u\n", variable);
```

Variablen deklarieren

Erstes Zeichen keine Ziffer, keine Umlaute und Sonderzeichen verwenden (ausser „_“). C ist CaseSensitive

```
#include <stdio.h>
int main (){
      int variable;           /* Initialisieren */
      variable = 523;         /* Deklarieren*/
      short variable2 = 666;
      long variable3, variable4, variable5;
      variable4 = variable; /* =666 */
      printf(
}
```

Deklarieren	Datentyp einer Variable zuweisen
Initialisieren	Erstbelegung bei der Deklaration der Variablen.

Variablen ausgeben

```
#include<stdio.h>
Int main(){
Short s_zahl= 123;
printf("Wert von s_zahl: %d\n", s_zahl);
return 0;
}
```

%d = Formatbezeichner

s_zahl = welche Variable

es können auch mehrere Fomatbezeichner und mehrere Variable angegeben werden (Reihenfolge beachgen!)

%d = dezimale Ganzzahl

%ld = lange dezimale Ganzzahl

Verwaltung von Ganzzahlen

Speicherbereich = Adresse im Arbeitsspeicher

Ich bin 2 Byte Gross und von Typ short															
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Das gäbe 1 + 4 + 8 = 13

Fliesskommazahlen

float	$+-3.4*10^{38}$	Einfache Genauigkeit	7 Stellen
double	$+-1.7*10^{308}$	Doppelte Genauigkeit	15 Stellen

Long double	+-3.4*10^{4932}	Sehr hohe Genauigkeit	19 Stellen

Verwaltung von Fliesskommazahlen

Exponentialschreibweise zur Basis 10 und formuliert diese so um, dass vor dem

Komma eine Null steht: 9.8 = 0.98*10^{1}.

Float basiert auf 4 Bytes. 1. – 3. Byte: Nachkomastellen, 4. Byte Exponent der Zehnerpotenz (=Position des Kommas)

Rechenoperationen

= Zuweisung, == vergleich, % Modulo (Rest der Division)

Datentypen umwandeln

`ergebnis = (float) x / (float) y`	Wenn x und y integer wäre, kann man es mit dem „Cast"- Operator in float umwandeln

Erweiterte Darstellung

`a +=2 // a = a + 2`	Zu a wird etwas hinzugezählt (+) und zwar ist das (=) 2

Zu verwenden bei: +=, -=, *=, /=, %=

Vorzeichenbehandlung

Bei einem 16-Bit Prozessor wird das erste Bit dazu verwendet um + oder – anzugeben:

Ich bin 2 Byte Gross und von Typ short															
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Diese Zahl währe negativ, da das erste Bit (16. Bit) 1 ist.

3. Daten formatiert einlesen und ausgeben

Printf() - Formatierte Ausgabe

`printf(„Formatstring %d", variable);`	Ausgabe von Text
`printf(„%d %c %i", a, b, c);`	Mehrere Variablen

Formatbezeichner

c	char oder int (0-255); einzelne Zeichen
d	int vorzeichenbehaftete Zahl (Dezimal)
ld	wie d, aber in Verbindung mit Ganzzahltyp long
i	wie d (i für Integer)

u	unsigned int; vorzeichenlos
lu	wie u, aber in Verbindung mit long
o	int oder unsigned int; vorzeichenlose Oktalzahl
x / X	int oder unsigned int; vorzeichenlose Hexadezimalzahl
lx / lX	wie x, aber in Verbindung mit long

Muss mit dem Typ der Variable übereinstimmen!

Oktale / Hexadezimale / ASCII- Darstellung

`printf(„Dezimal: %d“, wert);`	Dezimale Darstellung (Basis: 10 [0-10])
`printf(„Oktal: %o“, wert);`	Oktale Darstellung (Basis 8 [0-7])
`printf(„Hexadezimal: %x“, wert);`	Hexadezimale Darstellung (Basis 16 [0-F])
`printf(„ASCII: %c“, wert);`	ASCII-Darstellung

Formatbezeichner Fliesskommazahlen

f	float / double; Festkommaformat
e / E	float / double; Exponentialformat (e = kleines Exponentialzeichen, umgek.!)
g, G	float / double; kompaktere Art.

Bei long double „l“ voranstellen!

Formatierungen

`prinft(„%10d“);`	10 Stellen breit
`prinft(„%010d“);`	10 Stellen breit mit Nullen aufgefüllt.
`prinft(„%-10d“);`	linksbündig, 10 Stellen breit
`prinft(„%0.2f“);`	2 Kommastellen
`prinft(„%-010.3f“);`	linksbündig, 10-Stellen mit Nullen. 2 Kommastellen

Nicht druckbare Steuerzeichen

\a	Alert
\t	Horizontaler Tabulator
\n	New Line
\r	Carriage Return – Anfang der aktuellen Zeile
\0	Null Terminierungszeichen für

\b	Backspace (Cursor um eine Position nach links
\v	Vertikaler Tabulator
\f	FormFeed – Seitenumbruch
\“, ,, ?, \	Gibt das Zeichen nach dem „\“ aus
\ddd	Oktalen wert (ddd(octal)

	Strings

\xdd	Hex-Wert (dd(hexadezimal)

Einlesen von Daten mit scanf()

`printf(„Bitte Zahl eingeben:“);` `scanf(„%f“, &var);`	Liest eine Zahl in „var“ ein. KEINE Texteingaben im Befehl!
`scanf(„%5d“, zahl);`	Fünf Dezimalstellen
`scanf(„[^x]“, zahl);`	So lange einlesen, bis „x“ vorkommt.
`scanf(„[x]“, zahl);`	So lange einlesen, bis „x“ NICHT vorkommt.

`int zahl;`
`float wert;`
`printf(„Ganzzahl:“);`
`scanf(„%5“, &zahl);`
`fflush(stdin);// oder getchar();`
`printf(„Fliesskommazahl:“);`
`scanf(„%f“, &wert);`
`printf(„%d, %.2f“, zahl, wert);`
	Wenn man bei der Ganzzahl eine Zahl eingibt, die grösser 5 ist, dann wird der 6te wert für die Fliesszahl übernommen. Mit `fflush(stdin);` kann man das unterbinden. Hier wird der Tastaturpuffer geleert!
	ACHTUNG: Die Enter-Taste hat die Eingabe „\n“. Diese wird oft als Zeichen behandelt.
	Statt fflush(stdin); kann man jetzt getchar(); verwenden

4. Strukturierte Programmierung

- Sequenz: Abfolge von Befehlen, die nacheinander ausgeführt werden
- Selektion: Bedingungen
  - Einseitige Selektion (Es wird nur etwas ausgeführt, wenn es Wahr ist)
  - Zweiseitige Selektion (Es wird in jedem Fall etwas ausgeführt. Bei Wahr und bei Falsch)
  - Mehrseitige Selektion (Es werden mehrere Werte auf Wahrheit überprüft und dann verschiedene Anweisungen ausgeführt)
- Iteration: Schleifen
  - Kopfgesteuert, Wiederhole solange x == 1
  - Fussgesteuert, wird mindestens 1x durchgelaufen
  - Zählschleife, z.B. Wiederhole 10 mal

Vergleichsoperatoren

<, <=, >, >=, ==/*Gleich*/, != /*Ungleich*/

5. Kontrollstrukturen

If-Else

```
if (zahl < 50)
      {
            printf("Die Zahl ist kleiner als 50\n");
```

```
}
elseif (zahl == 50)
{
	printf("Die Zahl ist gleich 50\n");
}
else
{
	printf("Die Zahl ist grösser als 50\n");
}
```

Switch-Case

```
switch (zahl)
{
case 1:
	printf("1 ist eine gute Zahl\n");
	break;
case 99:
	printf("99 ist eine gute Zahl\n");
	break;
case 4:
	printf("4 ist eine gute Zahl\n");
	break;
default:
	printf("Unbekannte Zahl\n");
	break
}
```

While-Schleife

```
while (var < 100)
{
	var += 1;		// Wert wird um 1 erhöht
}
```

Do while-Schleife

```
do
{
	Anweisunge(en);
}while (Bedingung);
```

For-Schleife

```
int i = 0;
for(i=1; i<=200; i++)
{
	printf("%i.\tDurchlauf\n", i);
}
```

Schleifen abbrechen

break;	Beendet die komplette Schleife
continue;	Bricht die Schleife ab und setzt sie mit dem nächsten Durchlauf fort
return;	Beendet nicht nur die schleife, sondern auch die ganze Funktion (z.B. main())
exit;	Beendet das gesamte Programm

6. Funktionen

Funktion Definieren

```
Rückgabetyp Funktionsname(Parameter){}
```

Rückgabetyp	Datentyp; float, int, double, … // wenn kein Wert zurückgibt: void
Funktionsname	Beliebigen Namen der Funktion
Parameter	Argumente, die übergeben werden können. // Wenn kein Parameter: nichts oder void
Auweisungen	Anweisungen der Funktion

void: wenn man keinen Typ angeben will oder kann

Parameter: in den runden Klammern angegebene Werte

Argumente: an die Funktion gegebenen Argumente.

Beispiel

```
void multi_functi(int wert, char zeichen, float zahl)
{
      … source code …
}
int main()
{
      Multi_functi(10, ‚e‘, 23.5);
}
```

Call by Value: Die Werte, die an die Funktion übergegeben werden, werden mit diesem Verfahren übermittelt. Beim Aufruf wird eine Kopie des Originalwertes (Argument) angelegt. Wird dieser Wert geändert, ändert sich im Original nichts.

Wertrückgabe

Funktionen können auch Werte zurückgeben:

```
float function(float x, float y)
{
      float ergebnis = 0;
      ergebnis = x + y;
      return(ergebnis);
}
float ergebnis(float d)
{
      Return(d * PI); // PI = Konstante
}
```

```
int function(y)
{
      if(bedingung)
      return 1;
      else
      return 0;
}
void main(){
      x = function(y);
      printf(“%f”, x);
}
```

Lokale und globale Variablen

Lokal: in einer Funktion initialisiert und deklariert.

Global: ausserhalb einer Funktion initialisiert und definiert. Ist für alle Funktionen verfügbar.

7. Arrays und Strings

In einem Array kann man mehrere Variablen eines Datentyps speichern.

datentyp arrayname[anzahl_der_elemente] z.B. integer temperatur[24];

```
    int monat     =     0,
        i       =   0;

    double temperatur[31], durchschnitt    =        0;
```

```
    for(i = 0; i <= 30; i++)         // = 31 Tage
    {
            printf("Bitte die Temperatur vom %i. Tag eingeben: ", i+1);
            scanf("%lf", &temperatur[i]);

            durchschnitt = durchschnitt + temperatur[i];
    }


    for (i = 0; i <= 30; i++)
    {
        printf("Die Temperatur vom %i. Tag betraegt: %.2lf\n", i,
temperatur[i]);
    }

        durchschnitt = durchschnitt / 31;  // Durchschnitt berechnen


        printf("Der Durchschnitt betraegt: %.2lf\n\n", durchschnitt);
```

Arrays an Funktionen übergeben

Die Arrays werden mit Call-by-reference übergeben. Das heisst, es wird mit den Originalwerden gearbeitet.

void funktionsname(int arrayname_ohne_"[]", int anzahl elemente){}

rueckgabewert = funktionsname(arrayname_ohne_"[]");

Formatbezeichner: %c für ein einzelnes Zeichen. Mit der For-Schleife ausgeben:

```
for (i = 0; i <= 30; i++)
    {
        printf("Die Temperatur vom %i. Tag betraegt: %.2lf\n", i,
temperatur[i]);
    }
```

String

Einen String bezeichnet eine Kette von Zeichen.

```
char kette[] = {„Zeichen"};  // {} sind optional
char kette[] = {'H', 'a', 'l', 'l', 'o' '\0'};  //{} Obligatiorisch, \0
schliesst den String ab
```

Formatbezeichner: %s: printf(„%s", str); oder %c: printf(„%c", str[6]);

Sonderzeichen: für die Zeichen „ ' und \ muss man jeweils \ voranstellen!

`gets()`	Leerzeichen werden eigelesen. Kann aber nur String-Variablen einlesen. z.B. gets(txt);
`puts()`	Puts kann auch Leerzeichen ausgeben. Kann jedoch nur String-Variablen ausgeben. Z.B. puts(txt);

String-Operationen

Mit Strings kann man vieles anstellen. Man muss jeseils die Bibliothek **string.h** einbinden! Folgendes ist nice-to-know:

kopieren	`strcpy(txt_quelle, txt_ziel);`

vergleichen	`strcmp(txt1, txt2); // Ergibt true oder false!!!`