

Portfolioarbeit

Grundlagen der technischen Informatik, INF-P-IN005

Studiengang BSc Cyber Security 1. Semester

Themensteller: Prof. Martin Klaper

vorgelegt von: Valentino Totaro
Haldenstrasse 12
8302 Kloten
+41 78 645 45 59
valentino.totaro@students.ffhs.ch

Abgabetermin Portfolio-Arbeit Nr. 1: 09.09.2024

Inhaltsverzeichnis

Portfolioarbeit	0
1. Einleitung Portfolio-Arbeit 1	1
1.1 Aufgabenstellung	1
2. Portfolioarbeit 1	2
2.1 TA 1: Johnny installieren und einarbeiten	2
Aufgabenstellung	2
2.1.1 Johnny Installation	3
2.1.2 Einarbeitung	3
2.2 TA 2: Countdown von 10 bis 0	3
Aufgabenstellung	3
2.2.1 Analyse der Aufgabenstellung	4
2.2.2 Lösungsansatz Countdown	4
2.2.3 Ausgabe	6
2.3 TA 3: Ganze Zahlen Subtrahieren mit Johnny	7
Aufgabenstellung	7
2.3.1 Lösungsansatz Division mit Rest	7
2.3.2 Ausgabe	10
3. Abbildungsverzeichnis	18
4. Tabellenverzeichnis	18
5. Literaturverzeichnis	18
6. Abkürzungsverzeichnis	18
7. Eigenständigkeitserklärung für Portfolio-Arbeit Nr. 1	19

1. Einleitung Portfolio-Arbeit 1

Diese erste Portfolio-Arbeit (PVA 1) hat das Ziel, den von Neumann-Simulator Johnny zu verstehen und sich in dessen Anwendung einzuarbeiten. Im ersten Teil, der in Kapitel 2.1 behandelt wird, wird der Simulator installiert und erste Erfahrungen damit gesammelt. In der zweiten Aufgabe, Kapitel 2.2, soll ein Countdown-Algorithmus von 10 bis 0 programmiert werden. Schliesslich wird in Kapitel 2.3 ein Algorithmus zur Division zweier ganzer Zahlen implementiert, bei dem sowohl der Quotient als auch der Rest ermittelt werden.

1.1 Aufgabenstellung

Die jeweiligen Aufgabenstellungen sind in den Kapiteln zu den entsprechenden Teilaufgaben erläutert, um eine bessere Lesbarkeit zu gewährleisten und die Überprüfung der erarbeiteten Lösungen im Vergleich zur Aufgabenstellung zu erleichtern.

2. Portfolioarbeit 1

Im zweiten Kapitel dreht sich alles um die Portfolio-Arbeit (PVA 1), deren Schwerpunkt auf dem von Neumann-Simulator Johnny liegt. Die Aufgaben umfassen drei Teilaufgaben:

- **TA1:** Installation des von Neumann-Simulators Johnny.
- **TA2:** Entwicklung eines Algorithmus für einen Countdown im Johnny-Simulator.
- **TA3:** Programmierung eines Algorithmus zur Division zweier ganzer Zahlen im Johnny.

2.1 TA 1: Johnny installieren und einarbeiten

Die erste Aufgabe besteht darin, den Johnny-Simulator zu installieren und durch Tutorials ein Verständnis für seine Funktionsweise zu erlangen.

Aufgabenstellung

«Verschaffen Sie sich einen Überblick und ein Verständnis zum "JOHNNY- Modellrechner".

Dafür:

- *Installieren Sie den "JOHNNY-Modellrechner" auf Ihrem Computer. Alternativ nutzen Sie die Webversion (Nachteil: keine Editierung des Hauptspeichers möglich)*
- *Arbeiten Sie das Tutorial durch*
Hinweis: Weiterführende Links finden Sie in den Tipps zur Portfolioarbeit. Am besten starten Sie diese Aufgabe mit einer Recherche im Internet. Bewertung: gem. Bewertungsschema.» (Klapper, 2024)¹

¹ <https://moodle.ffhs.ch/mod/assign/view.php?id=4713977>

2.1.1 Johnny Installation

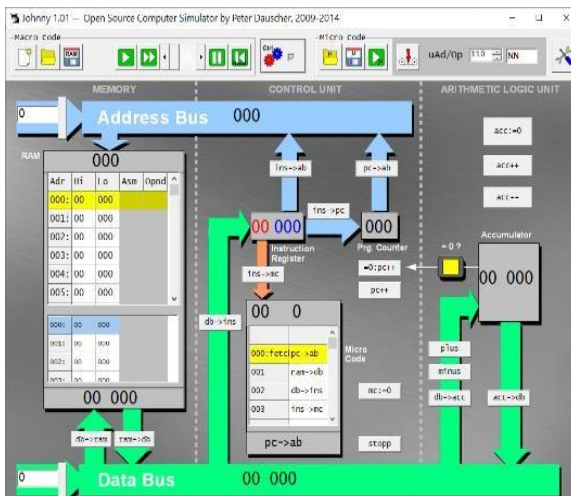


Abbildung 1: Johnny Version 1.0.1 (Screenshot)

optional aktiviert werden, um die einzelnen Programmierschritte bei der Makrobefehl-Programmierung zu erläutern.

2.1.2 Einarbeitung

Die bearbeiteten Online-Tutorials boten eine gute Möglichkeit, sich mit dem Johnny-Simulator vertraut zu machen, und ergänzten das Wissen aus den Vorlesungen. Dieser Aspekt wird hier nicht weiter vertieft.

2.2 TA 2: Countdown von 10 bis 0

In dieser Aufgabe wird ein Algorithmus programmiert, der einen Countdown von 10 bis 0 im Johnny-Simulator durchführt.

Aufgabenstellung

«Schreiben Sie einen Algorithmus in "JOHNNY-ASSEMBLER" für einen Countdown. Der Countdown startet bei 10 und endet bei 0

Es werden alle "Countdown-Werte" in einem darauffolgenden Register gespeichert (z. B. ADR 50=10 <> ADR 51=9 <> ... <> ADR 60=0)» (Klapper, 2024)²

² <https://moodle.ffhs.ch/mod/assign/view.php?id=4713977>

2.2.1 Analyse der Aufgabenstellung

Die Herausforderung dieser Aufgabe bestand darin, dass Johnny lediglich die Addition und Subtraktion von Adressbereichen zulässt und sicherzustellen ist, dass der Countdown vollständig bis zur Zahl 0 korrekt geschrieben wird.

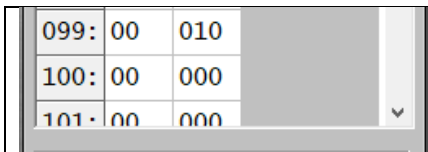
2.2.2 Lösungsansatz Countdown

Um die Benutzerfreundlichkeit zu erhöhen, wurde ein Algorithmus entwickelt, bei dem der Startwert „10“ automatisch im Register #099 festgelegt wird, ohne dass der Nutzer ihn manuell eingeben muss. Während der Ausführung wird der Wert bei jedem Schritt um 1 verringert, bis der Countdown die Zahl „0“ erreicht.

000				
Adr	Hi	Lo	Asm	Opnd
000:	01	099	TAKE	099
001:	04	010	SAVE	010
002:	04	015	SAVE	015
003:	08	010	DEC	010
004:	07	002	INC	002
005:	06	010	TST	010
006:	05	001	JMP	001
007:	09	025	NULL	025
008:	10	000	HLT	000
009:	00	000		
010:	00	000		
099:	00	010		
100:	00	000		
101:	00	000		

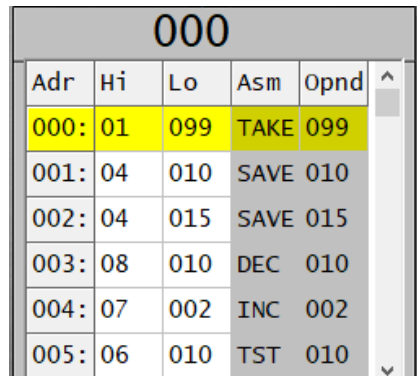
Abbildung 2: Schritte der Makrobefehle (Screenshot)

Bei der Ausführung als Makrocode finden folgende Schritte statt:



099:	00	010
100:	00	000
101:	00	000

Abbildung 3: Register 099 (Screenshot)



000				
Adr	Hi	Lo	Asm	Opnd
000:	01	099	TAKE	099
001:	04	010	SAVE	010
002:	04	015	SAVE	015
003:	08	010	DEC	010
004:	07	002	INC	002
005:	06	010	TST	010

Abbildung 4: Register #000 bis #005 des Countdowns (Screenshot)

Aus dem Reg. #099 (vgl. Abb. 3) wird der Wert «010» in die ALU eingelesen. (vgl. Abb. 4)

Der Wert «010» wird aus der ALU in das Reg. #010 gespeichert. Das Reg. dient als Variable, um die einzelnen Iterationen zu erfassen und dass der Wert «010» nicht bei jeder Ausführung von Hand in das Reg. geschrieben werden muss.

Der Wert «010» der sich noch in der ALU befindet wird in das Reg. #015 geschrieben. Dieses Reg. ist der Startpunkt des Countdowns und kann vom Anwender frei gewählt werden, sollte jedoch nicht kleiner als «009» oder grösser als «085» sein.

Das Reg. #010 wird um einen Wert dekliniert (-1). Mit diesem Makrobefehl wird sichergestellt, dass die jeweilige Iteration erkannt werden kann.

Das Reg. #002 wird um Eins erhöht (+1), damit in der nächsten Iteration der Wert aus Reg. #010 nachfolgend an Reg. #015 ergänzt wird.

Mit TST wird die ALU getestet, ob der Wert im Accumulator gleich «0» ist. Wenn nein, wird der Makrobefehl in Reg. #006 ausgeführt, bei Wert NULL der Befehl in Reg. #007.

006:	05	001	JMP 001
007:	09	025	NULL 025
008:	10	000	HLT 000
009:	00	000	
010:	00	000	

Abbildung 5: Register #006 bis #010 des Countdowns (Screenshot)

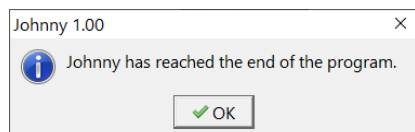


Abbildung 6: Abschluss des Algorithmus (Screenshot)

Basierend auf dem letzten Befehl #005 wird dieses Reg. #006 gewählt solange der Wert von Reg. #010 nicht NULL entspricht (vgl. Abb. 5). Mit dem Befehl «JMP» wird auf das Reg. #001 gesprungen und startet die Iteration neu.

Basierend auf dem letzten Befehl #005 wird dieses Reg. gewählt, sobald der Wert von Reg. #010 NULL entspricht. Die Anzahl Iterationen entspricht nun dem Wert aus dem Reg. #099 minus 1. Somit wird in diesem Schritt der letzte Wert des Countdowns in Reg. #025 auf NULL geschrieben.

Das Programm hält definiert an (vgl. Abb. 6) und gibt eine Meldung an den Nutzer. Diese Meldung ist nicht zwingend notwendig, aber für die Nutzerfreundlichkeit sinnvoll.

Tabelle 1: Lösungsbeschreibung Countdown

2.2.3 Ausgabe

Nach der Ausführung des Programms sind die Countdown-Werte in den Registern von #015 bis #025 gespeichert, wobei das Register #025 abschliessend den Wert „0“ enthält, wie in der Aufgabenstellung gefordert.

008				
Adr	Hi	Lo	Asm	Opnd
015:	00	010		
016:	00	009		
017:	00	008		
018:	00	007		
019:	00	006		
020:	00	005		
021:	00	004		
022:	00	003		
023:	00	002		
024:	00	001		
025:	00	000		
026:	00	000		

Abbildung 7: Register Werte #015 bis #026 (Screenshot)

2.3 TA 3: Ganze Zahlen Subtrahieren mit Johnny

In dieser Teilaufgabe soll ein Algorithmus programmiert werden, der eine ganzzahlige Division durchführt und dabei den Quotienten und den Rest der Division ermittelt.

Aufgabenstellung

«Schreiben Sie einen "JOHNNY-ASSEMBLER" Algorithmus für die ganzzahlige Division ("IDIV").

- Schreiben Sie den Algorithmus "IDIV"
- Als Ergebnis des Programms resultiert eine ganze Zahl, resp. zwei Zahlen: Ergebnis q und Rest r

Hinweis: Wird eine Zahl z durch eine Zahl n geteilt, dann kann man das Ergebnis wieder durch zwei ganze Zahlen darstellen, nämlich das Ergebnis q der „ganzzahligen Division“ und den „Rest“ r . Man schreibt auch $z : n = q \text{ Rest } r$ und es gilt: $q \cdot n + r = z$ (Klapper, 2024)³

Analyse der Aufgabenstellung

Die Herausforderung dieser Aufgabe lag aus Sicht des Autors darin, dass

1. Johnny in der Bedienung nicht sehr userfreundlich ist, d.h. wenn der Algorithmus getestet wird, kann nicht «einfach» eine Zeile mit fehlendem Code zwischen zwei Zeilen hinzugefügt werden. Es half den groben Code schriftlich ausserhalb von Johnny zu entwickeln.
2. alle relevanten Funktionen des Algorithmus bedacht werden mussten. Z.B. hat der Autor, die Division durch Null erst am Schluss eingebaut (vgl. Punkt 1 dieser Aufzählung), um ein falsches Resultat bei einer Division durch Null auszuschliessen.
3. Der Algorithmus für Division ohne Ausgabe des Rests erwies sich für den Autor als einfach zu realisieren. Für den zweiten Wert mit «Rest» benötigte es einige Tüftelei, um das mit wenig Variablen zu realisieren.

Dementsprechend konnte die nachfolgend beschriebene Lösung in 2.3.1 realisiert werden.

2.3.1 Lösungsansatz Division mit Rest

Der entwickelte Algorithmus überprüft zunächst, ob eine Division durch 0 erfolgt. Wenn dies der Fall ist, wird der Prozess abgebrochen. Ansonsten wird der Dividend so lange durch den Divisor subtrahiert, bis der Rest ermittelt wurde. Die Anzahl der Subtraktionen bestimmt den Quotienten.

³ <https://moodle.ffhs.ch/mod/assign/view.php?id=4713977>

000				
Adr	Hi	Lo	Asm	Opnd
000:	06	019	TST	019
001:	05	003	JMP	003
002:	05	016	JMP	016
003:	01	018	TAKE	018
004:	04	020	SAVE	020
005:	01	017	TAKE	017

006:	04	021	SAVE	021
007:	07	021	INC	021
008:	01	020	TAKE	020
009:	04	022	SAVE	022
010:	03	019	SUB	019
011:	04	020	SAVE	020

012:	06	020	TST	020
013:	05	007	JMP	007
014:	08	021	DEC	021
015:	05	016	JMP	016
016:	10	000	HLT	000
017:	00	000		

018:	00	020		
019:	00	008		
020:	00	000		
021:	00	002		
022:	00	004		
023:	00	000		

Abbildung 8: Algorithmus Division mit Rest mit Johnny (Screenshot)

Der geschriebene Code beginnt damit, zu prüfen, ob eine Division durch Null vorliegt, und falls dies zutrifft, wird der Prozess abgebrochen. Danach werden die erforderlichen Speicherbereiche vorbereitet und ein Zähler für die Anzahl der Divisionen erhöht (siehe Abb. 8 #007). Der Wert in der ALU wird zunächst als Rest zwischengespeichert (siehe Abb. 8 #022). Anschliessend wird der Divisor vom Dividend subtrahiert, der Quotient in einer Variablen abgelegt und der Vorgang in einer Schleife durch Tests und Sprünge (siehe Abb. 8 Speicher #012 und #013) so oft wiederholt, bis der Wert Null erreicht ist. Da es sich um eine "DO-WHILE"-Schleife handelt, bei der die Bedingung erst am Ende geprüft wird, wird der Quotient nach dem letzten Durchlauf um eins verringert (-1), um den korrekten Quotientenwert zu erhalten.

Nachfolgend werden die einzelnen Makrobefehle im Detail erläutert:

000				
Adr	Hi	Lo	Asm	Opnd
000:	06	019	TST	019
001:	05	003	JMP	003
002:	05	016	JMP	016
003:	01	018	TAKE	018
004:	04	020	SAVE	020
005:	01	017	TAKE	017

Abbildung 9: Register #000 bis #005 Division mit Rest mit Johnny (Screenshot)

Dieser Schritt prüft den Speicher #019, der den zu berechnenden Divisor erhält, ob der Wert Null entspricht.

Wenn Reg. #019 nicht 0 entspricht, dann wird auf Reg. #003 gesprungen.

Wenn der Divisor Wert 0 entspricht, wird auf Reg. #016 gesprungen, welches das Programm beendet.

Der Dividend wird aus dem Reg. #018 in die ALU geladen. Dies und die nächsten zwei Reg. dienen lediglich der Nutzerfreundlichkeit des Algorithmus.

Der Wert aus der ALU wird in Reg. #020 gespeichert.

Der Wert null wird aus dem Reg. #017 in die ALU geladen

006:	04	021	SAVE	021
007:	07	021	INC	021
008:	01	020	TAKE	020
009:	04	022	SAVE	022
010:	03	019	SUB	019
011:	04	020	SAVE	020

Abbildung 10: Register #006 bis #011 Division mit Rest mit Johnny (Screenshot)

Der Wert null aus der ALU wird in das Reg. #021 gespeichert, welches dem Quotienten entspricht.

Der Quotient in Reg. #021 wird um eins inkrementiert (+1).

Der Dividend wird aus #020 geladen.

Der Inhalt der ALU entspricht beim letzten Durchgang dem Rest, welcher in Reg. #022 gespeichert wird.

Nun wird der Divisor aus Reg. #019 dem Dividenten abgezogen.

Und der erhaltene Wert in Reg. #020, also dem Dividenten gespeichert.

012:	06	020	TST	020
013:	05	007	JMP	007
014:	08	021	DEC	021
015:	05	016	JMP	016
016:	10	000	HLT	000
017:	00	000		

Abbildung 11: Register #012 bis #017 Division mit Rest mit Johnny (Screenshot)

Der Wert der ALU wird nun geprüft, ob dieser Null entspricht. Ist dies nicht der Fall, wird Reg. #013 ausgeführt, ansonsten Reg. #014.

Dieses Reg. ist ein Loop, solange der Wert von Reg. #012 nicht 0 entspricht, springt es auf Reg. #007 zurück, welches die Berechnung weiter ausführt.

Der Quotient wird um eins dekliniert (vgl. Abs. 2 dieses Kapitels)
Anschliessend wird auf das nächste Reg. #016 gesprungen. Dieser Schritt könnte gespart werden, ist aber für die Leserlichkeit beibehalten worden.

Stoppt den Algorithmus.
Reg. für Wert «0», wird nie verändert.

018:	00	020		
019:	00	008		
020:	00	000		
021:	00	002		
022:	00	004		
023:	00	000		

Abbildung 12: Register #018 bis #023 Division mit Rest mit Johnny (Screenshot)

Ist der Dividend und soll nach den Anwenderwünschen verändert werden.

Ist der Divisor und soll nach den Anwenderwünschen verändert werden.

Reg. für den Quotient, wird für die Zwischenresultate benötigt.

Resultat: **Quotient** q

Resultat: **Rest** r

Tabelle 2: Lösungsbeschreibung IDIV

2.3.2 Ausgabe

Der Quotient und der Rest werden in den Registern #021 und #022 gespeichert und angezeigt.

3. Einleitung Portfolio-Arbeit 2

In dieser Aufgabe geht es darum, ein grundlegendes Verständnis für das Dokumentieren und Modellieren von Algorithmen zu erlangen. Mithilfe von UML 2.0 soll der Ablauf eines Algorithmus visuell dargestellt werden, um die einzelnen Schritte und deren Zusammenhänge klar zu erkennen. Ein weiterer wichtiger Teil der Aufgabe ist die Implementierung eines Algorithmus in JOHNNY-ASSEMBLER, um das Konzept hinter dem Sieb des Eratosthenes zu verstehen. Dieses klassische Verfahren dient zur Bestimmung von Primzahlen und ist ein grundlegendes Beispiel für die effiziente Lösung eines mathematischen Problems. Abschliessend wird der Algorithmus durch eine Flowchart-Darstellung dokumentiert, um den Ablauf und die Funktionsweise möglichst anschaulich darzustellen.

3.1 Aufgabenstellung

Der Auftrag umfasst die folgenden Schritte, die der Reihe nachbearbeitet werden sollen:

- Einführung in UML 2.0 und die Dokumentation von Algorithmen
In dieser Teilaufgabe sollen die Grundlagen der UML 2.0 erarbeitet und die entsprechenden Werkzeuge zur Erstellung von Flowcharts recherchiert werden. Das Ziel ist es, sich einen Überblick über die verschiedenen Möglichkeiten der Dokumentation und Modellierung von Algorithmen zu verschaffen.
- Implementierung eines Algorithmus in JOHNNY-ASSEMBLER: Das Sieb des Eratosthenes. Es soll ein Algorithmus in JOHNNY-ASSEMBLER geschrieben werden, der das Sieb des Eratosthenes zur Bestimmung der Primzahlen von 1 bis n verwendet. Der Algorithmus muss so konfigurierbar sein, dass der Parameter n frei gewählt werden kann. Außerdem soll der Speicher genutzt werden, um festzuhalten, ob eine Zahl eine Primzahl ist oder nicht.
- Dokumentation des Algorithmus mit einem Flowchart
Schließlich soll der erstellte Algorithmus mittels eines UML 2.0 konformen Flowcharts dokumentiert werden. Das Flowchart soll den Ablauf des Algorithmus so anschaulich wie möglich darstellen und alle notwendigen Legenden und Beschriftungen enthalten.

4. Portfolio-Arbeit 2

TA-1: Überblick über UML 2.0 und Werkzeuge zur Erstellung von Flowcharts

- **Beschreibung:** Verschaffen Sie sich ein grundlegendes Verständnis zu UML 2.0 und den verschiedenen Techniken zur Modellierung von Algorithmen.
- **Aufgabe:**
 - Setzen Sie sich mit den Konzepten der UML 2.0 auseinander.
 - Suchen Sie geeignete Werkzeuge, um Flowcharts zu zeichnen (z. B. Activity Charts).
- **Hinweis:** Nutzen Sie die zur Verfügung gestellten Links sowie weiterführende Literatur oder das Internet für Ihre Recherche.

TA-2: Algorithmus für das Sieb des Eratosthenes in JOHNNY-ASSEMBLER

- **Beschreibung:** Schreiben Sie einen Algorithmus in JOHNNY-ASSEMBLER, der das Sieb des Eratosthenes zur Bestimmung der Primzahlen von 1 bis n implementiert.
- **Anforderung:**
 - Der Algorithmus muss konfigurierbar sein, d. h., der Parameter n muss frei gewählt werden können.
 - Die Ausgabe erfolgt im freien Hauptspeicher, wobei für jede Zahl festgehalten wird, ob es sich um eine Primzahl handelt.
 - Beispiel: Speicherstelle 101 repräsentiert die Zahl 1.
- **Zusätzliche Frage:** Was ist die höchste Primzahl, die Sie mit JOHNNY bestimmen können?

TA-3: Dokumentation des Algorithmus mittels Flowchart

- **Beschreibung:** Dokumentieren Sie den Algorithmus, den Sie in TA-2 erstellt haben, mithilfe eines Flowcharts gemäß UML 2.0.
- **Aufgabe:**
 - Erstellen Sie ein Flowchart, das den Algorithmus vollständig dokumentiert.
 - Achten Sie darauf, sich so weit wie möglich an den UML 2.0 Standard zu halten.
 - Ergänzen Sie das Diagramm mit allen notwendigen Legenden und Beschriftungen.

4.1 TA-1: Überblick über UML 2.0 und Werkzeuge zur Erstellung von Flowcharts

Um die Teilaufgabe TA-1 zu lösen, habe ich mich zuerst intensiv mit den Konzepten von UML 2.0 beschäftigt. Mir war es wichtig zu verstehen, welche Diagrammtypen verfügbar sind und welche sich für die Dokumentation von Algorithmen eignen. Besonders habe ich mich auf Activity Diagrams konzentriert, da diese den Ablauf des Sieb des Eratosthenes am besten darstellen. Für die Erstellung des Flowcharts habe ich verschiedene Werkzeuge recherchiert und mich schließlich für draw.io entschieden.

Überblick über UML 2.0

UML 2.0 (Unified Modeling Language) ist eine standardisierte Modellierungssprache, die in der Softwareentwicklung verwendet wird, um Systeme zu visualisieren, zu dokumentieren und zu spezifizieren. Besonders wichtig für diese Aufgabe sind Activity Diagrams oder Flowcharts, da sie den Ablauf eines Algorithmus beschreiben und modellieren. Diese Diagramme eignen sich ideal zur Darstellung der einzelnen Schritte des Sieb des Eratosthenes.

Werkzeug zur Erstellung von Flowcharts

Für die Erstellung des Flowcharts habe ich mich für draw.io entschieden. Dieses Tool ist kostenlos, einfach zu bedienen, unterstützt den UML 2.0-Standard und bietet eine intuitive Benutzeroberfläche sowie viele Vorlagen. Dies macht draw.io zu einer ausgezeichneten Wahl für die Umsetzung der Aufgaben.

4.2 TA-2: Algorithmus für das Sieb des Eratosthenes in JOHNNY-ASSEMBLER

4.2.1 In Python zur Veranschaulichung

```
# Create a list of True values representing numbers from 0 to n
# Initially, we assume every number is prime
primes = [True] * (n + 1)

# Mark 0 and 1 as not prime because they are not prime numbers by definition
primes[0] = False
primes[1] = False
```

Abbildung 13: Screenshot aus WebStorm. Code in Python

Zu Beginn wird ein Array mit dem Namen `primes` erstellt, das alle Zahlen von 0 bis n abdeckt. Jede Zahl wird zunächst als „True“ markiert, was bedeutet, dass wir annehmen, dass jede Zahl eine Primzahl ist. Anschliessend werden 0 und 1 als „False“ markiert, da sie per Definition keine Primzahlen sind.

```
# Start with the smallest prime number, which is 2
p = 2

# Continue looping as long as p * p <= n.
# This means we only check up to the square root of n.
while (p * p <= n):
```

Abbildung 14: Screenshot aus WebStorm. Code in Python

Der Algorithmus beginnt mit der kleinsten Primzahl, die 2 ist. Diese wird als Startwert festgelegt, um mit der Überprüfung der Primzahlen zu beginnen.

```
# If p is still marked as prime (True)
if primes[p]:
    # Mark all multiples of p as False (they are not prime)
    for i in range(p * p, n + 1, p):
        primes[i] = False
    # Move to the next number
    p += 1
```

Abbildung 15: Screenshot aus WebStorm. Code in Python

Die Schleife wird so lange fortgesetzt, wie das Quadrat von `p` kleiner oder gleich `n` ist. Dies bedeutet, dass die Berechnungen nur bis zur Quadratwurzel von `n` durchgeführt werden müssen, um alle Primzahlen bis `n` zu finden.

Wenn `p` immer noch als Primzahl (True) markiert ist, werden alle Vielfachen von `p` (beginnend mit `p * p`) als „False“ markiert, da sie keine Primzahlen sind. Anschliessend wird `p` um 1 erhöht, um mit der nächsten Zahl fortzufahren.

```
# Create a list of prime numbers by collecting all the numbers still marked as True
prime_list = [i for i in range(n + 1) if primes[i]]

# Return the list of prime numbers
return prime_list
```

Abbildung 16: Screenshot aus WebStorm. Code in Python

Nachdem die Schleife abgeschlossen ist, wird eine Liste der Primzahlen erstellt, indem alle noch als „True“ markierten Zahlen gesammelt werden. Diese Liste enthält alle gefundenen Primzahlen bis n.

In diesem Fall wäre der Code zu «High-Level» und nicht sehr nahe am Johnny-Assembler. Deswegen haben Joey und ich einen Code dann im Java geschrieben, der viel näher ist an den Instructions als der Code, den ich in Python geschrieben habe. Das ist Joeys Version:

```
public static void main(String[] args) {
    final int n = 1000;
    final int s = 2;
    final int[] array = new int[n];

    // Initialisierung des Arrays
    for (int i = 0; i != n; i++)
        array[i] = s + i;

    // Sieb des Eratosthenes
    for (int i = 0; i < n; i++) {
        if (array[i] == 0)
            continue;
        int p = array[i];
        for (int j = p + p - s; j < n; j += p) {
            array[j] = 0;
        }
    }

    // Auffüllen der lücken
    for (int i = 0; i < n; i++) {
        if (array[i] != 0)
            continue;
        for (int j = i + 1; j < n; j++) {
            if (array[j] != 0) {
                array[i] = array[j];
                array[j] = 0;
                break;
            }
        }
    }
}
```


4.2.2 In JOHNNY Assembly

Joey hat noch einen Compiler geschrieben, was für mich leider noch zu weit ist, habe aber trotzdem direkt im Johnny-Assembler versucht die Schritte vom Java Code zu übernehmen. Leider noch nicht fertig geworden, da es tatsächlich kompliziert ist. Aber hier ist mal einen Lösungsansatz bzw. Mal die Auflistung der Zahlen (Array):

4.3 TA-3: Dokumentation des Algorithmus mittels Flowchart

Sieve of Eratosthenes - Optimized UML 2.0 Activity Diagram

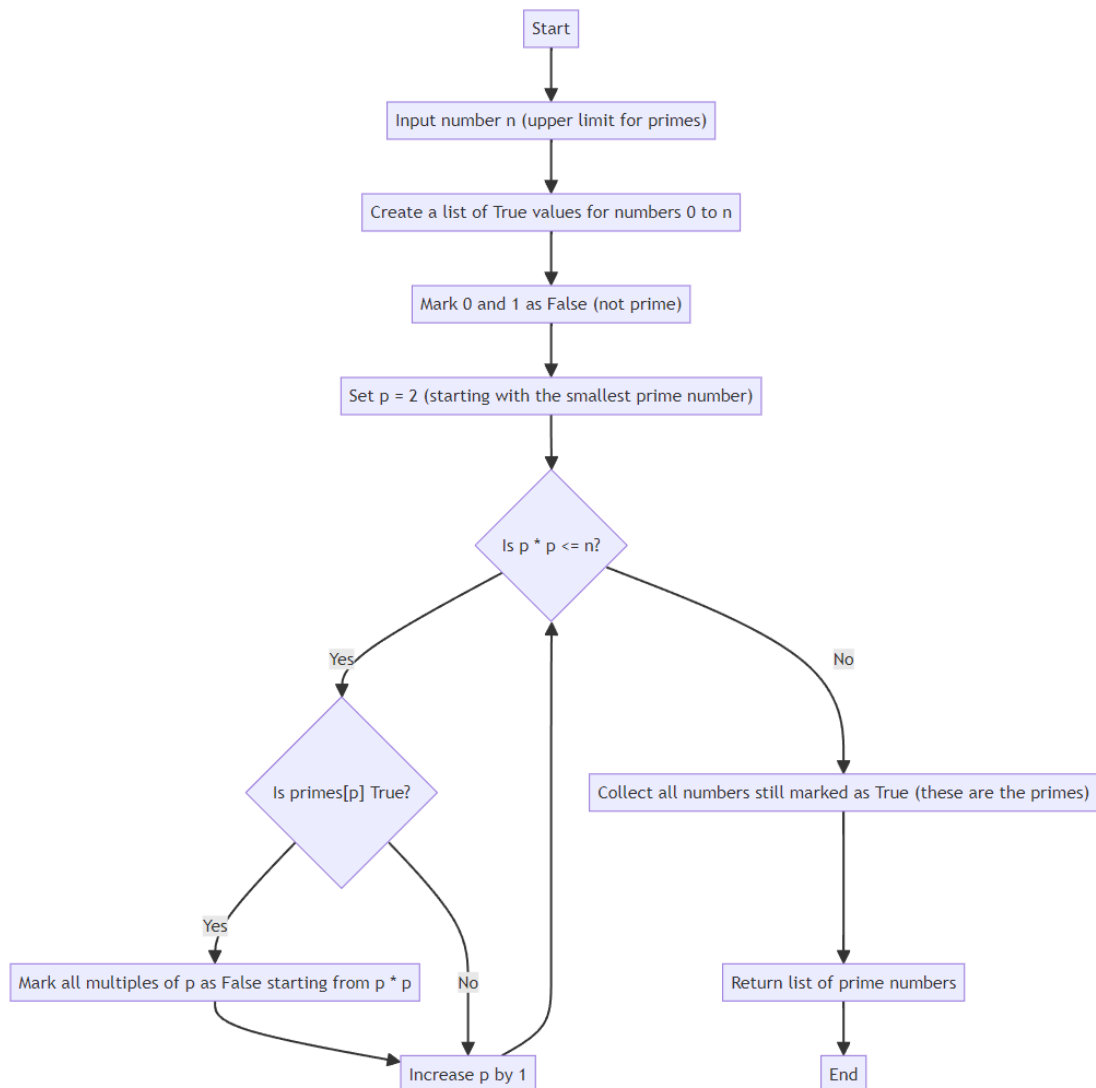


Abbildung 17: Diagramm in UML 2.0. Erstellt über WebStorm mit .html zusammen mit mermaid.js

Das Diagramm oben zeigt das Aktivitätsdiagramm des „Sieb des Eratosthenes“-Algorithmus, wie es in UML 2.0 dargestellt ist. Der Prozess ist in Schritte unterteilt, die verdeutlichen, wie der Algorithmus zur Berechnung von Primzahlen bis zu einer gegebenen Zahl n funktioniert.

Erklärung des Diagramms:

1. **Start:** Der Algorithmus beginnt mit dem Empfang einer Eingabezahl n , die als obere Grenze für die Berechnung der Primzahlen dient.
2. **Initialisierung:** Eine Liste von „True“-Werten für alle Zahlen von 0 bis n wird erstellt. Dies bedeutet, dass wir anfangs annehmen, dass jede Zahl eine Primzahl ist.
3. **Markierung von 0 und 1:** Die Zahlen 0 und 1 werden als „False“ markiert, da sie keine Primzahlen sind.
4. **Start mit der kleinsten Primzahl:** Der Algorithmus beginnt mit der Zahl 2, der kleinsten Primzahl.

5. **Schleife:**

- **Prüfung: Ist $p^2 \leq n$?** Der Algorithmus prüft, ob das Quadrat der aktuellen Zahl p kleiner oder gleich n ist. Wenn ja, fährt er mit der nächsten Überprüfung fort.
 - **Ist `primes[p]` True?** Wenn p als Primzahl (True) markiert ist, geht der Algorithmus dazu über, alle Vielfachen von p als „False“ zu markieren (nicht prim).
 - **Vielfache von p als „False“ markieren:** Alle Vielfachen von p (beginnend mit p^2) werden als „False“ markiert.
 - **Erhöhe p :** Danach wird p um 1 erhöht, und der Prozess wiederholt sich.
6. **Beendigung der Schleife:** Sobald $p^2 > n$ ist, wird die Schleife beendet.
7. **Sammeln der Primzahlen:** Die Zahlen, die immer noch als „True“ markiert sind, werden gesammelt, da dies die Primzahlen sind.
8. **Rückgabe der Primzahlen:** Die Liste der Primzahlen wird als Ergebnis zurückgegeben.
9. **Ende:** Der Algorithmus endet.

Das UML Diagramm würde natürlich anders aussehen mit dem Java Code, was dann noch aufgeführt werden wird.

Abbildungsverzeichnis

Abbildung 1: Johnny Version 1.0.1 (Screenshot).....	3
Abbildung 2: Schritte der Makrobefehle (Screenshot)	4
Abbildung 3: Register 099 (Screenshot)	5
Abbildung 4: Register #000 bis #005 des Countdowns (Screenshot)	5
Abbildung 5: Register #006 bis #010 des Countdowns (Screenshot)	6
Abbildung 6: Abschluss des Algorithmus (Screenshot)	6
Abbildung 7: Register Werte #015 bis #026 (Screenshot).....	6
Abbildung 8: Algorithmus Division mit Rest mit Johnny (Screenshot)	8
Abbildung 9: Register #000 bis #005 Division mit Rest mit Johnny (Screenshot)	9
Abbildung 10: Register #006 bis #011 Division mit Rest mit Johnny (Screenshot)	9
Abbildung 11: Register #012 bis #017 Division mit Rest mit Johnny (Screenshot)	10
Abbildung 12: Register #018 bis #023 Division mit Rest mit Johnny (Screenshot)	10
Abbildung 13: Screenshot aus WebStorm. Code in Python	13
Abbildung 14: Screenshot aus WebStorm. Code in Python	13
Abbildung 15: Screenshot aus WebStorm. Code in Python	13
Abbildung 16: Screenshot aus WebStorm. Code in Python	14
Abbildung 17: Diagramm in UML 2.0. Erstellt über WebStorm mit .html zusammen mit mermaid.js	16

Tabellenverzeichnis

Tabelle 1: Lösungsbeschreibung Countdown	6
Tabelle 2: Lösungsbeschreibung IDIV	10

Literaturverzeichnis

Klapper, M. (2024). *Moodle*. Retrieved from
<https://moodle.ffhs.ch/mod/assign/view.php?id=4713977>

Abkürzungsverzeichnis

ADR	Adresse
ALU	Arithmetic Logic Unit
CU	Control Unit
GTI	Grundlagen der technischen Informatik PVAPortfolioarbeit
Reg.	Register

Eigenständigkeitserklärung für Portfolio-Arbeit Nr. 1

Hiermit erkläre ich, dass:

- ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe.
- alle sinngemäss oder wörtlich übernommenen Passagen aus fremden Quellen kenntlich gemacht wurden.
- alle verwendeten Hilfsmittel ordnungsgemäss angegeben sind.
- keine anderen als die angegebenen Hilfsmittel verwendet wurden.
- das Thema dieser Arbeit oder Teile davon nicht bereits Gegenstand eines anderen Moduls waren, es sei denn, dies wurde vorher mit der entsprechenden Betreuungsperson vereinbart.
- mir bewusst ist, dass meine Arbeit auf Plagiate überprüft wird und ich der FFHS dafür die nötigen Rechte einräume.

Valentino Totaro

Fernfachhochschule Schweiz (FFHS)