

9 Datenstrukturen

Datenstrukturen sind als Begriff bereits in Abschnitt 2.3 eingeführt worden. In diesem Kapitel werden die LabVIEW-spezifischen Eigenschaften von Datenstrukturen behandelt, wobei eine Beschränkung auf die grundlegenden Datenstrukturen *Arrays* (Datenfelder) und *Cluster* (Verbund) erfolgt. Datenstrukturen wie zum Beispiel Bäume, Graphen und Heaps, die insbesondere beim Sortieren und Suchen von großer Bedeutung sind, werden hier nicht behandelt, da in LabVIEW im Allgemeinen für diese Aufgaben bereits vorgefertigte Funktionen zur Verfügung stehen, die bei einer Einführung in die systematische Programmierung mit LabVIEW zumeist ausreichend sind.

Weitere Datenstrukturen können mit Hilfe von *Arrays* erzeugt werden, da diese als grundlegende Datenstruktur näherungsweise die lineare Ablage von Daten im Hauptspeicher eines Rechners widerspiegeln. Beispiele für die Realisierung verschiedener Datenstrukturen in LabVIEW zeigt J. P. Damico in *LabVIEW Power Programming* [28].

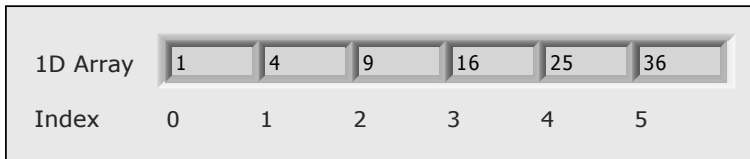
Die Datenstruktur *String* (Zeichenkette) ist bereits in Abschnitt 8.3 behandelt worden, da in LabVIEW der elementare Datentyp *Character* nicht benötigt wird (vgl. Abschn. 2.3.3).

9.1 Arrays (Datenfelder)

In diesem Abschnitt soll zunächst gezeigt werden, wie *Arrays* in LabVIEW manuell (s. Abschn. 9.1.1) und programmgesteuert (s. Abschn. 9.1.2) erzeugt werden können. Anschließend werden die Funktionen zur Bearbeitung von *Arrays* vorgestellt (s. Abschn. 9.1.3) und es werden die Möglichkeiten zur Formatumwandlung zwischen verschiedenen Datentypen behandelt, die nur mit Hilfe von Array-Funktionen vorgenommen werden können (s. Abschn. 9.1.4). Am Ende des Abschnitts soll die Verwendung von Array-Funktionen anhand dreier Beispiele weiter veranschaulicht werden (s. Abschn. 9.1.6).

Ein *Array* enthält eine Menge von Daten gleichen Typs. Zum Aufbau eines *Arrays* können alle in LabVIEW zur Verfügung stehenden Datentypen verwendet werden. Für die Realisierung komplexer Datenstrukturen ist es aber nicht möglich, *Arrays* ineinander zu verschachteln. Eine Einschränkung ist damit nicht verbunden, da komplexere Datenstrukturen in Kombination mit *Clusters*, die in Abschnitt 9.2 vorgestellt werden, realisiert werden können. Ein *Array* kann mehrere Dimensionen mit jeweils maximal $2^{31} - 1$ Elementen pro Dimension aufweisen. Der Zugriff auf einzelne oder mehrere Elemente erfolgt über Indizes. Zu beachten ist, dass die Indizierung bei 0

beginnt, d. h. das erste Element eines Arrays hat den Index 0. In einem Array mit N Elementen weist der Index damit den Wertebereich von 0 bis $N - 1$ auf. Bei der beispielhaften Darstellung eines eindimensionalen Arrays auf dem *Front Panel* in LabVIEW hat dementsprechend das erste Element mit dem Wert 1 den Index 0 und das sechste Element mit dem Wert 36 den Index 5 (Abb. 9.1).



1D Array	1	4	9	16	25	36
Index	0	1	2	3	4	5

Abb. 9.1: Eindimensionales Array mit sechs Elementen

9.1.1 Deklaration von Arrays

Arrays können in LabVIEW auf dem *Front Panel* sowohl als Eingabe- als auch als Ausgabeelemente sowie als Konstante im *Block Diagram* erzeugt werden. Das Erzeugen eines Arrays soll zunächst anhand eines eindimensionalen Arrays erläutert werden, um anschließend die Erzeugung mehrdimensionaler Arrays vorzunehmen. Dabei soll in diesem Abschnitt auch die Darstellung von Arrays berücksichtigt werden, obwohl diese für die Funktionsweise eines Algorithmus nicht von Bedeutung ist. Sinnvoll ist dieses Vorgehen aber, da die Darstellung von Arrays in LabVIEW für das grundsätzliche Verständnis von Bedeutung ist.

Erzeugen von eindimensionalen Arrays

Für die Erzeugung eines eindimensionalen Arrays sind einige wenige Schritte ausreichend. Zunächst wird aus der *Controls Palette* » *Array, Matrix & Cluster* eine *Array Shell* (Array-Container) ausgewählt und auf dem *Front Panel* platziert (Abb. 9.2a). Das korrespondierende Element im *Block Diagram* wird zunächst schwarz dargestellt, da dem Array noch kein Datentyp zugeordnet worden ist. In einem zweiten Schritt kann aus der *Controls Palette* ein beliebiger Datentyp ausgewählt und mit dem *Operating Tool* in die *Array Shell* eingesetzt werden (Abb. 9.2b). Nach dem Einsetzen des ausgewählten Datentyps nimmt das korrespondierende Element im *Block Diagram* die dem Datentyp zugeordnete farbliche Darstellung an. Auf dem *Front Panel* wird der eingesetzte Datentyp zunächst gedimmt dargestellt und die geometrischen Abmessungen der *Array Shell* werden automatisch an die des eingefügten Objektes angepasst. Damit ist dem Array ein Datentyp zugeordnet worden. In diesem Beispiel ist in die *Array Shell* ein Eingabeelement vom Typ DBL eingesetzt worden. Das Array enthält aber noch keine Elemente und ist zunächst leer (Abb. 9.2c). In dem zugehörigen Element im *Block Diagram* symbolisieren die eckigen Klammern in Anlehnung an textbasierte Programmiersprachen die Array-Struktur. Mit dem Beschriftungswerkzeug oder über die Inkrement/Dekrement-Schaltflächen des *Controls* kann schließlich

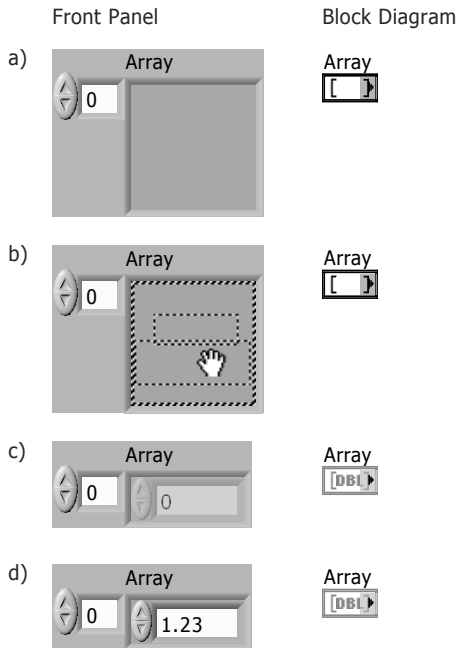


Abb. 9.2: Erstellen eines Arrays auf dem *Front Panel*

der gewünschte Wert in das Eingabeelement eingetragen werden. Das Array enthält nun ein Element (Abb. 9.2d).

In analoger Weise kann im *Block Diagram* eine Array-Konstante erzeugt werden, indem zuerst aus der *Functions Palette* » *Array* die Array-Konstante ausgewählt wird (Abb. 9.3a). Anschließend kann aus der *Functions Palette* der gewünschte Datentyp, in diesem Fall eine String-Konstante, ausgewählt und in die leere Array-Konstante eingesetzt werden (Abb. 9.3b). Danach nimmt die Array-Konstante wiederum die dem Datentyp zugeordnete Farbe an (Abb. 9.3c).

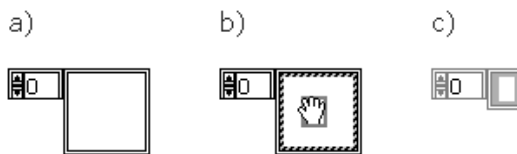


Abb. 9.3: Erstellen einer Array-Konstanten im *Block Diagram*

Im Folgenden soll die Visualisierung von Arrays behandelt werden. Beispielhaft wird dies für die Darstellung auf dem *Front Panel* vorgenommen. Diese Ausführungen sind in analoger Weise aber auch für die Darstellung von Array-Konstanten im *Block Diagram* gültig. Die Darstellung von ein- und mehrdimensionalen Arrays auf dem *Front Panel* ist auf unterschiedliche Weise möglich, lässt aber in der Regel keinen Rückschluss auf die tatsächliche Anzahl der im Array enthaltenen Elemente zu. Abbildung 9.4a zeigt zunächst wieder ein eindimensionales Array mit einem Element.

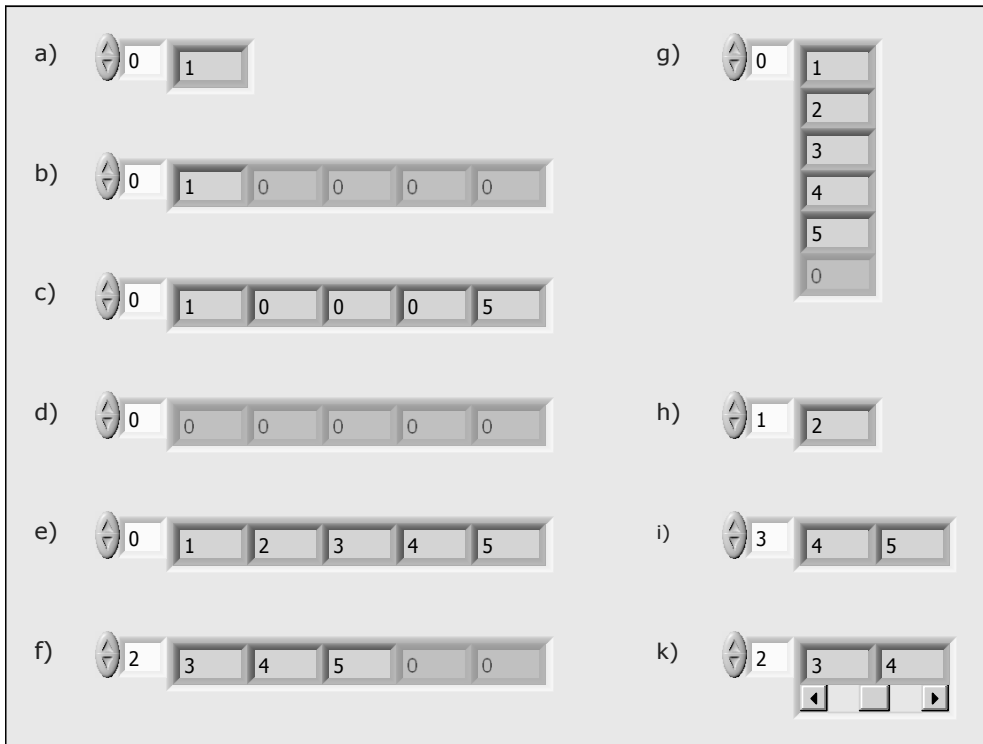


Abb. 9.4: Darstellung eines eindimensionalen Arrays auf dem *Front Panel*

Wird das Positionierwerkzeug über eine Ecke der *Array Shell* gebracht, nimmt der Cursor die Form eines rechten Winkels an und die *Array Shell* kann in die gewünschte Richtung erweitert werden. In Abbildung 9.4b werden beispielsweise fünf Elemente dargestellt, von denen die letzten vier jedoch gedimmt dargestellt werden, d. h. sie enthalten keine Daten. Um diesen Sachverhalt zu verdeutlichen, enthält die Darstellung von Arrays in einigen der folgenden Beispielen mehr Felder als tatsächlich vorhandene Elemente. Das Positionierwerkzeug ermöglicht es also, die Anzahl der dargestellten Elemente eines Arrays zu ändern. Um die geometrische Größe des eingesetzten Datentyps zu verändern, kann das Positionierwerkzeug zudem auf den Rand des Datentyps gebracht werden und anschließend kann der Rahmen des Datentyps auf die gewünschte Größe aufgezogen werden.

Wenn in das fünfte Element des Arrays eine fünf eingegeben wird, werden die Elemente mit dem Index 1 bis 3 automatisch mit dem Standardwert, in diesem Fall einer Null, belegt (Abb. 9.4c). Um alle Elemente aus einem Array zu entfernen besteht die Möglichkeit, im Kontextmenü der *Array Shell* (nicht im Kontextmenü des eingesetzten Datentyps) die Option *Data Operations* » *Empty Array* (Datenoperationen » Leeres Array) auszuwählen (Abb. 9.4d).

Alle weiteren Darstellungen in Abbildung 9.4 zeigen dasselbe Array mit den fünf Elementen 1, 2, 3, 4, 5. Bei einem eindimensionalen Array ist es möglich, die Elemente in einer Zeile (Abb. 9.4e) oder in einer Spalte (Abb. 9.4g) anzuordnen. Bei der Darstel-

lung von Arrays ist das *Index Display* (Anzeige indizieren) am linken Rand des Arrays von besonderer Bedeutung. Dieses ermöglicht in großen Arrays die Darstellung eines Ausschnitts der im Array enthaltenen Daten. In Abbildung 9.4f weist das *Index Display* den Wert 2 auf. Da die Indizierung bei Null beginnt, wird dementsprechend das dritte Element des Arrays unmittelbar neben dem *Index Display* dargestellt. Direkt vergleichbar ist die Anzeige eines einzelnen Wertes des Arrays in Abbildung 9.4h. Da die Anzahl der Elemente in einem Array bei der dynamischen Erzeugung von Arrays nicht immer bekannt ist, kann über das Kontextmenü des Arrays *Advanced* » *Show Last Element* (Fortgeschritten » Letztes Element anzeigen) das letzte Element zur Anzeige gebracht werden (Abb. 9.4i). Dieses ist dann nicht neben dem *Index Display* sondern am rechten Rand der *Array Shell* angeordnet. Um Teile größerer Arrays darzustellen, eignet sich schließlich der in Abbildung 9.4k dargestellte *Scrollbar*, der im Kontextmenü über *Visible Items* » *Scrollbar* (Sichtbare Objekte » Bildlaufleiste) eingeblendet werden kann.

LV8.0

Grundsätzlich sollte bedacht werden, dass für die Anzeige eines Arrays auf dem *Front Panel*, ähnlich wie bei globalen und lokalen Variablen, eine Kopie des Arrays im Hauptspeicher des Rechners erzeugt wird, was sich bei sehr großen Arrays nachteilig auf die Ausführung des Programms auswirken kann.

Erzeugen von mehrdimensionalen Arrays

Natürlich können in LabVIEW auch mehrdimensionale Arrays erstellt und verarbeitet werden. Die prinzipielle Anordnung (aber nicht die Funktionalität, vgl. Abschn. 9.1.4) der Elemente in einem zweidimensionalen Array ist vergleichbar mit der in einer Matrix (Abb. 9.5a), in der die Elemente spalten- und zeilenweise (*columns* & *rows*) angeordnet sind. Ein dreidimensionales Array ergibt sich, wenn diese einzelnen *pages* (Seiten) in einem Stapel übereinander angeordnet werden (Abb. 9.5b).

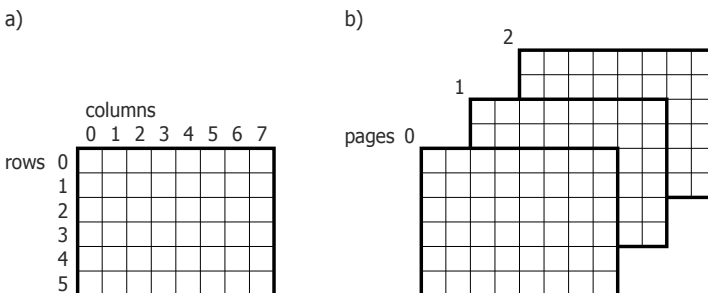


Abb. 9.5: Prinzipielle Struktur eines zwei- und dreidimensionalen Arrays

Die folgenden Abbildungen zeigen beispielhaft die Darstellung mehrdimensionaler Arrays auf dem *Front Panel*. Um einem Array weitere Dimensionen hinzuzufügen, kann das *Index Display* mit dem Positionierwerkzeug nach unten aufgezogen werden oder im entsprechenden Kontextmenü kann die Option *Add Dimension* (Dimension hinzufügen) ausgewählt werden. Für jede Dimension steht dann eine Index-Anzeige zur Verfügung.

Abbildung 9.6 zeigt ein zweidimensionales Array mit fünf Spalten und vier Zeilen. Auch hier ist es möglich, nur einen Teil des Arrays darzustellen. Eine Indizierung mit 0,0 sorgt dann dafür, dass das erste Element des Arrays, die 1, unmittelbar neben der Index-Anzeige dargestellt wird. Dementsprechend wird eine Indizierung mit 1,2 zur Anzeige des Elementes 8 der zweiten Zeile und dritten Spalte führen.

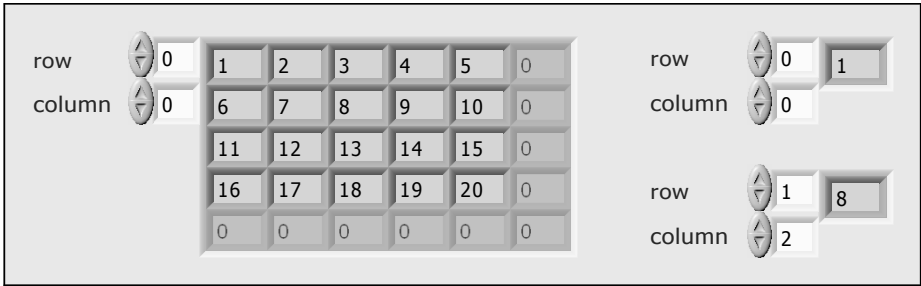


Abb. 9.6: Darstellung eines zweidimensionalen Arrays auf dem *Front Panel*

Dazu analog wird auch ein dreidimensionales Array dargestellt, wie z. B. in Abbildung 9.7 mit fünf Spalten, vier Zeilen und drei Seiten skizziert. Zur Verdeutlichung sind hier, wie auch in einigen weiteren Beispielen, die einzelnen Seiten des Arrays versetzt hintereinander dargestellt worden. Grundsätzlich kann in LabVIEW nur jeweils eine Seite eines dreidimensionalen Arrays dargestellt werden. Wenn wiederum nur ein Element des Arrays angezeigt wird, führt die Indizierung mit 1,1,3 unmittelbar neben der Index-Anzeige zur Darstellung des Elementes 29 aus der zweiten Seite, zweiten Zeile und der vierten Spalte und eine Indizierung mit 2,0,1 führt zur Anzeige des Elementes 42 aus der dritten Seite, der ersten Zeile und der zweiten Spalte.

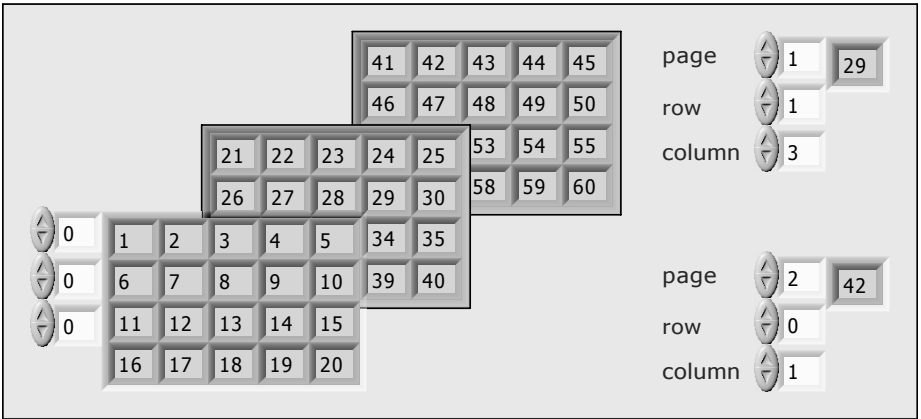


Abb. 9.7: Darstellung eines dreidimensionalen Arrays auf dem *Front Panel*

Über der Index-Anzeige für die erste Dimension (Spalten) ist die Index-Anzeige für die zweite Dimension (Zeilen) angeordnet. Alle weiteren Index-Anzeigen für höhere Dimensionen werden dann jeweils oben hinzugefügt (vgl. Abb. 9.8).

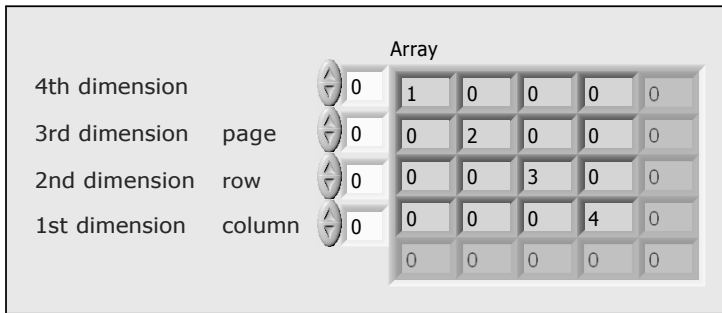


Abb. 9.8: Reihenfolge der Indizierung bei mehrdimensionalen Arrays

Abbildung 9.9 zeigt in einer Übersicht noch einmal die Darstellung eines numerischen Wertes und eines ein-, zwei- und dreidimensionalen Arrays auf dem *Front Panel* und im *Block Diagram*. Im *Block Diagram* gibt sowohl die Breite der Verbindungsleitungen einen Hinweis auf die Komplexität der Datenstruktur als auch die Breite der Klammersymbole in den jeweiligen Terminals. Bei Arrays die numerische Daten enthalten, wird die Komplexität der Datenstruktur dagegen durch zwei parallele Linien angedeutet (vgl. Abb. 9.10).

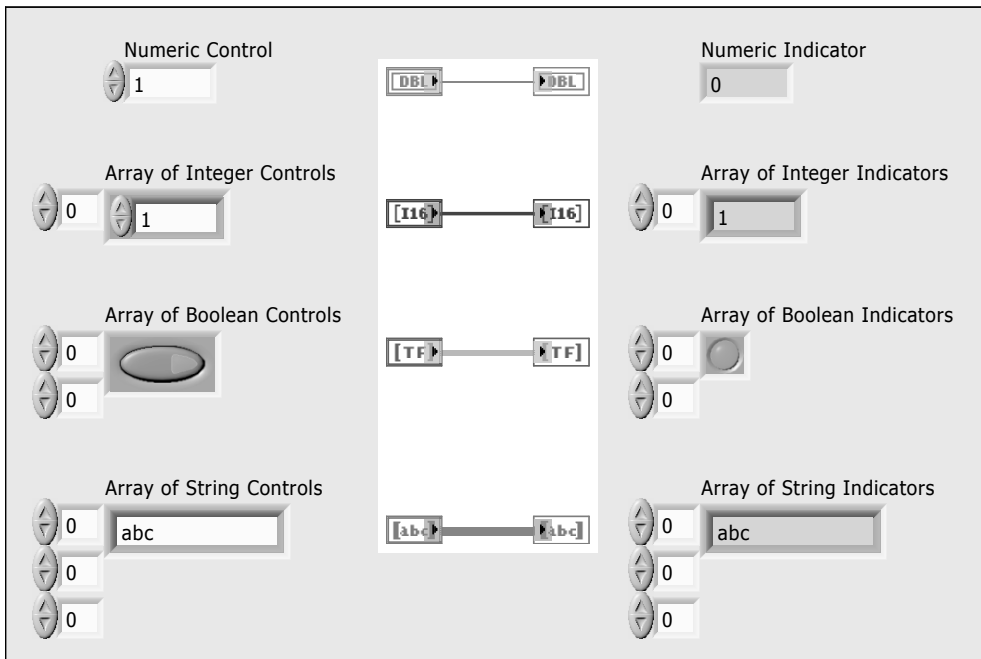


Abb. 9.9: Darstellung von Arrays auf dem *Front Panel* und im *Block Diagram*

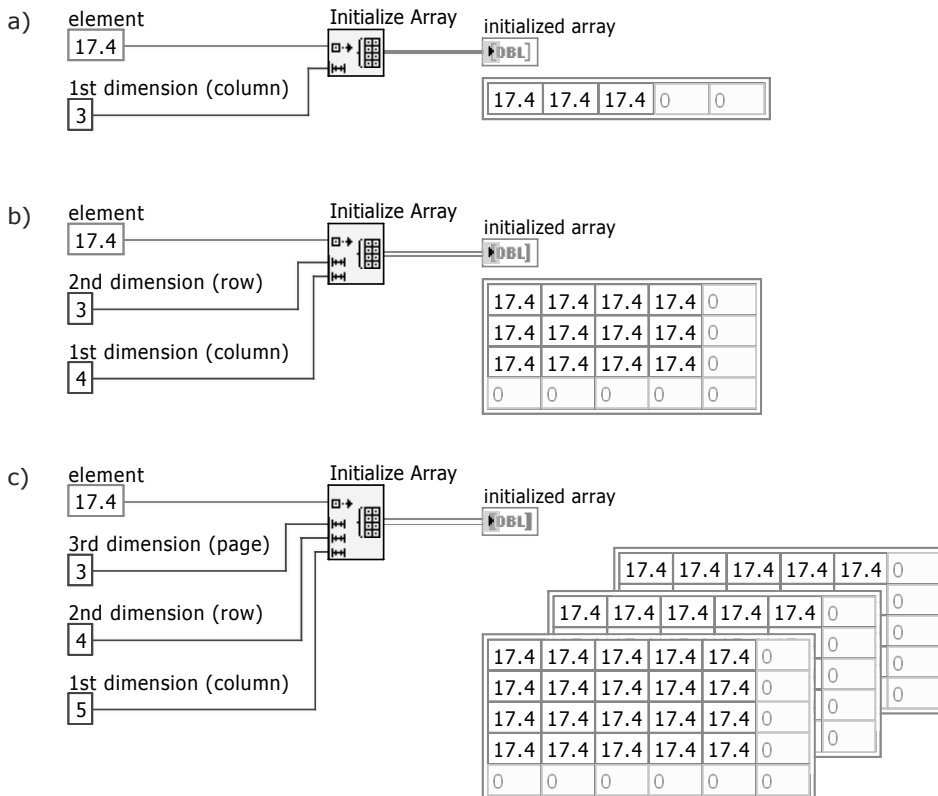


Abb. 9.10: Beispiele für die Initialisierung von ein- und mehrdimensionalen Arrays mit der Funktion *Initialize Array*

9.1.2 Programmgesteuerte Erzeugung und Verarbeitung von Arrays

Die programmgesteuerte Erzeugung von Arrays kann auf unterschiedliche Weise erfolgen. Es ist möglich, ein n-dimensionales Array mit einem vorgegebenen Element zu besetzen, ein Array aus einzelnen Teilen zusammensetzen und schließlich bietet die Indexing-Funktionalität am Rand von Wiederholschleifen eine äußerst komfortable Möglichkeit, um Arrays programmgesteuert zu generieren bzw. auszuwerten.

Initialize Array (Array initialisieren)

Die Funktion *Initialize Array*, die aus der Palette *Functions* » *Array* ausgewählt werden kann (s. Abb. 9.20), dient dazu, ein n-dimensionales Array eines beliebigen Datentyps mit einem gewünschten Wert zu generieren. In Abbildung 9.10a wird ein eindimensionales Array mit drei Elementen erzeugt, indem an das Terminal *Element* die Konstante 17.4 mit dem Datentyp DBL angeschlossen und dem Terminal *Dimension Size* (Dimensionsgröße) die Anzahl der Werte übergeben wird. Bei einem eindimensionalen Array ist es möglich, die Elemente des Arrays in einer Zeile oder in einer

Spalte darzustellen. Bei mehrdimensionalen *Arrays* bezieht sich die erste Dimension aber immer auf die *columns* (Spalten).

Mit dem Positionier-Werkzeug kann der Funktionsknoten nach unten aufgezogen werden, um weitere Eingänge *Dimension Size* für weitere Dimensionen bereitzustellen. Abbildung 9.10b zeigt ein Beispiel für ein zweidimensionales *Array* mit vier Spalten und drei Zeilen. Wird die Funktion *Initialize Array* um einen weiteren Eingang erweitert, um ein dreidimensionales *Array* zu erzeugen, ist zu beachten, dass sich für die Anordnung der Eingänge *Dimension Size* die gleiche Anordnung ergibt wie in Abbildung 9.8. Über das oberste Terminal *Dimension Size* wird nun die Anzahl der Seiten festgelegt, so dass sich ein dreidimensionales *Array* mit fünf Spalten, vier Zeilen und drei Seiten ergibt (Abb. 9.10c).

Prinzipiell ist es möglich, *Arrays* mit der Dimensionsgröße 0 zu initialisieren. Dann wird ein leeres *Array* erzeugt, welches lediglich die Informationen über den Datentyp und die Anzahl der Dimensionen enthält. Dies ist insbesondere bei der Initialisierung von *Shift Register* von Bedeutung (vgl. Abb. 9.13).

Build Array (Array erstellen)

Ebenfalls aus der Palette *Functions* » *Array* kann die Funktion *Build Array* ausgewählt werden (s. Abb. 9.20). Mit dieser können aus einzelnen Elementen oder aus Teil-Arrays, die den Eingängen der Funktion zugeführt werden, *Arrays* zusammengesetzt werden. Abbildung 9.11a zeigt zunächst eine einfache Möglichkeit, um ein Element, hier vom Datentyp *String*, in ein *Array* mit einem Element umzuwandeln. Sollen mehrere Einzelelemente zu einem *Array* zusammengesetzt werden, kann auch dieser Funktionsknoten mit dem Positionier-Werkzeug nach unten erweitert werden, um die erforderliche Anzahl von Eingängen zu erzeugen. Es ist aber auch möglich, im Kontextmenü der Funktion die Option *Add Input* (Eingang hinzufügen) auszuwählen. So können dann wie in Abbildung 9.11b drei einzelne Elemente zu einem eindimensionalen *Array* zusammengesetzt werden oder es können wie in Abbildung 9.11c einzelne Elemente an ein eindimensionales *Array* angehängt werden.

Weiterhin besteht die Möglichkeit, Teil-Arrays zu einem *Array* zusammenzufassen. In Abbildung 9.11d wird an ein zweidimensionales *Array* eine weitere Reihe hinzugefügt. Verallgemeinert bedeutet dies, dass einem *Array* mit der Dimension n ein Teil-*Array* mit der Dimension $n - 1$ hinzugefügt werden kann. In der n -ten Dimension wird das *Array* dann um ein Element erweitert. Ein vergleichbarer Fall ist in Abbildung 9.11e dargestellt. Da das eindimensionale *Array* eine Spalte mehr aufweist als das zweidimensionale *Array*, werden bei letzterem die Elemente der dritte Spalte mit dem Standardwert besetzt. Im vorliegenden Fall, in dem die *Array*-Elemente den Datentyp *String* aufweisen, ist dies ein leerer *String*.

Wenn an die Eingänge der Funktion *Build Array* ein oder mehrere *Arrays* mit gleicher Dimension angeschlossen werden, kann im Kontextmenü der Funktion *Build Array* die Option *Concatenate Inputs* (Eingänge verknüpfen) aktiviert bzw. deaktiviert werden (Abb. 9.12). Bei aktivierter Option *Concatenate Inputs*, werden die Teil-Arrays zu einem *Array* mit gleicher Dimension zusammengesetzt (Abb. 9.12a,b). Bei deaktivierter Option *Concatenate Inputs* erstellt die Funktion *Build Array* aus

den beiden zugeführten Teil-Arrays ein neues Array dessen Dimension um eins größer ist, als die Dimension der zugeführten Arrays (Abb. 9.12c,d).

Eine für LabVIEW typische Programmstruktur, in der die Funktion *Build Array* innerhalb einer Wiederholschleife verwendet wird, zeigt Abbildung 9.13. Das Schieberegister einer For-Schleife (oder While-Schleife) wird mit einer Array-Konstanten initialisiert. Die Array-Konstante enthält in diesem Beispiel zunächst keine Elemente. Während der Ausführung des Programms wird dem Array bei jedem Schleifendurchlauf der aktuelle Wert des Iterationszählers hinzugefügt.

Diese kleine Beispielpogramm veranschaulicht einen enormen Vorteil von LabVIEW (und anderen, eher problemorientierten Programmiersprachen). Arrays werden dynamisch verwaltet, d. h. der Software-Entwickler ist von der Spezifikation und der anschließenden Verwaltung einer festen Array-Größe entbunden. Zwar ist der Hinweis in der Online-Hilfe korrekt, dass durch die dynamische Verwaltung von Arrays die Leistungsfähigkeit eines Algorithmus abnimmt. Aber solange die Ausführungsgeschwindigkeit unkritisch ist, sollte der dynamischen Verwaltung von Arrays der Vorzug gegeben werden, da so im Allgemeinen sehr gut lesbare und damit in hohem Maße selbst dokumentierende Programme entstehen.

Aus dem gleichen Grund wird in LabVIEW der Datentyp *Character* nicht benötigt und es ist ausreichend, dem Software-Entwickler die Datenstruktur *String* zur Verfügung zu stellen, da auch hier die Verwaltung der Zeichenanzahl einer Zeichenkette dynamisch erfolgt.

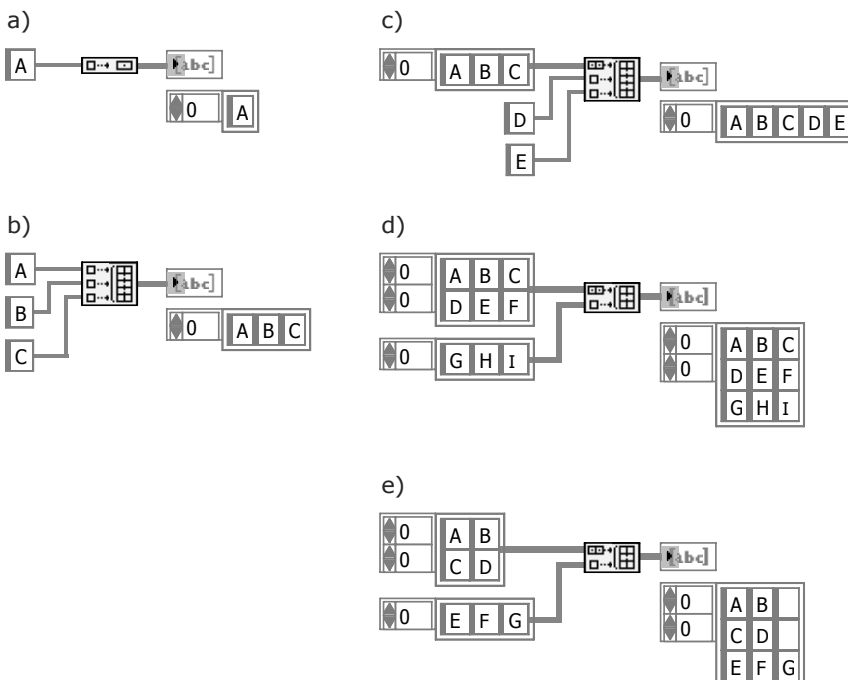


Abb. 9.11: Beispiele für die Anwendung der Funktion *Build Array*

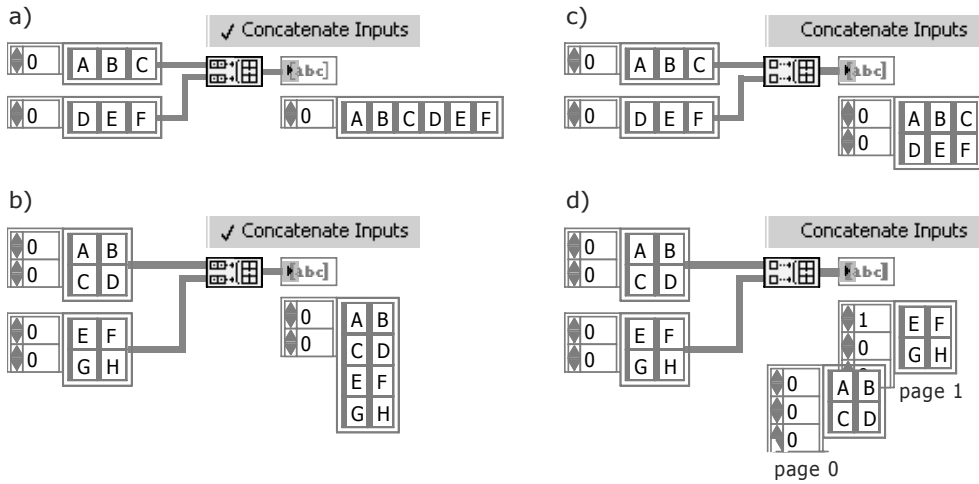


Abb. 9.12: Anwendungsbeispiele für die Funktion *Build Array*, a, b) mit aktivierter und c, d) deaktivierter Option *Concatenate Inputs*

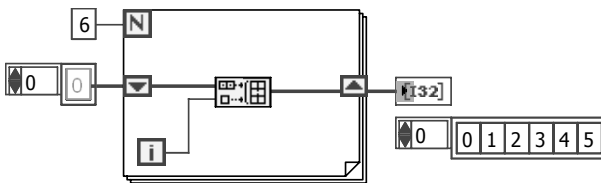


Abb. 9.13: Typische Verwendung der Funktion *Build Array* innerhalb einer Wiederholschleife

► Beispiel:

Die Verwendung der Funktion *Build Array* soll an einem Beispiel veranschaulicht werden, in dem zwei Arrays erzeugt werden, wobei das eine die geraden und das andere die ungeraden Zahlen enthält (vgl. Abb. 9.14). Dies wird erreicht, indem mit Hilfe eines Schieberegisters der Funktion *Build Array* das Ergebnis des vorletzten Schleifendurchlaufs zugeführt wird. Dies hat zur Folge, dass im Schieberegister kontinuierlich zwischen den letzten beiden Schleifendurchläufen (Iteration $n-2$ und $n-1$) gewechselt wird. Eine Realisierung dieser Aufgabenstellung ist natürlich auch möglich, indem ein Laufindex, z. B. der Iterationszähler i der For-Schleife, zum einen mit 2 multipliziert wird und zum anderen mit 2 multipliziert und um eins inkrementiert wird und das Resultat jeweils einem Funktionsknoten *Build Array* zugeführt wird. ◀

Indizierung bei Wiederholschleifen

Wie bereits in Abschnitt 7.1.3 erläutert wurde, können Datenflüsse einer Wiederholschleife über den Rand zugeführt bzw. entnommen werden. Dafür stehen Schleifentunnel zur Verfügung, die über die äußerst komfortable Möglichkeit verfügen, Arrays (Datenfelder) automatisch zu indizieren bzw. zu generieren, wenn im Kontext-

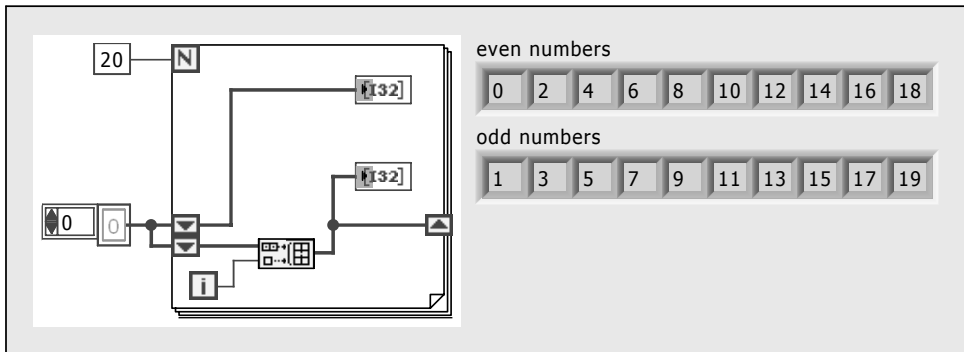


Abb. 9.14: Beispiel für die Erzeugung von zwei eindimensionalen Arrays mit geraden und ungeraden Zahlen

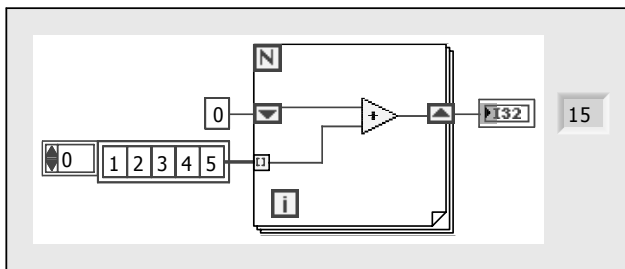


Abb. 9.15: Beispiel für die Automatische Indizierung am Eingangstunnel einer For-Schleife

Menü eines Schleifentunnels die Option *Enable Indexing* (Indizierung aktivieren) ausgewählt wird.

Wird ein Array bei aktivierter Option *Enable Indexing* dem Rand einer Wiederholschleife zugeführt, wird dem Array bei jedem Schleifendurchlauf ein Element entnommen und verarbeitet. Bei einem eindimensionalen Array ist dies ein Skalar und bei einem mehrdimensionalen Array mit der Dimension n ist dies ein Teil-Array mit der Dimension $n - 1$. Abbildung 9.15 zeigt ein Beispiel für die automatische Indizierung auf der Eingangsseite einer For-Schleife. Bei jedem Schleifendurchlauf wird dem zugeführten Array ein Element entnommen und zu dem aktuellen Wert des Schieberegisters addiert, so dass die Summe aller im Array enthaltenen Elemente gebildet wird. Dieses Beispiel soll vornehmlich die Funktionsweise der automatischen Indizierung veranschaulichen. Auf einfachere Weise ist die Summenbildung mit Hilfe der Funktion *Add Array Elements* (Array-Elemente addieren) möglich (vgl. Abb. 9.46).

Wenn die automatische Indizierung an einem Schleifentunnel auf der Ausgangsseite einer Wiederholschleife aktiviert worden ist, wird bei jedem Schleifendurchlauf ein neues Element erzeugt und die Ergebnisse aller Schleifendurchläufe werden am Schleifenrand gesammelt. Wenn die Wiederholschleife beendet worden ist, werden alle Ergebnisse als Array ausgegeben. Im Beispiel nach Abbildung 9.16 wird der Wert des Iterationszählers quadriert und nach fünf Schleifendurchläufen wird ein Array mit den ersten fünf Quadratzahlen ausgegeben.

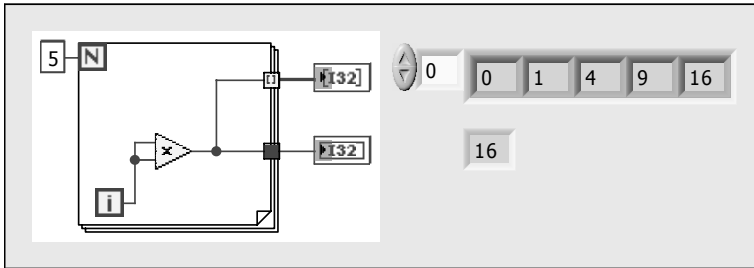


Abb. 9.16: Beispiel für die Automatische Indizierung am Ausgangstunnel einer For-Schleife

Ist die automatische Indizierung hingegen deaktiviert, wird nur das Ergebnis des letzten Schleifendurchlaufs am Schleifentunnel ausgegeben (Abb. 9.17). Bei einer deaktivierten automatischen Indizierung an einem Schleifentunnel auf der Eingangsseite einer Wiederholschleife wird dagegen bei jedem Schleifendurchlauf das vollständige Array in die Wiederholschleife übergeben (Abb. 9.17). Da bei einer For-Schleife die Anzahl der Schleifendurchläufe festliegt, ist die automatische Indizierung standardmäßig aktiviert, während sie bei einer While-Schleife standardmäßig deaktiviert ist.

Bei einer For-Schleife ist es bei aktivierter automatischer Indizierung am Eingangstunnel nicht erforderlich, das Zählterminal mit einem Vorgabewert zu belegen, da der Vorgabewert automatisch auf die Größe des Arrays gesetzt wird (vgl. Abb. 9.15). Unterscheidet sich der Vorgabewert am Zählterminal von der Array-Länge, so setzt LabVIEW den Vorgabewert für die Anzahl der Schleifenumläufe auf den kleineren der beiden Werte (Abb. 9.17).

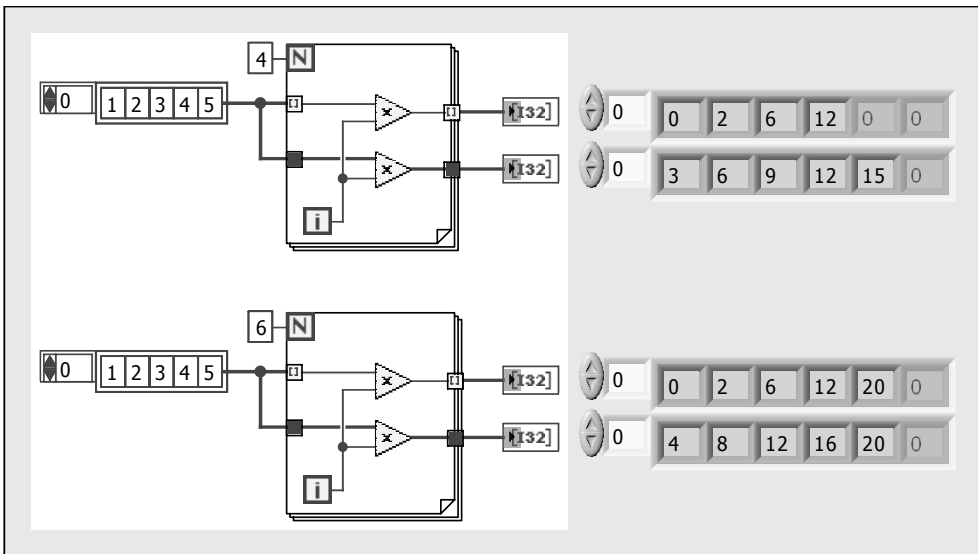


Abb. 9.17: Beispiel für die automatische Indizierung an der Ein- und Ausgangsseite einer For-Schleife

► **Programmiertipp:**

Gelegentlich ergeben sich bei der Programmierung Fehler, weil ein Ein- oder Ausgang mit dem falschen Datentyp oder der falschen Datenstruktur verbunden wird. Diese Fehler können vermieden werden, wenn die gewünschten Bedien- oder Anzeigeelemente im Kontextmenü eines Ein- oder Ausgangs über die Option *Create* » *Control* bzw. *Indicator* (Erstellen » Bedien- bzw. Anzeigeelement) generiert werden. Bei der automatischen Erzeugung von Bedien- oder Anzeigeelementen wird immer der korrekte Datentyp oder die korrekte Datenstruktur berücksichtigt, dies kann auch bei der Fehlersuche genutzt werden. ◀

	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{		}	~	.
€	•	,	f	"	...	†	‡	^	%	Š	<	œ	•	ž	•
•	`	'	"	"	•	—	—	~	™	š	>	œ	•	ž	ÿ
	i	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Abb. 9.18: Darstellung einer ASCII-Tabelle (PC-ANSI) in einem zweidimensionalen Array

► Beispiel: Erzeugen von mehrdimensionalen Arrays

Zwei- und mehrdimensionale *Arrays* können auf einfache Weise mit Hilfe der automatischen Indizierung erzeugt werden, wenn Wiederholschleifen mit aktivierter automatischer Indizierung ineinander geschachtelt werden. Beispielhaft soll dies an der Erzeugung einer ASCII-Tabelle gezeigt werden, bei der die 256 Zeichen in einem zweidimensionalen Array mit jeweils 16 Zeilen und 16 Spalten dargestellt werden (Abb. 9.18).

Für die Generierung der ASCII-Tabelle ist es ausreichend, zwei For-Schleifen ineinander zu schachteln, deren Zählterminal jeweils mit 16 belegt wird (Abb. 9.19). Zur Erzeugung der einzelnen Zeichen wird zunächst ein Schieberegister mit Null initialisiert und bei jedem Schleifendurchlauf wird der Wert des Schieberegisters inkrementiert. Damit stehen dann insgesamt Ganzzahlen im Bereich von 0 bis 255 zur Verfügung. Diese Ganzzahlen können mit Hilfe der Funktion *Type Cast* (Typenformung) (vgl. Abschn. 8.5) in einen *String* umgewandelt werden. Die innere For-Schleife erzeugt jeweils ein eindimensionales Array mit 16 Elementen (Spalten) und die äußere For-Schleife setzt diese eindimensionalen Arrays zu einem zweidimensionalen Array mit 16 Zeilen zusammen.

Natürlich ist es möglich, diese Aufgabenstellung auch auf andere Weise zu realisieren. Ein weiteres Beispiel unter Verwendung der Funktion *Reshape Array* (Array umformen) zeigt Abbildung 9.42. Auch ist es nicht zwingend erforderlich, eine Typumwandlung mittels *Type Cast* vorzunehmen. Eine alternative Möglichkeit zeigt das Beispiel in Abbildung 9.55. ◀

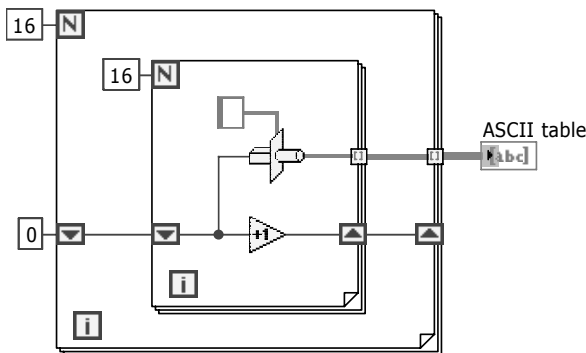


Abb. 9.19: Programm für die Erzeugung einer ASCII-Tabelle

9.1.3 Funktionen für das Arbeiten mit Arrays

In diesem Abschnitt sollen die Möglichkeiten zur Auswertung und Manipulation von *Arrays* vorgestellt werden. Abbildung 9.20 zeigt alle Elemente der Palette *Functions* » *Array*. Die Möglichkeiten zur manuellen und programmgesteuerten Erzeugung von *Arrays* mit Hilfe der Funktionen *Array Constant*, *Initialize Array* und *Build Array* sind bereits in den beiden vorigen Abschnitten vorgestellt worden. Hier soll nun zunächst auf eine Gruppe von *Array*-Funktionen eingegangen werden, die die Indizierung eines

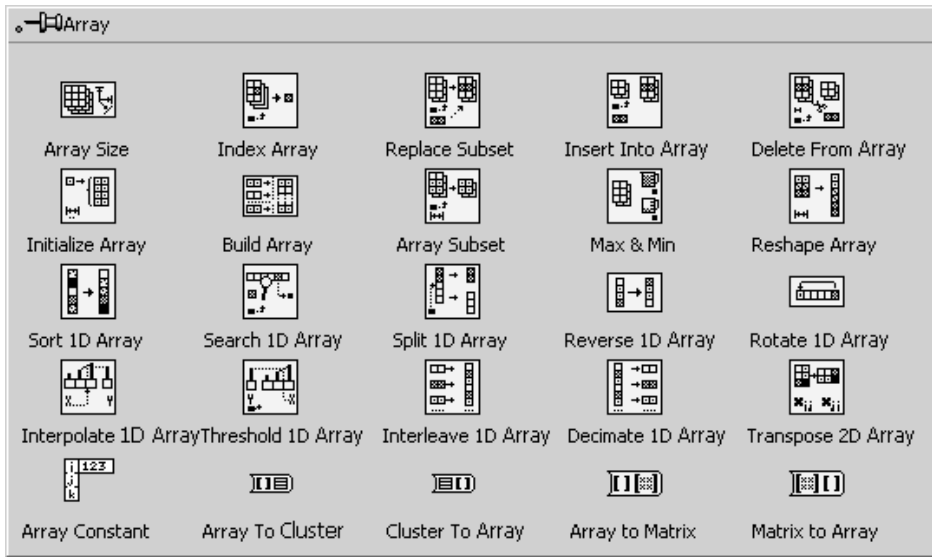


Abb. 9.20: Array-Funktionen in der Palette *Array*

Arrays sowie das Ersetzen, Löschen und Einfügen von einzelnen Elementen oder Teil-Arrays ermöglicht. Dabei soll schon an dieser Stelle darauf hingewiesen werden, dass die meisten Funktionen in LabVIEW polymorph sind. Das heißt, sie können unterschiedliche Datentypen und Datenstrukturen verarbeiten. Daher sind auch die meisten Funktionen aus den Funktionspaletten *Numeric* und *Comparison* direkt verwendbar.

Array Size (Array Größe)

Eine einfache Möglichkeit zur Ermittlung der Array-Größe von ein- und mehrdimensionalen Arrays bietet die Verwendung der Funktion *Array Size*. Sie liefert die Anzahl der Elemente des am Eingang angeschlossenen Arrays. Bei einem eindimensionalen Array ist das Ergebnis ein Skalar (Abb. 9.21a) und bei einem n -dimensionalen Array ist das Ergebnis ein eindimensionales Array mit n Elementen, wobei die Elemente der Funktion *Array Size* jeweils die dimensionsspezifische Länge des Eingangs-Arrays angeben (Abb. 9.21b).

Index Array (Array indizieren)

Die Funktion *Index Array* kann genutzt werden, um ein bestimmtes Element eines Arrays oder ein Teil-Array zu indizieren und am Ausgang der Funktion auszugeben. Für jede Dimension des zugeführten Arrays wird ein Index-Anschluss zur Verfügung gestellt. Wird beispielsweise, das in Abbildung 9.22a skizzierte eindimensionale Array mit dem Index 2 indiziert, wird das dritte Element, C, ausgegeben. Bei der in Abbildung 9.22b dargestellten Indizierung eines zweidimensionalen Arrays mit dem Index 2 für die Zeile und dem Index 3 für die Spalte wird das Element M in der dritten Zeile und vierten Spalte ausgegeben.

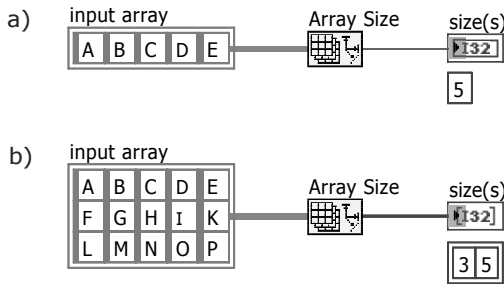


Abb. 9.21: Beispiel für die Verwendung der Funktion *Array Size*, a) mit einem eindimensionalen und b) mit einem mehrdimensionalen Array am Eingang der Funktion

Grundsätzlich ist es nicht erforderlich, alle Index-Eingänge der Funktion *Index Array* mit Werten zu belegen. Wird nur ein Index für die Zeile angegeben, stellt die Funktion *Index Array* die ganze Zeile am Ausgang bereit (Abb. 9.22c) und wenn nur ein Index für die Spalte angegeben wird, stellt die Funktion dementsprechend die ganze Spalte am Ausgang bereit (Abb. 9.22d).

In analoger Weise kann die Indizierung eines dreidimensionalen Arrays erfolgen, die in den Abbildungen 9.22e bis 9.22g beispielhaft gezeigt wird. Die Indizierung mit 1,1,1 für die Seite, Zeile und Spalte des zugeführten Arrays liefert das Element 5 der zweiten Seite in der zweiten Zeile und zweiten Spalte (Abb. 9.22e). Wenn der Index-Eingang für die Seite eines Arrays nicht verwendet wird, stellt die Funktion einen Querschnitt über alle Seiten, also z. B. alle Elemente der zweiten Zeile und ersten Spalte am Ausgang bereit (Abb. 9.22f). Für den Fall, dass nur der Eingang für die Indizierung einer Seite mit einem Wert belegt wird, stellt die Funktion die entsprechende Seite am Ausgang bereit (Abb. 9.22g).

Grundsätzlich ist es möglich, die Indizierung eines Arrays mit Hilfe der Funktion *Index Array* mehrfach vorzunehmen. Dazu kann der Funktionsknoten mit dem Positionierwerkzeug nach unten aufgezogen werden. Abbildung 9.23 zeigt dafür ein Beispiel, in dem ein zweidimensionales Array dreimal indiziert wird. Beispielsweise wird mit der Indizierung 1,1 das Element F aus der zweiten Spalte und aus der zweiten Zeile am obersten Ausgang der Funktion bereit gestellt.

► Beispiel: Erzeugen einer Q-Folge

Ein gutes Beispiel für die Verwendung der Funktion *Index Array* stellt die Erzeugung einer Q-Folge dar. Bei einer Q-Folge ergibt sich jedes neue Glied der Folge mit Hilfe der beiden vorigen Glieder nach Gleichung 9.1

$$Q_n = Q_{(n-Q_{n-1})} + Q_{(n-Q_{n-2})}, \quad (9.1)$$

wobei die ersten beiden Glieder der Folge mit $Q_1 = Q_2 = 1$ vorgegeben sind. Die Zahlenwerte der zwei vor dem aktuell zu berechnenden Glied geben an, wie weit jeweils rückwärts gezählt werden muss, um die beiden Zahlenwerte zu ermitteln, die für das neue Glied der Folge addiert werden sollen. Der Anfang der Zahlenfolge lautet somit: 1, 1, 2, 3, 3, 4, 5, 5, 6, 6, 6, 8, 8, 8, 10, 9, 10...

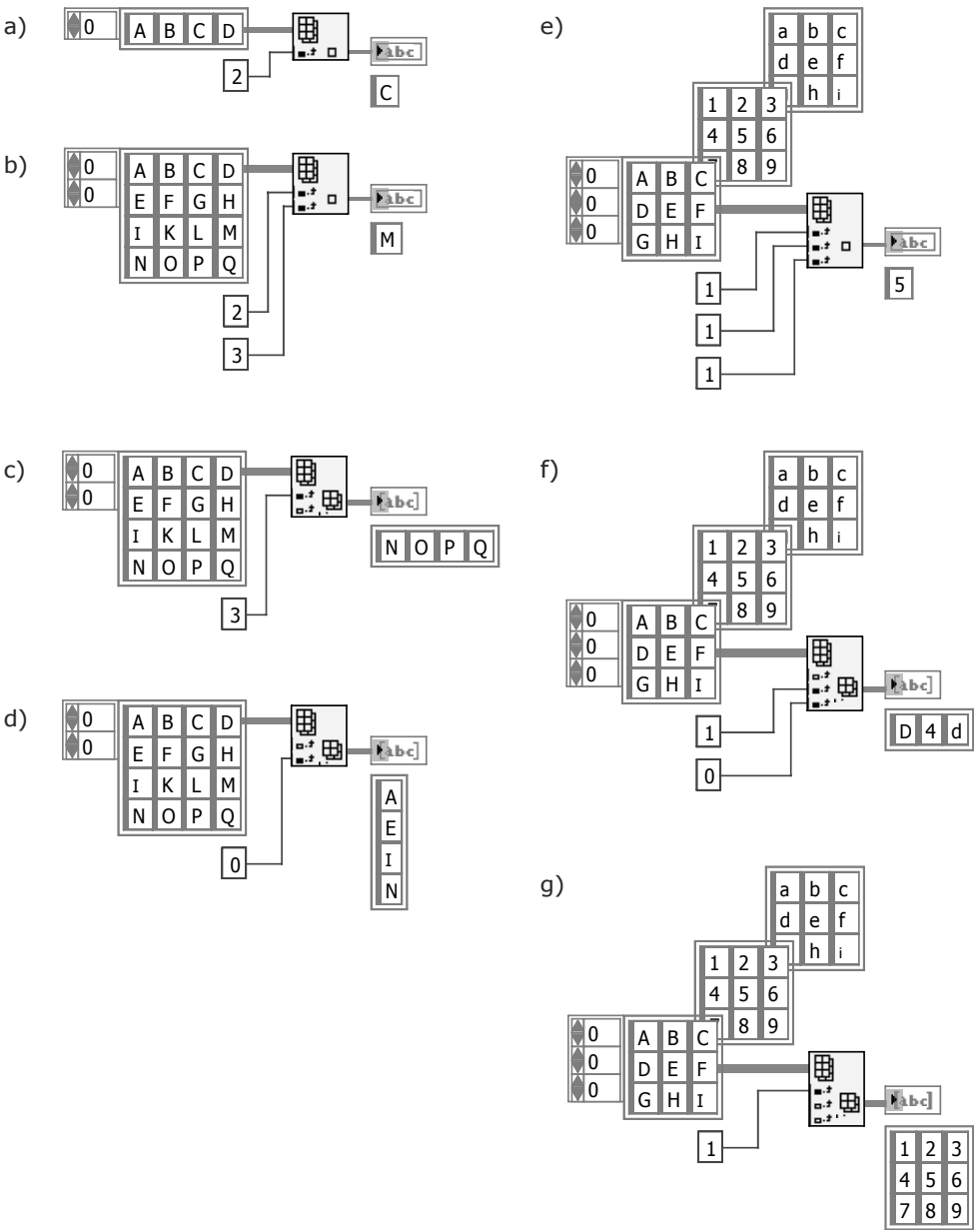


Abb. 9.22: Anwendungsbeispiele für die Funktion *Index Array*

Beispielsweise ergibt sich das 18. Glied der Q-Folge aus der zweiten 5 und der ersten 6, da einmal 10 und einmal 9 Elemente in der Folge zurückgegangen werden muss, um die Zahlenwerte zu ermitteln, die für die Bildung des 18. Gliedes addiert werden müssen.

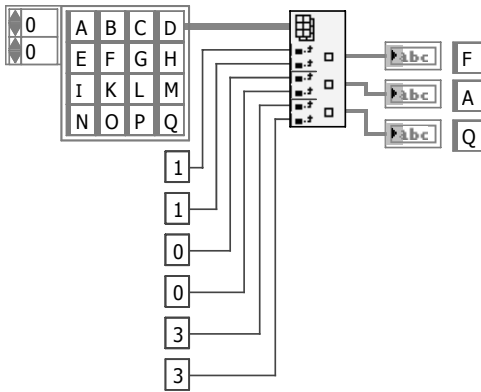


Abb. 9.23: Beispiel für eine mehrfache Indizierung eines Arrays mit Hilfe der Funktion *Index Array*

Abbildung 9.24 zeigt das *Block Diagram* des Programms *QSeries.vi*, welches die ersten 100 Glieder der Q-Folge berechnet. Da die ersten beiden Glieder der Folge gegeben sind, wird das Schieberegister der For-Schleife mit einem eindimensionalen Array, welches die ersten beiden Glieder der Q-Folge enthält, initialisiert und der Iterationszähler der For-Schleife wird um 2 erhöht. Um die Werte der Glieder $i-1$ und $i-2$ zu ermitteln, wird das Array der Q-Folge entsprechend indiziert. Diese beiden Glieder geben an, wie weit jeweils innerhalb der Q-Folge rückwärts gegangen werden muss. Mit diesen beiden Ergebnissen wird das Array der Q-Folge erneut indiziert, die beiden resultierenden Werte werden addiert und mit Hilfe der Funktion *Build Array* als neues Element an die bisherigen Glieder der Q-Folge angehängt.

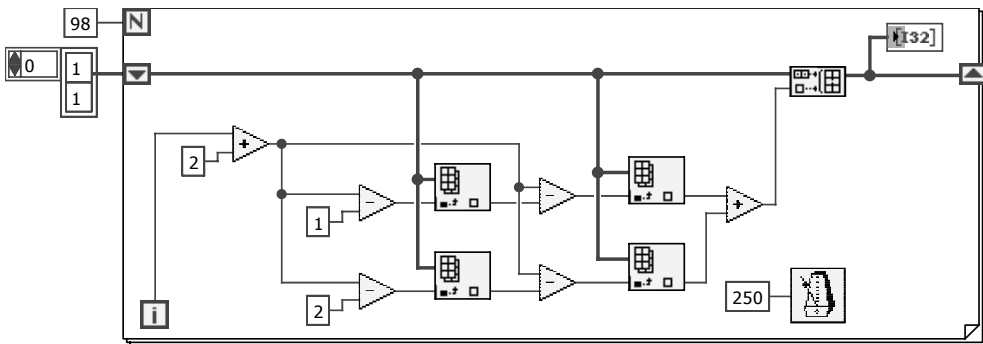


Abb. 9.24: *Block Diagram* des Programms *QSeries.vi* zur Ermittlung der ersten 100 Elemente einer Q-Folge

Bei den vier folgenden Funktionen zum Ersetzen, Einfügen, Löschen und Ausschneiden eines Elementes oder eines Teil-Arrays wird die Indizierung des zugeführten Arrays auf die gleiche Weise vorgenommen, wie bei der Funktion *Index Array*.

Grundsätzlich ist es bei mehrdimensionalen Arrays nicht erforderlich, alle Index-Eingänge der jeweiligen Funktion mit einem Zahlenwert zu belegen.

Replace Array Subset (Teilarray ersetzen)

Abbildung 9.25 zeigt beispielhaft das Ersetzen eines Elementes in einem zweidimensionalen Array. Die Indizierung 2,3 des zugeführten Arrays führt dazu, dass das Element M in der dritten Zeile und vierten Spalte durch das neue Element (*new element*) X ersetzt wird.

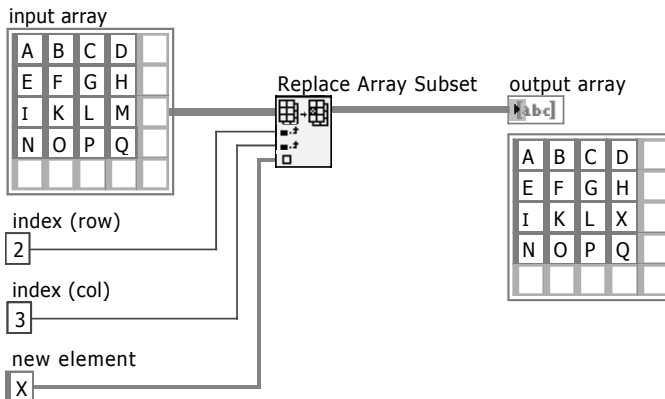


Abb. 9.25: Ersetzen eines Elementes in einem zweidimensionalen Array mit Hilfe der Funktion *Replace Array Subset*

Insert Into Array (In Array einfügen)

Um in ein bestehendes Array ein Element oder ein Teil-Array einzufügen, kann die Funktion *Insert Into Array* verwendet werden. Abbildung 9.26 zeigt ein Beispiel für diese Anwendung. In ein zweidimensionales Array wird ein eindimensionales Array, welches ein Element enthält, in der zweiten Zeile eingefügt. Da das zweidimensionale Array vier Spalten aufweist, werden die verbleibenden Elemente des zweidimensionalen Arrays mit dem Standardwert, in diesem Fall ein leerer String, aufgefüllt.

In diesem Beispiel ist der Index-Eingang für die Zeilen angeschlossen worden. Wäre der Index-Eingang für die Spalten mit einer 2 belegt worden, wäre das eindimensionale Array mit dem Element X als zweite Spalte in das eingehende Array eingefügt worden. Bei der Funktion *Insert Into Array* ist es bei einem zweidimensionalen Array grundsätzlich nur möglich, einen der beiden Index-Eingänge mit einem Zahlenwert zu belegen. Abbildung 9.27 zeigt ein weiteres Beispiel, bei dem in ein zweidimensionales Array ein zweidimensionales Array in der dritten Zeile eingefügt wird.

Handbuch für die Programmierung mit LabVIEW
mit Studentenversion LabVIEW 2009

Mütterlein, B.

2009, XVIII, 516 S., Hardcover

ISBN: 978-3-8274-1761-9