

Kapitel 11

Kombinatorik

Kombinatorik

Kombinatorik ist ein Teilgebiet der Mathematik, das sich, vereinfacht gesprochen, mit den Möglichkeiten beschäftigt, Elemente aus einer Menge in verschiedenartiger Weise auszuwählen und zusammenzustellen. Für die Programmierung interessieren uns im Wesentlichen zwei kombinatorische Fragestellungen:

- Wie viele verschiedene, einem bestimmten Schema folgende Auswahlen gibt es?
- Wie können alle, einem bestimmten Schema folgende Auswahlen erzeugt werden?

Die Antwort auf die erste Frage sagt uns, wie groß der Suchraum eines Problems ist. Durch die Beantwortung dieser Frage hoffen wir Formeln zu finden, die uns helfen, die zu erwartende Rechenzeit zur Lösungssuche vorab zu bestimmen.

Durch die Beantwortung der zweiten Frage hoffen wir, konkrete Algorithmen zu finden, die uns bei der "erschöpfenden Lösungssuche" helfen.

In diesem Kapitel formulieren wir vier kombinatorische Grundaufgaben, für die wir dann die beiden oben gestellten Fragen beantworten werden.

Kombinatorische Grundaufgaben

Man könnte die Ziehung der Lottozahlen (6 aus 49) in zweierlei Hinsicht ändern:

1. Man könnte zulassen, dass eine Zahl mehrfach gezogen wird. Man würde dazu die gezogene Kugel immer wieder in das Ziehungsgerät zurücklegen. Sechsmal die 1 wäre dann ein gültiger Tipp und eine mögliche Ziehung.
2. Man könnte verlangen, dass man die gezogenen Zahlen in der korrekten Ziehungsreihenfolge getippt haben muss, um zu gewinnen. 1, 2, 3, 4, 5, 6 wäre dann ein anderes Ziehungsergebnis als 6, 5, 4, 3, 2, 1. Das Ziehungsverfahren müsste man dazu nicht ändern. Man müsste nur das Ziehungsergebnis in der Reihenfolge der Ziehung bekannt geben.

Insgesamt ergeben sich durch die Kombination der beiden Varianten vier verschiedene Ziehungsmodalitäten.

1. Ziehung mit Wiederholungen und mit Beachtung der Reihenfolge
2. Ziehung mit Wiederholungen und ohne Beachtung der Reihenfolge
3. Ziehung ohne Wiederholungen und mit Beachtung der Reihenfolge
4. Ziehung ohne Wiederholungen und ohne Beachtung der Reihenfolge

Diese vier Fälle wollen wir im Folgenden diskutieren.

Permutationen mit Wiederholungen

In einer Lostrommel befinden sich n unterscheidbare Kugeln. Wir ziehen k mal eine Kugel aus dieser Lostrommel, notieren uns das Ziehungsergebnis und legen die Kugel wieder in die Trommel zurück. Gewonnen hat, wer die gezogenen Kugeln in der richtigen Reihenfolge getippt hat. Wir führen also eine Ziehung von k Kugeln aus einer Grundgesamtheit mit n Kugeln mit Zurücklegen und mit Beachtung der Reihenfolge durch.

Wie viele verschiedene Ziehungsergebnisse gibt es?

- Es gibt n Möglichkeiten, die erste Kugel zu ziehen.
- In jedem dieser n Fällen gibt es n Möglichkeiten, die zweite Kugel zu ziehen. Das sind insgesamt n^2 Fälle.
- In jedem dieser n^2 Fällen gibt es n Möglichkeiten, die dritte Kugel zu ziehen. Das macht insgesamt n^3 Fälle.
- Setzt man diese Überlegung auf alle k zu ziehenden Kugeln fort, so ergibt sich, dass es insgesamt n^k mögliche Ziehungsergebnisse gibt.

Eine Auswahl von k Elementen aus einer n -elementigen Menge, bei der es auf die Reihenfolge der Auswahl ankommt, bezeichnen wir als **n - k -Permutation mit Wiederholungen**, wenn in der Auswahl Wiederholungen von Elementen vorkommen dürfen.
Es gibt n^k solcher Permutationen.

Beispiel: Fahrradschloss



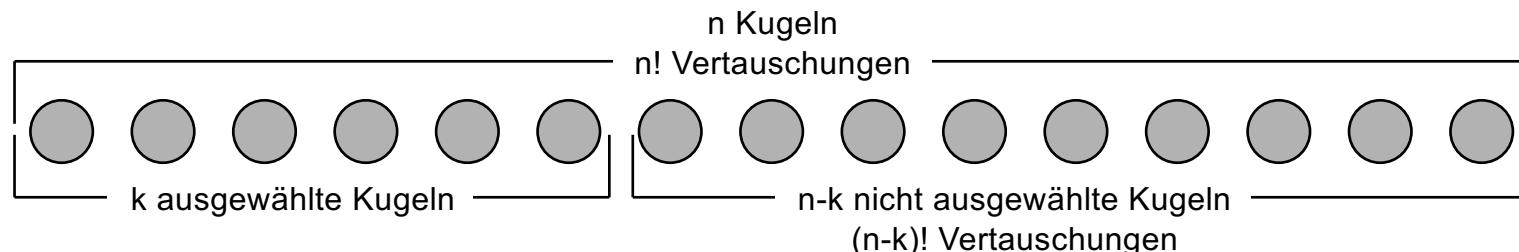
Das Schloss hat $k = 4$ Ringe mit jeweils $n = 9$ Einstellmöglichkeiten.
Es gibt insgesamt $n^k = 9^4 = 6561$ mögliche einstellbare Codes.

Permutationen ohne Wiederholungen

In einer Lostrommel befinden sich n unterscheidbare Kugeln. Wir ziehen k mal eine Kugel aus dieser Lostrommel, notieren uns das Ziehungsergebnis und legen die Kugel nicht wieder in die Trommel zurück. Gewonnen hat, wer die gezogenen Kugeln in der richtigen Reihenfolge geraten hat.

Wie viele verschiedene Ziehungsergebnisse gibt es?

- Es gibt n Möglichkeiten, die erste Kugel zu ziehen.
- In jedem dieser n Fällen gibt es $n-1$ Möglichkeiten, die zweite Kugel zu ziehen. Das sind insgesamt $n \cdot (n - 1)$ Fälle.
- In jedem dieser $n(n - 1)$ Fällen gibt es $n-2$ Möglichkeiten, die dritte Kugel zu ziehen. Das macht insgesamt $n(n - 1)(n - 2)$ Fälle.
- Setzt man diese Überlegung auf alle k zu ziehenden Kugeln fort, so ergibt sich, dass es insgesamt $n \cdot (n - 1) \cdots (n - k + 1)$ mögliche Ziehungsergebnisse gibt.



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

Eine Auswahl von k Elementen aus einer n -elementigen Menge, bei der es auf die Reihenfolge der Auswahl ankommt, bezeichnen wir als **$n-k$ -Permutation ohne Wiederholungen**, wenn in der Auswahl keine Wiederholungen von Elementen vorkommen dürfen.

Es gibt $\frac{n!}{(n-k)!}$ solcher Permutationen.

Beispiel: Finale im 100m Lauf bei den Olympischen Spielen



Im Finale kämpfen $n = 8$ Läufer um $k = 3$ Medaillen (Gold, Silber und Bronze).

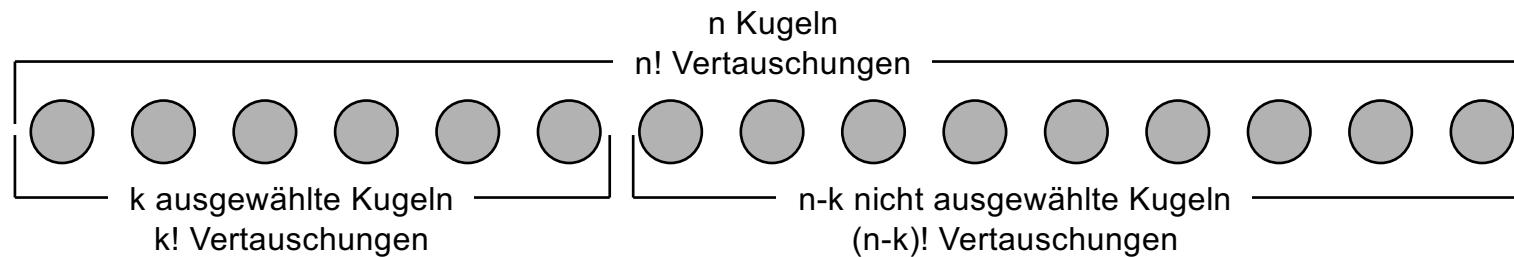
Es gibt insgesamt $\frac{n!}{(n-k)!} = 8 \cdot 7 \cdot 6 = 336$ mögliche Medaillenvergaben.

Kombinationen ohne Wiederholungen

In einer Lostrommel befinden sich n unterscheidbare Kugeln. Wir ziehen k mal eine Kugel aus dieser Lostrommel, notieren uns das Ziehungsergebnis und legen die Kugel nicht wieder in die Trommel zurück. Gewonnen hat, wer die gezogenen Kugeln ohne Berücksichtigung der Ziehungsreihenfolge richtig getippt hat.

Wie viele verschiedene Ziehungsergebnisse gibt es?

Wir legen die Kugeln in allen möglichen Reihenfolgen vor uns hin. Dazu gibt es $n!$ Möglichkeiten:



Die zu betrachtenden Ziehungen kommen dabei vielfach vor. Um nur die zu betrachtenden Ziehungen zu erhalten, müssen wir durch die Anzahl der Vertauschungen im vorderen Teil – das sind $k!$ – und durch die Anzahl der Vertauschungen im hinteren Teil – das sind $(n-k)!$ – dividieren. Es bleiben

$$\frac{n!}{k! (n - k)!}$$

Fälle übrig.

Binomialkoeffizienten

Der Ausdruck $\frac{n!}{k!(n-k)!}$ ist so bedeutsam für die Mathematik, dass man eine eigene Notation eingeführt hat. Man nennt diesen Ausdruck Binomialkoeffizient und schreibt:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \text{ (sprich "n über k")}$$

In gekürzter Form ist dies:

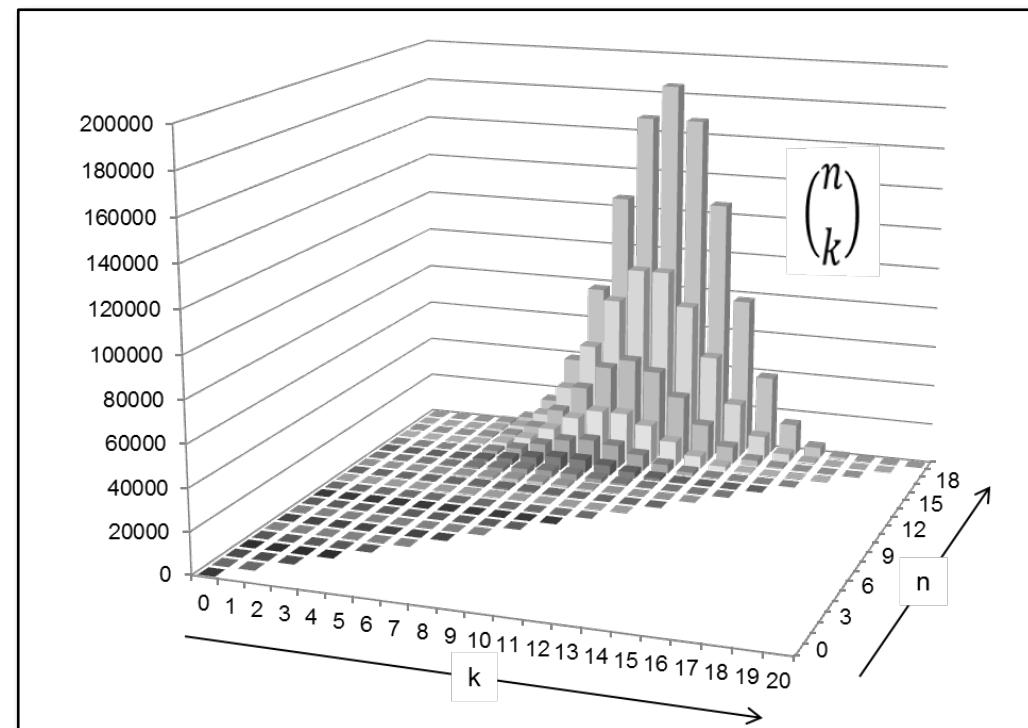
$$\binom{n}{k} = \frac{\overbrace{n(n-1)(n-2)\cdots(n-k+1)}^{k-\text{Faktoren}}}{\overbrace{k(k-1)(k-2)\cdots1}^{}}$$

Man schreibt einen Bruch mit k Faktoren in Zähler und Nenner- im Zähler von n und im Nenner von k absteigend.

Beispiel:

$$\binom{100}{5} = \frac{100 \cdot 99 \cdot 98 \cdot 97 \cdot 96}{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 75287520$$

Mehr dazu in den Übungen.



Eine Auswahl von k Elementen aus einer n -elementigen Menge, bei der es auf die Reihenfolge der Auswahl nicht ankommt, bezeichnen wir als **n - k -Kombination ohne Wiederholungen**, wenn in der Auswahl keine Wiederholungen von Elementen vorkommen dürfen.

Es gibt $\binom{n}{k}$ solcher Kombinationen.

Beispiel: Ziehung der Lottozahlen "6 aus 49"



Es gibt $n = 49$ Kugeln, von denen $k = 6$ gezogen werden.

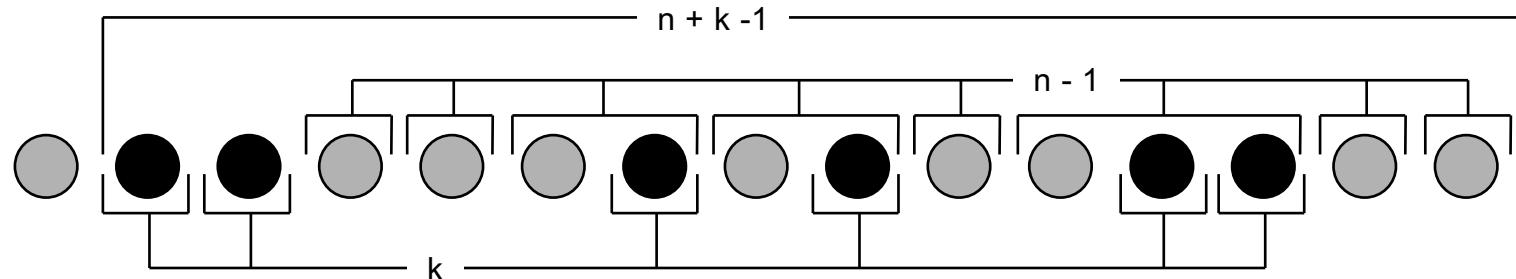
Dann gibt es $\binom{n}{k} = \binom{49}{6} = \frac{49 \cdot 48 \cdot 47 \cdot 46 \cdot 45 \cdot 44}{6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 13983816$ mögliche Ziehungsergebnisse.

Kombinationen mit Wiederholungen

In einer Lostrommel befinden sich n unterscheidbare Kugeln. Wir ziehen k mal eine Kugel aus dieser Lostrommel, notieren uns das Ziehungsergebnis und legen die Kugel wieder in die Trommel zurück. Gewonnen hat, wer die gezogenen Kugeln (incl. ihrer Vielfachheit) ohne Berücksichtigung der Ziehungsreihenfolge richtig getippt hat.

Wie viele verschiedene Ziehungsergebnisse gibt es?

Wir stellen uns vor, dass wir eine Kugel aus der Menge der n Kugeln herausnehmen vor uns hinlegen. Zu den restlichen $n-1$ Kugeln fügen wir k neue, von den anderen Kugeln unterscheidbare Kugeln hinzu. Die $n+k-1$ Kugeln legen wir jetzt in allen möglichen Reihenfolgen hinter die am Anfang herausgelegte Kugel:



Ausgewählt sind diejenigen grauen Kugeln, die von mindestens einer schwarzen Kugel gefolgt werden, und zwar so oft, wie schwarze Kugeln folgen. Um Vertauschungen auszuscheiden, müssen wir noch durch die Anzahl der möglichen Vertauschungen der schwarzen Kugeln untereinander (das sind $k!$) und durch die Anzahl der möglichen Vertauschungen der grauen Kugeln incl. ihrer schwarzen Nachfolger untereinander (das sind $(n-1)!$) dividieren. Es gibt also

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

Ziehungsergebnisse.

Eine Auswahl von k Elementen aus einer n -elementigen Menge, bei der es auf die Reihenfolge der Auswahl nicht ankommt, bezeichnen wir als **n - k -Kombination mit Wiederholungen**, wenn in der Auswahl Wiederholungen von Elementen vorkommen dürfen.

Es gibt $\binom{n+k-1}{k}$ solcher Kombinationen.

Beispiel: Verteilung von Bonbons an Kinder

Es sollen $k = 100$ Bonbons an $n = 5$ Kinder verteilt werden (ohne dabei auf Gerechtigkeit zu achten).

$$\binom{n}{k} = \binom{n}{n-k}$$

Dann gibt es $\binom{n+k-1}{k} = \binom{104}{100} = \binom{104}{4} = \frac{104 \cdot 103 \cdot 102 \cdot 101}{4 \cdot 3 \cdot 2 \cdot 1} = 4598126$ mögliche Verteilungen.

Beachten Sie, dass hier streng genommen nicht Bonbons an Kinder sondern Kinder an Bonbons verteilt werden. Jedes Bonbon bekommt ein Kind zugelost. In der Lostrommel sind 5 Namensschilder der Kinder und es wird 100 mal mit Zurücklegen gezogen

Zusammenfassung:

Zahlenschloss $k = 4$ Ringe mit
 $n = 9$ Einstellmöglichkeiten:
 $n^k = 9^4 = 6561$
 verschiedene Zahlencodes

Finale mit $n = 8$ Läufern und $k = 3$ Medaillen
 $\frac{n!}{(n-k)!} = 8 \cdot 7 \cdot 6 = 336$
 verschiedene Medaillenvergaben

$n-k$ -Auswahl	mit Wiederholungen	ohne Wiederholungen $k \leq n$
mit Reihenfolge	Permutation mit Wiederholungen n^k	Permutation ohne Wiederholungen $\frac{n!}{(n-k)!}$
ohne Reihenfolge	Kombination mit Wiederholungen $\binom{n+k-1}{k}$	Kombination ohne Wiederholungen $\binom{n}{k}$

Verteilung von $k = 100$ Bonbons an $n = 5$ Kinder:
 $\binom{n+k-1}{k} = \binom{104}{100} = \binom{104}{4} = \frac{104 \cdot 103 \cdot 102 \cdot 101}{4 \cdot 3 \cdot 2 \cdot 1} = 4598126$
 verschiedene Verteilungen

Ziehung der Lottozahlen $k = 6$ aus $n = 49$
 $\binom{n}{k} = \binom{49}{6} = \frac{49 \cdot 48 \cdot 47 \cdot 46 \cdot 45 \cdot 44}{6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = 13983816$
 verschiedene Ziehungsergebnisse

Kombinatorische Algorithmen

Erstelle Programme, die die vier verschiedenen n-k-Auswahlen für beliebige Werte von n und k konkret erzeugen.

n-k-Auswahl	mit Wiederholungen	ohne Wiederholungen $k \leq n$
mit Reihenfolge	4-2-Permutationen mit Wiederholungen 1: < 0, 0> 2: < 0, 1> 3: < 0, 2> 4: < 0, 3> 5: < 1, 0> 6: < 1, 1> 7: < 1, 2> 8: < 1, 3> 9: < 2, 0> 10: < 2, 1> 11: < 2, 2> 12: < 2, 3> 13: < 3, 0> 14: < 3, 1> 15: < 3, 2> 16: < 3, 3>	4-2-Permutationen ohne Wiederholungen 1: < 0, 1> 2: < 1, 0> 3: < 0, 2> 4: < 2, 0> 5: < 0, 3> 6: < 3, 0> 7: < 1, 2> 8: < 2, 1> 9: < 1, 3> 10: < 3, 1> 11: < 2, 3> 12: < 3, 2>
ohne Reihenfolge	4-2-Kombinationen mit Wiederholungen 1: < 0, 0> 2: < 0, 1> 3: < 0, 2> 4: < 0, 3> 5: < 1, 1> 6: < 1, 2> 7: < 1, 3> 8: < 2, 2> 9: < 2, 3> 10: < 3, 3>	4-2-Kombinationen ohne Wiederholungen 1: < 0, 1> 2: < 0, 2> 3: < 0, 3> 4: < 1, 2> 5: < 1, 3> 6: < 2, 3>

$$4^2 = 16$$

$$\frac{4!}{(4-2)!} = 12$$

$$\binom{4}{2} = 6$$

$$\binom{4+2-1}{2} = 10$$

Grundmenge sind immer die Zahlen von 0 bis n-1, aus denen die Auswahl getroffen wird.

Ausgabe einer Auswahl

```
void print_array( int k, int array[])
{
    static int count = 0;
    int i;

    printf( "%3d: (", ++count);
    for( i = 0; i < k-1; i++)
        printf( "%2d,", array[i]);
    printf( "%2d)\n", array[k-1]);
}
```

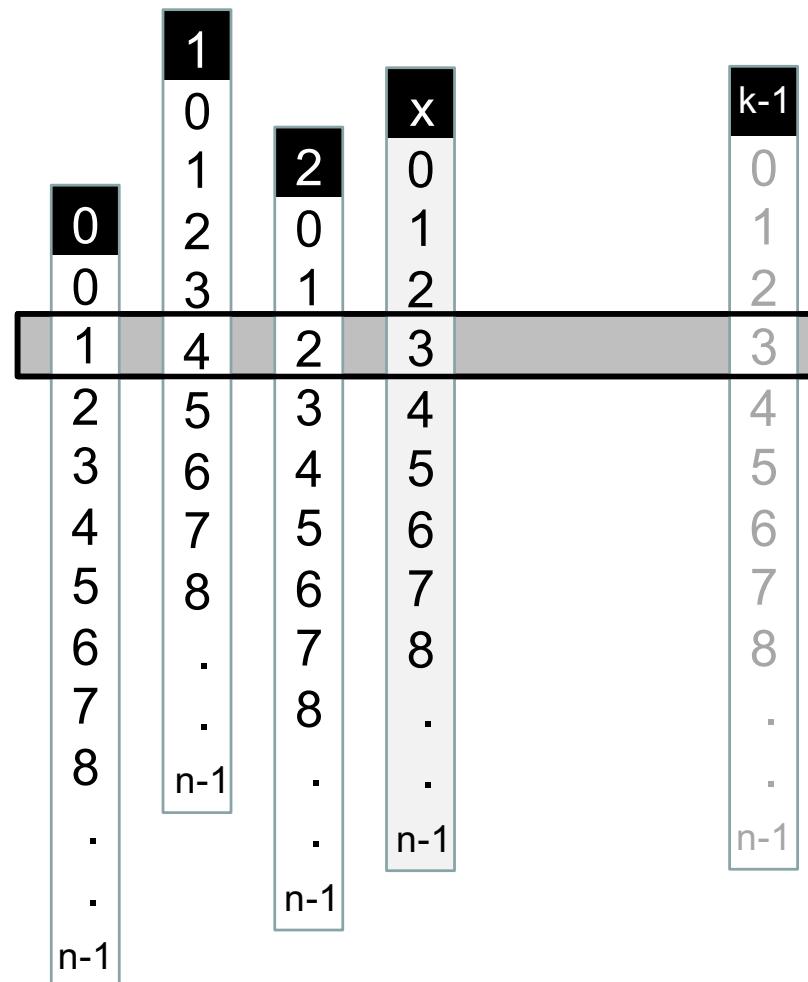
Der Funktion wird ein Array mit ganzen Zahlen und die Anzahl der Zahlen im Array übergeben.

Beim ersten Aufruf dieser Funktion wird der Zähler mit 0 initialisiert. Bei allen folgenden Aufrufen ist er mit dem zuletzt gesetzten Wert erneut verfügbar.

Diese Funktion wird immer dann gerufen, wenn eine neue Auswahl erzeugt wurde.

Zum Zählen der Aufrufe wird eine `static`-Variable innerhalb der Funktion `print_array` verwendet. Eine solche Variable ist nur innerhalb dieser Funktion bekannt und verwendbar. Sie behält aber über alle Funktionsaufrufe hinweg ihren Wert. Sie verhält sich also wie eine globale Variable, ist allerdings nicht global zugänglich.

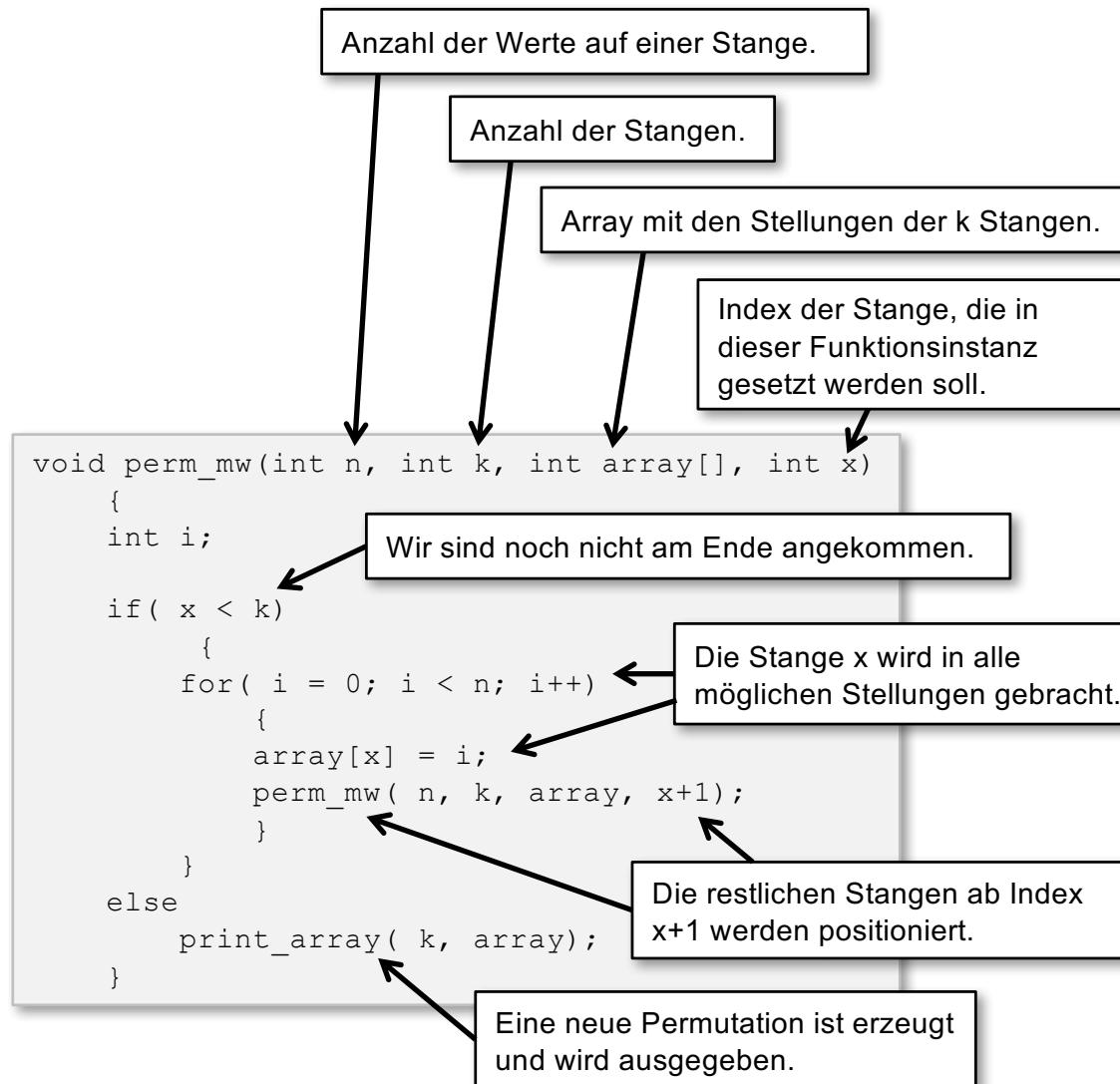
n-k-Permutationen mit Wiederholungen



Die k Stangen sind von 0 bis $k-1$ nummeriert und haben n Einstellungsmöglichkeiten, die von 0 bis $n-1$ nummeriert sind.

Die Stangen können beliebig verschoben werden, um eine gültige Auswahl zu erzeugen.

Rekursive Erzeugung von Permutationen mit Wiederholungen

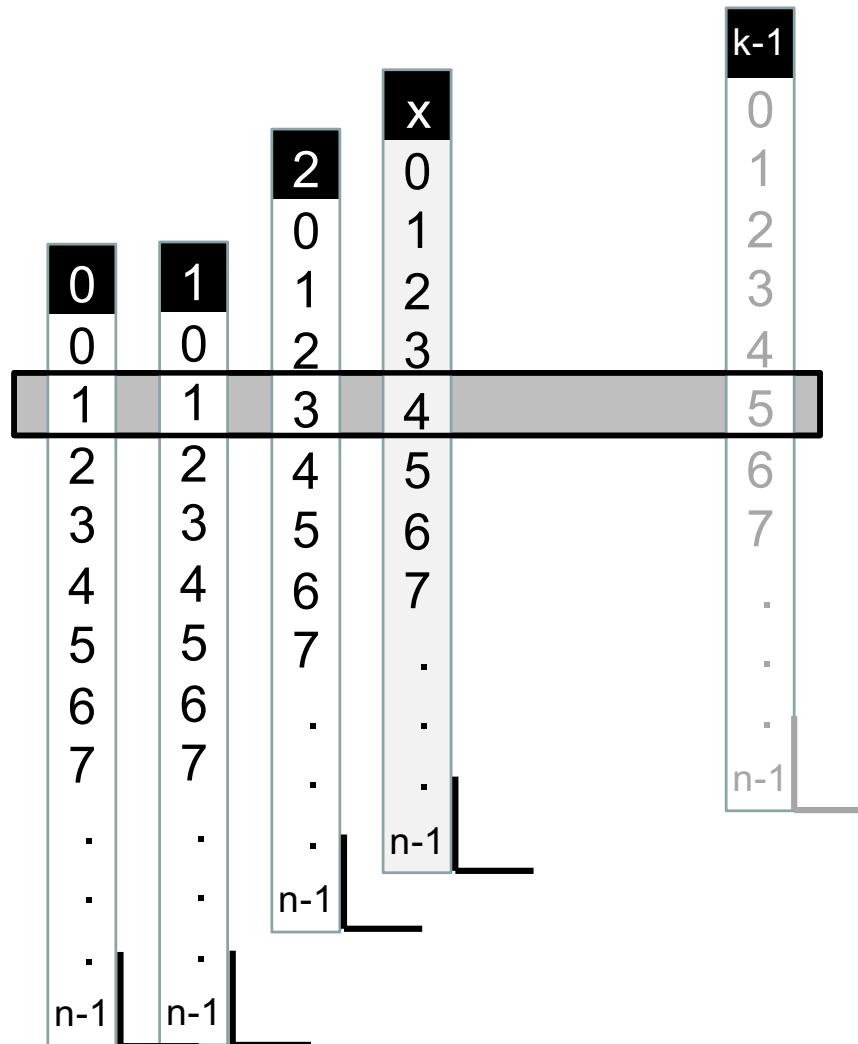


1	0	x		k-1
0	1	2	0	0
1	2	0	1	1
2	3	1	2	2
0	1	2	3	3
1	4	2	3	4
2	5	3	4	5
3	6	4	5	6
4	7	5	6	7
5	8	6	7	8
6	.	7	8	.
7	.	8	.	.
8	n-1	.	.	n-1
.
n-1				

Testrahmen und Ausgabe

```
Der Array ist auf die Größe  
der Auswahl ausgelegt.  
  
void main()  
{  
    int array[3];  
  
    printf( "2-3-Permutationen\nmit Wiederholungen\n");  
    perm_mw( 2, 3, array, 0);  
}  
  
2-3-Permutationen  
mit Wiederholungen  
1: < 0, 0, 0>  
2: < 0, 0, 1>  
3: < 0, 1, 0>  
4: < 0, 1, 1>  
5: < 1, 0, 0>  
6: < 1, 0, 1>  
7: < 1, 1, 0>  
8: < 1, 1, 1>
```

n-k-Kombinationen mit Wiederholungen



Durch zusätzlich angebrachte Winkel wird sichergestellt, dass die Auswahl aufsteigend ist.
Die Erzeugung gleicher Auswahlen in verschiedenen Reihenfolgen ist damit ausgeschlossen.

Rekursive Erzeugung von Kombinationen mit Wiederholungen:

```
void perm_mw(int n, int k, int array[], int x)
{
    int i;

    if( x < k)
    {
        for( i = 0; i < n; i++)
        {
            array[x] = i;
            perm_mw( n, k, array, x+1);
        }
    }
    else
        print_array( k, array);
}
```

Die Funktion zur Erzeugung von Permutationen wird nur geringfügig verändert, um Kombinationen zu erzeugen.

```
void komb_mw(int n, int k, int array[], int x, int min)
{
    int i;

    if( x < k)
    {
        for( i = min; i < n; i++)
        {
            array[x] = i;
            komb_mw( n, k, array, x+1, i);
        }
    }
    else
        print_array( k, array);
}
```

Zusätzlich wird der kleinste Wert für die aktuelle Stange übergeben.

Nur Werte ab dem Minimum werden eingestellt.

Die nächste Stange muss mindestens den Wert der aktuellen Stange haben.

Testrahmen und Ausgabe

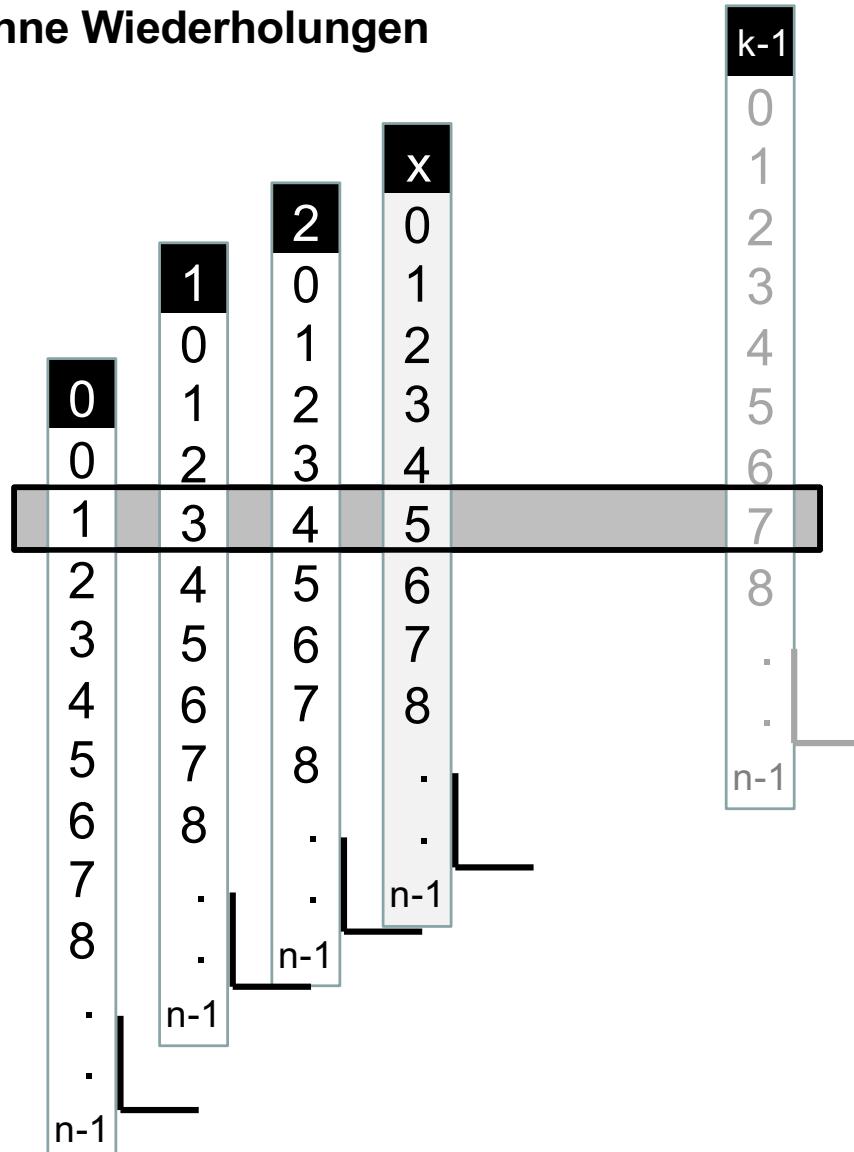
```
void main()
{
    int array[4];

    printf( "3-4-Kombinationen\nmit Wiederholungen\n");
    komb_mw( 3, 4, array, 0, 0);
}
```

```
3-4-Kombinationen
mit Wiederholungen
1: < 0, 0, 0, 0>
2: < 0, 0, 0, 1>
3: < 0, 0, 0, 2>
4: < 0, 0, 1, 1>
5: < 0, 0, 1, 2>
6: < 0, 0, 2, 2>
7: < 0, 1, 1, 1>
8: < 0, 1, 1, 2>
9: < 0, 1, 2, 2>
10: < 0, 2, 2, 2>
11: < 1, 1, 1, 1>
12: < 1, 1, 1, 2>
13: < 1, 1, 2, 2>
14: < 1, 2, 2, 2>
15: < 2, 2, 2, 2>
```

Starte mit Minimalwert 0.

n-k-Kombinationen ohne Wiederholungen



Rekursive Erzeugung von Kombinationen ohne Wiederholungen:

```
void komb_mw(int n, int k, int array[], int x, int min)
{
    int i;

    if( x < k)
    {
        for( i = min; i < n; i++)
        {
            array[x] = i;
            komb_mw( n, k, array, x+1, i);
        }
    }
    else
        print_array( k, array);
}
```

Die Funktion zur Erzeugung von Kombinationen mit Wiederholungen wird nur geringfügig verändert, um Kombinationen ohne Wiederholungen zu erzeugen.

```
void komb_ow(int n, int k, int array[], int x, int min)
{
    int i;

    if( x < k)
    {
        for( i = min; i <= n-k+x; i++)
        {
            array[x] = i;
            komb_ow( n, k, array, x+1, i+1);
        }
    }
    else
        print_array( k, array);
}
```

Nach oben muss Platz für die noch folgenden Stangen gelassen werden.

Die nächste Stange muss mindestens 1 höher als die aktuelle Stange sein.

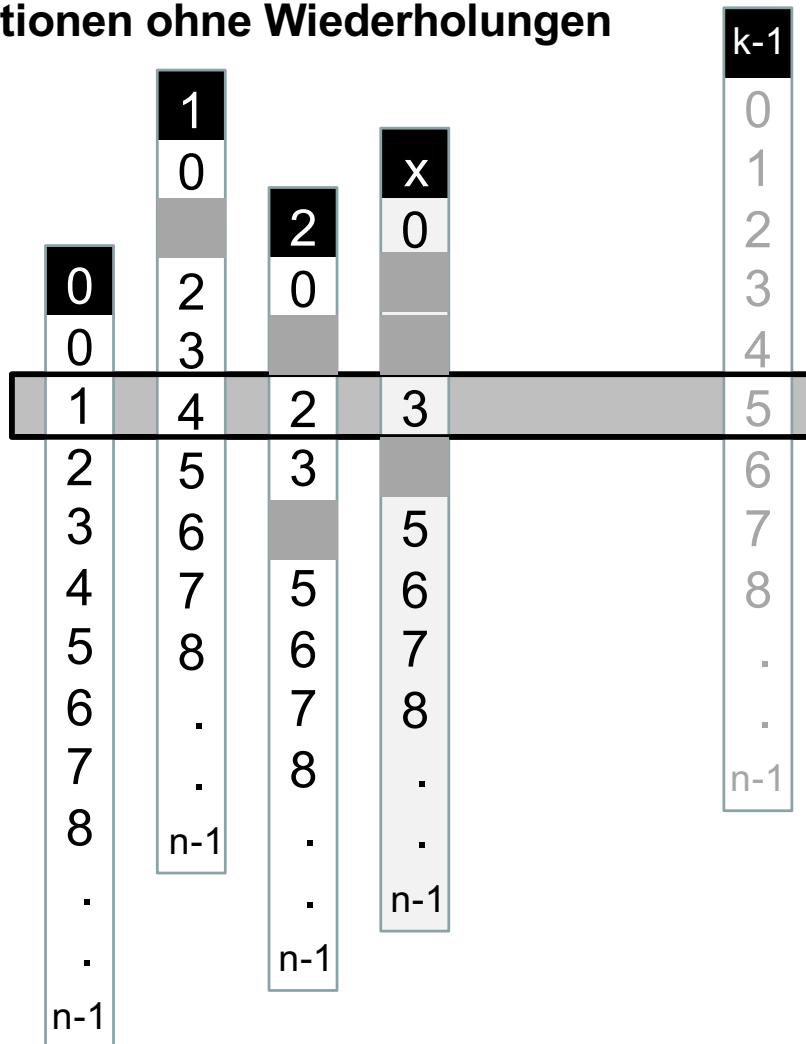
Testrahmen und Ausgabe

```
void main()
{
    int array[3];

    printf( "5-3-Kombinationen\nohne Wiederholungen\n");
    komb_ow( 5, 3, array, 0, 0);
}
```

```
5-3-Kombinationen
ohne Wiederholungen
1: < 0, 1, 2>
2: < 0, 1, 3>
3: < 0, 1, 4>
4: < 0, 2, 3>
5: < 0, 2, 4>
6: < 0, 3, 4>
7: < 1, 2, 3>
8: < 1, 2, 4>
9: < 1, 3, 4>
10: < 2, 3, 4>
```

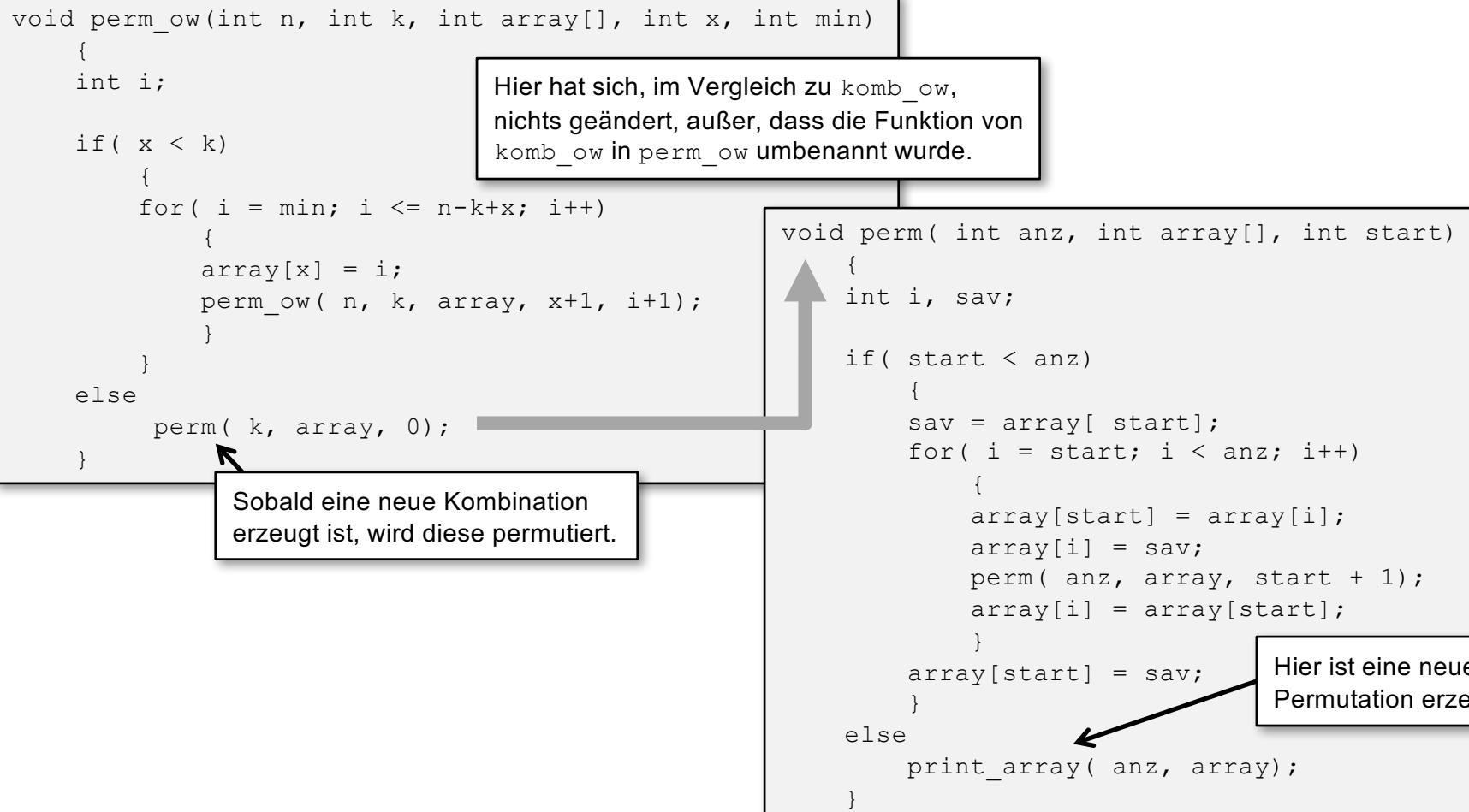
n-k-Permutationen ohne Wiederholungen



Eine Stange darf auf keinen Wert eingestellt werden, der auf einer vorherigen Stange bereits eingestellt wurde. Mit einfachen mechanischen Hilfsmitteln lässt sich das, im Vergleich zu den zuvor betrachteten Fällen, nicht realisieren.

Rekursive Erzeugung von Permutationen ohne Wiederholungen

Idee: Man erzeugt Kombinationen ohne Wiederholungen und permutiert diese dann mit der Funktion `perm`, die wir im Kapitel über Rekursion bereits kennengelernt haben



Testrahmen und Ausgabe

```
void main()
{
    int array[2];

    printf( "4-2-Permutationen\nohne Wiederholungen\n");
    perm_ow( 4, 2, array, 0, 0);
}
```

```
4-2-Permutationen
ohne Wiederholungen
1: < 0, 1>
2: < 1, 0>
3: < 0, 2>
4: < 2, 0>
5: < 0, 3>
6: < 3, 0>
7: < 1, 2>
8: < 2, 1>
9: < 1, 3>
10: < 3, 1>
11: < 2, 3>
12: < 3, 2>
```

Juwelenraub

Zwei Ganoven haben die Scheibe eines Juwelierladens eingeschlagen und in aller Eile 10 Schmuckstücke zusammengerafft. Wieder zu Hause angekommen, streiten sie sich um eine gerechte Verteilung der Beute. Zum Glück sind alle Beutestücke mit einem Preisschild versehen, aber wie soll man eine Verteilung vornehmen, bei der beide einen annähernd gleichen Anteil erhalten?

Lösungsidee: Betrachte alle denkbaren Teilauswahlen mit 1, 2, 3, 4 oder 5 Beutestücken und berechne jeweils den Wert der Teilauswahl. Wähle die Teilauswahl, deren Wert der halben Gesamtsumme am nächsten liegt. Teilauswahlen sind Kombinationen ohne Wiederholungen. Teilauswahlen mit mehr als 5 Beutestücken müssen nicht betrachtet werden, da dann ja die gegenteilige Auswahl weniger als 5 Beutestücke hat und bereits in der Betrachtung enthalten ist.

Bei einer $n-k$ -Auswahl ohne Beachtung der Reihenfolge und ohne Wiederholungen gibt es " n über k " mögliche Ergebnisse.

$\binom{10}{k}$	
k	Auswahlen
1	10
2	45
3	120
4	210
5	252
Summe:	
637	

Insgesamt sind 637 Fälle zu betrachten.

Juwelenraub 2

Zunächst werden die benötigten Datenstrukturen definiert:

```
Preise der 10 Beutestücke.  
double beute[10] = { 333.33, 655.99, 387.50, 1420.10, 4583.17,  
                     7500.00, 215.12, 3230.17, 599.00, 3775.11};  
  
double summe; ← Gesamtwert der Beute.  
  
int anzahl;  
int auswahl[10]; ← Diese vier Variablen beschreiben eine Teilauswahl der Beute:  
double teilsumme;  
double abweichung;
```

Diese vier Variablen beschreiben eine Teilauswahl der Beute:
anzahl ist die Anzahl der ausgewählten Beutestücke.
auswahl ist der Array mit den Indices der ausgewählten Beutestücke.
teilsumme ist der Wert der Teilauswahl.
abweichung ist die Abweichung der Teilsumme von der Hälfte des
Gesamtwerts.

Juwelenraub 3

In der Funktion `vorbereitung` wird die Beute aufgelistet. Dabei wird der Gesamtwert der Beute in der Variablen `summe` berechnet:

```
void vorbereitung()
{
    int i;

    for( i = 0, summe = 0.0; i < 10; i++)
    {
        printf( "Beutestueck %2d: %10.2f Euro\n", i+1, beute[i]);
        summe += beute[i];
    }
    printf( "Gesamtbeute:      %10.2f Euro\n\n", summe);
    abweichung = summe + 1;
}
```

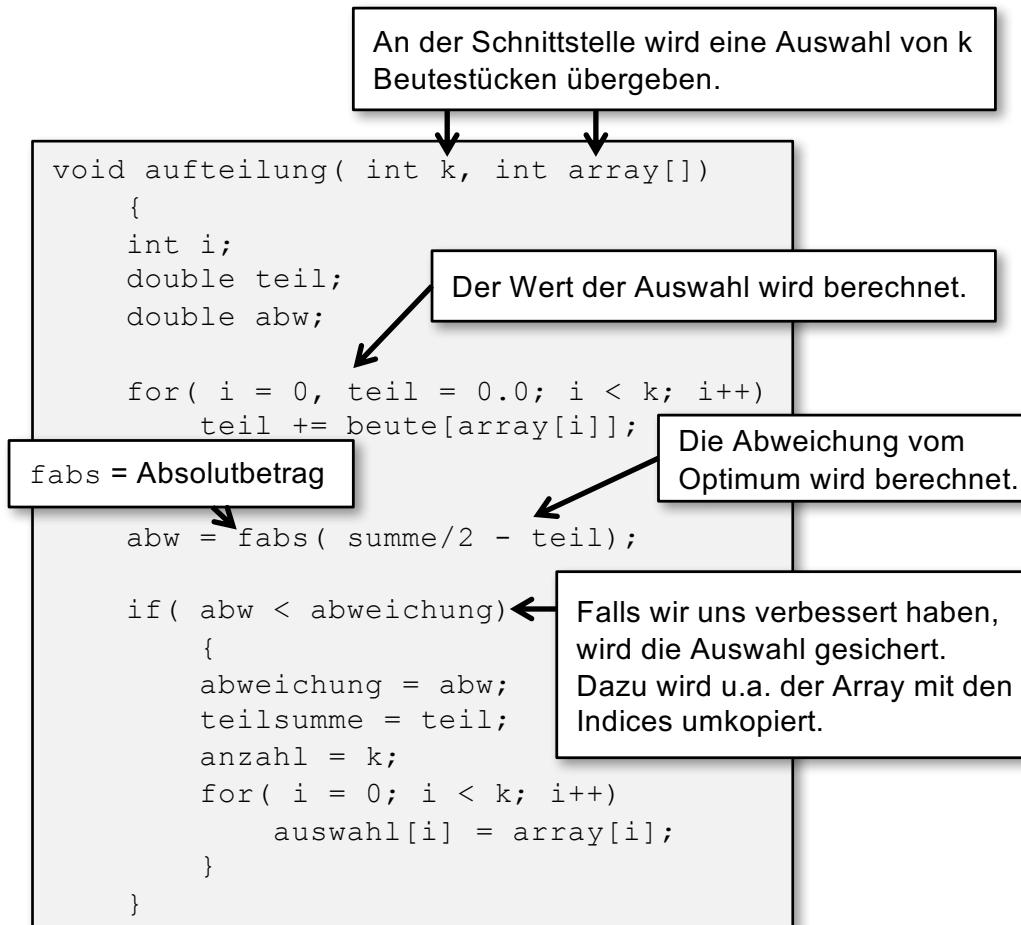
Beutestueck 1:	333.33	Euro
Beutestueck 2:	655.99	Euro
Beutestueck 3:	387.50	Euro
Beutestueck 4:	1420.10	Euro
Beutestueck 5:	4583.17	Euro
Beutestueck 6:	7500.00	Euro
Beutestueck 7:	215.12	Euro
Beutestueck 8:	3230.17	Euro
Beutestueck 9:	599.00	Euro
Beutestueck 10:	3775.11	Euro
Gesamtbeute:	22699.49	Euro

Der Gesamtwert der Beute wird berechnet.

Die Abweichung wird auf einen großen Anfangswert gesetzt, damit beliebige Auswahlen diesen Wert später unterbieten.

Juwelenraub 4

Der Funktion aufteilung wird eine Auswahl von Beutestücken übergeben. Die Funktion berechnet den Wert dieser Auswahl und die Abweichung dieses Werts vom Optimum (halbe Gesamtsumme)



Ist die Abweichung kleiner als die kleinste bisher gefundene Abweichung, so wird die Auswahl als bisher beste Auswahl gespeichert.

Juwelenraub 5

Mit der Funktion `komb_ow` erzeugen wir alle $n-k$ -Auswahlen ohne Wiederholungen und ohne Beachtung der Reihenfolge

```
void komb_ow(int n, int k, int array[], int x, int min)
{
    int i;

    if( x < k)
    {
        for( i = min; i <= n-k+x; i++)
        {
            array[x] = i;
            komb_ow( n, k, array, x+1, i+1);
        }
    }
    else
        aufteilung( k, array);
}
```

Hier wird jetzt `aufteilung` statt bisher `print_array` gerufen.

Ist eine neue Auswahl erzeugt, so wird durch Aufruf von `aufteilung` geprüft, ob diese Auswahl eine bessere Aufteilung liefert.

Juwelenraub 6

Im Hauptprogramm werden, nach der Vorbereitung der Datenstrukturen, alle zu betrachtenden Kombinationen (10-1, 10-2, 10-3, 10-4 und 10-5) erzeugt und überprüft.

```
void main( )
{
    int array[5];
    int i;

    vorbereitung();
    for( i = 1; i <= 5; i++)
        komb_ow( 10, i, array, 0, 0);
    auswertung();
}
```

Schritt für Schritt werden alle 10-1, 10-2, 10-3, 10-4 und 10-5-Kombinationen ohne Wiederholungen erzeugt und getestet.

Die Funktion `auswertung` gibt nach Überprüfung aller Kombinationen die beste Auswahl aus:

```
void auswertung()
{
    int i;

    printf( "Der Komplize erhält:\n\n");
    for( i = 0; i < anzahl; i++)
    {
        printf( "    Beutestueck %2d %10.2f Euro\n", auswahl[i]+1,
               beute[auswahl[i]]);
    }
    printf( "\nTeilsumme %10.2f Euro\n", teilsumme);
    printf( "\nAbweichung %10.2f Euro\n", abweichung);
}
```

Der Komplize erhält:		
Beutestueck 3	387.50	Euro
Beutestueck 6	7500.00	Euro
Beutestueck 7	215.12	Euro
Beutestueck 8	3230.17	Euro
Teilsumme	11332.79	Euro
Abweichung	16.95	Euro

Geldautomat 1

Programmiere einen Geldautomaten, der intern unbegrenzt viele 5-, 10-, 20-, 50-, 100-, 200-, 500-EURO-Scheine vorhält, so, dass er einen Geldbetrag mit bis zu 20, aber möglichst wenig Geldscheinen auszahlt. Wenn eine solche Auszahlung nicht möglich ist, soll eine entsprechende Meldung ausgegeben werden.

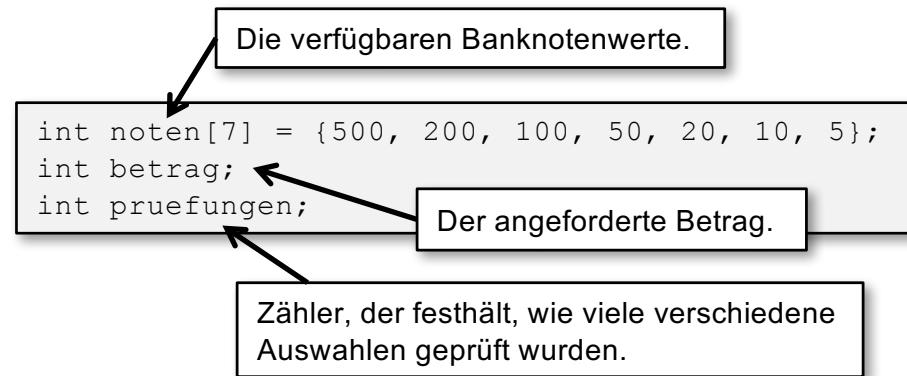
Eine Auszahlung mit k Geldscheinen ist eine $n-k$ -Kombination mit Wiederholungen der $n = 7$ Notenwerte.

$\binom{7+k-1}{k}$	Auswählen
1	7
2	28
3	84
4	210
5	462
6	924
7	1716
8	3003
9	5005
10	8008
11	12376
12	18564
13	27132
14	38760
15	54264
16	74613
17	100947
18	134596
19	177100
20	230230
Summe:	
888029	

Insgesamt sind 888029 Fälle zu betrachten.

Geldautomat 2

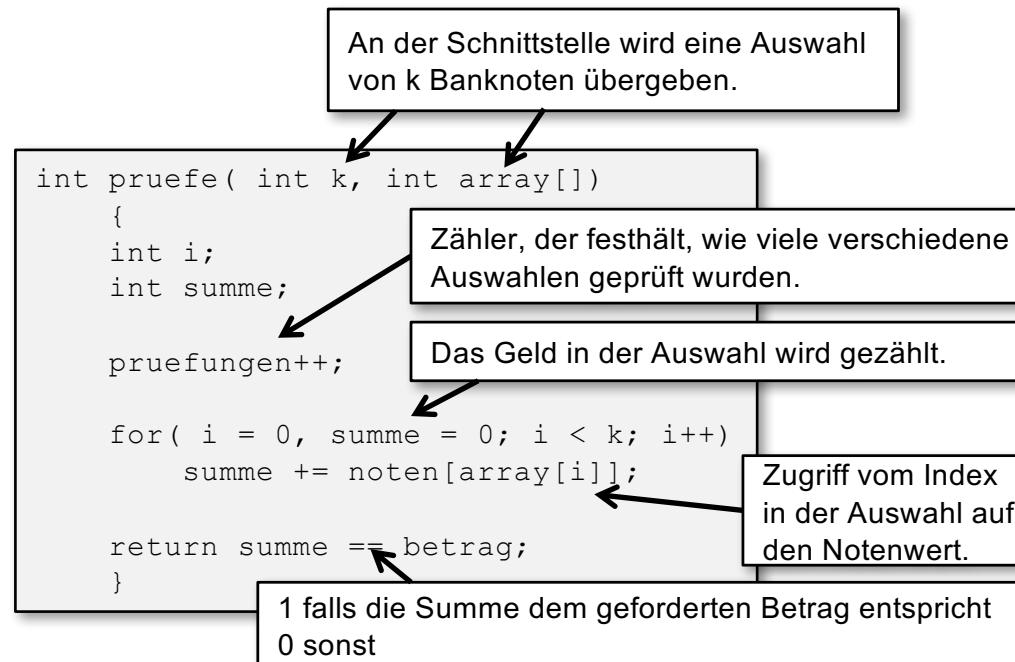
Zunächst werden wieder die benötigten Datenstrukturen definiert:



Das Zwischenspeichern von Auswahlen ist nicht erforderlich, da wie die Suche abbrechen können, sobald wir eine korrekte Auszahlung gefunden haben.

Geldautomat 3

In der Funktion pruefe überprüfen wir, ob eine Auswahl exakt den gesuchten Betrag liefert:



Wenn der Betrag stimmt, gibt die Funktion 1 zurück, ansonsten 0.

Geldautomat 4

Zur Lösungssuche verwenden wir eine modifizierte Funktion `komb_mw`, die wir abbrechen, sobald eine Lösung gefunden wurde:

```
int komb_mw(int n, int k, int array[], int x, int min)
{
    int i;

    if( x < k)
    {
        for( i = min; i < n; i++)
        {
            array[x] = i;
            if( komb_mw( n, k, array, x+1, i))
                return 1;
        }
        return 0;
    }
    else
        return pruefe( k, array);
}
```

Wenn in der Rekursion eine Lösung gefunden wird, dann brich mit einer Erfolgsmeldung ab.
Ansonsten suche weiter.

Die Auswahl lässt sich auf dieser Aufrufe Ebene nicht zu einer Lösung fortsetzen . Suche ggf. auf höheren Aufrufe ebene n weiter.

Wenn eine vollständige Auswahl erzeugt wurde, dann gib das Ergebnis der Prüfung zurück.

Am Returnwert der Funktion lässt sich erkennen, ob eine Lösung gefunden wurde oder nicht.

Geldautomat 5

Hauptprogramm

```

void main()
{
    int array[20];
    int k, i;

    for( ; ;)
    {
        printf( "Betrag: ");
        scanf( "%d", &betrag);

        if( betrag <= 0)          ← Abbruch, wenn ein Betrag ≤ 0 eingegeben wurde.
            break;
        pruefungen = 0;           ← Zähler der Prüfungen zurücksetzen.

        for( k = 1; k <= 20; k++) ← Suche nach Lösungen mit 1, 2, 3, ... 20
            {
                if( komb_mw( 7, k, array, 0, 0))
                    {
                        printf( "Auszahlung: ");
                        for( i = 0; i < k; i++)
                            printf( "%d ", noten[array[i]]);
                        printf( "\n");
                        break; ← Die Suche war erfolgreich.
                    }
                }
            if( k > 20)             ← Keine Lösung gefunden.
                printf( "Keine Auszahlung möglich\n");
            printf( "Es wurden %d Prüfungen durchgeführt\n\n", pruefungen);
    }
}

```

```

Betrag: 885
Auszahlung: 500 200 100 50 20 10 5
Es wurden 2353 Prüfungen durchgeführt

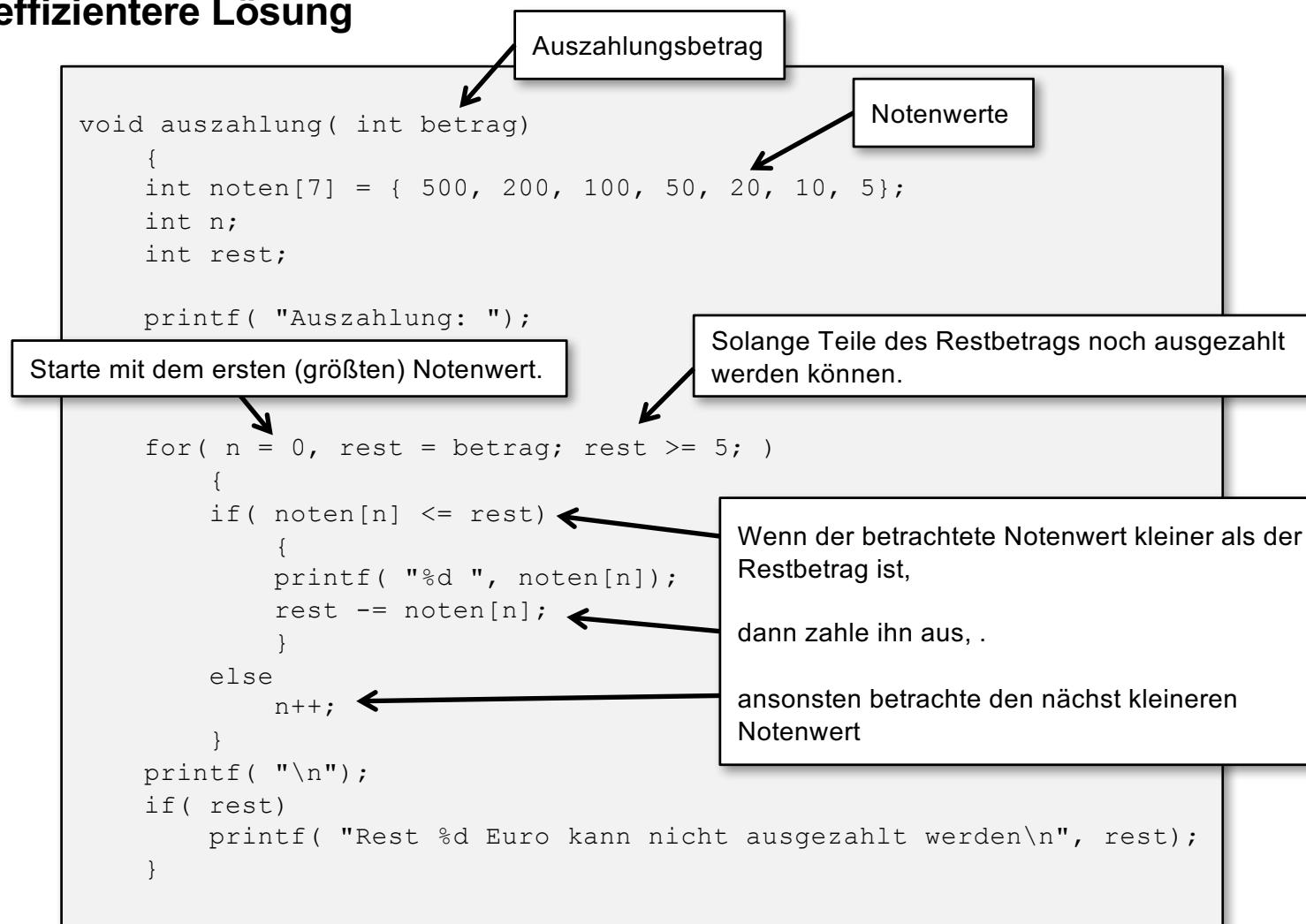
Betrag: 1235
Auszahlung: 500 500 200 20 10 5
Es wurden 926 Prüfungen durchgeführt

Betrag: 500
Auszahlung: 500
Es wurden 1 Prüfung durchgeführt

Betrag: 1
Keine Auszahlung möglich
Es wurden 888029 Prüfungen durchgeführt

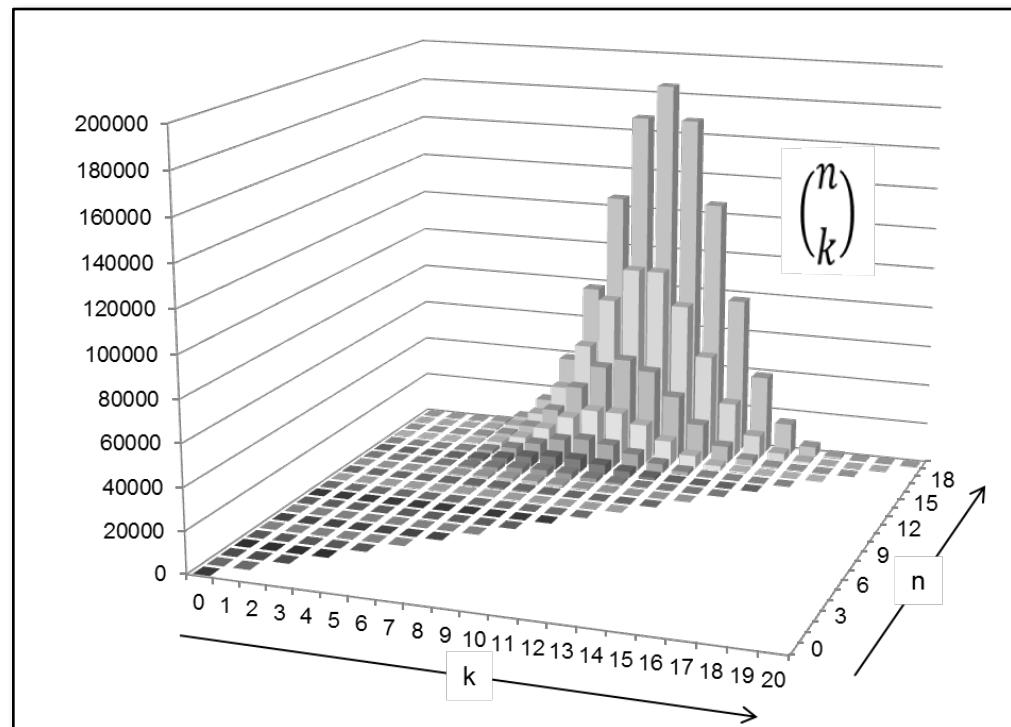
```

Alternative, effizientere Lösung



Den hier verwendeten Algorithmus nennt man einen Greedy-Algorithmus, weil er sich immer den größten Happen schnappt und so versucht, das Problem dadurch möglichst schnell zu lösen. Gier kann allerdings blind machen. Wenn der Automat nur mit Noten ab 20 € bestückt wäre und 60 € auszuzahlen wären, dann würde der Algorithmus zunächst 50 € auszahlen und wäre in eine Sackgasse gelaufen, obwohl er mit drei 20 € Noten den Betrag hätte auszahlen können.

Kombinatorische Algorithmen versuchen mit brutaler Gewalt (brute force) die Lösung eines Problems zu finden, indem sie den Suchraum des Problems vollständig aufbauen. Der Suchraum kann dabei so groß sein, dass eine Lösung des Problems nicht in akzeptabler Zeit gefunden werden kann. Selbst beim kleinsten Suchraum unserer vier Fragestellungen (Kombinationen ohne Wiederholungen) ergeben sich schon bei relativ kleinen Werten für n und k sehr große Suchräume (kombinatorische Explosion):



Wo immer möglich, sollte man versuchen, kombinatorische Algorithmen zu vermeiden und Algorithmen verwenden, die bessere Laufzeiteigenschaften haben. Ob dies immer möglich ist, ist eine offene Frage der Theoretischen Informatik, auf die ein Preisgeld von 1 Million Dollar ausgesetzt ist.