Aufgabe 4.2

// Algorithmus zur Berechnung eines Quadervolumens
// Es bedeuten: l Länge, b Breite, h Höhe, V Volumen

// Eingabe mit Überprüfung:
1. Wiederhole die folgenden Aktionen:
   1.1 Erfrage die Werte für *l*, *b* und *h* (Eingabe über Tastatur)
   bis *l*, *b* und *h* reelle Zahlen größer Null sind;

// Berechnung des Quadervolumens (Verarbeitung):
2. Bilde das Produkt $l \cdot b \cdot h$;
3. Weise das Ergebnis *V* zu;

// Ausgabe:
4. Gib den Wert von *V* auf dem Bildschirm aus;

Bemerkung: Bezgl. des Detaillierungsgrads der Schritte 2. und 3. sind andere Lösungen möglich.

Aufgabe 4.3

Die mehrfache Alternative

Falls *Selektor* =
$Wert_1$: $Verarbeitung_1$
$Wert_2$: $Verarbeitung_2$
...
sonst $Verarbeitung_n$

kann durch sukzessive Schachtelung der einfachen Alternative wie folgt ersetzt werden:

Falls $Bedingung_1$
dann $Verarbeitung_1$;
sonst Falls $Bedingung_2$
dann $Verarbeitung_2$;
sonst Falls ...
...
Falls $Bedingung_n$
dann $Verarbeitung_n$;

Aufgabe 4.4

a) Die Bedingungen $a > 0$ = w und $a < 0$ = w können niemals gleichzeitig erfüllt sein. Daher gilt folgende Vereinfachung:

Falls *a > 0*
dann *a := 2;* .

b) Das gegebene Kontrollkonstrukt realisiert die Betragsfunktion $b = |a|$ und kann wie folgt vereinfacht werden:
Falls *a > 0*
dann *b := a;*

sonst *b := -a;* .

Aufgabe 4.5

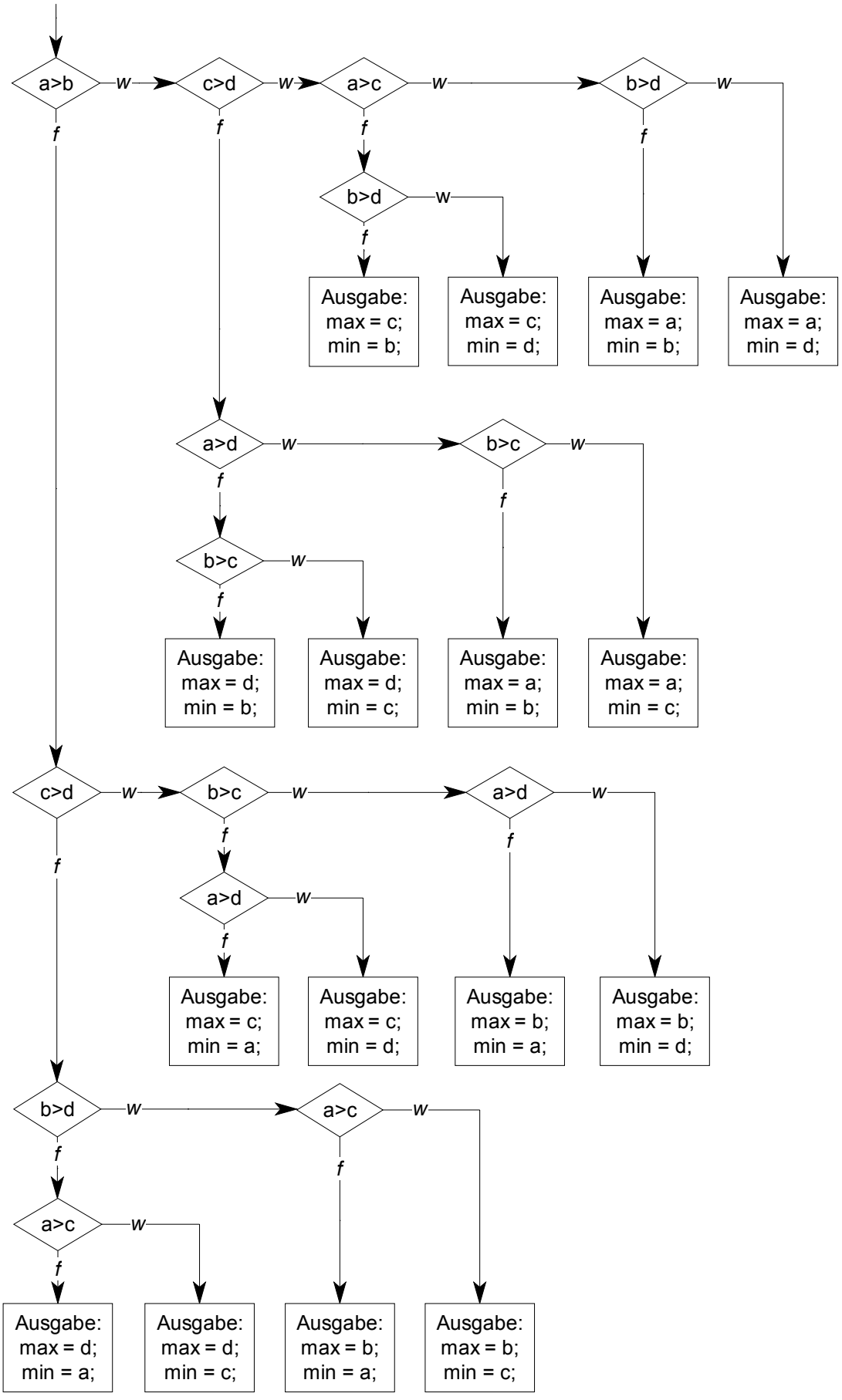
a) Max(*a*, *b*) bzw. Min(*a*, *b*) bezeichne im Folgendem die Maximum- bzw. Minimumfunktion im mathematischen Sinne.

// Eingabe der zu untersuchenden Zahlen
Tastatureingabe: a, b, c, d;

// Berechnung von Max(*a*, *b*, *c*, *d*) *und* Min(*a*, *b*, *c*, *d*);
Falls *a > b*
dann *{max := a; min := b;}*
sonst *{max := b; min := a;}* *// max* = Max(*a*, *b*); *min* = Min(*a*, *b*)
Falls *c > max*
dann *max := c;* *// c* > Max(*a*, *b*); *max* = Max(*a*, *b*, *c*)
Falls *c < min*
dann *min := c;* *// c* < Min(*a*, *b*); *min* = Min(*a*, *b*, *c*)
Falls *d > max*
dann *max := d;* *// d* > Max(*a*, *b*, *c*); *max* = Max(*a*, *b*, *c*, *d*)
Falls *d < min*
dann *min := d;* *// c* < Min(*a*, *b*, *c*); *min* = Min(*a*, *b*, *c*, *d*)

// Ausgabe des Ergebnisses
Bildschirmausgabe: "Max(a, b, c, d) = ", max;
Bildschirmausgabe: "Min(a, b, c, d) = ", min;

b) Prinzip: der Programmzustand wird nicht in Variablen sondern im Kontrollfluss festgehalten.



c) Lösung mit Hilfe eines lokalen Blocks *MaxMin2* zur Bestimmung von Max(*a*, *b*) und Min(*a*, *b*).

```
Block MaxMin4 ( )
    Daten a, b, c, d, max1, min1, max2, min2 ∈ R;
    Block MaxMin2 (ein: a, b ∈ R, mit: max, min ∈ R )
    // Berechnet Max(a, b) und Min(a, b);
    {   Falls a > b
        dann {max := a; min := b;}
        sonst {max := b; min := a;}
    }
{   // Eingabe der zu untersuchenden Zahlen
    Tastatureingabe: a, b, c, d;

    // Berechnung von Max(a, b, c, d) und Min(a, b, c, d);

    MaxMin2(ein: a, b; mit: max1, min1);
    MaxMin2(ein: c, d; mit: max2, min2);
    MaxMin2(ein: max1, max2; mit: max, max1);
    MaxMin2(ein: min1, min2; mit: min1, min);

    // Ausgabe des Ergebnisses
    Bildschirmausgabe: "Max(a, b, c, d) = ", max;
    Bildschirmausgabe: "Min(a, b, c, d) = ", min;
}
```

Aufgabe 4.6 (⇒ Progammdatei: A4_6.CPP)

Der folgende Lösungsvorschlag untersucht zunächst, ob die Punkte P1 und P2 nicht zusammenfallen und ermittelt, falls dies nicht gegeben ist, qualitativ die Steigung *s* der Geraden ($s = 0$, $s > 0$, $s < 0$, $s \to \infty$). Sodann wird für jeden Steigungsfall die Lage von P3 relativ zur Geraden bestimmt. Die Geradenparameter *a* und *s* werden nur berechnet, wenn dies notwendig ist.

```
Block Punkt_zu_Gerade ();

    Block Eingabe (aus: x1, y1, x2, y2, x3, y3 ∈ R )
    // Liest die Koordinaten für die Gerade durch (P1, P2) und den Punkt P3 ein.
    {
        Tastatureingabe:x1, y1, x2, y2, x3, y3;
    }

    Block Position (ein: x1, y1, x2, y2, x3, y3 ∈ R; aus: Text ∈ N )
        Daten s, a, y_3 ∈ R;
                Fall ∈ N;
    {
        Falls x1 = x2
        dann Falls y1 = y2
                dann Fall = 0;                          // Gerade unbestimmt
                sonst Falls y1 < y2                     // s → ∞
                        dann Fall := 11;
                        sonst Fall := 12;


        Falls y1 = y2                                   // s = 0
        dann Falls x1 < x2
                dann Fall = 21;
```

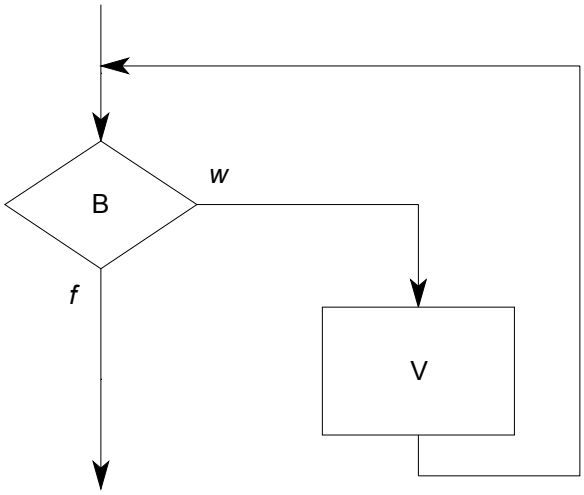
```
        sonst Falls x1 > x2
                dann Fall := 22;

Falls (x1 < x2) ∧ (y1 > y2)                     // s < 0
dann Fall := 31;
sonst Falls (x1 > x2) ∧ (y1 < y2)
        dann Fall := 32;

Falls (x1 > x2) ∧ (y1 > y2)                     // s > 0
dann Fall := 41;
sonst Falls (x1 < x2) ∧ (y1 < y2)
        dann Fall := 42;

Falls (Fall = 31) ∨ (Fall = 32) ∨ (Fall = 41) ∨ (Fall = 42)
dann { s := (y2 - y1) / (x2- x1);               // Berechnung der Geradenparameter
         a:= y1 - s ·x1;
         y_3 := s ·x3 + a;
       }

Falls Fall =
0:      Text := 4;
11:     {Falls x3 > x1
          dann Text := 2;
          sonst Falls x3 = x1
                  dann Text := 1;
                  sonst Text := 3;
        }

12:     {Falls x3 > x1
          dann Text := 3;
          sonst Falls x3 = x1
                  dann Text := 1;
                  sonst Text := 2;
        }

21:     {Falls y3 > y1
          dann Text := 3;
          sonst Falls y3 = y1
                  dann Text := 1;
                  sonst Text := 2;
        }

22:     {Falls y3 > y1
          dann Text := 2;
          sonst Falls y3 = y1
                  dann Text := 1;
                  sonst Text := 3;
        }


31, 42:
        {Falls y3 > y_3
          dann Text := 3;
          sonst Falls y3 = y_3
                  dann Text := 1;
                  sonst Text := 2;
```

```
            }

        32, 41:
            {Falls y3 > y_3
              dann Text := 2;
              sonst Falls y3 = y_3
                    dann Text := 1;
                    sonst Text := 3;
            }
    }

    Block Ausgabe (ein: Text ∈ N )
    // Gibt die Position des Punktes relativ zur Geraden aus.

    {
        Falls Text =
        1:      Bildschirmausgabe: „P3 liegt auf der Geraden“;
        2:      Bildschirmausgabe: „P3 liegt rechts der Geraden“;
        3:      Bildschirmausgabe: „P3 liegt links der Geraden“;
        4:      Bildschirmausgabe: „Wegen P1 = P2 ist das Problem unbestimmt“;
    }


Daten x1, y1, x2, y2, x3, y3 ∈ R;
       Text ∈ N;

{   Eingabe(aus: x1, y1, x2, y2, x3, y3);
    Position( ein: x1, y1, x2, y2, x3, y3; aus: Text);
    Ausgabe( ein: Text ∈ N);
}
```

Aufgabe 4.7

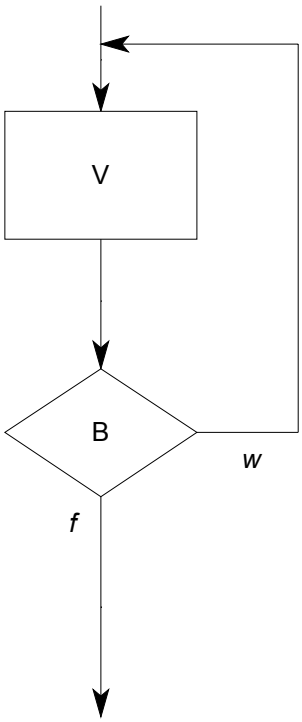
a) Wiederholung mit vorausgehender Bedingungsprüfung

Solange Bedingung (B) **führe aus**
Verarbeitung (V);



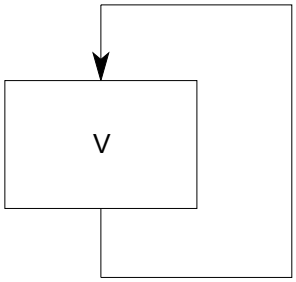
b) Wiederholung mit nachfolgender Bedingungsprüfung

Wiederhole
Verarbeitung (V)
solange Bedingung (B);



c) Wiederholung ohne Bedingungsprüfung

Wiederhole
Verarbeitung (V)
für immer;



Aufgabe 4.8 (⇒ Progammdatei: A4_8.CPP)

Der Block *Succ_Prim* verwendet einen Block *Test_Prim*, der eine gegebene Zahl *n* auf Primalität prüft (vgl. Beispiel 4.8).

```
Block Test_Prim (ein: n ∈ N; aus: b ∈ B2 )                    //  B2 = {w, f}
// Testet Zahlen n ∈ N auf Primalität
    Daten t ∈ N;
{   t := 3;
    Solange (t·t ≤ n) ∧ (n mod t ≠ 0) führe aus
        t := t + 2;
    Falls t·t > n
    dann b := w;
    sonst b := f;
}

Block Succ_Prim (ein: Prim ∈ N; aus: NPrim ∈ N )
// Ermittelt den Nachfolger NPrim ∈ N zu einer gegebenen Primzahl Prim ∈ N
    Daten b ∈ B2;
{   Falls Prim = 2
    dann Nprim = 3;
    sonst { Wiederhole
                { Prim := Prim + 2;                  // Bestimme den nächsten Kandidaten
                  Test_Prim( ein: Prim; aus: b ∈ B2 )
                }
            solange b ≠ w;                           // Prim ist Primzahl
            Nprim := Prim;
          }
}
```

Aufgabe 4.9 (⇒ Progammdatei: A4_9.CPP)

a) Iterative Lösung des Problems

$\binom{n}{0} = 1 \Rightarrow$ Intitialisierung der Zählschleife *binom := 1;*

$\binom{n}{k} = \binom{n}{k-1} \cdot \frac{n-k+1}{k}$ für k > 0

⇒ Schleifenkörper *binom := (binom·(n - i + 1)) div i;*
i ist die Zählvariable.

Es stellt sich die Frage, ob die ganzzahlige Division (div) tatsächlich ohne Rest aufgeht. Hierzu wird die folgende Darstellung des Binomialkoeffizienten betrachtet:

$$\binom{n}{k-1} = \frac{n!}{(k-1)!\cdot(n-k+1)!}$$

Wegen $n \geq k$ enthält $n! = 1 \cdot 2 \cdot \ldots \cdot n$ stets den Faktor k.

$\Rightarrow \binom{n}{k-1}$ ist stets ohne Rest durch k teilbar.

Damit kann der folgende Block *Binom_iter* zur iterativen Berechnung der Binominialkoeffizienten angegeben werden

```
Block Binom_iter (ein: n, k ∈ N; aus: binom ∈ N )
   Daten i ∈ N;
{  binom := 1;
   Für i := 1 solange i ≤ k mit i := i + 1 führe aus
      binom := (binom · (n - i + 1)) div i;
}
```

Man beachte: durch die äußere Klammersetzung ist gewährleistet, dass, unabhängig von der Reihenfolge der Auswertung von Ausdrücken in einer konkreten Programmiersprache, die Division auch tatsächlich aufgeht.

b) Rekursive Lösung

```
Block Binom_rek (ein: n, k ∈ N; aus: binom ∈ N )
{  Falls k = 0                                   // Endebedingung
   dann binom := 1;
   sonst { Binom_rek( ein: n, k-1; aus: binom );
           binom := (binom · (n - k + 1)) div k;
         }
}
```

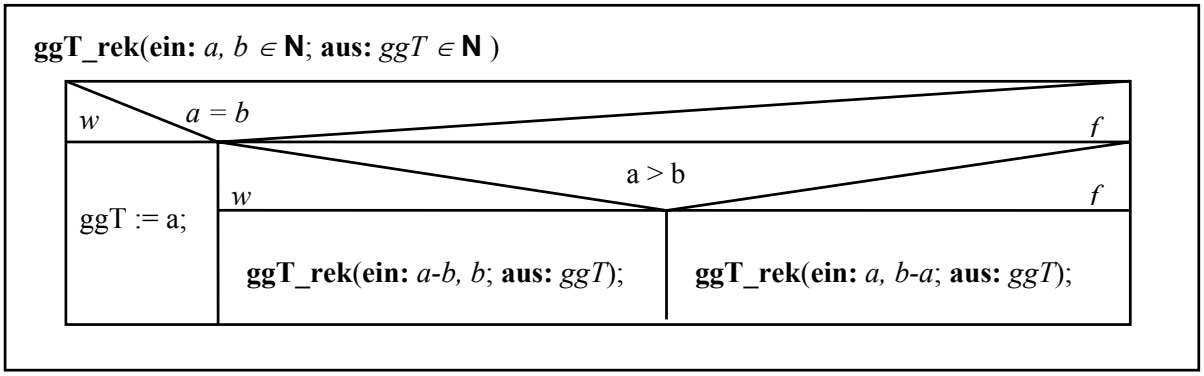
Aufgabe 4.10 (⇒ Progammdatei: A4_10.CPP)

Der im Beispiel 4.2 vorgestellte Algorithmus ist mathematisch durch die Gleichung

$$\mathrm{ggT}(a,b) = \begin{cases} a & \text{für } a = b \\ \mathrm{ggT}(a-b,b) & \text{für } a > b \\ \mathrm{ggT}(a,b-a) & \text{für } a < b \end{cases}$$

definiert.

Rekursiver Algorithmus



Aufgabe 4.11 (⇒ Progammdatei: A4_11.CPP)

Die Ziffer *d* mit dem niedrigsten Stellenwert einer Dezimalzahl *z* erhält man gemäß *d* = *z* mod 10. So erhält man z.B. für *z* = 1052 die letzte Ziffer *d* = 1052 mod 10 = 2. Mit dieser Überlegung kann der folgende Algorithmus angegeben werden:

```
Block Quersumme_rek (ein: Zahl ∈ N; aus: Quer ∈ N )
// Berechnet die Quersumme einer natürlichen Zahl
{  Falls Zahl = 0
   dann Quer := 0;
   sonst {   Quersumme_rek (ein: Zahl div 10; aus: Quer );
             Quer := Quer + Zahl mod 10;
         }
   }
```

Aufgabe 4.12 (⇒ Progammdatei: A4_12.CPP)

a) Rekursive Lösung

```
Block fn_rek (ein: n ∈ N; aus: f ∈ N )
// Berechnet rekursiv den ganzzahligen Logarithmus zur Basis 2 plus 1 von einer natürlichen Zahl n
{  Falls n = 1
   dann f := 1;
   sonst {   fn_rek (ein: (n div 2); aus: f );
             f := f + 1;
         }
}
```

b) Iterative Lösung

Block *fn_iter* (**ein:** $n \in \mathbf{N}$; **aus:** $f \in \mathbf{N}$)
// Berechnet iterativ den ganzzahligen Logarithmus zur Basis 2 plus 1 von einer natürlichen Zahl n
{ f := 1
solange *n > 1* **führe aus**
{ n := n div 2;
f := f + 1;
}
}