Formatted Printing Using `printf`

8

8.1 General Rules for `printf`

This chapter discusses the syntax of `printf`, which we have already seen before. The general usage of `printf` is

```
System.out.printf( "u0w1u1w2 ...wkuk", d1, ..., dk );
```

where `w1`, ..., `wk` are placeholders that each specify the formatting of a single data, `d1`, ..., `dk` are the data to be formatted with the placeholders `w1`, ..., `wk` in this order, and `u0`, ..., `uk` are character sequences without formatting placeholders. For each `i`, the data `di` must be of the type specified in `wi`; if not, it must be converted to a data of that type. Any data can be interpretable as a `String` data, and any whole number data can be interpretable as a floating point data.

If a data supplied to a placeholder cannot be converted to the required type (for example, a `String` is supplied to a placeholder for a whole number), a run-time error occurs.

```
1 %d|Exception in thread "main" java.util.IllegalFormatConversionException:
      d != java.lang.String
2 at java.util.Formatter$FormatSpecifier.failConversion(Formatter.java:4302)
3 at java.util.Formatter$FormatSpecifier.printInteger(Formatter.java:2793)
4 at java.util.Formatter$FormatSpecifier.print(Formatter.java:2747)
5 at java.util.Formatter.format(Formatter.java:2520)
6 at java.io.PrintStream.format(PrintStream.java:970)
7 at java.io.PrintStream.printf(PrintStream.java:871)
8 at PrintFString.printStringWithFormat(PrintFString.java:28)
9 at PrintFString.main(PrintFString.java:22)
```

The same `IllegalFormatConversionException` error also occurs when the number of data supplied and the number of placeholders do not agree, as well as when a placeholder is incorrectly written.

Since each placeholder starts with a percent character, to include the percent character in a part that is not a placeholder, the `%%` is used via escaping. In `printf`, the newline character can be specified as `%n`.

M. Ogihara, *Fundamentals of Java Programming*,
https://doi.org/10.1007/978-3-319-89491-1_8

8.2 Formatted Printing of String Data

Next, we learn how to format String data using printf.

The general expression for String formatting is %Xs, where X consists of three options that can be selected independently of each other. Here are the three options, which must appear in this order:

1. The flush left positioning can be specified using a single minus sign -.
   If this option is not present, the data is automatically printed in flush right.
2. The minimum number of character spaces allocated for the data can be specified with a positive integer without leading 0's.
   If this option is not present, the default character space allocated is the exact number of character spaces required to print the data.
   If this option is present and the length of the data to be printed is smaller than the specified number of character spaces, the space character ' ' is added so as to make the length exactly the specified number. The adding of the space character occurs at the beginning of the data if the positioning is flush right (default), and at the end if flush left.
   However, if this option is present and the length of the String data to be printed is greater than the specified minimum character space, the entire data is printed with no padding.
3. It is possible to print only a prefix of the String data. The length of the prefix to be printed can be specified with a period immediately before the length.
   If this option is not present, the entire character sequence of the String data is printed.
   However, if the specified prefix length is greater than the length of the String data to be printed, the prefix length is automatically reduced to the exact length of the String data.

For example, "%-10s" specifies that the String data must be printed in flush left using at least ten character spaces, and "%10.5s" specifies that only the first five characters must be printed in flush right using at least ten character spaces. Thus, in

```
System.out.printf( "Message=%-10.5s-Mike", "Table tennis is fun!" );
```

the 10-space formatting of the String "Table tennis is fun!" will appear after "Message=", and "-Mike" after that. This results in:

```
Message=Table     --Mike
```

The next program shows different effects in printf for String. The program presents the results in a two-column table, with the left column showing the format used and the right column showing the formatting generated. To print a row of the table, the program uses a method named printStringWithFormat. The method takes two String parameters, fmt and message, where the first formal parameter is used to format the second parameter. The formatting of the second parameter using the first parameter can be accomplished by

```
System.out.printf( fmt, message );
```

This will be the second column of the row.

For the first column, we want to print the first parameter fmt. Assuming that the length of fmt is no more than 10, we allocate exactly ten character spaces in flush right and print the vertical line character | after that as follows:

```
"System.out.printf( "%10s|", fmt );"
```

We also want to show if padding appears in the formatting of message using fmt as the formatting. Since the white space is invisible, we add a colon at the end to indicate the end of the line for this purpose.

Altogether, the method has the following code:

```
19   public static void printStringWithFormat(
20       String fmt, String message )
21   {
22     System.out.printf( "%10s|", fmt );
23     System.out.printf( fmt, message );
24     System.out.println( ":" );
25   }
```

The method main prints the String variable t using various formats. The method first stores the literal "Welcome to the Club!" to t (Line 5). The method then prints the header lines (Lines 7–9). The method then prints t in seven different formats by calling the printStringWithFormat method (Lines 10–16). At the end, in Line 17, the method prints a line identical to the one that it printed in Line 9.

```
1  public class PrintFString
2  {
3    public static void main( String[] args )
4    {
5      String t = "Welcome to the Club!";
6      System.out.println( "theString=\"Welcome to the Club!\"" );
7      System.out.println( "Format    |Position" );
8      System.out.println( "String    |01234567890123456789012345" );
9      System.out.println( "----------+-------------------------" );
10     printStringWithFormat( "%s:", t );
11     printStringWithFormat( "%25s:", t );
12     printStringWithFormat( "%-25s:", t );
13     printStringWithFormat( "%10s:", t );
14     printStringWithFormat( "%-10s:", t );
15     printStringWithFormat( "%-25.14s:", t );
16     printStringWithFormat( "%25.14s:", t );
17     System.out.println( "----------+-------------------------" );
18   }
```

Listing 8.1 A code that shows various output formatting for String data. The main method

Executing the code produces the following:

```
theString="Welcome to the Club!"
Format    |Position
String    |012345678901234567890123456789
----------+------------------------
       %s:|Welcome to the Club!:
     %25s:|     Welcome to the Club!:
    %-25s:|Welcome to the Club!     :
     %10s:|Welcome to the Club!:
    %-10s:|Welcome to the Club!:
 %-25.14s:|Welcome to the           :
  %25.14s:|           Welcome to the:
----------+------------------------
```

`%c` is used to format a `char` data. Only two options are available: the number of character spaces allocated and the flush left positioning.

8.3 Formatted Printing of Integers

To use `printf` for printing a whole number, we use a format `String` of the form `%Xd`, where the part `X` consists of four options that can be selected independently of each other. Here are the four options (the fourth option must appear the last):

1. The forced plus sign for a strictly positive value can be specified with a single plus sign `+`.
   If this option is not present, a strictly positive value appears without the plus sign.
2. A single comma `,` specified the forced currency punctuation. If the option is present, the punctuation appears with every three digits if the environment information the JVM has access to state that the country where the computer is running is the United States of America.
   If this option is not present, there will be no punctuation.
3. Either the flush left or leading 0s can be specified with a single minus sign `-` or a single `0` respectively.
   At most one of the two can be specified.
   If neither options are present, the data is printed in flush right.
4. The minimum number of character spaces allocated for the data can be specified with a positive integer without leading 0s.
   If this option is not present, the character space allocated is the exact number of character spaces required to print the data, meaning that the "leading-0" option will be ignored.

Note that the comma, the plus sign, and the minus sign are counted towards the number of characters used.

Here is a code that demonstrates the different formatting of a whole number. The method responsible for printing a row of the table is very much similar to the previous one, except that the second parameter is an `int` data named `n`.

```
public class PrintFDecimal
{
  public static void main( String[] args )
  {
    int num = 456789;
    System.out.println( "number=" + num );
    System.out.println( "Format    |Position" );
    System.out.println( "String    |01234567890123456789" );
    System.out.println( "----------+--------------------" );
    printDecimalWithFormat( "%d:", num );
    printDecimalWithFormat( "%+d:", num );
    printDecimalWithFormat( "%,d:", num );
    printDecimalWithFormat( "%+,d:", num );
    printDecimalWithFormat( "%20d:", num );
    printDecimalWithFormat( "%+20d:", num );
    printDecimalWithFormat( "%,20d:", num );
    printDecimalWithFormat( "%+,20d:", num );
    printDecimalWithFormat( "%-20d:", num );
    printDecimalWithFormat( "%+-20d:", num );
    printDecimalWithFormat( "%,-20d:", num );
    printDecimalWithFormat( "%+-,20d:", num );
    printDecimalWithFormat( "%020d:", num );
    printDecimalWithFormat( "%+020d:", num );
    printDecimalWithFormat( "%,020d:", num );
    printDecimalWithFormat( "%+,020d:", num );
    System.out.println( "----------+--------------------" );
  }
  public static void printDecimalWithFormat( String fmt, int n )
  {
    System.out.printf( "%-10s|", fmt );
    System.out.printf( fmt, n );
    System.out.println();
  }
}
```

Listing 8.2 A code that shows various output formatting for whole number data

The code produces the following output:

```
          01234567890123456789
---------+--------------------
%d:      |456789:
%+d:     |+456789:
%,d:     |456,789:
%+,d:    |+456,789:
%20d:    |              456789:
%+20d:   |             +456789:
%,20d:   |             456,789:
%+,20d:  |            +456,789:
%-20d:   |456789              :
%+-20d:  |+456789             :
%,-20d:  |456,789             :
%+-,20d: |+456,789            :
%020d:   |00000000000000456789:
%+020d:  |+0000000000000456789:
%,020d:  |0000000000000456,789:
%+,020d: |+000000000000456,789:
---------+--------------------
```

8.4 Formatted Printing of Floating Point Numbers

To use printf for printing a floating point number, we use a format String of the form %Xf, where the part X consists of five options that can be selected independently of each other. The first four options are the same as the options for %d. The last option specifies the exact number of digits printed after the decimal point. It is specified with a single period followed by a strictly positive integer, which is the number of digits.

Since there are so many options to apply to a floating number, we demonstrate the effect using two programs.

Here is the first program.

```
1  public class PrintFFloat
2  {
3    public static void main( String[] args )
4    {
5      double num = 1974.9215;
6      System.out.println( "number=1974.9215" );
7      System.out.println( "Format    |Position" );
8      System.out.println( "String    |01234567890123456789" );
9      System.out.println( "----------+--------------------" );
10     printFloatWithFormat( "%f:", num );
11     printFloatWithFormat( "%+f:", num );
12     printFloatWithFormat( "%,f:", num );
13     printFloatWithFormat( "%+,f:", num );
14     printFloatWithFormat( "%20f:", num );
15     printFloatWithFormat( "%+20f:", num );
16     printFloatWithFormat( "%,20f:", num );
17     printFloatWithFormat( "%+,20f:", num );
18     printFloatWithFormat( "%-20f:", num );
19     printFloatWithFormat( "%+-20f:", num );
20     printFloatWithFormat( "%,-20f:", num );
21     printFloatWithFormat( "%+-,20f:", num );
22     printFloatWithFormat( "%020f:", num );
23     printFloatWithFormat( "%+020f:", num );
24     printFloatWithFormat( "%,020f:", num );
25     printFloatWithFormat( "%+,020f:", num );
26     System.out.println( "----------+--------------------" );
27   }
```

Listing 8.3 A code that shows various output formatting for floating point data (part 1). The main method

The auxiliary method is similar to the one from the previous programs.

```
28   public static void printFloatWithFormat( String fmt, double v )
29   {
30     System.out.printf( "%-10s|", fmt );
31     System.out.printf( fmt, v );
32     System.out.println();
33   }
34 }
```

Listing 8.4 A code that shows various output formatting for floating point data (part 2). A method used for printing the format String and the formatted data together

Here is the second program.

```
1  public class PrintFFloat2
2  {
3    public static void main( String[] args )
4    {
5      double num = 1974.9215;
6      System.out.println( "number=1974.9215" );
7      System.out.println( "Format    |Position" );
8      System.out.println( "String    |0123456789012345678 9" );
9      System.out.println( "----------+--------------------" );
10     printFloatWithFormat( "%.3f:", num );
11     printFloatWithFormat( "%+.3f:", num );
12     printFloatWithFormat( "%,.3f:", num );
13     printFloatWithFormat( "%+,.3f:", num );
14     printFloatWithFormat( "%20.3f:", num );
15     printFloatWithFormat( "%+20.3f:", num );
16     printFloatWithFormat( "%,20.3f:", num );
17     printFloatWithFormat( "%+,20.3f:", num );
18     printFloatWithFormat( "%-20.3f:", num );
19     printFloatWithFormat( "%+-20.3f:", num );
20     printFloatWithFormat( "%,-20.3f:", num );
21     printFloatWithFormat( "%+-,20.3f:", num );
22     printFloatWithFormat( "%020.3f:", num );
23     printFloatWithFormat( "%+020.3f:", num );
24     printFloatWithFormat( "%,020.3f:", num );
25     printFloatWithFormat( "%+,020.3f:", num );
26     System.out.println( "----------+--------------------" );
27   }
```

Listing 8.5 A code that shows various output formatting for floating point data (part 3). The `main` method of the second program

The auxiliary methods between the two programs are identical.

```
28   public static void printFloatWithFormat( String fmt, double v )
29   {
30     System.out.printf( "%-10s|", fmt );
31     System.out.printf( fmt, v );
32     System.out.println();
33   }
34 }
```

Listing 8.6 A code that shows various output formatting for floating point data (part 4). A method used for printing the format `String` and the formatted data together

Executing the two programs, `PrintFFloat` and `PrintFFloat2`, results in the following output.

The following is from `PrintFFloat`:

```
1            0123456789012345678 9
2  ----------+--------------------
3  %f:       |1974.921500:
4  %+f:      |+1974.921500:
5  %,f:      |1,974.921500:
6  %+,f:     |+1,974.921500:
```

```
%20f:     |          1974.921500:
%+20f:    |         +1974.921500:
%,20f:    |         1,974.921500:
%+,20f:   |        +1,974.921500:
%-20f:    |1974.921500          :
%+-20f:   |+1974.921500         :
%,-20f:   |1,974.921500         :
%+-,20f:  |+1,974.921500        :
%020f:    |0000000001974.921500:
%+020f:   |+000000001974.921500:
%,020f:   |000000001,974.921500:
%+,020f:  |+00000001,974.921500:
```

The following is from `PrintFFloat2`.

```
            01234567890123456789
----------+--------------------
%.3f:      |1974.922:
%+.3f:     |+1974.922:
%,.3f:     |1,974.922:
%+,.3f:    |+1,974.922:
%20.3f:    |             1974.922:
%+20.3f:   |            +1974.922:
%,20.3f:   |            1,974.922:
%+,20.3f:  |           +1,974.922:
%-20.3f:   |1974.922             :
%+-20.3f:  |+1974.922            :
%,-20.3f:  |1,974.922            :
%+-,20.3f:|+1,974.922           :
%020.3f:   |00000000000001974.922:
%+020.3f:  |+00000000001974.922:
%,020.3f:  |000000000001,974.922:
%+,020.3f:|+00000000001,974.922:
----------+--------------------
```

8.5 Printing the Fibonacci Sequence (Reprise)

In Chap. 7, we learned how to compute the Fibonacci numbers using two variables to record two immediate predecessors in the sequence. Now, consider printing the square and cubic roots of the numbers obtained as a table, where we generate the sequence up to F_{80}:

```
Enter n: 30
    i                F_i                sqrt                cbrt
-------------------------------------------------------------------
    2                  2             1.41421             1.25992
    3                  3             1.73205             1.44225
    4                  5             2.23607             1.70998
    5                  8             2.82843             2.00000
    6                 13             3.60555             2.35133
    7                 21             4.58258             2.75892
    8                 34             5.83095             3.23961
    9                 55             7.41620             3.80295
   10                 89             9.43398             4.46475
   11                144            12.00000             5.24148
   12                233            15.26434             6.15345
   13                377            19.41649             7.22405
   14                610            24.69818             8.48093
```

17	15	987	31.41656	9.95648
18	16	1597	39.96248	11.68876
19	17	2584	50.83306	13.72242
20	18	4181	64.66065	16.10992
21	19	6765	82.24962	18.91280
22	20	10946	104.62313	22.20335
23	21	17711	133.08268	26.06640
24	22	28657	169.28379	30.60156
25	23	46368	215.33230	35.92577
26	24	75025	273.90692	42.17632
27	25	121393	348.41498	49.51437
28	26	196418	443.19070	58.12912
29	27	317811	563.74728	68.24272
30	28	514229	717.09762	80.11593
31	29	832040	912.16227	94.05489
32	30	1346269	1160.28833	110.41904
33	31	2178309	1475.90955	129.63029
34	32	3524578	1877.38595	152.18402
35	33	5702887	2388.07182	178.66175
36	34	9227465	3037.67427	209.74622
37	35	14930352	3863.98137	246.23891
38	36	24157817	4915.06022	289.08079
39	37	39088169	6252.05318	339.37651
40	38	63245986	7952.73450	398.42293
41	39	102334155	10116.03455	467.74254
42	40	165580141	12867.79472	549.12272
43	41	267914296	16368.08773	644.66184
44	42	433494437	20820.52922	756.82333
45	43	701408733	26484.12228	888.49923
46	44	1134903170	33688.32394	1043.08477
47	45	1836311903	42852.21001	1224.56587
48	46	2971215073	54508.85316	1437.62195
49	47	4807526976	69336.33229	1687.74661
50	48	7778742049	88197.17710	1981.38919
51	49	12586269025	112188.54231	2326.12119
52	50	20365011074	142706.03027	2730.83137
53	51	32951280099	181524.87460	3205.95506
54	52	53316291173	230903.20737	3763.74314
55	53	86267571272	293713.41691	4418.57798
56	54	139583862445	373609.23763	5187.34425
57	55	225851433717	475238.29151	6089.86433
58	56	365435296162	604512.44500	7149.40935
59	57	591286729879	768951.70842	8393.29931
60	58	956722026041	978121.68264	9853.60747
61	59	1548008755920	1244189.99993	11567.98733
62	60	2504730781961	1582634.12764	13580.64357
63	61	4052739537881	2013141.70835	15943.47179
64	62	6557470319842	2560755.81027	18717.39668
65	63	10610209857723	3257331.70827	21973.94288
66	64	17167680177565	4143389.93791	25797.07926
67	65	27777890035288	5270473.41662	30285.38401
68	66	44945570212853	6704145.74818	35554.58645
69	67	72723460248141	8527805.12489	41740.55105
70	68	117669030460994	10847535.68609	49002.78069
71	69	190392490709135	13798278.54151	57528.52933
72	70	308061521170129	17551681.43427	67537.63032
73	71	498454011879264	22326083.66640	79288.16470
74	72	806515533049393	28399217.12036	93083.11578
75	73	1304969544928657	36124362.20792	109278.18137
76	74	2111485077978050	45950898.55463	128290.94539

```
77    75    3416454622906707      58450445.87432        150611.64511
78    76    5527939700884757      74350115.67499        176815.81168
79    77    8944394323791464      94574808.08223        207579.11010
80    78   14472334024676221     120301014.22962        243694.76088
81    79   23416728348467685     153025253.95655        286093.99305
82    80   37889062373143906     194651129.90462        335870.05549
```

We can generate this output using two coordinated formats, one for the first line of the header and the other for the data rows of the table. Both types have four entries, and the numbers of spaces allocated for them are 5, 20, 20, and 20 in this order. For the header line, all the entries are `String` data, so we use the place holders `"%5s"`, `"%20s"`, `"%20s"`, and `"%20s"`. For each data line, the first two entries are integers and the last entries are floating point numbers with five digits after the decimal point, so we use the placeholders `"%5d"`, `"%20d"`, `"%20.5f"`, and `"%20.5f"`. Each time a new element of the Fibonacci sequence is calculated, the element is stored in the variable `f`. The new element is calculated as the sum of the two variables, `fp` and `fpp`. `fp` and `fpp` hold the values of the previous Fibonacci number and the number right before the previous Fibonacci number. We calculate the two roots using the `Math` functions `sqrt` and `cbrt`.

Here is the code for this task, `FibonacciClean`.

```
1  import java.util.*;
2  public class FibonacciClean
3  {
4    public static void main( String[] args )
5    {
6      long f, fp = 1, fpp = 1;
7      double sqroot, cbroot;
8      Scanner keyboard = new Scanner( System.in );
9      System.out.print( "Enter n: " );
10     int n = keyboard.nextInt();
11     System.out.printf( "%5s%20s%20s%20s\n",
12         "i", "F_i", "sqrt", "cbrt" );
```

Listing 8.7 A program that prints the Fibonacci numbers and their square and cubic roots neatly (part 1)

```
13     for ( int i = 1; i <= 65; i ++ )
14     {
15       System.out.print( "-" );
16     }
17     System.out.println();
18     for ( int i = 2; i <= n; i ++ )
19     {
20       f = fp + fpp;
21       sqroot = Math.sqrt( f );
22       cbroot = Math.cbrt( f );
23       System.out.printf( "%5d%20d%20.5f%20.5f\n",
24           i, f, sqroot, cbroot );
25       fpp = fp;
26       fp = f;
27     }
28   }
29 }
```

Listing 8.8 A program that prints the Fibonacci numbers and their square and cubic roots neatly (part 2)

Summary

- The formatting options that are available for printing a `String` data with a `printf` statement include the choice between the flush left and the flush right positioning, the number of character spaces allocated, and the length of the prefix to be printed.
- The formatting options that are available for printing a whole number with a `printf` statement include the choice among flush left, flush right, and flush right with leading 0s, the forced plus sign, the forced punctuation, and the number of character spaces allocated.
- The formatting options that are available for printing a floating point number with a `printf` statement include the choice among flush left, flush right, and flush right with leading 0s, the forced plus sign, the forced punctuation, the number of character spaces allocated, and the number of digits after the decimal point.
- The formatting options that are available for printing a `char` data with a `printf` statement are the choice between flush left and flush right, and the number of character spaces allocated.
- To include `%` in the format, use `%%`.

Exercises

1. **Printf output** Let `myVar` be a `double` variable with the value of `10.345678`. State what output the following `printf` statements generate.
   (a) `System.out.printf( "value=%4.1f", myVar );`
   (b) `System.out.printf( "value=%5.2f", myVar );`
   (c) `System.out.printf( "value=%.3f", myVar );`
2. **(Approximately) simulating `printf` for a real number** Write a method named `dot2f` that acts as if it were `printf` with `"%.2f"` as the format. The method does not return a value. The method has one formal parameter `input`, which is a `double`. The method must print the value of `input`.
   The strategy for achieving the goal is as follows:
   - Execute `Math.round` of `input * 100.0` and convert it to an integer by casting `(long)`, and then store its value into a `long` variable `input100`.
   - Divide `input100` into two `long` values `a` and `b`, which are respectively the quotient of `input100` by 100 and the remainder of `input100` by 100.
   - Print `a`, the period, `b / 10`, and `b % 10` in this order.
3. **(Approximately) simulating `printf` for a real number, continued** Using the idea from the previous question, write a `void` method `dot4f` that takes a `double` variable `input` as its parameter and prints the value of `input` in `"%.4f"` format.
4. **A simple for-loop, leading 0s** Using a for-loop whose body consists of just one `printf` statement, write a code that produces the following output:

```
001:
003:
005:
007:
009:
```

5. **Two values per line for-loop** Using a for-loop whose body consists of just one `printf` statement, write a code that produces the following output:

```
001,002
003,004
005,006
007,008
009,010
```

6. **A simple for-loop** Using a for-loop whose body consists of just one `printf` statement, write a code that produces the following output:

```
100!
 88!
 76!
 64!
 52!
 40!
 28!
 16!
```

7. **Powering** Write a method named `powerList` that takes two parameters, a `double d` and an `int k`, and prints the i-th power of `d` for $i = 1, \ldots, k$. The output for each value of i appears in a single line along with the value of i, with r digits for i and four digits after the decimal point for the powers. The following is an example of the output produced by such a method:

```
   1 3.1000
   2 9.6100
   3 29.7910
   4 286.2915
```

Programming Projects

8. **Mimicking the simple formatting of a floating point number without rounding** Write a program named `PseudoPrintfF` that receives a `double` number `x` and a nonnegative `int` number `d` from the user, and then prints the absolute value of `x` with exactly `d` digits after the decimal point. If the user enters a negative integer for `d`, the program terminates with an error of `IllegalArgumentException`. If `d == 0`, the program prints only the integer part of the absolute value of `x` without a period. If `d > 0`, the program prints the integer part of the absolute value, prints a period, and prints `d` digits as follows: Let `y` be the absolute value of `x` minus the integer part of the absolute value. `y` is greater than or equal to 0 and strictly less than 1. The program repeats the following `d` times:
   - Multiply `y` by 10.
   - Print the integer part of `y`.
   - Subtract the integer part of `y` from `y`.
9. **Printing the calendar of a month** Write a program named `CalOfAMonth` that produces the calendar of a month. The program receives the number of days in the month and the position at which the first day of the month starts (0 for Sunday, 1 for Monday, etc.), and then prints the calendar in the example below, where the inputs are 31 and 2.

```
  Sun    Mon    Tue    Wed    Thu    Fri    Sat
+------+------+------+------+------+------+------+
|      |      |   1  |   2  |   3  |   4  |   5  |
|   6  |   7  |   8  |   9  |  10  |  11  |  12  |
|  13  |  14  |  15  |  16  |  17  |  18  |  19  |
|  20  |  21  |  22  |  23  |  24  |  25  |  26  |
|  27  |  28  |  29  |  30  |  31  |      |      |
+------+------+------+------+------+------+------+
```

The number of days must be 28, 29, 30, or 31. Let `numberOfDays` and `startDay` be the number of days and the position of the first day of the month. A key calculation to make is (excluding the fixed lines) how many rows are in the table. The minimum number of cells required to produce the table is the sum of `numberOfDays` and `startDay`. If the sum is a multiple of 7, then the number of rows is the sum divided by 7; otherwise, it is the quotient plus 1. Based on this observation, we can write the code using a double for-loop. The exterior loop of the double for-loop produces the row values in a variable `row` starting from 0. The interior loop of the double for-loop produces the column values in a variable `col` between 1 and 7. The cell at `(row,col)` corresponds to the day index `7 * row + col - startDay`. If the day index is greater than or equal to 0 and less than or equal to `numberOfDays`, we need to print the value of the index; otherwise, the index is invalid, so the cell should be six white spaces.

10. **Parsing `Prinrtf` parameter** Write a program named `ParsePrintfD` that receives a `String` data, `w` (which is a syntactically correct integer format placeholder for `printf`), and produces the answers to the following questions on the screen:
    - Does it contain `'+'`?
    - Does it contain `','`?
    - Does it contain `'-'`?
    - Does it contain a sequence of numerals?
    - Does it contain a sequence of numerals and the sequence start with a `'0'`?

    Here is an execution example:

```
Enter your format string: +-,45
Has a '+' = true
Has a '-' = true
Has a ',' = true
Has a number = true
Has a zero = false
```

Here is one more:

```
Enter your format string: 034,3
Has a '+' = false
Has a '-' = false
Has a ',' = true
Has a number = true
Has a zero = true
```

11. **Generating a BMI table** Write a program named `BMITable` that generates a table of BMI values for a range of weights in pounds and for a range of heights in inches. The weight range is from 260 pounds down to 80 pounds with the value gap decreasing by 5 (the values are 260, 255, 250, . . . , 80). The height range is from 56 to 76 with the value increasing by 2 (the values are 56, 58, . . . , 76). The rows are the weights and the columns are the heights. Use the `printf` format of `"%5.1f"` to print the values. The output of the program should start with:

```
-------+-------------------------------------------------------
Weight | height (in.)
(lbs.) |   56   58   60   62   64   66   68   70   72   74   76
-------+-------------------------------------------------------
260    | 58.3 54.3 50.8 47.5 44.6 42.0 39.5 37.3 35.3 33.4 31.6
255    | 57.2 53.3 49.8 46.6 43.8 41.2 38.8 36.6 34.6 32.7 31.0
250    | 56.0 52.2 48.8 45.7 42.9 40.3 38.0 35.9 33.9 32.1 30.4
```

and end with:

```
95     | 21.3 19.9 18.6 17.4 16.3 15.3 14.4 13.6 12.9 12.2 11.6
90     | 20.2 18.8 17.6 16.5 15.4 14.5 13.7 12.9 12.2 11.6 11.0
85     | 19.1 17.8 16.6 15.5 14.6 13.7 12.9 12.2 11.5 10.9 10.3
80     | 17.9 16.7 15.6 14.6 13.7 12.9 12.2 11.5 10.8 10.3  9.7
-------+-------------------------------------------------------
```

Write and use a method for printing the separator line, a method for printing the two header lines, and a method for printing the rest of the table.