Die Programmiersprache $P_{(asta)}$

Uwe Naumann

Informatik 12 (STCE)
RWTH Aachen

Vorwort

In dieser Vorlesung stelle ich die Programmiersprache *P*(asta) vor. Habe sogar einen Compiler (bzw. Interpreter) dafür geschrieben!

Was auch immer in *P*(asta) geht, kann (und werde) ich live mithilfe eines Stapels von Tellern und einer Tüte voller Nudeln zeigen. Nachdem ich diverse Nudelumschichtungsdurchgänge, z.B. zur Berechnung der Fakultät einer gegeben Zahl (von Nudeln), hinter mich gebracht haben werde, wird sich der Nutzen der Programmiersprache *P*(asta) eindrucksvoll bestätigt haben.

Die Programmiersprache *P*(asta) ist in der Entwicklung. Version 1 soll nicht der Weisheit letzter Schluss sein. Einerseits sind weitere Beispiele herzlich willkommen! Sendet sie bitte inklusive

1. einer kurzen Beschreibung der zu lösenden Aufgabe
2. eines möglichst detailliert dokumentieren Quellcodes
3. einer (oder bei Bedarf auch mehrerer) dokumentierten Beispielsitzung

an naumann@stce.rwth-aachen.de ... vielen Dank! Andererseits bin ich auch für Ideen zur Weiterentwicklung von *P*(asta) dankbar. Dabei soll vor allem die Natürlichsprachigkeit von *P*(asta) sichergestellt werden, was gar nicht so einfach ist, wie es klingt ...

Viel Spaß

Uwe Naumann

Inhaltsverzeichnis

Kapitel 1

Sprache

P(asta) Programme beginnen mit der Bereitstellung von Speicherplatz in Form von Tellern (siehe Abschnitt 1.1) und werden durch

```
Bis bald!
```

beendet, worauf sich das Programm mittels

```
Guten Appetit!
```

verabschiedet.

1.1 Variablen und Werte

Der Hauptspeicher des Computers wird durch Teller repräsentiert. Dessen Größe wird mithilfe des [Hole dir ... Teller]-Sprachkonstrukts explizit festgelegt:

```
Hole dir <wieviele> Teller!
```

Individuelle Anweisungen werden mit Ausrufezeichen abgeschlossen. So wird z.B. mittels

```
Hole dir 3 Teller!
```

Speicher der Größe 3 reserviert (alloziert).

Auf den Tellern liegende Nudeln stellen (ganzzahlige) Werte dar. Die aktuelle Version von *P*(asta) kennt nur ganze Zahlen. Nach ihrer Allozierung sind alle Teller leer.

Dasselbe in C++ In C++ könnte man ein Feld der Größe 3 von Elementen ganzzahligen Typs allozieren und zu Null initialisieren.

```
int t[3]={0,0,0};
```

1.2 Zuweisungen

Das Hinzufügen von Nudeln zu Tellern geschieht durch Verwendung des [Lege ... auf]-Sprachkonstrukts:

```
Lege <wieviele> Nudeln auf <welchen Teller>!
```

Analog, erlaubt das [Nimm ... von]-Sprachkonstrukt das Entfernen von Nudeln:

```
Nimm <wieviele> Nudeln von <welchem Teller>!
```

Der Zugriff auf die einzelnen Teller geschieht über eine eindeutige Nummer $1, 2, \ldots$, z.B. Teller 1, Teller 2, und Teller 3. Alle Zuweisungen sind inkrementell bzw. dekrementell. So fügt z.B. die Anweisung

```
Lege 9 Nudeln auf Teller 1!
```

neun zu den bereits auf Teller 1 liegenden Nudeln hinzu. Mittels

```
Nimm 4 Nudeln von Teller 1!
```

werden vier Nudeln von Teller 1 entfernt.

Dasselbe in C++ In C++ könnte man mittels

```
t[0]+=9;
```

neun Nudeln auf Teller 1 legen bzw. mittels

```
t[0]-=4;
```

davon vier wieder entfernen.

1.3 Ein- und Ausgabe

Zur Ein- bzw. Ausgabe stellt *P*(asta) [Frag mich]- bzw. [Sag mir]-Sprachkonstrukte zur Verfügung.

Die Anzahl der auf einem Teller liegenden Nudeln kann durch Benutzereingabe explizit gesetzt werden, z.B. resultiert

```
Frag mich, wieviele Nudeln auf Teller 1 liegen sollen!
```

in der Ausgabe

```
Wieviele Nudeln liegen auf Teller 1?
:-)
```

Von den Benutzer(innen) des Programms wird daraufhin die Eingabe einer ganzen Zahl, z.B. 3, erwartet. Infolgedessen wird die Anzahl der Nudeln auf Teller 1 auf drei gesetzt, was einer nichtinkrementellen Zuweisung von drei (Nudeln) zu Teller 1 entspricht.

Möchte man wissen, wieviel Nudeln auf einem Teller liegen, so kann man das z.B. mittels

```
Sag mir, wieviele Nudeln auf Teller 1 liegen!
```

abfragen. Weitere zulässige Fragen in P(asta) resultieren in Ja (1) order Nein (0) als möglich Antworten. So kann man z.B. mithilfe der Anweisung

```
Sag mir, ob Teller 1 voller als Teller 2 ist!
```

erfahren, ob auf Teller 1 mehr Nudeln liegen als auf Teller 2. Alternativ, könne man mittels

```
Sag mir, ob Teller 1 genauso voll wie Teller 2 ist!
```

erfragen, ob auf beiden Tellern die gleiche Anzahl von Nudeln liegt.

Weitere mögliche Bedingungen sind

```
... Teller 1 leerer als Teller 2 ...
... Teller 1 nicht leerer als Teller 2 ...
... Teller 1 nicht voller als Teller 2 ...
... Teller 1 nicht genauso voll wie Teller 2 ...
... Teller 1 leer...
... Teller 1 nicht leer...
```

wobei die zu vergleichenden Teller natürlich beliebige innerhalb des allozierten Indexbereichs liegende Nummern haben können. Desweiteren können Bedingungen durch und bzw. oder verknüpft werden, z.B. wäre

```
Teller 1 voller als Teller 2
oder
Teller 1 nicht genauso voll wie Teller 2
```

äquivalent zu

```
Teller 1 nicht leerer als Teller 2
```

Dasselbe in C++ Eine Aufforderung zur Eingabe in *P*(asta) kann in C++ unter Verwendung von

```
#include<iostream>
using namespace std;
```

z.B. wie folgt implementiert werden.

```
cout << "Wieviele Nudeln liegen auf Teller 1?" << endl;
cin >> t[0];
```

Zur Ausgabe der Anzahl von Nudeln auf Teller 1 schreibt man z.B.

```
cout << "Auf Teller 1 ";
if (t[0]==1)
  cout << "liegt 1 Nudel."
else
  cout << "liegen " << t[0] << " Nudeln." << endl;
```

Die Ausgabe von Wahrheitswerten kann z.B. wie folgt geschehen.

```
bool c=t[0]==t[1];
// bool c=t[0]<t[1];
// bool c=!(t[0]<t[1]);
// bool c=!(t[0]>t[1]);
// bool c=t[0]!=t[1];
// bool c=t[0]==0;
// bool c=t[0]>0;
if (c)
   cout << "Ja." << endl;
else
   cout << "Nein." << endl;
```

1.4 Optionale Anweisungen

Optionale Anweisungen, deren Ausführung vom Wahrheitswert einer Bedingung abhängt, werden mittels eines [Mache folgendes ... wenn]-Sprachkonstrukts implementiert:

```
Mache folgendes,
  <anweisung>,
wenn <Bedingung>!
```

Dabei wird die optionale Anweisung klein geschrieben und mit einem Komma abgeschlossen. So wird z.B. aufgrund

```
Mache folgendes,
  lege 1 Nudel auf Teller 2,
wenn Teller 1 voller als Teller 2 ist!
```

nur dann eine zusätzliche Nudel auf Teller 1 gelegt, wenn auf Teller 1 bereits mehr Nudeln als auf Teller 2 liegen.

Dasselbe in C++ Bedingte Folgen von Anweisungen können in C++ in mittels möglicherweise geschachtelter if-Anweisungen implementiert werden, z.B.

```
if (c) {
  ...
}
```

wobei c einen entsprechenden Wahrheitswert bezeichnet.

1.5 Schleifen

Soll eine Anweisungen sooft wiederholt werden bis eine zu spezifizierende Bedingung nicht mehr erfüllt ist, dann bietet sich die Verwendung des [Wiederhole ... solange]-Sprachkonstrukts an:

```
Wiederhole folgendes,
  <anweisung>,
solange <Bedingung>!
```

Wie bei optionalen Anweisungen wird die potentiell zu wiederholende Anweisung klein geschrieben und mit einem Komma abgeschlossen. So wird z.B. aufgrund

```
Wiederhole folgendes,
  lege 1 Nudel auf Teller 2,
solange Teller 1 voller als Teller 2 ist!
```

wiederholt eine zusätzliche Nudel auf Teller 2 gelegt, bis auf Teller 2 genauso viele Nudeln wie auf Teller 1 liegen.

Dasselbe in C++ *P*(asta)-Schleifen können in C++ wie folgt imlementiert werden

```
While (c) {
  ...
}
```

wobei c einen entsprechenden Wahrheitswert bezeichnet.

1.6 Anweisungssequenzen

Sequenzen von Anweisungen werden durch einfache Aneinanderreihung von Anweisungen gebildet:

```
<Anweisung>!
<Anweisung>!
<Anweisung>!
```

Z.B. wird man mittels

```
Frag mich,
  wieviele Nudeln auf Teller 1 liegen sollen!
Sag mir,
  wieviele Nudeln auf Teller 1 liegen!
```

erst zur Eingabe eines Wertes aufgefordert, welcher dann umgehend wieder ausgegeben wird.

Innerhalb von optionalen Anweisungen ([Mache folgendes ... wenn]) und Schleifen ([Wiederhole folgendes ... solange]) werden einzelne Anweisungen klein geschrieben, mit Komma abgeschlossen und mittels und getrennt:

```
...
  <anweisung>, und
  <anweisung>, und
  <anweisung>,
...
```

Im folgenden Beispiel wird ein Wert solange abgefragt und unmittelbar wieder ausgegeben, bis eine Null eingegeben wird.

```
Lege 1 Nudel auf Teller 1!
Wiederhole folgendes,
  frag mich,
    wieviele Nudeln auf Teller 1 liegen sollen,
  und
  sag mir,
    wieviele Nudeln auf Teller 1 liegen,
solange Teller 1 nicht leer ist!
```

Kapitel 2

Beispiele

2.1 Grundrechenarten

2.1.1 Addition

Zur Addition zweier natürlicher Zahlen werden die beiden Summanden durch Nudeln auf den Tellern 1 und 2 repräsentiert. Entsprechende Zahlen von Nudeln werden dann jeweils auf Teller 3 gelegt, was das gesuchte Resultat ergibt.

P(asta) **Program**

```
// Addition der Anzahl der Nudeln auf Teller 1
// und der Anzahl der Nudeln auf Teller 2
// Resultat auf Teller 3

// Uwe Naumann, 2012


Hole dir 3 Teller!
// Eingabe des ersten Summanden
Frag mich,
  wieviele Nudeln auf Teller 1 liegen sollen!
// Eingabe des zweiten Summanden
Frag mich,
  wieviele Nudeln auf Teller 2 liegen sollen!
// Addition
Lege
```

```
  soviele Nudeln, wie auf Teller 1 liegen,
auf Teller 3!
Lege
  soviele Nudeln, wie auf Teller 2 liegen,
auf Teller 3!
// Ausgabe des Resultats
Sag mir,
  wieviele Nudeln auf Teller 3 liegen!
Bis bald!
```

Beispielsitzung

```
Wieviele Nudeln liegen auf Teller 1?
:-) 47
:
Wieviele Nudeln liegen auf Teller 2?
:-) 34
:
Auf Teller 3 liegen 81 Nudeln.
:
Guten Appetit!
```

Dasselbe in C++

```
#include<iostream>
using namespace std;

int main() {
        int t[3]={0,0,0};
        cin >> t[0];
        cin >> t[1];
        t[2]+=t[0];
        t[2]+=t[1];
        cout << t[2] << endl;
        return 0;
}
```

2.1.2 Subtraktion

Zur Subtraktion zweier natürlicher Zahlen wird der Minuend durch Nudeln auf den Teller 1 und der Subtrahend entsprechend durch Nudeln auf

Teller 2 repräsentiert. Dabei muss sichergestellt werden, dass der Subtrahend nicht größer als der Minuend ist. Sollte das nicht der Fall sein, so wird der Wahrheitswert der Bedingung

```
Teller 2 voller als Teller 1
```

also Ja auf den Bildschirm ausgegeben. Anderenfalls werden soviele Nudeln, wie auf Teller 2 liegen, von Teller 1 entfernt. Die auf Teller 1 verbleibende Anzahl von Nudeln stellt das Resultat der Subtraktion dar.

P(asta) Program

```
// Subtraktion der Anzahl der Nudeln auf Teller 2
// von der Anzahl der Nudeln auf Teller 1
// Resultat auf Teller 1 falls nicht negativ

// Uwe Naumann, 2012

Hole dir 2 Teller!
// Eingabe des Minuenden
Frag mich,
  wieviele Nudeln auf Teller 1 liegen sollen!
// Eingabe des Subtrahenden
Frag mich,
  wieviele Nudeln auf Teller 2 liegen sollen!
// Ist das Resultat <0?
Sag mir,
  ob Teller 2 voller als Teller 1 ist!
// Falls nicht ...
Mache folgendes,
// Subtraktion
  nimm
    soviele Nudeln, wie auf Teller 2 liegen,
  von Teller 1,
  und
// Ausgabe des Resultats
  sag mir,
    wieviele Nudeln auf Teller 1 liegen,
wenn Teller 2 nicht voller als Teller 1 ist!
Bis bald!
```

Beispielsitzung

```
Wieviele Nudeln liegen auf Teller 1?
:-) 47
:
Wieviele Nudeln liegen auf Teller 2?
:-) 34
:
Nein.
:
Auf Teller 3 liegen 13 Nudeln.
:
Guten Appetit!
```

Dasselbe in C++

```
#include<iostream>
using namespace std;

int main() {
        int t[2]={0,0};
        cin >> t[0];
        cin >> t[1];
        cout << (t[1]>t[0]) << endl;
        if (!(t[1]>t[0])) {
          t[0]-=t[1];
          cout << t[0] << endl;
        }
        return 0;
}
```

2.1.3 Multiplikation

Zur Multiplikation zweier natürlicher Zahlen werden die beiden Faktoren durch Nudeln auf den Tellern 1 und 2 repräsentiert. Während einer Anzahl von Schleifeniterationen, die gleich der auf Teller 2 liegenden Zahl von Nudeln ist, werden jeweils soviele Nudeln auf Teller 3 gelegt, wie sich auf Teller 1 befinden. Das Resultat ergibt sich anschließend zur Anzahl der auf Teller 3 liegenden Nudeln.

P(asta) Program

```
// Multiplikation der Anzahl der Nudeln auf Teller 1
// mit der Anzahl der Nudeln auf Teller 2
// Resultat auf Teller 3

// Uwe Naumann, 2012

Hole dir 3 Teller!
// Eingabe des ersten Faktors
Frag mich,
  wieviele Nudeln auf Teller 1 liegen sollen!
// Eingabe des zweiten Faktors
Frag mich,
  wieviele Nudeln auf Teller 2 liegen sollen!
// Produkt
Wiederhole folgendes,
  lege
    soviele Nudeln, wie auf Teller 1 liegen,
  auf Teller 3,
  und
  nimm
    1 Nudel von Teller 2,
solange Teller 2 nicht leer ist!
// Ausgabe des Resultats
Sag mir,
  wieviele Nudeln auf Teller 3 liegen!
Bis bald!
```

Beispielsitzung

```
Wieviele Nudeln liegen auf Teller 1?
:-) 47
:
Wieviele Nudeln liegen auf Teller 2?
:-) 34
:
Auf Teller 3 liegen 1598 Nudeln.
:
Gute Appetit!
```

Dasselbe in C++

```
#include<iostream>
using namespace std;

int main() {
        int t[3]={0,0,0};
        cin >> t[0];
        cin >> t[1];
        while (t[1]>0) {
          t[2]+=t[0];
          t[1]-=1;;
        }
        cout << t[2] << endl;
        return 0;
}
```

2.1.4 Division

Zur ganzzahligen Division zweier natürlicher Zahlen wird der Dividend durch Nudeln auf den Teller 1 und der Divisor entsprechend durch Nudeln auf Teller 2 repräsentiert. Zur Berechnung des Quotienten und des Rests werden innerhalb einer Schleife jeweils soviele Nudeln, wie auf Teller 2 liegen, von Teller 1 entfernt. Pro Schleifeniteration wird der Wert des Quotienten auf Teller 3 um eins erhöht. Die Schleife wird verlassen, sobald auf Teller 1 weniger Nudeln liegen als auf Teller 2. Die auf Teller 1 verbleibenden Nudeln stellen den Rest der ganzzahligen Division dar.

P(asta) Program

```
// Division der Anzahl der Nudeln auf Teller 1
// durch die Anzahl der Nudeln auf Teller 2
// Resultat auf Teller 3; Rest auf Teller 1

// Uwe Naumann, 2012

Hole dir 3 Teller!
// Eingabe des Dividenden
Frag mich,
  wieviele Nudeln auf Teller 1 liegen sollen!
// Eingabe des Divisors
Frag mich,
```

```
  wieviele Nudeln auf Teller 2 liegen sollen!
// Division
Wiederhole folgendes,
  mache folgendes,
    // Reduziere Rest
    nimm
      soviele Nudeln, wie auf Teller 2 liegen,
    von Teller 1,
    und
    // Inkrementiere Quotient
    lege
      1 Nudel
    auf Teller 3,
  wenn
    Teller 1 nicht leerer als Teller 2
  ist,
solange
  Teller 1 nicht leerer als Teller 2
ist!
// Ausgabe des ganzzahligen Quotienten
Sag mir,
  wieviele Nudeln auf Teller 3 liegen!
// Ausgabe des Rests
Sag mir,
  wieviele Nudeln auf Teller 1 liegen!
Bis bald!
```

Beispielsitzung

```
Wieviele Nudeln liegen auf Teller 1?
:-) 47
:
Wieviele Nudeln liegen auf Teller 2?
:-) 34
:
Auf Teller 3 liegt 1 Nudel.
:
Auf Teller 1 liegen 13 Nudeln.
:
Guten Appetit!
```

Dasselbe in C++

```
#include<iostream>
using namespace std;

int main() {
        int t[3]={0,0,0};
        cin >> t[0];
        cin >> t[1];
        while (!(t[0]<t[1])) {
          if (!(t[0]<t[1])) {
            t[0]-=t[1];
            t[2]+=1;
          }
        }
        cout << t[2] << endl;
        cout << t[0] << endl;
        return 0;
}
```

2.2 Was schon der kleine Gauss machen musste ...

Vom Schüler Carl-Friedrich Gauss wird sich folgende Geschichte erzählt:

"Eines Tages hatte sein Mathelehrer mal keine Lust und wollte sich durch eine seiner Meinung nach besonders aufwändige Aufgabe etwas Freizeit beschaffen. Er ließ die Klasse alle Zahlen von 1 bis 100 aufsummieren, in der Hoffnung, dass diese damit wenigstens 15 Minuten beschäftigt sei.

Anstatt wild darauf los zu addieren, schaltete der kleine Gauss seinen schon damals hellen Kopf ein und erkannte, dass sowohl 1+100 als auch 2+99 als auch 3+98 ... als auch 50+51 jeweils 101 ergaben. Er zählte insgesamt 50 dieser Summen und meldete nach wenigen Sekunden mit dem Resultat 5050=50*101 zurück. Die erhoffte Pause des Mathelehrers wurde siginfikant verkürzt..."

Dank *P*(asta) müssen wir nicht ganz so schlau wie Gauss sein. Der Computer ist bekanntermaßen bei der Summation großer Zahlenfolgen signifikant effizienter als wir. Übergeben wir also ihm den Job...

***P*(asta) Program**

```
// Summe aller Zahlen bis zur Anzahl der Nudeln
// auf Teller 1
// Resultat auf Teller 3

// Uwe Naumann, 2012


Hole dir 3 Teller!
// Die Summe der ersten wieviel Zahlen
// soll berechnet werden?
Frag mich,
  wieviele Nudeln auf Teller 1 liegen sollen!
Wiederhole folgendes,
  lege 1 Nudel auf Teller 2,
  und
  lege
    soviele Nudeln, wie auf Teller 2 liegen,
  auf Teller 3,
solange Teller 2 leerer als Teller 1 ist!
Sag mir,
  wieviele Nudeln auf Teller 3 liegen!
Bis bald!
```

Beispielsitzung

```
Wieviele Nudeln liegen auf Teller 1?
:-) 10
:
Auf Teller 3 liegen 55 Nudeln.
:
Guten Appetit!
```

Dasselbe in C++

```
#include<iostream>
using namespace std;

int main() {
        int t[3]={0,0,0};
        cin >> t[0];
        while (t[1]<t[0]) {
```

```
        t[1]+=1;
        t[2]+=t[1];
      }
      cout << t[2] << endl;
      return 0;
}
```

2.3 Fakultät

Die Berechnung der Fakultät einer natürlichen Zahl ähnelt dem Problem des kleinen Gauss bis auf die Tatsache, dass die Elemente der Zahlenfolge nicht addiert sondern multipliziert werden. Desweiteren ist per Definition $0! = 1$, was durch die Initialisierung des Resultatstellers (Teller 3) mit einer Nudel gewährleistet ist. Anschließend werden die benötigten Multiplikationen durchgeführt und das Resultat ausgegeben.

P(asta) Programm

```
// Berechnung der Fakultaet
// Teller 1 haelt das Argument
// Das Resultat erscheint auf Teller 3

// Uwe Naumann, 2012

Hole dir 3 Teller!
// Eingabe des Arguments
Frag mich,
  wieviele Nudeln auf Teller 1 liegen sollen!
// 0!=1
Lege 1 Nudel auf Teller 3!
// Produktkette
Wiederhole folgendes,
  // Definition der Faktoren Teller 1 und Teller 2
  lege
    soviele Nudeln, wie auf Teller 3 liegen,
  auf Teller 2,
  und
  nimm
    soviele Nudeln, wie auf Teller 2 liegen,
  von Teller 3,
```

```
  und
  // Multiplikation Teller 1 * Teller 2
  // Resultat wird zu Teller 3 hinzugefuegt
  wiederhole folgendes,
    lege
      soviele Nudeln, wie auf Teller 1 liegen,
    auf Teller 3,
    und
    nimm 1 Nudel von Teller 2,
  solange Teller 2 nicht leer ist,
  und
  nimm 1 Nudel von Teller 1,
solange Teller 1 nicht leer ist!
// Ausgabe des Resultats
Sag mir,
  wieviele Nudeln auf Teller 3 liegen!
Bis bald!
```

Beispielsitzung

```
Wieviele Nudeln liegen auf Teller 1?
:-) 5
:
Auf Teller 3 liegen 120 Nudeln.
:
Guten Appetit!
```

Dasselbe in C++

```
#include<iostream>
using namespace std;

int main() {
        int t[3]={0,0,0};
        cin >> t[0];
        t[2]+=1;
        while (!(t[0]==0)) {
          t[1]+=t[2];
          t[2]-=t[1];
          while (!(t[1]==0)) {
            t[2]+=t[0];
```

```
            t[1]-=1;
          }
          t[0]-=1;
        }
        cout << t[2] << endl;
        return 0;
}
```