Block 2 (6.11.2023: NUR Diskussion/Fragen)

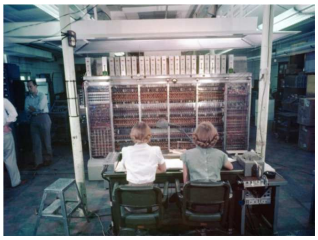
Schauen Sie sich vorab die Videos an!!

3 Einführung in C

Erste Programmiersprache (mit Unterprogrammen etc für das *Analytical Engine* von Charles Babbage)
Augusta Ada Byron King, Countess of Lovelace (*Ada Lovelace*), 1815-1852 (England).
(Zu ihren Ehren später: Programmiersprache ADA).

Erster Compiler (1952 für UNIVAC Computer) von
Grace Brewster Murray Hopper (1906-1992, USA)
(→ später die Programmiersprache COBOL)

Erster physikalische Computersimulationen ("Monte Carlo") programmiert 1953 von
Arianna Rosenbluth (USA)
Augusta H. Teller (1909-2001, Ungarn/USA)



3.1 Mein erstes Programm

Inhalt dieses Abschnitts sollen Sie selber am Computer nachvollziehen

Benutze Text Editor für `first.c`

```
#include <stdio.h>

int main()
{
  printf("my first program");
  return(0);
}
```

Compiliere

```
cc -o first first.c -Wall
```

Starte:

```
first
```

Um mehr über Funktion von `printf` zu erfahren: Schreibe `man 3 printf` in Shell.

3.2 Variablen, Befehle, Ausdrücke

Variablen müssen deklariert werden. Namen (z.B. für Variablen) müssen mit Buchstaben (a...z,A...Z) anfangen, enthalten Buchstaben, zahlen und Unterstrich (_).

```
#include <stdio.h>

int main()
{
  int counter;

  for(counter=0; counter<10; counter=counter+1)
    printf("%d\n", counter);

  return(0);
}
```

Zählen Sie Typen von Variablen auf

Wichtige Typen:

- int
- double
- char
- Arrays=Zeiger
- Funktionen
- Strukturen
- Selbst-definierte Datentypen
- beliebige Kombinationen

Mehrere Variablen auf einmal deklarieren: `double number1, average, sum`.
Wichtige Befehle

- Zuweisung, arithmetische Ausdrücke, z.B.

```
a = b + c;
value = 3.0*sin(angle);
hours = minutes/60;
mod = x % y;
value += delta;
counter++;
number--;
code = input & mask     /* bitwise AND */
code2 = input | mask2   /* bitwise OR */
rest = sequence << n    /* bit shift left by n bits */
```

- for-Schleife

```
for( <initial> ;  <condition> ;  <command>)
  <command>;
```

- while Schleife

```
while( <condition> )
  <command>;
```

  Bsp.:

```
while( counter < 10 )
  printf("counter=%d\n", counter++);
```

- if-Befehl

```
if ( <condition >)
  <command1>;
else         /* optional */
  <command2>;
```

Mögliche `<conditions>`

```
a==b   !!!!!!
a!=b
a<b
a<=b
a>b
a>=b
```

`a` und `b` können nicht nur Variablen sondern letztlich bliebige Ausdrücke sein.

Bedingungen können per AND (&&) oder/und OR (||) verknüpft werden, z.B.

```
if( (money>100) || ((price < 10) && (money>20)))
  printf("Buy it!!\n");
```

- Blöcke können anstatt eines einzelnen Befehls stehen

```
{
  <command1>;
  <command2>;
  ...
}
```

Achtung

```
double x;
x = 2/3;
printf("%f\n", x)
```

ergibt 0.
Casts übersetzen Datentypen, falls möglich:

```
  int z;

  z = (int) 3.2;
  printf("%d\n", z);
```

ergibt 3;

3.3 Arrays

Arrays = Vektoren oder "Listen" von Variablen. Deklaration: `<type> <name>[<size>]`. Arrays starten mit Index 0, laufen bis `<size>-1`. Bsp.:

```
#include <stdio.h>

int main()
{
  int counter;
  double value[10];
  for(counter=0; counter<10; counter++)
    value[counter]=counter*counter + 0.1;
  return(0);
}
```

Matrix = Array von Arrays: `double matrix[10][10]`
Summe zweier 10x10 Matrizen

```
for(row=0; row<10; row++)
  for(column=0; column<10; column++)
    result[row][column] = matrix1[row][column] + matrix2[row][column];
```

3.4 Unterprogramme, Funktionen

Es gibt vordefinierte Funktionen, z.B. mathematische Funktionen. Man benötigt header-file, z.B. : `math.h` (seltsame Resultat falls nicht verwendet)

```
#include <stdio.h>
#include <math.h>

int main()
{
  double argument;
```

```
  argument = 0.0;
  while (argument < 2*M_PI)
  {
    printf("sin(%f)=%f\n", argument, sin(argument));
    argument += 0.1;
  }
  return(0);
}
```

Linke Bibliothek beim Compilieren: `-l<lib-name>`

```
cc -o first first.c -Wall -lm
```

Zufallszahlen

```
#include <stdlib.h>

...

value = drand48();
```

erzeugt (pseudo) Zufallszahl in $[0, 1)$
Eigene Funktionen definieren:

```
#include <stdio.h>

/*************** calc_average()**************/
/** Calculates average of values passed      **/
/** uses periodic boundary conditions        **/
/** PARAMETERS: (*)= return-paramter         **/
/**       number: .. of values               **/
/**     value: array of values               **/
/** RETURNS:                                 **/
/**    average                               **/
/*********************************************/
double calc_average(int number, double value[])
{
  int counter;  /* local variable, not visible e.g. in main() */
  double sum;   /* the same */
  sum = 0.0;
  for(counter=0; counter<number; counter++)
    sum += value[counter];
  return(sum/number);
}
```

```
int main()
{
  int counter;
  double value[10];
  double average;
  for(counter=0; counter<10; counter++)
    value[counter]=counter*counter;
  average = calc_average(10, value);
  printf("avg=%f\n", average);
  return(0);
}
```

Unterprogramme = Funktionen ohne Rückgabewert:

```
void print_value_and_flowers(int x)
{
  printf("flowers\n, value=%d, flowers\n", x);
}
```

Mehr als ein Rückgabewert: benutze Strukturen oder Zeiger (siehe später)

3.5 Geltungsbereich von Variablen

Variablen sind lokal, bis auf (SEHR BÖSE!!) globale

```
#include <stdio.h>

int x;

void change(int z)
{
  x= 100;
  printf("local x=%d, z=%d\n", x, z);
  z= 101;
  printf("local x=%d, z=%d\n", x, z);
}

int main()
{
  int z;

  x = 200; z =201;     /* one can have several commands in a line */
```

```
  printf("x=%d, z=%d\n", x, z);
  change(z);
  printf("x=%d, z=%d\n", x, z);

  return(0);
}
```

[Selbsttest]

Frage: Was wird ausgegeben (1-2 min nachdenken)

TAKE HOME WORK

Besorgen Sie sich das des C Tutorials vom StudIP, lesen sie die Seiten 1-40. Vollziehen sie die Programmteile praktisch nach!