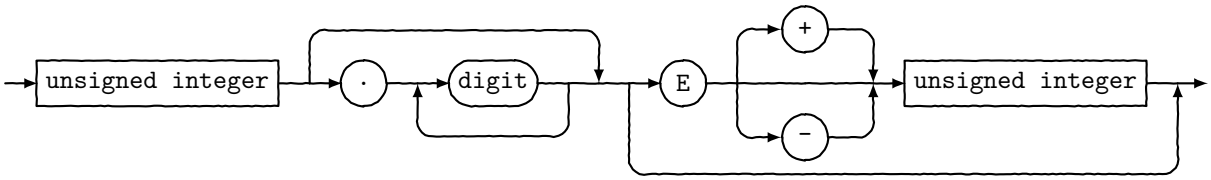
5 Tutorial: Diagrams as Simple Graphs
教程：将图表绘制为简单图形

In this tutorial we have a look at how graphs and matrices can be used to typeset a diagram.

在本教程中，我们将探讨如何使用图和矩阵来绘制图表。

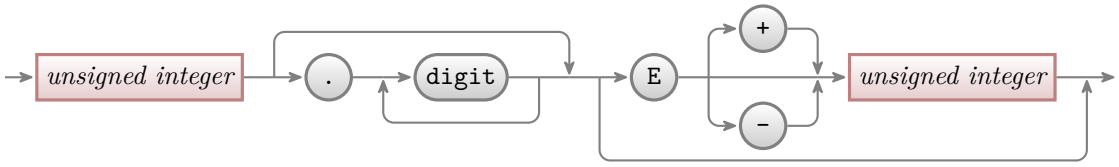
Ilka, who just got tenure for her professorship on Old and Lovable Programming Languages, has recently dug up a technical report entitled *The Programming Language Pascal* in the dusty cellar of the library of her university. Having been created in the good old times using pens and rules, it looks like this[1]:

Ilka 刚刚因其对古老且可爱的编程语言的教授职位获得终身教职，并最近在大学图书馆的地下室中挖掘出一份名为《Pascal 编程语言》的技术报告。这份报告是在过去使用钢笔和规则制作的，看起来像这样[2]：



For her next lecture, Ilka decides to redo this diagram, but this time perhaps a bit cleaner and perhaps also bit "cooler".

为了下一堂课，Ilka 决定重新制作这个图表，但这次可能会更加清晰，也更加“酷”。



Having read the previous tutorials, Ilka knows already how to set up the environment for her diagram, namely using a `tikzpicture` environment. She wonders which libraries she will need. She decides that she will postpone the decision and add the necessary libraries as needed as she constructs the picture.

在阅读了之前的教程后，Ilka 已经知道如何设置图表的环境，即使用`tikzpicture` 环境。她想知道她将需要哪些库。她决定在构建图表时根据需要添加所需的库。

5.1 Styling the Nodes
为节点设置样式

The bulk of this tutorial will be about arranging the nodes and connecting them using chains, but let us start with setting up styles for the nodes.

本教程的主要部分将涉及安排节点并使用链将它们连接起来，但让我们首先为节点设置样式。

There are two kinds of nodes in the diagram, namely what theoreticians like to call *terminals* and *nonterminals*. For the terminals, Ilka decides to use a black color, which visually shows that "nothing needs to be done about them". The nonterminals, which still need to be "processed" further, get a bit of red mixed in.

图表中有两种类型的节点，即理论家所称的“终端”和“非终端”。对于终端，Ilka 决定使用黑色，以视觉上表明“无需对其做任何处理”。非终端仍需要进一步的“处理”，因此会混入一些红色。

Ilka starts with the simpler nonterminals, as there are no rounded corners involved. Naturally, she sets up a style:

Ilka 从较简单的非终端开始，因为它们没有圆角。自然地，她设置了一个样式：

[1] The shown diagram was not scanned, but rather typeset using Ti*k*Z. The jittering lines were created using the `random steps` decoration.

[2] 所示的图表不是扫描的，而是使用 Ti*k*Z 排版得到的。抖动的线是使用`random steps` 装饰实现的。


```
\usetikzlibrary {positioning}
\begin{tikzpicture}[
    nonterminal/.style={
      % The shape:
      rectangle,
      % The size:
      minimum size=6mm,
      % The border:
      very thick,
      draw=red!50!black!50,          % 50% red and 50% black,
                                     % and that mixed with 50% white
      % The filling:
      top color=white,               % a shading that is white at the top...
      bottom color=red!50!black!20, % and something else at the bottom
      % Font
      font=\itshape
    }]
  \node [nonterminal] {unsigned integer};
\end{tikzpicture}
```

Ilka is pretty proud of the use of the `minimum size` option: As the name suggests, this option ensures that the node is at least 6mm by 6mm, but it will expand in size as necessary to accommodate longer text. By giving this option to all nodes, they will all have the same height of 6mm.

对于`minimum size` 选项，Ilka 感到非常自豪：正如其名称所示，该选项确保节点至少为 6mm 乘 6mm，但它会根据需要扩展大小以适应更长的文本。通过将此选项应用于所有节点，它们将都具有相同的高度为 6mm。

Styling the terminals is a bit more difficult because of the round corners. Ilka has several options how she can achieve them. One way is to use the `rounded corners` option. It gets a dimension as parameter and causes all corners to be replaced by little arcs with the given dimension as radius. By setting the radius to 3mm, she will get exactly what she needs: circles, when the shapes are, indeed, exactly 6mm by 6mm and otherwise half circles on the sides:

对终端的样式设置要困难一些，因为要处理圆角。Ilka 有几种方法可以实现它们。一种方法是使用`rounded corners` 选项。它接受一个尺寸作为参数，并将所有角替换为具有给定尺寸作为半径的小弧。通过将半径设置为 3mm，她将得到所需的效果：当形状确实为 6mm 乘 6mm 时，得到圆形，否则在两侧得到半圆：


```
\usetikzlibrary {positioning}
\begin{tikzpicture}[node distance=5mm,
                    terminal/.style={
                      % The shape:
                      rectangle,minimum size=6mm,rounded corners=3mm,
                      % The rest
                      very thick,draw=black!50,
                      top color=white,bottom color=black!20,
                      font=\ttfamily}]
  \node (dot)    [terminal]                {.};
  \node (digit) [terminal,right=of dot]   {digit};
  \node (E)      [terminal,right=of digit] {E};
\end{tikzpicture}
```

Another possibility is to use a shape that is specially made for typesetting rectangles with arcs on the sides (she has to use the `shapes.misc` library to use it). This shape gives Ilka much more control over the appearance. For instance, she could have an arc only on the left side, but she will not need this.

另一种可能性是使用一种专门用于绘制带有侧边弧的矩形的形状（她必须使用`shapes.misc` 库才能使用它）。该形状使 Ilka 对外观有更多的控制权。例如，她可以仅在左侧使用弧，但她不需要这样做。


```
\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm,
                    terminal/.style={
                      % The shape:
                      rounded rectangle,
                      minimum size=6mm,
                      % The rest
                      very thick,draw=black!50,
                      top color=white,bottom color=black!20,
                      font=\ttfamily}]
  \node (dot)    [terminal]                {.};
  \node (digit) [terminal,right=of dot]   {digit};
  \node (E)      [terminal,right=of digit] {E};
\end{tikzpicture}
```

At this point, she notices a problem. The baseline of the text in the nodes is not aligned:
此时，她注意到了一个问题。节点中的文本的基线没有对齐：


```
\usetikzlibrary {calc,positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm]
  \node (dot)   [terminal]                {.};
  \node (digit) [terminal,right=of dot]   {digit};
  \node (E)     [terminal,right=of digit] {E};

  \draw [help lines] let \p1 = (dot.base),
                         \p2 = (digit.base),
                         \p3 = (E.base)
                     in (-.5,\y1) -- (3.5,\y1)
                        (-.5,\y2) -- (3.5,\y2)
                        (-.5,\y3) -- (3.5,\y3);
\end{tikzpicture}
```

(Ilka has moved the style definition to the preamble by saying `\tikzset{terminal/.style=...}`, so that she can use it in all pictures.)
(Ilka 将样式定义移到导言区中，使用`\tikzset{terminal/.style=...}`，以便在所有图表中都可以使用它。)

For the `digit` and the `E` the difference in the baselines is almost imperceptible, but for the dot the problem is quite severe: It looks more like a multiplication dot than a period.
对于`digit` 和`E`，基线之间的差异几乎不可见，但对于点号，问题相当严重：它看起来更像是乘法点而不是句点。

Ilka toys with the idea of using the `base right=of...` option rather than `right=of...` to align the nodes in such a way that the baselines are all on the same line (the `base right` option places a node right of something so that the baseline is right of the baseline of the other object). However, this does not have the desired effect:
Ilka 心生一个想法，即使用`base right=of...` 选项而不是`right=of...` 选项来对齐节点，使得基线都在同一条线上（`base right` 选项将一个节点放置在另一个节点的右侧，以使得其基线位于另一个对象的基线的右侧）。然而，这没有产生期望的效果：


```
\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm]
  \node (dot)   [terminal]                     {.};
  \node (digit) [terminal,base right=of dot]   {digit};
  \node (E)     [terminal,base right=of digit] {E};
\end{tikzpicture}
```

The nodes suddenly “dance around”! There is no hope of changing the position of text inside a node using anchors. Instead, Ilka must use a trick: The problem of mismatching baselines is caused by the fact that `.` and `digit` and `E` all have different heights and depth. If they all had the same, they would all be positioned vertically in the same manner. So, all Ilka needs to do is to use the `text height` and `text depth` options to explicitly specify a height and depth for the nodes.
节点突然“乱动”起来！无法通过锚点更改节点内文本的位置。相反，Ilka 必须使用一个技巧：基线不匹配的问题是由于`.` 和`digit` 以及`E` 的高度和深度不同所导致的。如果它们都相同，它们将以相同的方式在垂直方向上定位。因此，Ilka 所需做的就是使用`text height` 和`text depth` 选项明确指定节点的高度和深度。


```
\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm,
                    text height=1.5ex,text depth=.25ex]
  \node (dot)   [terminal]                {.};
  \node (digit) [terminal,right=of dot]   {digit};
  \node (E)     [terminal,right=of digit] {E};
\end{tikzpicture}
```

5.2 Aligning the Nodes Using Positioning Options
使用定位选项对齐节点

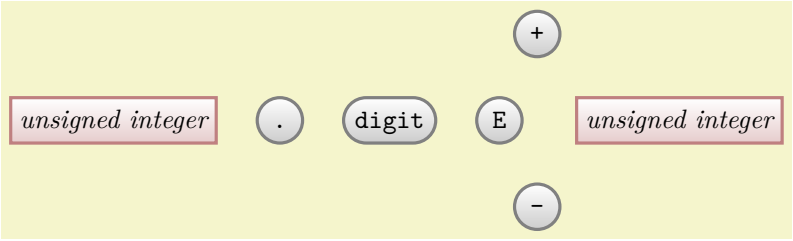
Ilka now has the “styling” of the nodes ready. The next problem is to place them in the right places. There are several ways to do this. The most straightforward is to simply explicitly place the nodes at certain coordinates “calculated by hand”. For very simple graphics this is perfectly alright, but it has several disadvantages:
Ilka 现在已经准备好节点的“样式”。下一个问题是将它们放置在正确的位置。有几种方法可以实现。最直接的方法是直接在“手工计算”的某些坐标处放置节点。对于非常简单的图形，这是完全可以的，但它有几个缺点：

1. For more difficult graphics, the calculation may become complicated.
   对于更复杂的图形，计算可能变得复杂。
2. Changing the text of the nodes may make it necessary to recalculate the coordinates.
   更改节点的文本可能需要重新计算坐标。
3. The source code of the graphic is not very clear since the relationships between the positions of the nodes are not made explicit.
   图形的源代码不太清晰，因为节点之间的位置关系没有明确表达。

For these reasons, Ilka decides to try out different ways of arranging the nodes on the page.
出于这些原因，Ilka 决定尝试在页面上排列节点的不同方式。

The first method is the use of *positioning options*. To use them, you need to load the `positioning` library. This gives you access to advanced implementations of options like `above` or `left`, since you can now say `above=of some node` in order to place a node above of `some node`, with the borders separated by `node distance`.
第一种方法是使用*定位选项*。要使用它们，你需要加载`positioning` 库。这使得你可以访问高级选项的实现，如`above` 或`left`，因为现在你可以说`above=of some node` 以将一个节点放置在`some node` 的上方，边界由`node distance` 分离。

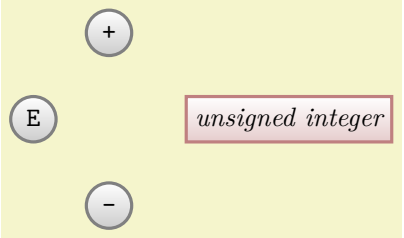
Ilka can use this to draw the place the nodes in a long row:
Ilka 可以使用这个选项将节点放置在一行中：




```
\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm]
  \node (ui1)   [nonterminal]                    {unsigned integer};
  \node (dot)   [terminal,right=of ui1]          {.};
  \node (digit) [terminal,right=of dot]          {digit};
  \node (E)     [terminal,right=of digit]        {E};
  \node (plus)  [terminal,above right=of E]      {+};
  \node (minus) [terminal,below right=of E]      {-};
  \node (ui2)   [nonterminal,below right=of plus] {unsigned integer};
\end{tikzpicture}
```

For the plus and minus nodes, Ilka is a bit startled by their placements. Shouldn't they be more to the right? The reason they are placed in that manner is the following: The `north east` anchor of the `E` node lies at the "upper start of the right arc", which, a bit unfortunately in this case, happens to be the top of the node. Likewise, the `south west` anchor of the `+` node is actually at its bottom and, indeed, the horizontal and vertical distances between the top of the `E` node and the bottom of the `+` node are both 5mm.
对于加号和减号节点，Ilka 对它们的位置有些惊讶。它们不应该更靠右吗？它们被放置在那个位置的原因是：`E` 节点的`north east` 锚点位于"右弧的上部起点"，这在这种情况下有点不幸，恰好是节点的顶部。同样，`+` 节点的`south west` 锚点实际上位于其底部，并且实际上`E` 节点的顶部和`+` 节点的底部之间的水平和垂直距离都为 5mm。

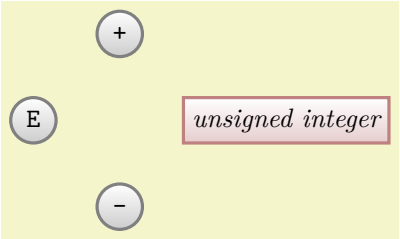
There are several ways of fixing this problem. The easiest way is to simply add a little bit of horizontal shift by hand:
有几种方法可以解决这个问题。最简单的方法是通过手动添加一点水平偏移量来解决：



```
\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm]
  \node (E)     [terminal]                                    {E};
  \node (plus)  [terminal,above right=of E,xshift=5mm]        {+};
  \node (minus) [terminal,below right=of E,xshift=5mm]        {-};
  \node (ui2)   [nonterminal,below right=of plus,xshift=5mm] {unsigned integer};
\end{tikzpicture}
```

A second way is to revert back to the idea of using a normal rectangle for the terminals, but with rounded corners. Since corner rounding does not affect anchors, she gets the following result:

第二种方法是回到使用普通矩形来表示终端的想法，但带有圆角。由于角的圆角不会影响锚点，因此可以得到以下结果：




```
\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm,terminal/.append style={rectangle,rounded corners=3mm}]
  \node (E)     [terminal]                       {E};
  \node (plus)  [terminal,above right=of E]      {+};
  \node (minus) [terminal,below right=of E]      {-};
  \node (ui2)   [nonterminal,below right=of plus] {unsigned integer};
\end{tikzpicture}
```

A third way is to use matrices, which we will do later.

第三种方法是使用后面将要介绍的矩阵。

Now that the nodes have been placed, Ilka needs to add connections. Here, some connections are more difficult than others. Consider for instance the "repeat" line around the `digit`. One way of describing this line is to say "it starts a little to the right of `digit` than goes down and then goes to the left and finally ends at a point a little to the left of `digit`". Ilka can put this into code as follows:

现在节点已经放置好了，Ilka 需要添加连接。这里，有些连接比其他连接更难处理。例如，考虑围绕`digit`的"repeat"线条。描述这条线条的一种方法是说"它从`digit` 的右侧稍微向右移动，然后向下移动，然后向左移动，最后以稍微在`digit` 的左侧的一个点结束"。Ilka 可以将其编码如下：


```
\usetikzlibrary {calc,positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm]
  \node (dot)   [terminal]                    {.};
  \node (digit) [terminal,right=of dot]       {digit};
  \node (E)     [terminal,right=of digit]     {E};

  \path (dot)   edge[->] (digit)  % simple edges
        (digit) edge[->] (E);

  \draw [->]
     % start right of digit.east, that is, at the point that is the
     % linear combination of digit.east and the vector (2mm,0pt). We
     % use the ($ ... $) notation for computing linear combinations
     ($ (digit.east) + (2mm,0) $)
     % Now go down
     -- ++(0,-.5)
     % And back to the left of digit.west
     -| ($ (digit.west) - (2mm,0) $);
\end{tikzpicture}
```

Since Ilka needs this "go up/down then horizontally and then up/down to a target" several times, it seems sensible to define a special *to-path* for this. Whenever the `edge` command is used, it simply adds the current value of `to path` to the path. So, Ilka can set up a style that contains the correct path:

由于 Ilka 需要多次执行"向上/向下，然后水平，再向上/向下到目标"操作，因此定义一个特殊的 *to-path* 似乎是合理的。每当使用`edge` 命令时，它会将当前的`to path` 值添加到路径中。因此，Ilka 可以设置一个包含正确路径的样式：


```
\usetikzlibrary {calc,positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm,
    skip loop/.style={to path={-- ++(0,-.5) -| (\tikztotarget)}}]
  \node (dot)   [terminal]                          {.};
  \node (digit) [terminal,right=of dot]             {digit};
  \node (E)     [terminal,right=of digit]           {E};

  \path (dot)   edge[->]           (digit)  % simple edges
        (digit) edge[->]           (E)
        ($ (digit.east) + (2mm,0) $)
                edge[->,skip loop] ($ (digit.west) - (2mm,0) $);
\end{tikzpicture}
```

Ilka can even go a step further and make her `skip loop` style parameterized. For this, the skip loop's vertical offset is passed as parameter `#1`. Also, in the following code Ilka specifies the start and targets differently, namely as the positions that are "in the middle between the nodes".

Ilka 甚至可以进一步使她的`skip loop` 样式参数化。为此，跳过循环的垂直偏移作为参数`#1` 传递。此外，在下面的代码中，Ilka 将开始和目标指定方式进行了不同的指定，即作为 "在节点之间的中间位置"。


```
\usetikzlibrary {calc,positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm,
    skip loop/.style={to path={-- ++(0,#1) -| (\tikztotarget)}}]
  \node (dot)   [terminal]                          {.};
  \node (digit) [terminal,right=of dot]             {digit};
  \node (E)     [terminal,right=of digit]           {E};

  \path (dot)   edge[->]               (digit)  % simple edges
        (digit) edge[->]               (E)
        ($ (digit.east)!.5!(E.west) $)
                edge[->,skip loop=-5mm] ($ (digit.west)!.5!(dot.east) $);
\end{tikzpicture}
```

5.3 Aligning the Nodes Using Matrices
使用矩阵对节点进行对齐

Ilka is still bothered a bit by the placement of the plus and minus nodes. Somehow, having to add an explicit `xshift` seems too much like cheating.

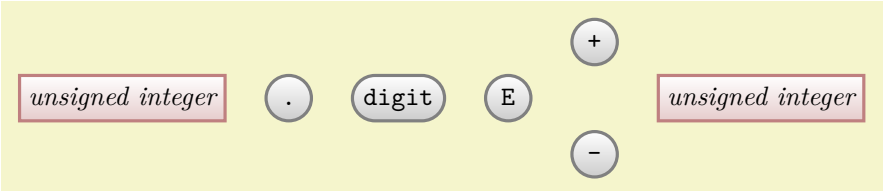
Ilka 仍然对加号和减号节点的位置感到困扰。不知何故，需要显式添加`xshift` 似乎太像作弊了。

A perhaps better way of positioning the nodes is to use a *matrix*. In Ti*k*Z matrices can be used to align quite arbitrary graphical objects in rows and columns. The syntax is very similar to the use of arrays and tables in TEX (indeed, internally TEX tables are used, but a lot of stuff is going on additionally).

一种更好的定位节点的方法是使用*矩阵*。在 Ti*k*Z 中，矩阵可用于在行和列中对几乎任意图形对象进行对齐。语法与在 TEX 中使用数组和表格非常相似（实际上，内部使用了 TEX 的表格，但还有很多其他操作）。

In Ilka's graphic, there will be three rows: One row containing only the plus node, one row containing the main nodes and one row containing only the minus node.

在 Ilka 的图形中，将有三行：一行仅包含加号节点，一行包含主要节点，一行仅包含减号节点。



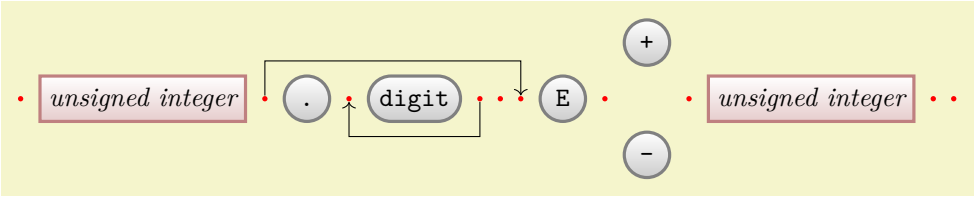
```
\usetikzlibrary {shapes.misc}
\begin{tikzpicture}
  \matrix[row sep=1mm,column sep=5mm] {
    % First row:
      & & & & \node [terminal] {+}; & \\
    % Second row:
    \node [nonterminal] {unsigned integer}; &
    \node [terminal]    {.};                &
    \node [terminal]    {digit};            &
    \node [terminal]    {E};                &
                                            &
    \node [nonterminal] {unsigned integer}; \\
    % Third row:
      & & & & \node [terminal] {-}; & \\
  };
\end{tikzpicture}
```

That was easy! By toying around with the row and columns separations, Ilka can achieve all sorts of pleasing arrangements of the nodes.

这很简单！通过调整行和列之间的间隔，Ilka 可以实现各种令人愉悦的节点布局。

Ilka now faces the same connecting problem as before. This time, she has an idea: She adds small nodes (they will be turned into coordinates later on and be invisible) at all the places where she would like connections to start and end.

现在，Ilka 面临的问题与以前一样。这次，她有了一个主意：她在所有她希望连接开始和结束的位置上添加了小节点（它们将在稍后转换为坐标并且不可见）。




```
\usetikzlibrary {shapes.misc}
\begin{tikzpicture}[point/.style={circle,inner sep=0pt,minimum size=2pt,fill=red},
                    skip loop/.style={to path={-- ++(0,#1) -| (\tikztotarget)}}]
  \matrix[row sep=1mm,column sep=2mm] {
    % First row:
    & & & & & & & & & \node (plus) [terminal] {+};\\
    % Second row:
    \node (p1) [point]  {}; &    \node (ui1)   [nonterminal] {unsigned integer}; &
    \node (p2) [point]  {}; &    \node (dot)   [terminal]    {.};                &
    \node (p3) [point]  {}; &    \node (digit) [terminal]    {digit};            &
    \node (p4) [point]  {}; &    \node (p5)    [point]  {};                       &
    \node (p6) [point]  {}; &    \node (e)     [terminal]    {E};                &
    \node (p7) [point]  {}; &                                                     &
    \node (p8) [point]  {}; &    \node (ui2)   [nonterminal] {unsigned integer}; &
    \node (p9) [point]  {}; &    \node (p10)   [point]       {};\\
    % Third row:
    & & & & & & & & & \node (minus)[terminal] {-};\\
  };

  \path (p4) edge [->,skip loop=-5mm] (p3)
        (p2) edge [->,skip loop=5mm]  (p6);
\end{tikzpicture}
```

Now, it's only a small step to add all the missing edges.

现在，只需要添加所有缺失的边。

5.4 The Diagram as a Graph
将图表视为图

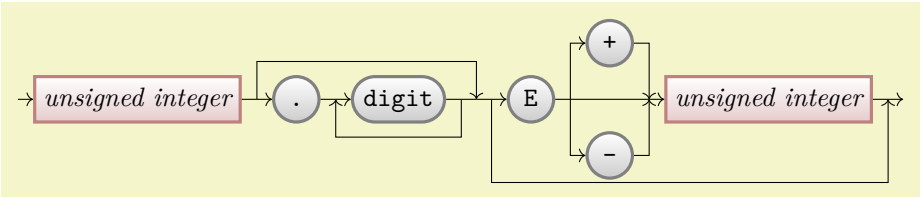
Matrices allow Ilka to align the nodes nicely, but the connections are not quite perfect. The problem is that the code does not really reflect the paths that underlie the diagram. For this, it seems natural enough to Ilka to use the `graphs` library since, after all, connecting nodes by edges is exactly what happens in a graph. The `graphs` library can both be used to connect nodes that have already been created, but it can also be used to create nodes "on the fly" and these processes can also be mixed.

矩阵使 Ilka 能够很好地对齐节点，但连接并不完美。问题在于代码实际上并没有反映出底层图表的路径。为此，使用graphs 库似乎是最自然的选择，毕竟，连接节点的边正是图中发生的事情。graphs 库既可用于连接已经创建的节点，也可用于“即时”创建节点，而且这两个过程也可以混合使用。

5.4.1 Connecting Already Positioned Nodes 连接已经定位的节点

Ilka has already a fine method for positioning her nodes (using a `matrix`), so all that she needs is an easy way of specifying the edges. For this, she uses the `\graph` command (which is actually just a shorthand for `\path graph`). It allows her to write down edges between them in a simple way (the macro `\matrixcontent` contains exactly the matrix content from the previous example; no need to repeat it here):

Ilka 已经有了一种很好的方法来定位她的节点（使用`matrix`），所以她所需要的就是一种简单的方法来指定边。为此，她使用了`\graph` 命令（实际上只是`\path graph` 的简写）。它允许她以简单的方式写出它们之间的边（宏`\matrixcontent` 包含了前面示例中的矩阵内容，这里不需要重复）：




```
\usetikzlibrary {graphs,shapes.misc}
\begin{tikzpicture}[skip loop/.style={to path={-- ++(0,#1) -| (\tikztotarget)}},
                    hv path/.style={to path={-| (\tikztotarget)}},
                    vh path/.style={to path={|- (\tikztotarget)}}]
  \matrix[row sep=1mm,column sep=2mm] { \matrixcontent };

  \graph {
    (p1) -> (ui1) -- (p2) -> (dot) -- (p3) -> (digit) -- (p4)
         -- (p5)  -- (p6) -> (e) -- (p7) -- (p8) -> (ui2) -- (p9) -> (p10);
    (p4) ->[skip loop=-5mm]  (p3);
    (p2) ->[skip loop=5mm]   (p5);
    (p6) ->[skip loop=-11mm] (p9);
    (p7) ->[vh path]         (plus)  -> [hv path] (p8);
    (p7) ->[vh path]         (minus) -> [hv path] (p8);
  };
\end{tikzpicture}
```

This is already pretty near to the desired result, just a few “finishing touches” are needed to style the edges more nicely.

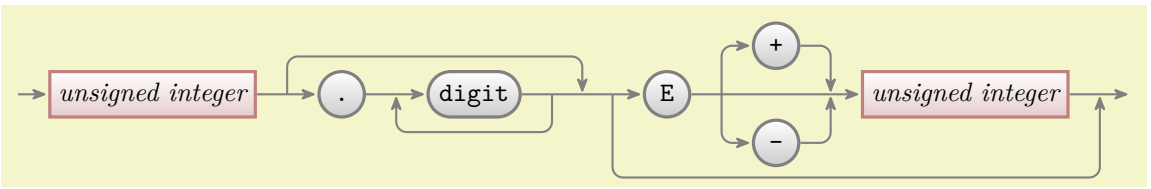
这已经非常接近预期的结果了，只需要一些“最后的修饰”来更好地设置边的样式。

However, Ilka does not have the feeling that the `graph` command is all that hot in the example. It certainly does cut down on the number of characters she has to write, but the overall graph structure is not that much clear – it is still mainly a list of paths through the graph. It would be nice to specify that, say, there the path from `(p7)` sort of splits to `(plus)` and `(minus)` and then merges once more at `(p8)`. Also, all these parentheses are bit hard to type.

然而，Ilka 并不认为`graph` 命令在这个例子中十分出色。它确实减少了她需要编写的字符数，但整体的图结构并不是那么明显——它仍然主要是图中路径的列表。可以指定，例如，从`(p7)` 到`plus` 和`minus` 之间的路径在某种程度上分叉，然后在`(p8)` 处再次合并。而且，所有这些括号都很难键入。

It turns out that edges from a node to a whole group of nodes are quite easy to specify, as shown in the next example. Additionally, by using the `use existing nodes` option, Ilka can also leave out all the parentheses (again, some options have been moved outside to keep the examples shorter):

事实证明，从一个节点到一组节点的边很容易指定，如下面的示例所示。此外，通过使用`use existing nodes` 选项，Ilka 还可以省略所有括号（再次，一些选项已经移到外面以缩短示例）：

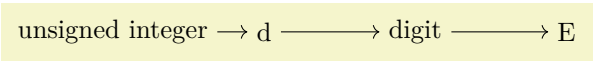


```
\usetikzlibrary {arrows.meta,graphs,shapes.misc}
\begin{tikzpicture}[>={Stealth[round]},thick,black!50,text=black,
                    every new ->/.style={shorten >=1pt},
                    graphs/every graph/.style={edges=rounded corners}]
  \matrix[column sep=4mm] { \matrixcontent };

  \graph [use existing nodes] {
    p1 -> ui1 -- p2 -> dot -- p3 -> digit -- p4 -- p5  -- p6 -> e -- p7 -- p8 -> ui2 -- p9 -> p10;
    p4 ->[skip loop=-5mm]  p3;
    p2 ->[skip loop=5mm]   p5;
    p6 ->[skip loop=-11mm] p9;
    p7 ->[vh path] { plus, minus } -> [hv path] p8;
  };
\end{tikzpicture}
```

5.4.2 Creating Nodes Using the Graph Command
使用图命令创建节点

Ilka has heard that the `graph` command is also supposed to make it easy to create nodes, not only to connect them. This is, indeed, correct: When the `use existing nodes` option is not used and when a node name is not surrounded by parentheses, then TikZ will actually create a node whose name and text is the node name:

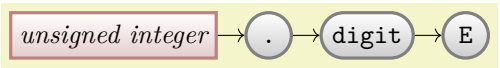
Ilka 听说`graph` 命令还可以方便地创建节点，不仅可以连接它们。这是正确的：当没有使用`use existing nodes` 选项并且节点名称没有用括号括起来时，TikZ 实际上会创建一个节点，其名称和文本都是节点名称：




```
\usetikzlibrary {graphs}
\tikz \graph [grow right=2cm] { unsigned integer -> d -> digit -> E };
```

Not quite perfect, but we are getting somewhere. First, let us change the positioning algorithm by saying `grow right sep`, which causes new nodes to be placed to the right of the previous nodes with a certain fixed separation (`1em` by default). Second, we add some options to make the node "look nice". Third, note the funny `d` node above: Ilka tried writing just `.` there first, but got some error messages. The reason is that a node cannot be called `.` in TikZ, so she had to choose a different name – which is not good, since she wants a dot to be shown! The trick is to put the dot in quotation marks, this allows you to use "quite arbitrary text" as a node name:

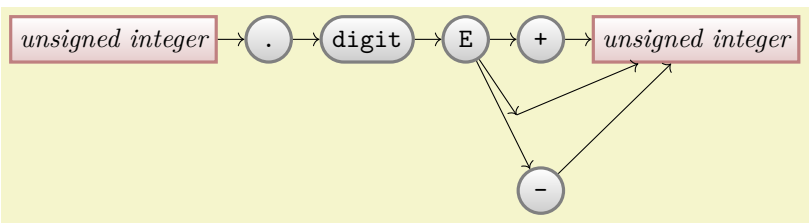
还不是很完美，但我们正在取得进展。首先，让我们通过使用`grow right sep` 命令来更改定位算法，这将使新节点放置在前一个节点的右侧，并保持一定的固定间距（默认为`1em`）。其次，我们添加一些选项来使节点"看起来不错"。第三，在上面的例子中注意到有一个有趣的节点`d`：Ilka 首先尝试只在那里写一个`.`，但是得到了一些错误消息。原因是在 TikZ 中节点不能被称为 `.`，所以她不得不选择一个不同的名称——这并不好，因为她希望显示一个点！诀窍是将点放在引号中，这样你就可以使用"相当任意的文本"作为节点名称：




```
\usetikzlibrary {graphs,shapes.misc}
\tikz \graph [grow right sep] {
  unsigned integer[nonterminal] -> "."[terminal] -> digit[terminal] -> E[terminal]
};
```

Now comes the fork to the plus and minus signs. Here, Ilka can use the grouping mechanism of the `graph` command to create a split:

现在到了分支加减号的部分。在这里，Ilka 可以使用`graph` 命令的分组机制来创建一个分支：



```
\usetikzlibrary {graphs,shapes.misc}
\tikz \graph [grow right sep] {
  unsigned integer  [nonterminal] ->
  "."               [terminal] ->
  digit             [terminal] ->
  E                 [terminal] ->
  {
    "+"             [terminal],
    ""              [coordinate],
    "-"             [terminal]
  } ->
  ui2/unsigned integer [nonterminal]
};
```

Let us see, what is happening here. We want two `unsigned integer` nodes, but if we just were to use this text twice, then Ti*k*Z would have noticed that the same name was used already in the current graph and, being smart (actually too smart in this case), would have created an edge back to the already-created node. Thus, a fresh name is needed here. However, Ilka also cannot just write `unsigned integer2`, because she wants the original text to be shown, after all! The trick is to use a slash inside the node name: In order to "render" the node, the text following the slash is used instead of the node name, which is the text before the slash. Alternatively, the `as` option can be used, which also allows you to specify how a node should be rendered.

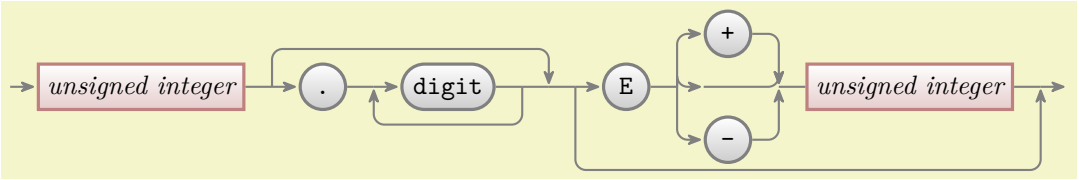
让我们看看这里发生了什么。我们想要两个`unsigned integer` 节点，但是如果我们只是两次使用相同的文本，那么 Ti*k*Z 会注意到在当前图中已经使用了相同的名称，并且会创建一条返回到已创建节点的边。因此，在这里需要一个新的名称。然而，Ilka 也不能只是写`unsigned integer2`，因为她希望显示原始文本！诀窍是在节点名称中使用斜杠：为了"渲染"节点，斜杠后面的文本将被用于替代节点名称，节点名称则是斜杠前面的文本。或者，也可以使用`as` 选项，它还允许指定节点的呈现方式。

It turns out that Ilka does not need to invent a name like `ui2` for a node that she will not reference again anyway. In this case, she can just leave out the name (write nothing before `/`), which always stands for a "fresh, anonymous" node name.

事实证明，Ilka 不需要为她不会再次引用的节点发明一个像`ui2` 这样的名称。在这种情况下，她可以只留空名称（在`/`之前不写任何内容），它始终代表一个"新的匿名"节点名称。

Next, Ilka needs to add some coordinates in between of some nodes where the back-loops should got and she needs to shift the nodes a bit:

接下来，Ilka 需要在一些节点之间添加一些坐标，用于连接反向循环，并且她需要稍微移动节点：



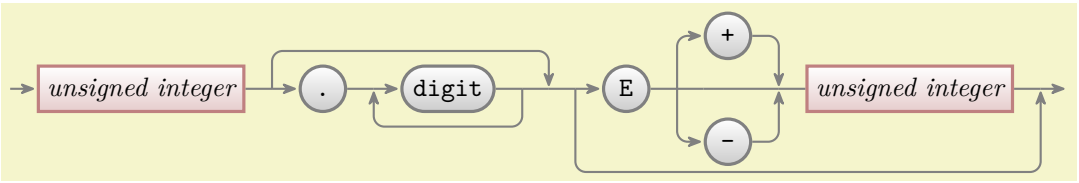
```
\usetikzlibrary {arrows.meta,graphs,shapes.misc}
\begin{tikzpicture}[>={Stealth[round]}, thick, black!50, text=black,
                    every new ->/.style={shorten >=1pt},
                    graphs/every graph/.style={edges=rounded corners}]
  \graph [grow right sep, branch down=7mm] {
    /                   [coordinate] ->
    unsigned integer    [nonterminal] --
    p1                  [coordinate] ->
    "."                 [terminal] --
    p2                  [coordinate] ->
    digit               [terminal] --
    p3                  [coordinate] --
    p4                  [coordinate] --
    p5                  [coordinate] ->
    E                   [terminal] --
    q1                  [coordinate] ->[vh path]
    { [nodes={yshift=7mm}]
      "+"                 [terminal],
      q2/                 [coordinate],
      "-"                 [terminal]
    } -> [hv path]
    q3                  [coordinate] --
    /unsigned integer   [nonterminal] --
    p6                  [coordinate] ->
    /                   [coordinate];

    p1 ->[skip loop=5mm]   p4;
    p3 ->[skip loop=-5mm]  p2;
    p5 ->[skip loop=-11mm] p6;
  };
\end{tikzpicture}
```

All that remains to be done is to somehow get rid of the strange curves between the `E` and the unsigned integer. They are caused by Ti*k*Z's attempt at creating an edge that first goes vertical and then horizontal but is actually just horizontal. Additionally, the edge should not really be pointed; but it seems difficult to get rid of this since the *other* edges from `q1`, namely to `plus` and `minus` should be pointed.

剩下的就是如何消除`E` 和 unsigned integer 之间的奇怪曲线了。它们是由 Ti*k*Z 试图创建首先垂直然后水平的边而实际上只是水平的边引起的。此外，该边实际上不应该有箭头；但是似乎很难摆脱，因为从`q1` 到其他边，即`plus` 和`minus`，应该有箭头。

It turns out that there is a nice way of solving this problem: You can specify that a graph is `simple`. This means that there can be at most one edge between any two nodes. Now, if you specify an edge twice, the options of the second specification "win". Thus, by adding two more lines that "correct" these edges, we get the final diagram with its complete code:

事实证明，有一种很好的方法可以解决这个问题：您可以指定图是`simple` 的。这意味着任何两个节点之间最多只能有一条边。现在，如果您指定了两次边，第二次规定的选项将"胜出"。因此，通过添加两行来"修正"这些边，我们得到了带有完整代码的最终图表：



```
\usetikzlibrary {arrows.meta,graphs,shapes.misc}
\tikz [>={Stealth[round]}, black!50, text=black, thick,
       every new ->/.style          = {shorten >=1pt},
       graphs/every graph/.style    = {edges=rounded corners},
       skip loop/.style             = {to path={-- ++(0,#1) -| (\tikztotarget)}},
       hv path/.style               = {to path={-| (\tikztotarget)}},
       vh path/.style               = {to path={|- (\tikztotarget)}},
       nonterminal/.style           = {
         rectangle, minimum size=6mm, very thick, draw=red!50!black!50, top color=white,
         bottom color=red!50!black!20, font=\itshape, text height=1.5ex,text depth=.25ex},
       terminal/.style              = {
         rounded rectangle,  minimum size=6mm, very thick, draw=black!50, top color=white,
         bottom color=black!20, font=\ttfamily, text height=1.5ex, text depth=.25ex},
       shape                        = coordinate
       ]
  \graph [grow right sep, branch down=7mm, simple] {
    / -> unsigned integer[nonterminal] -- p1 -> "." [terminal] -- p2 -> digit[terminal] --
    p3 -- p4 -- p5 -> E[terminal] -- q1 ->[vh path]
    {[nodes={yshift=7mm}]
      "+"[terminal], q2, "-"[terminal]
    } -> [hv path]
    q3 -- /unsigned integer [nonterminal] -- p6 -> /;

    p1 ->[skip loop=5mm]   p4;
    p3 ->[skip loop=-5mm]  p2;
    p5 ->[skip loop=-11mm] p6;

    q1 -- q2 -- q3;  % make these edges plain
  };
```

5.5 Using Chains
使用链表

Matrices allow Ilka to align the nodes nicely, but the connections are not quite perfect. The problem is that the code does not really reflect the paths that underlie the diagram.

矩阵允许 Ilka 将节点对齐得很好，但连接并不完美。问题在于代码实际上并不反映图表的底层路径。

For this reason, Ilka decides to try out *chains* by including the `chain` library. Basically, a chain is just a sequence of (usually) connected nodes. The nodes can already have been constructed or they can be constructed as the chain is constructed (or these processes can be mixed).

由于这个原因，Ilka 决定尝试使用包含`chain` 库的 *chains*。基本上，链是一系列（通常）相连的节点。这些节点可以已经构造好，也可以在构造链的过程中构造（或者这些过程可以混合进行）。

5.5.1 Creating a Simple Chain
创建一个简单的链

Ilka starts with creating a chain from scratch. For this, she starts a chain using the `start chain` option in a scope. Then, inside the scope, she uses the `on chain` option on nodes to add them to the chain.

Ilka 首先从零开始创建一条链。为此，她在作用域内使用`start chain` 选项开始一条链。然后，在作用域内，她使用`on chain` 选项将节点添加到链中。


```
\begin{tikzpicture}[start chain,node distance=5mm]
  \node [on chain,nonterminal]  {unsigned integer};
  \node [on chain,terminal]     {.};
  \node [on chain,terminal]     {digit};
  \node [on chain,terminal]     {E};
  \node [on chain,nonterminal]  {unsigned integer};
\end{tikzpicture}
```

(Ilka will add the plus and minus nodes later.)

（Ilka 稍后将添加加号和减号节点。）

As can be seen, the nodes of a chain are placed in a row. This can be changed, for instance by saying `start chain=going below` we get a chain where each node is below the previous one.

可以看到，链的节点被放置在一行中。这可以改变，例如通过使用`start chain=going below`，我们可以得到一个每个节点都在前一个节点下方的链。

The next step is to *join* the nodes of the chain. For this, we add the `join` option to each node. This joins the node with the previous node (for the first node nothing happens).

下一步是*连接*链的节点。为此，我们在每个节点上添加`join` 选项。这将节点连接到前一个节点（对于第一个节点，不会发生任何操作）。


```
\begin{tikzpicture}[start chain,node distance=5mm]
  \node [on chain,join,nonterminal]  {unsigned integer};
  \node [on chain,join,terminal]     {.};
  \node [on chain,join,terminal]     {digit};
  \node [on chain,join,terminal]     {E};
  \node [on chain,join,nonterminal]  {unsigned integer};
\end{tikzpicture}
```

In order to get a arrow tip, we redefine the `every join` style. Also, we move the `join` and `on chain` options to the `every node` style so that we do not have to repeat them so often.

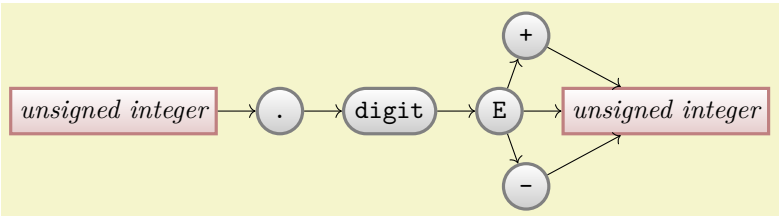
为了获得箭头标记，我们重新定义`every join` 样式。此外，我们将`join` 和`on chain` 选项移到`every node` 样式中，这样我们就不必经常重复它们。


```
\begin{tikzpicture}[start chain,node distance=5mm, every node/.style={on chain,join}, every join/.style={->}]
  \node [nonterminal]  {unsigned integer};
  \node [terminal]     {.};
  \node [terminal]     {digit};
  \node [terminal]     {E};
  \node [nonterminal]  {unsigned integer};
\end{tikzpicture}
```

5.5.2 Branching and Joining a Chain
分支和连接链

It is now time to add the plus and minus signs. They obviously *branch off* the main chain. For this reason, we start a branch for them using the `start branch` option.

现在是时候添加加号和减号符号了。它们显然是从主链上分支*出*来的。因此，我们使用`start branch` 选项为它们启动一个分支。




```
\begin{tikzpicture}[start chain,node distance=5mm, every node/.style={on chain,join}, every join/.style={->}]
  \node [nonterminal]  {unsigned integer};
  \node [terminal]     {.};
  \node [terminal]     {digit};
  \node [terminal]     {E};
  \begin{scope}[start branch=plus]
    \node (plus)  [terminal,on chain=going above right] {+};
  \end{scope}
  \begin{scope}[start branch=minus]
    \node (minus) [terminal,on chain=going below right] {-};
  \end{scope}
  \node [nonterminal,join=with plus,join=with minus]  {unsigned integer};
\end{tikzpicture}
```

Let us see, what is going on here. First, the `start branch` begins a branch, starting with the node last created on the current chain, which is the `E` node in our case. This is implicitly also the first node

on this branch. A branch is nothing different from a chain, which is why the plus node is put on this branch using the `on chain` option. However, this time we specify the placement of the node explicitly using `going` ⟨*direction*⟩. This causes the plus sign to be placed above and right of the `E` node. It is automatically joined to its predecessor on the branch by the implicit `join` option.

让我们看看这里发生了什么。首先，`start branch` 开始一个分支，从当前链上最后创建的节点开始，也就是我们的例子中的`E` 节点。这也是这个分支上的第一个节点。分支与链没有任何区别，这就是为什么使用`on chain` 选项将加号节点放在这个分支上。然而，这次我们使用`going` ⟨*direction*⟩ 显式地指定节点的位置。这使得加号标记被放置在`E` 节点的上方和右方。它通过隐式的`join` 选项自动与其在分支上的前一个节点连接在一起。

When the first branch ends, only the plus node has been added and the current chain is the original chain once more and we are back to the `E` node. Now we start a new branch for the minus node. After this branch, the current chain ends at `E` node once more.

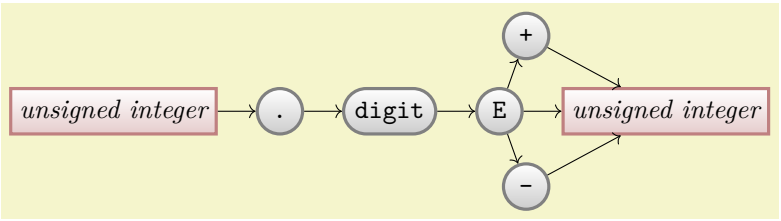
当第一个分支结束时，只添加了加号节点，当前链再次变为原始链，我们回到`E` 节点。现在我们为减号节点开始一个新的分支。在这个分支之后，当前链再次结束于`E` 节点。

Finally, the rightmost unsigned integer is added to the (main) chain, which is why it is joined correctly with the `E` node. The two additional `join` options get a special `with` parameter. This allows you to join a node with a node other than the predecessor on the chain. The `with` should be followed by the name of a node.

最后，将最右边的无符号整数添加到（主）链中，这就是为什么它与`E` 节点正确连接的原因。两个额外的`join` 选项使用了特殊的`with` 参数。这允许您将一个节点与链上的前任节点之外的节点连接起来。`with` 后面应该跟着一个节点的名称。

Since Ilka will need scopes more often in the following, she includes the `scopes` library. This allows her to replace `\begin{scope}` simply by an opening brace and `\end{scope}` by the corresponding closing brace. Also, in the following example we reference the nodes `plus` and `minus` using their automatic name: The ith node on a chain is called `chain-`⟨i⟩. For a branch ⟨*branch*⟩, the ith node is called `chain/`⟨*branch*⟩`-`⟨i⟩. The ⟨i⟩ can be replaced by `begin` and `end` to reference the first and (currently) last node on the chain.

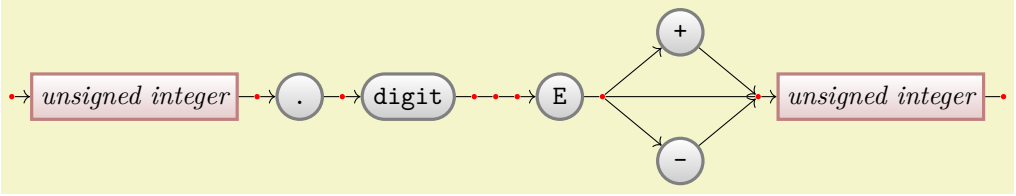
由于 Ilka 在接下来的工作中经常需要作用域，她包含了`scopes` 库。这使得她可以用一个左花括号替换`\begin{scope}`，用相应的右花括号替换`\end{scope}`。此外，在下面的示例中，我们使用它们的自动名称引用节点`plus` 和`minus`：链上的第 i 个节点称为`chain-`⟨i⟩。对于一个分支⟨*branch*⟩，第 i 个节点称为`chain/`⟨*branch*⟩`-`⟨i⟩。可以将⟨i⟩ 替换为`begin` 和`end`，以引用链上的第一个节点和（当前的）最后一个节点。




```
\begin{tikzpicture}[start chain,node distance=5mm, every on chain/.style={join}, every join/.style={->}]
  \node [on chain,nonterminal]  {unsigned integer};
  \node [on chain,terminal]     {.};
  \node [on chain,terminal]     {digit};
  \node [on chain,terminal]     {E};
  { [start branch=plus]
    \node (plus)  [terminal,on chain=going above right] {+};
  }
  { [start branch=minus]
    \node (minus) [terminal,on chain=going below right] {-};
  }
  \node [nonterminal,on chain,join=with chain/plus-end,join=with chain/minus-end]  {unsigned integer};
\end{tikzpicture}
```

The next step is to add intermediate coordinate nodes in the same manner as Ilka did for the matrix. For them, we change the `join` style slightly, namely for these nodes we do not want an arrow tip. This can be achieved either by (locally) changing the `every join` style or, which is what is done in the below example, by giving the desired style using `join=by ...`, where `...` is the style to be used for the join.

接下来的步骤是以与 Ilka 为矩阵做的方式相同的方式添加中间坐标节点。对于它们，我们稍微改变`join` 样式，即对于这些节点，我们不希望有箭头标记。这可以通过（局部）更改`everyjoin` 样式实现，或者如下示例所示，通过使用`join=by ...` 给出所需的样式，其中`...` 是要用于连结的样式。




```
\begin{tikzpicture}[start chain,node distance=5mm and 2mm,
                    every node/.style={on chain},
                    nonterminal/.append style={join=by ->},
                    terminal/.append style={join=by ->},
                    point/.style={join=by -,circle,fill=red,minimum size=2pt,inner sep=0pt}]
  \node [point] {};  \node [nonterminal] {unsigned integer};
  \node [point] {};  \node [terminal]    {.};
  \node [point] {};  \node [terminal]    {digit};
  \node [point] {};  \node [point]       {};
  \node [point] {};  \node [terminal]    {E};
  \node [point] {};
  { [node distance=5mm and 1cm] % local change in horizontal distance
    { [start branch=plus]
      \node (plus)  [terminal,on chain=going above right] {+};
    }
    { [start branch=minus]
      \node (minus) [terminal,on chain=going below right] {-};
    }
    \node [point,below right=of plus,join=with chain/plus-end by ->,join=with chain/minus-end by ->] {};
  }
  \node [nonterminal] {unsigned integer};
  \node [point]       {};
\end{tikzpicture}
```

5.5.3 Chaining Together Already Positioned Nodes
连接已经定位的节点的链

The final step is to add the missing arrows. We can also use branches for them (even though we do not have to, but it is good practice and they exhibit the structure of the diagram in the code).

最后一步是添加缺失的箭头。我们也可以为它们使用分支（即使我们不必这样做，但这是一个好的实践，并且它们展示了代码中的结构）。

Let us start with the repeat loop around the `digit`. This can be thought of as a branch that starts at the point after the digit and that ends at the point before the digit. However, we have already constructed the point before the digit! In such cases, it is possible to "chain in" an already positioned node, using the `\chainin` command. This command must be followed by a coordinate that contains a node name and optionally some options. The effect is that the named node is made part of the current chain.

让我们从围绕`digit` 的重复循环开始。可以将其视为从数字之后的位置开始的一个分支，到数字之前的位置结束。然而，我们已经构造了数字之前的位置！在这种情况下，可以使用`\chainin` 命令将已经定位的节点“链入”当前链中。该命令必须后跟一个包含节点名称和可选选项的坐标。效果是将命名节点作为当前链的一部分。


```
\begin{tikzpicture}[start chain] % plus some styles that are not shown
  \node                 [point] {};
  \node (before digit)  [point] {};
  \node                 [terminal]    {digit};
  \node                 [point] {};
  { [start branch=digit loop]
    \chainin (before digit) [join=by {->,skip loop=-5mm}];
  }
  \node                 [point] {};
\end{tikzpicture}
```

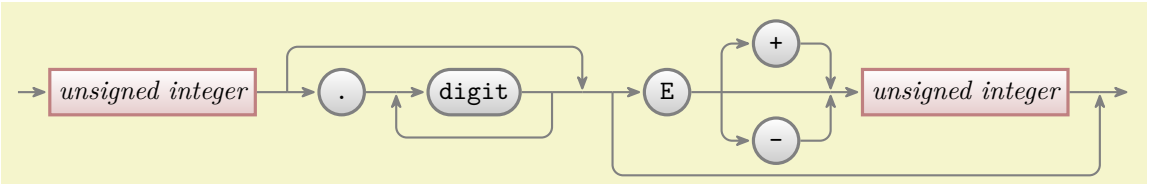
5.5.4 Combined Use of Matrices and Chains
矩阵和链的组合使用

Ilka's final idea is to combine matrices and chains in the following manner: She will use a matrix to position the nodes. However, to show the logical "flow structure" inside the diagram, she will create chains and branches that show what is going on.

Ilka 的最后一个想法是以以下方式结合矩阵和链：她将使用矩阵来定位节点。然而，为了显示图表中的逻辑“流结构”，她将创建显示正在进行的链和分支。

Ilka starts with the matrix we had earlier, only with slightly adapted styles. Then she writes down the main chain and its branches:

Ilka 首先从我们之前使用的矩阵开始，只是稍微调整了样式。然后她写下了主链和它的分支：




```
\usetikzlibrary {arrows.meta}
\begin{tikzpicture}[point/.style={coordinate},>={Stealth[round]},thick,draw=black!50,
                    tip/.style={->,shorten >=1pt},every join/.style={rounded corners},
                    hv path/.style={to path={-| (\tikztotarget)}},
                    vh path/.style={to path={|- (\tikztotarget)}}]
  \matrix[column sep=4mm] {
    % First row:
    & & & & & & & & & & \node (plus) [terminal] {+};\\
    % Second row:
    \node (p1) [point]  {}; &    \node (ui1)   [nonterminal] {unsigned integer}; &
    \node (p2) [point]  {}; &    \node (dot)   [terminal]    {.};                &
    \node (p3) [point]  {}; &    \node (digit) [terminal]    {digit};            &
    \node (p4) [point]  {}; &    \node (p5)    [point]  {};                       &
    \node (p6) [point]  {}; &    \node (e)     [terminal]    {E};                &
    \node (p7) [point]  {}; &                                                     &
    \node (p8) [point]  {}; &    \node (ui2)   [nonterminal] {unsigned integer}; &
    \node (p9) [point]  {}; &    \node (p10)   [point]       {};\\
    % Third row:
    & & & & & & & & & & \node (minus)[terminal] {-};\\
  };

  { [start chain]
    \chainin (p1);
    \chainin (ui1)   [join=by tip];
    \chainin (p2)    [join];
    \chainin (dot)   [join=by tip];
    \chainin (p3)    [join];
    \chainin (digit) [join=by tip];
    \chainin (p4)    [join];
    { [start branch=digit loop]
      \chainin (p3) [join=by {skip loop=-6mm,tip}];
    }
    \chainin (p5)    [join,join=with p2 by {skip loop=6mm,tip}];
    \chainin (p6)    [join];
    \chainin (e)     [join=by tip];
    \chainin (p7)    [join];
    { [start branch=plus]
      \chainin (plus)  [join=by {vh path,tip}];
      \chainin (p8)    [join=by {hv path,tip}];
    }
    { [start branch=minus]
      \chainin (minus) [join=by {vh path,tip}];
      \chainin (p8)    [join=by {hv path,tip}];
    }
    \chainin (p8)    [join];
    \chainin (ui2)   [join=by tip];
    \chainin (p9)    [join,join=with p6 by {skip loop=-11mm,tip}];
    \chainin (p10)   [join=by tip];
  }
\end{tikzpicture}
```

```
\setcounter{section}{5}
\setcounter{subsection}{5}
\setcounter{subsubsection}{4}
```