# Übung Calculation Pi

## Aufgabenstellung

Die Aufgabe war es drei verschiedene Tasks zu programmieren. Einen Task der die Leibniz-Reihe berechnet, einen weiteren Task der Pi berechnet und einen Task der die Zahl auf dem Display ausgibt, die Tasten und die Tasks ansteuert.

Der Leibniz Algorithmus:

Mathematisch sieht der Algorithmus so aus, wie

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} \cdots = \frac{\pi}{4}$$

### Genauigkeit des Algorithmus:

| n = Anzahl durchläufe | Ergebnis |
|---|---|
| 2 | 2.6666666666666665 |
| 4 | 2.8952380952380952 |
| 8 | 3.0170718170718169 |
| 16 | 3.0791533941974261 |
| 32 | 3.1103502736986859 |
| 64 | 3.1259686069732875 |
| 100 | 3.1315929035585528 |
| 1000 | 3.1405926538397928 |
| 10000 | 3.1414926535900429 |
| 100000 | 3.1415826535897935 |
| 1000000 | 3.1415916535897930 |
| 10000000 | 3.1415925535897928 |

Der Algorithmus ist nicht wirklich effizient man braucht extrem viele Durchläufe bis man ein paar passende Komastellen hat.

Da dieser Algorithmus sich stetig steigert respektive die Brüche immer kleiner werden. Wurde mit einer FOR schleife gearbeitet. Dabei wird die Schleife bis zum gegebenen Wert «Genauigkeit», durchlaufen. Die Mathematik lässt sich dann in einer Zeile berechnen.

Mit dieser Formel: $zahl \mathrel{+}= \frac{(-1)^{i+1}}{2*i-1} = \frac{\pi}{4}$

Am Schluss wurde noch das Zwischenresultat «zahl» mit 4 multipliziert.

```
    for(i = 1; i <= genauigkeit; i++)
    {
        zahl += pow(-1.0,i+1)/(2*i-1);
        pi_leibniz = zahl*4;
```

## Verfahren von Gauss, Brent und Salamin

Das Verfahren von Gauss, Brent und Salamin funktioniert nach der berechnung einer Lemniskate mit hilfe von ellptischer Integrale und des Arithemtisch-geometrischen Mittel. Gauss vermutete damal Pi mit der lemniskatischen Konstante:

$$\pi = \frac{4AGM\left(1,\frac{1}{\sqrt{2}}\right)^2}{1-\sum_{j=1}^{\infty} 2^{j+1}c_j^2}$$

Das Arithmetisch-geometrische Mittel wird über die Iteration:

$$a_n = \frac{a_{n-1}+b_{n-1}}{2} \; und \; b_n = \sqrt{a_{n-1}\cdot b_{n-1}}$$

$C_n$ wird dann folgendermassen berechnet:

$$c_n^2 = a_n^2 - b_n^2$$

a und b müssen hierbei grösser 0 sein, da das sonnst nicht funktioniert.

Zuerst müssen folgende Startwerte festgelegt werden:

```
    float a = 1;
    float b = 1/sqrt(2);
    float s = 0.5;
    float hilf;
    float c;
```

Hilf und c haben keinen Startwert werden aber später benötigt.

Die effektive berechnung beginnt im nächsten Schritt. Da man nach 3 Durchläufen schon eine Genauigkeit von 9 Stellen nach dem Komma hat, macht es nicht viel Sinn den Algorithmus länger rechnen zu lassen. Float kann eh nur knapp 5 stellen nach dem Koma richtig anzeigen.

```
for(n = 1; n <= 3; n++)
{
    hilf = a;
    a = (a+b)/2;
    b = sqrt(hilf*b);
    c = pow(a,2)-pow(b,2);
    s = s-pow(2,n)*c;
    pi_Gauss = (2*pow(a,2))/s;
```

### Zur berechnung

Als erstes setzt man hilf = a um den wert bei jedem Durchlauf zwischen zuspeichern, dieser wird im 3. Schritt noch mals verwendet jedoch ändert man den Wert von a unterdessen. Als erstes berechnet man das Arithmetische Mittel «a». Im nächstenschritt berechnet man das Geometrische Mittel «b». Als vierten Schritt rechnet man das Arithmetisch geometrische Mittel aus. Nun zum Komplizierten Bruch, «s» wiederspiegelt den Nenner. Zum Schluss wird noch ausgerechnet und man hat eine Funktion geschrieben die quadratisch zu PI konvergiert.

## Zwischenergebnisse:

| Index n | $a_n$ | $b_n$ | $c_n$ | $s_n$ | pi |
|---|---|---|---|---|---|
| **n = 0** | 1 | 0,70710 67811 86547 | | 0,5 | |
| **n = 1** | 0,85355 33905 93274 | 0,84089 64152 53715 | 0,02144 66094 06726 | 0,45710 67811 86547 | **3,1**8767 26427 12110 |
| **n = 2** | 0,84722 49029 23494 | 0,84720 12667 46891 | 0,00004 00497 56187 | 0,45694 65821 61801 | **3,141**68 02932 97660 |
| **n = 3** | 0,84721 30848 35193 | 0,84721 30847 52765 | 0,00000 00001 39667 | 0,45694 65810 44462 | **3,14159 2653**8 95460 |

## Beschreibung des Display Tasks

Als erstes werden die Tasten eingelesen anschliessend wird mit einer Statemachine ausgewertet welche Taste gedrückt wurde und dem entsprechend den Task wechelt die Tasks Startet und auf den Bildschirm schreibt. In der Statemachine gibt es einen State Displaymode, dieser soll immer der normal zustand sein, um auf das display zu schreiben. Der Leibnizstate soll den Leibniz Task anfragen ob er gerade am rechnen ist oder nicht und dann die aktuellen Daten auf das Display schreiben. Zusätzlich wird darin gewartet ob das Pi Fertig eventbit gesetzt wurde. Das selbe passiert im Gauss state.

## Eventbits:

```
#define interuppt10ms 0
#define Algorithmus 1
#define Starten 2
#define Leibniz_PI_RDY 3
#define Gauss_PI_RDY 4
#define reset 5
#define Starten_Leibniz 6
#define Ready 7
```

Interuppt10ms → soll verwendet werden um um den 500ms Timer zu einzustellen

Algorithmus → wird verwendet um zwischen den algorithmen auszuwählen

Starten → wird verwendet um die Algorithme zu starten

Leibniz_Pi_RDY → wird verwendet um zu signalisieren ob der Leibniztask fertig ist

Gauss_Pi_RDY → wird verwendet um zu signalisieren ob der Leibniztask fertig ist

Reset → wird verwendet um den Algorithmus zurück zusetzten

Start_Leibniz → wird nicht verwendet

Ready → wird verwendet um nachzufragen ob der Task gearade am rechnen ist oder nicht

## Zeitmessung

Die Zeitmessung konnte leider nicht durchgeführt werden, da das Programm nich zu 100% funktioniert hat. Jedoch wurde festgestellt, dass der Leibniz Task ca 2 Minuten benötigt wobei der Gauss Task nicht spürbar ist.

Was jedoch gesagt werden kann ist, dass der Gauss Task sehr viel schneller ist. Dieser benötigt genau 3 durchläufe um die Float Variable zu füllen. Der Leibniz Task braucht für 5 Komma stellen ungefähr 160'000 durchläufe. Klar muss man sagen, dass die berechnung vom Gauss Task im prozessor etwas anspruchsvoller ist. Jedoch ist dieser immer noch viel Schnäller da er quadratisch zu pi konvergiert.