Chapter 2

Getting Started

Chapter Overview

This chapter introduces you to RPG IV specifications. You'll learn how to write a simple file read/write program using a procedural approach. You'll also learn how to include comments within your programs as documentation. Finally, you'll see how RPG's techniques of output editing let you control the appearance of values on reports.

RPG IV Specifications

RPG IV programs consist of different kinds of lines, called specifications. Each type of specification has a particular purpose. The specification types are

- Header (Control) specifications—provide default options for the source
- File specifications—identify the files a program will use
- Definition specifications—define variables and other data items the program will use
- Input specifications—describe the record layout for program-described input files
- Calculation specifications—detail the procedure the program will perform
- Output specifications—describe the program output (results)
- Procedure boundary specifications—segment the source into units of work, called procedures

Not every program requires every kind of specification. Most of the specification types require a different identifier, or form type, which must appear in position 6 of each program line. A File specification line of code, for example, must include an F in position 6; for this reason, File specifications are commonly called F-specs.

Specifications that you use must appear in a specific order, or sequence, within your source code, with all program lines that represent the same kind of specification grouped together. Figure 2.1 illustrates the order in which the specifications are grouped.

Figure 2.1
Order of Specifications in an RPG Program

H	Header (control) specifications
F	File specifications
D	Definition specifications
I	Input specifications
C	Calculation specifications
O	Output specifications
P	Procedure boundary
D	Definition specifications for procedure
C	Calculation specifications for procedure
P	Procedure boundary

Most RPG IV specifications require fixed-position entries in at least part of the specification. **Fixed position**, or **fixed format**, means that the location of an entry within a program line is critical to the entry's interpretation by the RPG IV compiler. The editor you use to enter your source code can provide you with prompts to facilitate making your entries in the proper location. (Appendix C provides more information about editors.)

Most specifications also support a **free-form** area of the specification, where you can code keywords and values with little or no regard to their specific location within the free-form portion of the specification.

The code samples in this book use two (or more) ruler lines to help you determine where to make your entries. The first ruler line indicates column position; the following line (or lines) contains "prompts" similar to those given by an editor. Most editors also provide a similar ruler line near the top of the editing window. These ruler lines should not appear in your source code; they are provided to help you understand where entries should appear.

Tip

As you begin to work with RPG specifications, don't be overwhelmed by what appear to be hundreds of entries with multiple options. Fortunately, many entries are optional, and you will use them only for complex processing or to achieve specific effects. This book introduces these entries gradually, initially showing you just those entries needed to write simple programs. As your mastery of the language grows, you will learn how to use additional specification entries that may be required for more complex programs.

When you begin writing your first program, you will notice that an entry does not always take up all the positions allocated for it within a specification. When that happens, a good rule of thumb is that alphabetic entries start at the leftmost position of the allocated space, with unused positions to the right, while numeric entries are usually right-adjusted, with unused positions to the left.

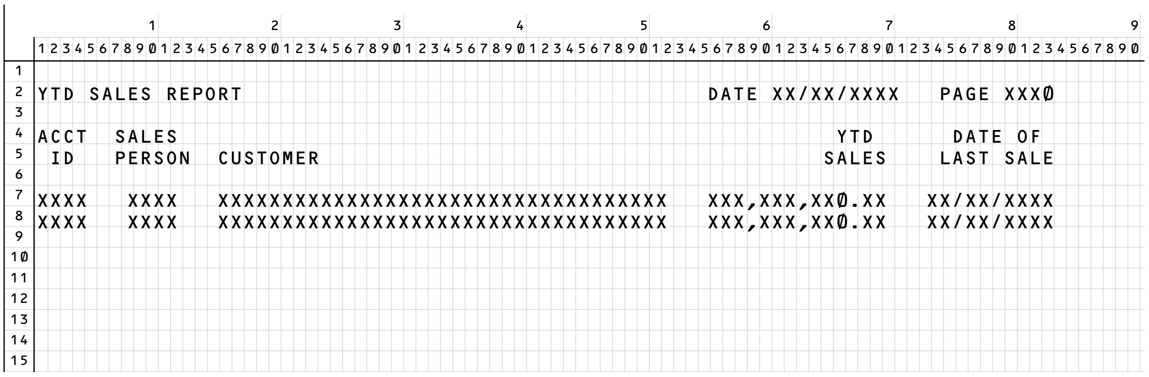
RPG Specifications for a Sample Program

Let's start with the minimal entries needed to procedurally code a simple read/write program. To help you understand how to write such a program, we will walk through writing an RPG IV program to solve the following problem.

We have a file—Customers—with records laid out as follows:

Field	Data Type	Length	Decimal Positions
Account Identifier	Alphanumeric	4	-
Salesperson	Alphanumeric	4	-
Customer Name	Alphanumeric	35	-
Customer Address	Alphanumeric	35	-
City	Alphanumeric	21	-
State/Province	Alphanumeric	2	-
Postal Code	Alphanumeric	10	-
Foreign Country	Alphanumeric	20	-
Date of Last Sale	Date (*mm/dd/yyyy*)	10	-
Year-to-Date Sales	Numeric	11	2

You want to produce a report laid out as follows:



```

YTD SALES REPORT                                      DATE XX/XX/XXXX    PAGE XXXØ

ACCT  SALES                                                      YTD      DATE OF
 ID   PERSON  CUSTOMER                                          SALES    LAST SALE

XXXX   XXXX   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX     XXX,XXX,XXØ.XX    XX/XX/XXXX
XXXX   XXXX   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX     XXX,XXX,XXØ.XX    XX/XX/XXXX
```

When you compare the desired output with the input record layout, you can see that all the output fields are present on the input records. No data transformation, data generation, or arithmetic calculation needs to take place within the program. Not all of the input fields are used in the report, but their locations in the input record layout will need to be considered when we are coding the RPG program. The required processing consists of reading each record from the input file, writing that data to the report with appropriate headings, and formatting the variable data.

Control Specifications

Although our sample program doesn't include them, **Control specifications** (sometimes called Header specifications, or H-specs) may be useful to control an RPG program's behavior. Control specifications provide the following functions:

- default formats (e.g., date formats) for the program
- changes to normal processing modes (e.g., changing the internal method the program uses to evaluate expressions)
- special options to use when compiling the program
- language enhancements that affect the entire program

Control specifications require an H in position 6. The remaining positions, 7–80, consist of reserved **keywords**, which have special values and meanings associated with them. There are no strict positional requirements for the keywords; they may appear in any order and in any position 7–80. The following header shows the layout of a Control specification:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

In the following example, Control specification keywords dictate the date and time formats to be used:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H Datfmt(*USA) Timfmt(*HMS)
```

A Control specification can include more than one keyword (with at least one space between them), and a program can have multiple Control specifications. Appendix A includes a complete list of Control specification keywords and their usage. Not all programs require Control specifications, but if they are present, Control specifications must appear as the first specifications in a program.

File Description Specifications

Our introductory RPG IV program will begin with File description specifications (also known by the shorter names **File specifications** and **F-specs**). All File specifications include an F in position 6. File specifications describe the files our program uses and define how the files will be used within the program. Each file used by a program requires its own File specification line. In our illustrative problem, the file Customers contains the data we want to process.

The output of our program is a printed report. Although you usually think of a report as hard copy rather than as a file per se, in RPG we produce a report through a printer file. Our introductory program will use a system-supplied printer file, Qprint, as the destination file for our report lines. This file then resides as a spooled file in an output queue, where it will wait until we release it to the printer. Your instructor will tell you which printer file to use in your programs and explain how to work with spooled files in the output queue.

You must code one File specification for each file the program uses. Although you can describe the files in any order, it is customary to describe the input file first. The following header shows the layout of a File specification. Note that in addition to column positions, the layout includes prompts to help you remember where to insert required entries.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
```

The completed File specifications for our program are shown below. We'll explain in detail each of the necessary entries for our sample program, and in subsequent chapters we will explain the entries not described here. (Appendix A includes a complete summary of all the RPG IV specifications.)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
FCustomers IF   F  152        Disk
FQprint    O    F  132        Printer Oflind(*Inof)
```

RPG IV lets you use both uppercase and lowercase alphabetic characters, but the language is not **case sensitive**. Thus, any lowercase letter you use within a file or variable name will be interpreted as its uppercase equivalent by the compiler. To aid in the program's readability, many programmers use **title case**, wherein each word in the source code is capitalized.

File Name (Positions 7–16)

First, in positions 7–16 (labeled *Filename++* on the ruler line), you enter the name of the file. In RPG IV, file names can be a maximum of 10 characters long. They must begin with an alphabetic character or the special character $, #, or @; the remaining characters may be alphabetic characters, numbers, or any of the four special characters _, #, $, and @. A file name cannot contain blanks embedded within the permissible characters.

Our practice problem input file is called Customers. The report file is Qprint. Note that you code file names like other alphabetic entries: beginning in the leftmost position allowed for that entry—in this case, position 7. Simply leave blank any unneeded positions to the right of the name.

File Type (Position 17)

Position 17 (labeled *I* on the ruler line) specifies the type of file or how the file will be used by the program. The two types we will work with in this program are input (type I) and output (type O). An **input file** contains data to be read by the program; an **output file** is the destination for writing output results from the program. In our example, Customers is an input file, and Qprint is an output file.

File Designation (Position 18; Input Files Only)

Every input file requires a file designation entry (position 18, labeled *P*). File designation refers to the way the program will access, or retrieve, the data in the file. In our example, we are going to retrieve data by explicitly reading records within our program rather than by using the built-in retrieval of RPG's fixed logic cycle. In RPG terminology, that makes the input file **full procedural**, so F is the appropriate entry for position 18. Since this designation applies only to input files, we'll leave it blank for the Qprint specification line.

File Format (Position 22)

The next required entry is file format. An F in position 22 (labeled *F*) stands for fixed format, which means that file records will be described within this program and that each record has the same fixed length. Although it is preferable to describe files externally using OS/400's built-in database facilities, for simplicity's sake we will start with program-described files and progress to externally described files in the next chapter. Because our files will be program described, an F is appropriate for both files of our sample program. All files, regardless of type, require an entry for file format.

Record Length (Positions 23–27)

You need to define the record length for each program-described file. Data file records can be of any length from 1 to 32,766 bytes; it is important that you code the correct value for this specification. When we add up the lengths of all the fields in Customers, we come up with a length of 152 bytes, so we enter 152 in positions 23–27. Note that record length is right-adjusted within the positions allocated for this entry. This is typical of most RPG IV entries that require a numeric value.

Most printers support a line of 132 characters. As a result, records of printer files (which correspond to lines of report output) are usually 132 positions long. Accordingly, output file Qprint is assigned a record length of 132 on its File specification.

Device (Positions 36–42)

The Device entry indicates the device associated with a file. Database files are stored on disk; accordingly, Disk is the appropriate device entry for the Customers file. The device associated with printer files is Printer. You enter these device names, left-adjusted, in positions 36–42 (labeled *Device+*).

Keywords (Positions 44–80)

The Keywords area of the File specification gives you an opportunity to amplify and specialize the basic file description in the positional area (positions 6–43) of the F-spec. RPG allows a number of reserved keywords (listed in Appendix A) in this area of the specification. Typically, they are coded with one or more values (**arguments**) in parentheses immediately following the keyword itself. You can code more than one keyword on a specification line in positions 44–80 without being too concerned about any other positional requirements. Most RPG programmers, however, prefer to limit their code to one keyword per line; if a specification requires more than one keyword, you can simply continue coding them in the Keywords area on subsequent F-spec lines.

Our sample program will use only one keyword: Oflind (Overflow indicator). **Overflow** is the name given to the condition that occurs when a printed report reaches the bottom of a page. Usually, when overflow occurs you will want to eject the printer to the next page and print a new set of heading lines before printing the next detail line. Your program can automatically detect overflow through the use of a reserved variable called an **overflow indicator**. The overflow indicators provided by RPG are called OA, OB, OC, OD, OE, OF, OG, and OV; you would code these indicators as *INOA, *INOB, and so on. The Oflind keyword associates one of these indicators with a printer device file—if that file signals overflow, the file's overflow indicator will be automatically set to *On. You can then test that indicator just before printing a detail line to determine whether or not you want to print headings first. In our sample, we name indicator OF as the overflow indicator for Qprint by coding Oflind(*Inof) in the Keywords area of the appropriate F-spec.

No other File specification entries are required to describe the files used by our sample program. In this introductory explanation, we've skipped over some of the entries that are not needed in this program; we'll cover them later. The completed File specifications for the program are shown again below.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
FCustomers IF   F  152        Disk
FQprint    O    F  132        Printer Oflind(*Inof)
```

Input Specifications

Input specifications, identified by an I in position 6, come after the File specifications in our introductory program. **Input specifications** (I-specs) describe the records within program-described input files and define the fields within those records. Every program-described input file defined on the File specifications must be represented by a set of Input specification lines.

Input specifications use two types of lines:

- **Record identification entries**, which describe the input records at a general level
- **Field description entries**, which describe the specific fields within the records

Together, these two types of lines describe the structure of the record layout for each program-described input file in the program. Each record identification line must precede the field entries for that record. The general layout for these two kinds of Input specifications is shown below.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..................................
I.......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrP1MnZr......
```

The I-specs for our introductory program are shown here. We'll explain in detail those entries required by our sample program.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..................................
I.......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrP1MnZr......
ICustomers NS
I                                  1    4  Accountid
I                                  5    8  Salesperson
I                                  9   43  Name
I                                 44   78  Address
I                                 79   99  City
I                                100  101  State
I                                102  111  Postalcode
I                                112  131  Country
I                       *USA D  132  141  Lastsaledate
I                                142  152 2Ytdsales
```

Record Identification Entries

Record identification lines describe the input records at a general level. Each line takes the following form:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..................................
```

File Name (Positions 7–16)

A record identification line must contain the name of the input file in positions 7–16 (labeled *Filename++* on the specification line). This name must match the entry on the File specification—in our case, Customers. The file name is a left-adjusted entry.

Sequence (Positions 17–18)

The next required record identification entry is Sequence, in positions 17–18 (labeled *Sq*). This entry signals whether the system should check the order of records in the file as the records are read during program execution. **Sequence checking** is relevant only when a file contains multiple record formats (that is, records with different field layouts). When sequence checking is not appropriate (which is usually the case), code any two alphabetic characters in positions 17–18 to signal that sequence checking is not required. Many programmers use NS to signal “no sequence.” Because the Customers file contains a single record format, we enter NS in positions 17–18.

The complete record identification specification is illustrated below.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.................................
ICustomers NS
```

Note

Note that with the specification coded as shown, the compiler will issue a warning that a record identification indicator is missing from the line. Although record identification indicators are relevant in fixed logic processing (discussed in Appendix E), they are not used in modern RPG programming. Simply ignore the compiler warning; it will not prevent your program from being compiled successfully.

Field Description Entries

Field description entries immediately follow the record identification entry. You define each field within the record by giving the field a valid name, specifying its length, and declaring its data type. Although you can define the fields of a record in any order, convention dictates that fields be described in order from the beginning of the record to the record’s end.

Each field description entry takes the following form:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
I.......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrP1MnZr......
```

Field Location (Positions 37–46)

You define a field’s length by specifying the beginning position and the ending position of the field within the input record. Length notation is not allowed. The beginning position is coded as the “from” location (positions 37–41 of the Input specifications, labeled *From+*); the ending position is the “to” location (positions 42–46, labeled *To+++*). If the field is 1 byte long, the from and to entries will be identical because the field begins and ends in the same location of the record.

Character fields may be up to 65,535 bytes long. Numeric fields may be up to 63 digits long. The length of native dates depends on their format but may be up to 10 bytes long. The beginning and ending positions are right-adjusted within the positions allocated for these entries. You do not need to enter leading, nonsignificant zeros.

Decimal Positions (Position 47–48)

Numeric fields require a decimal position entry in positions 47–48 (labeled *Dc*) indicating the number of decimal positions to the right of the decimal point. In RPG IV, a field must be numeric to be used in arithmetic calculations or to be edited for output, so it is important to *not* overlook the decimal position entry. If a numeric field represents whole numbers, the appropriate entry for its decimal positions is 0 (zero). Numeric fields can contain up to 63 positions to the right of the decimal point. Remember that the total length of the numeric field *includes* any decimal places (but not the decimal point itself or comma separators).

To define a field as a **character field**, simply leave the decimal position entry blank. Date fields (with a *D* in position 36) are also coded with a blank in the decimal position entry. Chapter 4 provides a more complete discussion of RPG IV data types.

Field Name (Positions 49–62)

The last required entry for a field description specification is a name for the field being described. This name, entered left-adjusted in positions 49–62 (labeled *Field*+++++++++) must adhere to the rules for valid field names in RPG IV. Within a record, a valid field name

- Uses letters, digits, or the special characters _, #, @, and $
- Does not begin with a digit or an underscore
- Does not include embedded blanks

In addition, a field name generally is 14 characters long or less. This is a practical limit, imposed by the fixed-format nature of the input specification.

The alphabetic characters can be either uppercase or lowercase or a combination (mixed case). RPG IV does not distinguish between letters on the basis of their case, but using a combination of upper- and lowercase characters—for example, capitalizing each word in the source code—makes your field names easier for others to understand.

Tip

Although RPG allows them, you should avoid the use of special characters $, #, and @ in RPG names. These special characters may not exist in all the languages or the character sets your program may use to compile. If the language or character set cannot recognize the character, the compiler will not be able to successfully translate the code. You should also avoid the underscore (_) in an RPG name; it's a "noisy" character and doesn't significantly aid the readability of your program.

Although not an RPG IV requirement, it is good programming practice to choose field names that reflect the data they represent by making full use of the 14-character limit for names. For example, "Loannumber" is far superior to "X" for the name of a field that will store loan numbers. Choosing good field names can prevent your accidental use of the wrong field as you write your program and can help clarify your program's processing to others who may have to modify the program.

Data Attributes (Positions 31–34)

RPG most commonly uses positions 31–34 to specify a format for date or time fields. RPG supports native date and time data types to enable date calculations and manipulation—important factors in modern business processing. We'll discuss dates and date formats in more detail in Chapter 7. The entry *USA in positions 31–34 of an I-spec indicates that the field Lastsaledate is in *mm/dd/yyyy* format, including the slash (/) separator characters. Since the other fields in the record layout are not date fields, this entry for those fields is left blank.

Data Type (Position 36)

For most alphanumeric (character) or numeric fields, you may leave position 36 blank. But for fields that represent other types of data, you must make an entry in position 36 to tell the compiler the external data type of the field. In our sample program, the D entry in this position indicates that the field Lastsaledate is a native date. The other fields in the record layout are character or numeric fields and do not require an entry here. Chapter 4 provides a more complete discussion of RPG data types.

Note

***Native date fields* are *not* the same as numeric fields that may be used to store date information. A native date field is stored in a special format that the computer will implicitly recognize as a date. Numeric fields require specific arithmetic or conversion coding to treat them as dates. We'll cover more about native dates in Chapter 7.**

You'll recall that not all the fields in the Customers file will appear on our desired report. If a program does not use all the fields coded in the Input specifications, the compiler will issue a warning, but this is not necessarily an error condition that will prevent a successful compile. You can omit the unused fields from the I-specs, but the remaining entries must reflect their correct position in the record layout. To review, the field description entries of the Input specifications for our sample program are shown below. Unused fields have been omitted.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC...................................
I.............................Fmt+SPFrom+To+++DcField+++++++++L1M1FrP1MnZr......
ICustomers NS
I                                        1    4  Accountid
I                                        5    8  Salesperson
I                                        9   43  Name
I                             *USA D   132  141  Lastsaledate
I                                      142  152 2Ytdsales
```

In these Input specifications, we define field Ytdsales as numeric by including a decimal position entry in positions 47–48. Lastsaledate is a native date (with a D in position 36). The remaining fields are character fields.

Output Specifications

Calculation specifications follow immediately after Input specifications in RPG programs. We, however, will discuss **Output specifications** next because their required entries parallel those required on Input specifications in many ways. Every program-described output file named on the File specifications needs a set of Output specifications that provide details about the required output. All Output specification lines require an O in position 6.

Output specifications, like Input specifications, include two kinds of lines: record identification lines, which deal with the output at the record level; and field description lines, which describe the content of a given output record. When the output is a report rather than a data file, “record” roughly translates to “report line.” Most reports include several different report-line formats; each needs definition on the Output specifications.

To refresh your memory, our output file, Qprint, is to contain a weekly sales report, formatted as shown in the printer spacing chart on page 21.

Our report includes four kinds of lines, or **record formats**. Three of the lines are headings, which should appear at the top of the report page, while the fourth is a **detail line** of variable information. The term “detail line” means that one line is to be printed for each record in an input file. The line contains detailed information about the data records being processed.

The following RPG IV code shows the complete Output specifications to produce the report described above. You should refer to this code again as you read about the required Output specification entries.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
OFilename++DF..NØ1NØ2NØ3Excnam++++B++A++Sb+Sa+.............................
O..............NØ1NØ2NØ3Field+++++++++YB.End++PConstant/editword/DTformat++
OQprint    E            Headings       2  2
O                                        16 'YTD SALES REPORT'
O                                        56 'DATE'
O                       *Date          Y     67
O                                        74 'PAGE'
O                       Page           Z     79

O          E            Headings       1
O                                         4 'ACCT'
O                                        11 'SALES'
O                                        65 'YTD'
O                                        78 'DATE OF'

O          E            Headings       2
O                                         3 'ID'
O                                        12 'PERSON'
O                                        22 'CUSTOMER'
O                                        66 'SALES'
O                                        79 'LAST SALE'

O          E            Detail         1
O                       Acountid               4
O                       Salesperson           11
O                       Name                  49
O                       Ytdsales       1      66
O                       Lastsaledate          79
```

Record Identification Entries

Output specifications require a record identification entry for each different line of the report. Each of these lines represents a record format and must be followed with detailed information about what that record format (or report line) contains. Because our report has four types of lines to describe, we have four record format descriptions in our Output specifications.

The header that follows illustrates the layout for record identification entries. The following discussions refer to this layout.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................
```

File Name (Positions 7–16)

The first record identification entry requires a file name entry in positions 7–16 (labeled *Filename++*). This file name serves to associate the record being described with the output file described on the File specifications. Thus Qprint, our output file, appears as the file name entered on the first record identification line of the Output specifications above. Although the Output specifications include four record format descriptions, because each describes a format to be written to the same file (Qprint), you do not have to repeat the file name entry on subsequent record identification entry lines.

Type (Position 17)

Each record format description requires an entry in position 17 (labeled *D*) to indicate the type of line being described. In this context, "type" refers to the way RPG IV is to handle printing the line. Because we will be using procedural techniques to generate the report instead of relying on RPG's fixed logic cycle, all the record format lines are **Exception lines**. As a consequence, we enter an *E* in position 17 of each record format line.

Exception Name (Position 30–39)

In RPG IV, it is common practice to provide a name in positions 30–39 (labeled *Excnam++++*) for each exception line. Although not required, such names let you control printing without the use of indicators. By using **exception names**, you can easily refer to lines to be printed from within your Calculation specifications.

Moreover, you can assign the same name to lines that need to be printed as a group at the same time. Because our report has three lines that should be printed together at the top of the page, we have given each the name Headings. The fourth line, which will contain the variable information from our data file, is identified as Detail. Note that Headings and Detail are arbitrarily assigned names, not RPG-reserved terms. Exception names follow the same rules of naming as field names (up to 10 characters long), and they are left-adjusted within positions 30–39.

Space and Skip Entries (Positions 40–51)

One more set of entries is needed to complete the record format line definitions. These entries describe the vertical alignment of a given line within a report page or relative to other report lines. Two kinds of entries control this vertical alignment: **Space entries** and **Skip entries**. Each variant offers "before" and "after" options.

It is important to understand the differences between Space entries and Skip entries. Space entries specify vertical printer positioning *relative to the current line*. Space is analogous to the carriage return on a typewriter or the Enter key on a computer. Each Space is the equivalent of hitting the Return (or Enter) key. Space before (positions 40–42, labeled *B*++) is like hitting the Return key before you type a line; Space after (positions 43–45, labeled *A*++) is like hitting Return after you type a line.

The same record format line can include both a Space before and a Space after entry. If both the Space before and the Space after entries are left blank within a record format description, the system defaults to Space 1 after printing—the equivalent of single-spacing. If you have either a Space before or a Space after entry explicitly coded and the other entry is blank, the blank entry defaults to 0. The maximum value you can specify for any Space entry is 255.

In contrast to Space entries, Skip entries instruct the printer to "skip to" the designated line on a page. Skip entries specify an *absolute vertical position on the page.* Skip 3 before printing causes the printer to advance to the third line on a page before printing; Skip 20 after printing causes the printer to advance to the 20th line on the page after printing a line. If the printer is already past that position on a given page, a Skip entry causes the paper to advance to the designated position on the next page. Most often, you will have a Skip before entry only for the first heading line of a report. Programmers most often use Skip entries to advance to the top of each new report page. Skip entries are also useful when you are printing information on a preprinted form, such as a check or an invoice.

You code any Skip before entry in positions 46–48 (labeled *Sb*+); Skip after entries are made in positions 49–51 (labeled *Sa*+). If you do not code any Skip entries, the system assumes that you do not want any skipping to occur. The maximum value you can specify for any Skip entry is 255.

The following record format lines show the spacing and skipping entries for our sample program:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
OFilename++DF..NØ1NØ2NØ3Excnam++++B++A++Sb+Sa+.............................
OQprint    E            Headings       2  2
 ...
O          E            Headings       1
 ...
O          E            Headings       2
 ...
O          E            Detail         1
```

Because we want the first heading of our report to print on the second line of a page, we code a Skip 2 before entry in positions 46–48 of the record format line describing that line. The Space 2 after entry (positions 43–45) for that same heading line will advance the printer head to the correct position for the second Headings line—that is, line 4.

The second Headings line, with its Space 1 after entry, positions the printer head for the third Headings line, which in turn, with its Space 2 after entry, positions the printer head for the first Detail line of data to print. Because the report-detail lines are to be single spaced, exception line Detail contains a Space 1 after entry.

Field Description Entries

Each record format line of the Output specifications is followed by field description entries that describe the contents of the line. Each field description specification

- Identifies an item to appear on the line
- Indicates where the item is to appear horizontally on the line
- Specifies any special output formatting for that item

The **field-level** items to be printed will be either a variable (field) or a constant (literal). Field-level items to be included within a record format may be entered in any order, although by convention programmers enter them in the order in which they are to appear in the output. The code below illustrates the layout for these field description entries.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
O..............NØ1NØ2NØ3Field+++++++++YB.End++PConstant/editword/DTformat++
```

Field Name (Positions 30–43)

The name of each field whose value is to appear as part of the output record is coded in positions 30–43 (labeled *Field*+++++++++). Any field appearing as part of the Output specifications must have been defined earlier in the program.

In our sample program, most of the fields to be printed are part of the Detail record format. These are the same fields—Accountid, Salesperson, Name, Ytdsales, and Lastsaledate—that we defined as part of our input record (though not necessarily in the same order as they appear in the record layout). When we include these field names in the output, each time our program processes a successive record from the input file, each Detail line printed will contain the data values present in those fields of the input record.

In addition to the input fields, two RPG IV reserved words that function as built-in, predefined fields appear as part of the report headings. In the first Headings line defined on page 29, notice the field name **Page**. RPG supplies this field to automatically provide the correct page numbers for a report. Page, a four-digit numeric field, has an initial value of 1; this value is automatically incremented by 1 each time the report begins a new page.

The **Date* field, which also appears as part of the first Headings line, is another RPG IV **reserved word**. *Date, an 8-digit numeric field, stores the current date, typically in *mmddyyyy* format. Any time your program needs to access the date on which the program is running, you can simply use *Date as a field. RPG IV also stores a 6-digit version of the date in reserved word *Udate*. Reserved words **Day*, *Uday*, **Month*, *Umonth*, **Year* (4 digits), and *Uyear* (2 digits) let you individually access the day, month, and year portions of the current date. Note that these reserved words refer to numeric fields, not native dates; the RPG program will treat them as numbers.

Constants (Positions 53–80)

In addition to fields, whose values change through the course of a program's execution, Output specifications typically contain **constants**, or **literals**—characters that do *not* change and instead represent the actual values that are to appear on the report. You enter each constant, enclosed within apostrophes ('), in positions 53–80 (labeled *Constant/editword/DTformat*++) of the Output specifications. The apostrophe on the left of the code should appear in position 53; in other words, you enter constants left-adjusted within positions 53–80. A constant cannot appear on the same Output specification line as a field; each needs its own line.

In our sample program, the first heading is to contain the word PAGE as well as the page number. Accordingly, we code PAGE as a constant within the first Heading line. Also, part of this first heading is the title—YTD SALES REPORT. Although several words make up this constant, you enter the group of words as a single constant, enclosed in apostrophes; the spaces between the words form part of the constant.

The second and third report lines, or record formats, consist of column headings for the report. These, too, are handled as constants, with the appropriate values entered in positions 53–80.

Notice that in the sample program, the column heading lines are broken up into conveniently sized logical units and that each unit is then coded as a separate constant.

Note also that you can ignore blank, or unused, positions in output lines unless they appear within a string of characters that you want to handle as a single constant (e.g., 'DATE OF' or 'LAST SALE').

End Position in Output Record (Positions 47–51)

You denote where a field or constant appears horizontally within a line by coding its **end position**—that is, the position of its last, or rightmost, character—within the line. To specify an end position, enter a numeric value that is right-adjusted within positions 47–51 (labeled *End*++); this represents the actual position desired for the rightmost character of the field or constant.

For example, because we want the E in constant PAGE to appear in column 74 of the first heading line of our sample report, we code a 74 in positions 50–51 of the specification entry for the constant PAGE. The printer spacing chart indicates that the rightmost digit of the page number should appear in column 79 of the report line. Accordingly, 79 is the specified end position for field PAGE within its Output specification line.

Our Output specifications include an end position for each field or constant that is part of our report. If you omit an end position for a field or constant, that item is output immediately adjacent to the previous item, with no blanks separating the items.

You can also optionally specify the placement of a field or constant relative to the end position of the previously defined field. To use this alternative method, you put a plus sign (+) in position 47 and a right-adjusted numeric value in the remaining positions. The value tells how many blanks you want between the end of the previous field and the beginning position of the current field. The listing below illustrates how you would code the Detail line of our report using this relative notation.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++
O          E            Detail         1
O                       Acountid             4
O                       Salesperson      +   3
O                       Name             +   3
O                       Ytdsales      1  +   3
O                       Lastsaledate     +   3
```

The above code will end field Accountid in position 4 and put three blanks between the end of Accountid and the start of Salesperson, three blanks between the end of Salesperson and the start of Name, and so on.

Edit Codes (Position 44)

Three of the fields appearing in the output—Page, *Date, and Ytdsales—have an entry in position 44, **Edit codes** (labeled *Y*). An edit code formats numeric values to make them more readable. The Z edit code associated with Page suppresses leading zeros when printing the value; so if Page shows a value of 0001, it will print as 1.

The Y edit code associated with *Date inserts slashes within the printed number. Thus, if *Date has a value of 12202009, it will be printed as 12/20/2009. Note that edit codes apply to numeric fields only. Lastsaledate, which is a native date field—not a number—already has separator characters as part of its value, so it does not require an edit code.

Edit code 1 causes commas and a decimal point to be inserted within the printed value of Ytdsales, and it signals that if the Ytdsales value is 0, the zero balance should appear on the report rather than being completely suppressed. RPG IV includes a large selection of editing alternatives to let you print or display values using a format most appropriate to your needs. A detailed discussion of these editing features appears at the end of this chapter.

Output Continuation Lines

Although they are not appropriate for our current report, output **continuation lines** introduced in RPG IV let you code long constants as a single entry that spans more than one specification line. The layout for the continuation form of the Output specification is as follows:

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
O..............................................Constant/Editword-continues+
```

Assume, for example, that you're defining a report that is to be captioned "ACME EXPLOSIVES SALES REPORT"—a constant too long to fit on one specification line. You can code this caption as a single constant on two (or more) specification lines by using the continuation feature. (Of course, you could also break the constant into two or more constants and then just code each constant on its own output line.)

To use the continuation feature, you code the end position for the entire constant on the first line—here, we use 90—together with some portion of the constant; then, signal that the constant is continued by terminating the entry on the first line with a hyphen (-) or a plus sign (+). A hyphen signals that the continuation resumes with the *first position* (i.e., position 53) of the continued constant on the next line, while a plus signals that the continuation resumes with the *first nonblank character* encountered in the continued constant on the next line.

The following code illustrates this output feature. Notice that you use an apostrophe only at the very beginning and the very end of the continued constant, rather than needing a set of apostrophes on each line.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
O..............NØ1NØ2NØ3Field+++++++++YB.End++PConstant/editword/DTformat++
O..............................................Constant/Editword-continues+
  // The two examples below would produce the same output because of the
  // use of the + and -.
O                                           9Ø 'ACME EXPLOSIVES SALES +
O                                                                  REPORT'

O                                           9Ø 'ACME EXPLOSIVES SALES -
O                                              REPORT'
```

Calculation Specifications

We have now defined the files to be used by our application, the format of the input records to be processed, and the desired output of the application. All we need to complete our program is a description of the processing steps required to obtain the input and write the report. We use **Calculation specifications** to describe these processing steps.

Before coding Calculation specifications, you need to develop the logic required to produce the desired output. In general, you would complete this stage of the program-development cycle—designing the solution—before doing any program coding, but we delayed program design to introduce you to some of the RPG IV specifications and give you a taste of the language.

We can sketch out the required processing of our program using **pseudocode**. Pseudocode is simply stylized English that details the underlying logic needed for a program. Although no single standard exists for formatting pseudocode, it consists of key control words and indentation to show the scope of control of the logic structures. It is always a good idea to work out the design of your program before actually coding it in RPG IV (or in any other language). Pseudocode is language independent and lets you focus on what needs to be done rather than on the specific syntax requirements of a programming language.

Our program exemplifies a simple read/write program in which we want to read a record, write a line on the report, and repeat the process until no more records exist in the file (a condition called **end-of-file**). This kind of application is termed **batch processing** because once the program begins, a "batch" of data, accumulated in a file, directs its execution. Batch programs can be run unattended because they do not require control or instructions from a user.

The logic required by our read/write program is quite simple:

Correct algorithm

```
Print headings
Read a record
While there are more records
   Print headings if necessary
   Write a detail line
   Read the next record
Endwhile
End program
```

Note that *While* indicates a repeated process, or loop. Within the loop, the processing requirements for a single record—in this case, simply writing a report line—are detailed and then the next record is read. Because we want to print report headings just once at the beginning of the report rather than once for each record, that step is listed at the beginning of the pseudocode *outside* the loop.

You may wonder why the pseudocode contains two read statements. Why can't there be just a single read, as in the first step within the While loop below?

Incorrect algorithm

```
Print headings
While there are more records
   Read the next record
   Print headings if necessary
   Write a detail line
Endwhile
End program
```

The preceding algorithm would work fine as long as each read operation retrieved a data record from the file. The problem is that eventually the system will try to read an input record and fail because there are no more records in the file to read. Once a program has reached end-of-file, it should not attempt to process any more input data. The incorrect algorithm above would inappropriately write a detail line after reaching end-of-file.

The correct algorithm places the read statement as the last step within the While loop so that as soon as end-of-file is detected, no further writing will occur. However, if that were the only read, our algorithm would try to write the first detail line before reading any data. That's why the algorithm also requires an initial read (often called a **priming read**) just before the While loop to "prime" the processing cycle.

After you have designed the program, it is a simple matter to express that logic in a programming language—that is, once you have learned the language's syntax. The following free-format Calculation specifications show the correct algorithm expressed in RPG IV. Notice the specifications' striking similarity to the pseudocode we sketched out earlier.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
 /Free
   Except Headings;
   Read Customers;

   Dow Not %Eof(Customers);

     If *Inof;
       Except Headings;
       Eval *Inof = *Off;
     Endif;

     Except Detail;
     Read Customers;
   Enddo;

   Eval *Inlr = *On;
   Return;
 /End-Free
```

Calculation specifications specify the processing that needs to be done. Free-format Calculation specifications are specified between /Free and /End-free instructions. These instructions, called **compiler directives**, direct the RPG IV compiler to use free-format syntax rules for any of the instructions within the block of code between the directives. The /Free and /End-free directives must be coded exactly as shown, beginning with a slash (/) character in position 7.

The instructions within the /FREE block usually begin with an operation that specifies an action to be taken. RPG IV supports a number of reserved words to identify valid operations. Many of these operations are followed by **operand values**, which RPG calls **factors**, to provide the compiler with the details necessary to perform an operation; other operation codes (Dow, If, and Eval in our example) are followed by expressions that the program will evaluate. Finally, each free-format Calculation specification must end with a semicolon (;).

Spacing is not usually critical in a free-format Calculation specification. You may code the specification in any position from 8 to 80; positions 6 and 7 *must* be blank. You may also indent operations to clarify the flow of the program.

RPG IV Operations

The RPG program executes the Calculation specifications sequentially (from beginning to end) unless the computer encounters an operation that redirects flow of control. Our program uses eight operations: Eval, Except, Read, Dow, Enddo, If, Endif, and Return. Let's look at the specific operations used within the calculations of our program. The intent here is to provide you with sufficient information to understand our basic program and to write similar programs. Several of the operations described in the following section are discussed in more detail in subsequent chapters of this book.

Except (Calculation Time Output)

An **Except** operation directs the program to output one or more E lines from the Output specifications. If no factor is coded with Except, the operation causes the system to output all unnamed E lines. In general, however, RPG programmers name their E lines and use the Except operation with an E-line name to state explicitly which line or lines are to be involved in the output operation. In the sample program, the first Except operation specifies Headings as the name of the group of E-lines to print. As a result, the three heading lines of our report will be printed. Later on, a second Except also prints heading lines if overflow has been reached. A third Except specifies Detail. When the program executes this line of code, our exception line named *Detail* will be printed, using the values of the fields from the currently retrieved Customers record.

Read (Read Sequentially)

Read is an input operation that instructs the computer to retrieve the next sequential record from the named input file—in this case, our Customers file. To use the Read operation with a file, you must have defined that file as input-capable on the File specifications.

Dow (Do While)

The **Dow** operation establishes a loop in RPG IV. An Enddo operation signals the end of the loop. Note that this Dow and Enddo correspond to the While and Endwhile statements in our pseudocode. The Dow operation repeatedly executes the block of code in the loop as long as the condition associated with the Dow operation is true. Because our program's Dow condition is preceded by the word *Not*, this line reads "Do while the end-of-file condition is not true." It is the direct equivalent of the pseudocode statement "While there are more records" because the end-of-file condition will come on only when our Read operation runs out of records.

The *%Eof* entry in this statement is an RPG IV built-in function that returns a true ('1' or *ON) or false ('0' or *OFF) value to indicate whether the file operation encountered end-of-file. **Built-in functions** (sometimes called *BIF*s) perform specific operations and then return a value to the expression in which they are coded. Most built-in functions allow you to enter values called *arguments* in parentheses immediately following the built-in function to govern the function. In this case, %Eof(Customers) means that we want our program to check the end-of-file condition specifically for the Customers file.

Enddo (End Do Group)

The **Enddo** operation serves to mark the end of the scope of a Do operation, such as Dow. All the program statements between the Dow operation and its associated Enddo are repeated as long as the Dow condition is true.

If

RPG's primary decision operation is the **If** operation. If the relationship expressed in the conditional expression coded with the If operation is true, all the calculations between the If and its associated Endif operation are executed; if the relationship is not true, those statements are bypassed. By coding

```
If *Inof = *On;
```

or simply

```
If *Inof;
```

we are telling the program that it should execute the following lines of code only if the overflow indicator *OF* is on:

```
Except Headings;
Eval *Inof = *Off;
```

Endif (End If Group)

The **Endif** operation marks the end of the scope of an If operation. All the program statements between the If operation and its associated Endif are executed as long as the If condition is true.

Tip

It is common practice to indent blocks of code that appear between Dow or If and their associated Enddo or Endif operations. By indenting the blocks, you can easily see which code is associated with the Dow or If operation. Don't overdo it, though. Indenting a couple of spaces is enough.

Eval (Evaluate Expression)

Eval is an operation used to assign a value to a variable. In the sample program, by coding

```
Eval *Inof = *Off;
```

we are assigning the value *Off to the overflow indicator OF (coded as *Inof). We do this after printing the heading lines, so that the program will know that it is no longer necessary to print the headings until indicator OF is once again set to *On automatically.

Later in the program, we use the line

```
Eval *Inlr = *On;
```

to assign the value *On to a special reserved indicator variable called **Last Record** (coded as ***Inlr**, read as **indicator** LR). *Inlr (commonly referred to as LR) performs a special function within RPG IV. If LR is on when the program ends, it signals the computer to close the files and free the memory associated with the program. If LR is not on, the program continues to tie up some of the system's resources even though the program is no longer running.

In most cases, specifying Eval is optional in a free-format Calculation specification; you can simply code the assignment expression without explicitly coding the Eval operation. Eval is included in this example to provide easy comparison with the fixed-format code—where it is required—but it could have been left out in the free-format C-spec.

Return (Return to Caller)

The **Return** operation returns control to the program that called it—either the computer's operating system or perhaps another program. Program execution stops when a Return is encountered. Although your program will end correctly without this instruction—provided you have turned on LR—including it is a good practice. Return clearly signals the endpoint of your program and lets the program become part of an application system of called programs. Chapter 12 deals in detail with called programs.

Fixed-Format Calculations

You may wonder why the free-format section of code is called *Calculation specifications*. Earlier versions of RPG IV (before Version 5) did not support free-format specifications. Instead, this function was handled by a fixed-format specification, which had a C in column 6. These specifications were called Calculation specifications (C-specs). Though modern RPG programming style encourages free format, this section of code is still commonly called Calculation specifications—even though the C in column 6 is no longer used.

The following shows the fixed-format C-specs equivalent to our free-format code.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CLØNØ1Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CLØNØ1Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++++
C                   Except    Headings
C                   Read      Customers
C                   Dow       Not %Eof(Customers)
C                   If        *Inof
C                   Except    Headings
C                   Eval      *Inof = *Off
C                   Endif
C                   Except    Detail
C                   Read      Customers
C                   Enddo
C                   Eval      *Inlr = *On
C                   Return
```

Fixed-format Calculation specifications do not require the /Free and /End-free directives. Instead, each line requires a C in position 6. Each Calculation specification contains an operation entered in positions 26–35 (labeled *Opcode(E)+*). Depending on the operation, specifications may also include a value in Factor 1 (positions 12–25), Factor 2 (positions 36–49), the Result field (positions 50–63), or the extended Factor 2 field (positions 36–80). Indicators associated with operations may also appear in positions 71–76. Fixed-format Calculation specifications do not require a semicolon delimiter at the end of the line. Appendix E covers fixed-format Calculation specifications in more detail.

Internal Documentation

You might think that once you have a program written and running, you are done with it forever and can move forward and develop new programs. Actually, about 70 percent of all programming is maintenance programming rather than new application development. Maintenance programming involves modifying existing programs to fix problems, address changing business needs, or satisfy user requests for modifications.

Because of the high probability that any program you write will be revised sometime in the future—either by yourself or by some other programmer in your company—it is your responsibility to make your program as understandable as possible to facilitate these future revisions. RPG programmers use several techniques to document their programs.

Program Overview

Most companies require overview documentation at the beginning of each program. This documentation, coded as a block of comments, states the function or purpose of the program, the program's author, the date when the program was written, and any special instructions or peculiarities of the program that those working with it should know.

If the program is revised, entries detailing the revisions—including the date of the revisions and their author—are usually added to that initial documentation. If a program uses several indicators, many programmers will provide an indicator "dictionary" as part of their initial set of comments to state the function or role of each indicator used within the program.

Comments

Another good way to help others understand what your program does is to include explanatory documentation internal to your program through the use of **comment lines**. RPG IV comments begin with **double slashes** (//) entered anywhere within positions 8–80. In free-format specifications, these comments can make up an entire line or a portion of the line. Once the compiler encounters the // characters, it will ignore the rest of the line, treating the remainder as a comment. Using // to specify comments is not limited to free-format specifications; you can enter comment lines anywhere in the program. In fixed-format specifications, the comments make up an entire line (positions 7–80); the line must begin with // characters and cannot include any compilable code.

Comments exist within the program at a source-code level only; they are for the benefit of programmers who may have to work with the program later. You should include comments throughout your program as needed to help explain specific processing steps that are not obvious. In adding such comments, you should assume that anyone looking at your program has at least a basic proficiency with RPG IV; your documentation should help clarify your program to such a person. Documenting trivial, obvious aspects of your program is a waste of time. On the other hand, failing to document difficult-to-grasp processing can cost others valuable time. Inaccurate documentation is worse than no documentation at all because it supplies false clues that may mislead the person responsible for program modification.

Appropriately documenting a program is an important learned skill. If you are uncertain about what to document, ask yourself, "What would I want to know about this program if I were looking at it for the first time?"

Fixed-Format Comments

In addition to the preferred // comment notation, fixed-format RPG statements allow an older alternative: asterisk comments. In fixed-format RPG IV syntax, an asterisk (*) in position 7 of any line—regardless of the specification type—designates that line as a comment; you can enter any documentation, in any form that you like, within the remaining portion of the line.

Free-format specifications do not allow asterisk comments. This book uses // comments exclusively. All the specification forms also include a comment area in positions 81–100 so that you can easily add a short comment to any line of code.

Blank Lines

In addition to the use of comments, many programmers find that a program's structure is easier to understand when blank lines are used to break the code into logical units. To facilitate using blank lines within your code, RPG IV treats two types of lines as blank: first, any line that is completely blank between positions 6 and 80 can appear anywhere within your program. Second, if position 6 contains a valid specification type and positions 7–80 are blank, the line is treated as a blank line; however, the line must be located in that portion of the program appropriate for its designated specification type.

The Completed Program

Our completed sample RPG IV program is shown below. Note that the order of the program statements is File, Input, Calculations, and Output. RPG requires this order. Also note that you can use blank comment lines or lines of asterisks to visually break the program into logical units and that using lowercase lettering within internal documentation helps it stand out from program code.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
  // -----------------------------------------------------------------------
  // This program produces a year-to-date sales report. The report data
  // comes directly from input file Customers.
  //      Date Written:  12/15/2006
  // -----------------------------------------------------------------------

FCustomers IF   F  152        Disk
FQprint    O    F  132        Printer Oflind(*Inof)

ICustomers NS
I                                  1    4  Accountid
I                                  5    8  Salesperson
I                                  9   43  Name
I                         *USA D  132  141  Lastsaledate
I                                142  152 2Ytdsales

 /Free

   Except Headings;
   Read Customers;

   Dow Not %Eof(Customers);

     If *Inof;
       Except Headings;
       *Inof = *Off;
     Endif;

     Except Detail;
     Read Customers;
   Enddo;

   *Inlr = *On;
   Return;

 /End-Free
```

continued...

continued...

```
OQprint    E           Headings       2  2
O                                          16 'YTD SALES REPORT'
O                                          56 'DATE'
O                      *Date         Y     67
O                                          74 'PAGE'
O                      Page          Z     79

O          E           Headings       1
O                                           4 'ACCT'
O                                          11 'SALES'
O                                          65 'YTD'
O                                          78 'DATE OF'

O          E           Headings       2
O                                           3 'ID'
O                                          12 'PERSON'
O                                          22 'CUSTOMER'
O                                          66 'SALES'
O                                          79 'LAST SALE'

O          E           Detail         1
O                      Accountid            4
O                      Salesperson         11
O                      Name                49
O                      Ytdsales      1     66
O                      Lastsaledate        79
```

To create this program, you would use an editor to enter the RPG IV code into a source member with an SEU member type of RPGLE. Once the source member contains all the required code, then you would compile the source using the CRTBNDRPG (Create Bound RPG Program) command to compile the source and create the program. Once the program is successfully created, you would execute it, using the CALL command.

Now that you have seen how to write a complete RPG IV program, we can return to the concept of output editing to learn RPG IV's editing features in greater detail.

Output Editing

Output editing refers to formatting output values by suppressing leading zeros and adding special characters—such as decimal points, commas, and dollar signs—to make the values easier for people looking at the output to comprehend. RPG IV allows numeric fields (but not character fields) to be edited as part of the Output specifications. You often will use editing to obtain the output format requested in a printer spacing chart.

Editing is used in part because of the way numbers are stored in the computer. For example, if Amount—a field six bytes long with two decimal positions—is assigned the value 31.24, the computer stores that value as 003124. Although the computer keeps track of the decimal position, a decimal point is not actually stored as part of the numeric value. If you were to specify that Amount be printed without editing, the number would be printed as 003124; the nonsignificant zeros would appear, and there would be no indication of where the decimal point should be.

Edit Codes

To make it easier to specify the most commonly needed kinds of editing, RPG IV includes several built-in edit codes you can use to indicate how you want a field's value to be printed. You associate an edit code with a field by entering the code in position 44 of the Output specification containing that field. All commonly used edit codes automatically result in **zero suppression**—that is, printing blanks in place of nonsignificant leading zeros—because that is a standard desired format.

Editing Numbers

Some editing decisions vary with the application. Do you want numbers to be printed with commas (or other appropriate grouping separator for your region) inserted? How do you want to handle negative values—ignore them and omit any sign, print CR immediately after a negative value, print a floating minus sign (-) after the value, or print a floating negative sign to the left of the value? And if a field has a value of zero, do you want to print a zero or leave that spot on the report blank? A set of 16 edit codes—1 through 4, A through D, and J through Q—cover all combinations of these three options: commas, sign handling, and zero balances. The following table details the effects of the 16 codes (in the shaded area).

Options and Edit Codes

Print commas	Print zero balance	No sign	CR	Right -	Floating -
Yes	Yes	1	A	J	N
Yes	No	2	B	K	O
No	Yes	3	C	L	P
No	No	4	D	M	Q

Thus, if you want commas, zero balances to print, and a floating negative sign, you would use edit code N; if you did not want commas or any sign but did want zero balances to print, you would use edit code 3.

To give you a clearer understanding of the effects of each of these edit codes, the following table demonstrates how various values would appear when printed with each of the edit codes. The position of the decimal place in the values is indicated by a caret (^). Notice that if you use edit codes 1–4 with a field containing a negative value, the field will be printed like a positive number.

Edit code	Value				
	1234^56	1234^56-	0234^56-	0000^00	000000^
1	1,234.56	1,234.56	234.56	.00	.00
2	1,234.56	1,234.56	234.56		
3	1234.56	1234.56	234.56	.00	.00
4	1234.56	1234.56	234.56		
A	1,234.56	1,234.56CR	234.56CR	.00	.00
B	1,234.56	1,234.56CR	234.56CR		
C	1234.56	1234.56CR	234.56CR	.00	.00
D	1234.56	1234.56CR	234.56CR		
J	1,234.56	1,234.56-	234.56-	.00	.00

continued…

continued...

Edit code	**Value**				
	1234^56	1234^56-	0234^56-	0000^00	000000^
K	1,234.56	1,234.56-	234.56-		
L	1234.56	1234.56-	234.56-	.00	.00
M	1234.56	1234.56-	234.56-		
N	1,234.56	-1,234.56	-234.56	.00	.00
O	1,234.56	-1,234.56	-234.56		
P	1234.56	-1234.56	-234.56	.00	.00
Q	1234.56	-1234.56	-234.56		

RPG provides three additional useful edit codes: X, Y, and Z. Edit code *Y* results in slashes being printed as part of a date. For example, if you run your program on December 11, 2009, the reserved field *Date will contain 12112009. If edited with edit code Y, this date will be printed as 12/11/2009. Although edit code Y is normally used to edit dates, you can also use it with any field for which slash insertion is appropriate.

Edit code *Z* simply zero suppresses leading nonsignificant zeros. Z does *not* enable the printing of a decimal point or a negative sign, so if a field contained a value of -234.56, the Z edit code would cause the field to be printed as 23456. The use of Z is usually limited to whole number fields.

With one exception, all the edit codes suppress leading zeros. Edit code *X*, however, retains them. For this reason, the X edit code is useful when you want to convert a numeric value to a character string and retain the leading zeros.

Currency Output Editing

You occasionally will want dollar signs (or other local currency symbols) to be printed as part of your report. As we mentioned in Chapter 1, you can position dollar signs in a fixed column of the report or you can place them just to the left of the first significant digit of the values with which they are associated. This latter type of dollar sign is called a *floating dollar sign*.

Fixed currency symbol	Floating currency symbol
$ 12.34	$12.34
$5,432.10	$5,432.10
$ .00	$.00

In general, you want to use a dollar sign in addition to one of the editing codes. To specify a floating dollar sign, code '$' (apostrophes included) in the constant/edit word positions (columns 53–80) of the Output specifications *on the same line* as the field and its edit code. To specify a fixed dollar sign, code '$' as a constant *on its own line* with its own end position.

You can use one additional feature along with edit codes. An asterisk coded in the constant/ edit word position on the same line as the field and edit code specifies that insignificant leading zeros be replaced by asterisks rather than simply being suppressed. This feature is called **asterisk fill** or sometimes **check protection** because its most common use is in printing checks—to prevent tampering with a check's face value. For example, a check worth $12.15 might include the amount written as $****12.15.

The following examples illustrate how to code the various currency output options.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
O..............NØ1NØ2NØ3Field+++++++++YB.End++PConstant/editword/DTformat++
  // The following line illustrates a floating dollar sign.
O                       Amount        1     65 '$'
  // The next two lines illustrate a fixed dollar sign.
O                                           56 '$'
O                       Amount        1     65
  // The following line illustrates asterisk fill.
O                       Amount        1     65 '*'
  // The next two lines combine a fixed dollar sign with asterisk fill.
O                                           56 '$'
O                       Amount        1     65 '*'
```

Edit Words

You would think that given the variety of edit codes built into RPG IV, you would be able to find a code to fit your every need. Unfortunately, that is not the case. Social Security and telephone numbers represent good examples of values that we are used to seeing in a format that an edit code cannot supply. RPG IV includes an alternative to edit codes, called **edit words**, that can help in this kind of situation.

You code an edit word in the constant/edit word portion of the Output specifications on the same line as the field with which it is to be used. Note that because they perform the same function, edit words and edit codes are never used together for the same field. An edit word supplies a template into which a number is inserted. The template is enclosed with apostrophes. Within the template, a blank position indicates where a digit should appear, while a 0 indicates how far zero suppression should take place. With no zero in the edit word, the default is to zero suppress to the first significant digit.

You can use any character—including commas—as an insertion character within the template. The insertion characters will be printed in the specified place as long as they are to the right of a significant digit. A dollar sign at the left of the edit word signals a fixed dollar sign; a dollar sign adjacent to a zero denotes a floating dollar sign. To indicate a blank as an insertion character, use an ampersand (&).

Examine the table below to see how edit words work.

Raw value	Edit word	Printed result
999999999	‘ - - ’	999-99-9999
999999999	‘ & & ’	999 99 9999
1234123412	‘0( ) - ’	(123)412-3412
00012^14	‘ $0. ’	$12.14
00012^14	‘$ 0. ’	$ 12.14
00012^14	‘ *0. ’	***12.14
05678^90	‘$, 0. ’	$ 5,678.90
05678^90-	‘ , $0. CR’	$5,678.90CR
05678^90-	‘ , $0. -’	$5,678.90-
05678^90	‘ , **DOLLARS* *CENTS’	*5,678*DOLLARS*90*CENTS

You can duplicate the effects of any edit code with an edit word. In general, RPG programmers use edit words only when there is no edit code that provides the format they want for their output.

Chapter Summary

RPG IV programs are written as fixed-format or free-format specifications. Different specification forms convey different kinds of information to the RPG IV compiler, which translates the program into machine language.

File specifications contain descriptions of all files used within a program. Input specifications provide detailed information about each program-described input file used by a program. There are two kinds of Input specification lines: one that contains record identification entries to generally describe a record format within a file, and one that contains field identification entries to define the fields of the record. Each field is described on a separate line.

Calculation specifications center on operations, or processing steps, to be accomplished by the computer. Each Calculation specification includes an RPG IV operation either by coding it explicitly (or, in the case of the Eval operation, implicitly) in an expression. Depending on the specific operation, it may also include additional entries. The computer executes operations in the order specified on the Calculation specifications unless the computer encounters an operation that specifically alters this flow of control.

Output specifications provide details about each program-described output file. You use two kinds of Output specification lines: a record identification line to describe an output record format at a general level and field description lines to describe each field or constant that appears as part of a record format. When the output is a report, you need a record identification line and corresponding field-identification entries for each kind of line to appear on the report.

An important part of programming is documenting the program. Comment lines—signaled by double slashes (//) in nearly any position of a specification line—can appear anywhere within a program. Most comments are coded on a separate line, but in free-format specifications, they may be included on the same line as executable RPG statements but positioned after the RPG statements. The RPG IV compiler ignores such comments.

Within your code, you can insert completely blank lines and lines that are blank except for the specification type to visually break the code into sections.

It is customary to edit printed numeric values. RPG IV supplies ready-made edit codes for common editing requirements and lets you create special editing formats by using edit words.

The following table summarizes the operation codes and built-in functions discussed in this chapter. Optional entries are shown within curly braces ({}).

Function or operation	Description	Syntax
Dow	Do while	Dow *logical-expression*
Enddo Endif	End a structured group	End*xx*
%Eof	End of file	%Eof{(*file-name*)}
Eval	Evaluate expression	{Eval} *result* = *expression*
Except	Calculation time output	Except {*name*}
If	If	If *logical-expression*
Read	Read a record	Read *file-name*
Return	Return to caller	Return

Key Terms

arguments
asterisk fill
batch processing
built-in functions
Calculation specifications
case sensitive
character field
check protection
comment lines
compiler directives
constants
continuation lines
Control specifications
detail line
double slashes (//)
Dow (Do While)
edit code
edit words
end position
Enddo (End Do Group)
Endif
end-of-file
Eval
Except
exception lines
exception names
factors
field-level
field description entries
File specifications/F-specs
fixed format
fixed position
free form
full procedural
If
indicator
*Inlr
Input specifications
keywords
Last Record
literals
native date fields
numeric fields
operand values
output editing
output file
Output specifications
overflow
overflow indicator
Page
priming read
pseudocode
Read
record formats
record identification entries
reserved word
Return
sequence checking
Skip entries
Space entries
title case
zero suppression

Discussion/Review Questions

1. What is a fixed-format language? Can you give an example of a free-format language? Which form offers the most advantages?
2. Why do reports generated by RPG IV programs need to appear on File specifications?
3. Why don't you need to enter a File Designation for output files?

X	1STQTR	#3
ABC	QTY-OH	CustNo
@end	SALES	$AMT
_YTD_Sales	CUST#	Day1
YR END	YR_END	Yearend
InvoiceNumber	avg.sales	cusTnbR

4. Which of the following are invalid RPG IV variable names? Why?
5. What is an indicator? What specific methods of turning on indicators were introduced in this chapter? How can you use indicators to control processing? What alternative RPG IV feature can be used to reduce or eliminate indicators in a program?
6. Describe the difference between a Skip entry and a Space entry on the Output specifications.
7. How could you obtain five blank lines between detail lines of a report?
8. What is the advantage of giving the same name to several exception lines of output?
9. What are some fields that are automatically provided by RPG IV for your use?
10. Why do you often need two read statements within a program?
11. What is the correct order of specifications within an RPG IV program?
12. What is the purpose of each kind of RPG IV specification introduced in this chapter?
13. What is LR? Why is it used?
14. What is maintenance programming? And what programming techniques can you adopt to facilitate it?
15. Why does RPG IV include both edit codes and edit words? What exceptions are there to the rule that an edit code and an edit word or constant should never appear together on the same Output specification line?
16. What are the programming implications of the fact that RPG IV is not case sensitive?
17. Describe internal and external documentation. Why is there so much importance placed on correctly documenting a program?
18. Research Control specifications. What are some of the advantages of using them? Are there disadvantages?

Exercises

1. A Customer listing program uses data from file Customers to generate a report that reflects all the data in the file. The record layout of file Customers follows:

Description	Positions	(Decimal positions)	Notes
Customer number	1–5	(0)	—
Customer name	6–25	—	—
Last order date	26–33	—	*mmddyyyy*
Balance owed	34–43	(2)	—

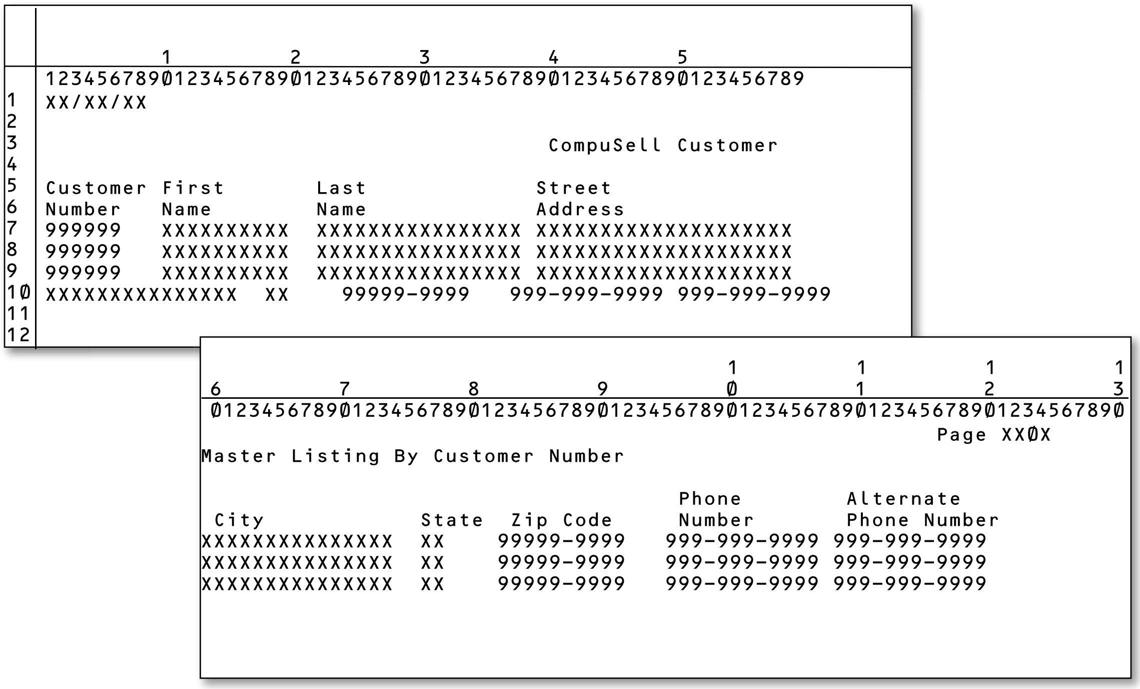
   Write the File specifications for this program.
2. Given the above problem definition, write the Input specifications.
3. Write the pseudocode to produce the report.
4. Design a report for the application in Exercise 1, using the printer spacing chart notation of Chapter 1.
5. Develop Output specifications based on your printer spacing chart from Exercise 4 and the File specifications of Exercise 1.

Programming Assignments

All five of the programming assignments below center on a single company, CompuSell. CompuSell is a mail-order company specializing in computers and computer supplies. Appendix F provides a description of the company and the record layouts of its data files.

1. CompuSell would like you to write a program to produce a listing of all its customers. Use data file CSCSTP, the customer master file for CompuSell, as your input file. The listing should exactly match the format described in the following printer spacing chart.

Note: the grid has been split to allow the report to be formatted for the textbook.

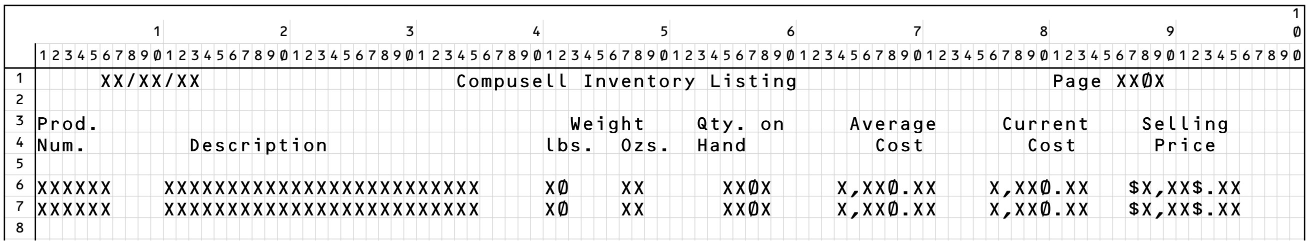


```
            1         2         3         4         5
   123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789
1  XX/XX/XX
2
3                                         CompuSell Customer
4
5  Customer First       Last             Street
6  Number   Name        Name             Address
7  999999   XXXXXXXXXX  XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
8  999999   XXXXXXXXXX  XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
9  999999   XXXXXXXXXX  XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
1Ø XXXXXXXXXXXXXXX  XX    99999-9999   999-999-9999 999-999-9999
11
12
```

```
                                         1         1         1         1
 6         7         8         9         Ø         1         2         3
 Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø
                                                         Page XXØX
Master Listing By Customer Number

                                     Phone        Alternate
 City            State  Zip Code     Number       Phone Number
XXXXXXXXXXXXXXX  XX    99999-9999   999-999-9999 999-999-9999
XXXXXXXXXXXXXXX  XX    99999-9999   999-999-9999 999-999-9999
XXXXXXXXXXXXXXX  XX    99999-9999   999-999-9999 999-999-9999
```

2. CompuSell wants an inventory listing, formatted as shown in the following printer spacing chart. Write the program to produce this report, exactly matching the printer spacing chart specifications. The input file is CSINVP; its record layout is given in Appendix F.



```
                                                                                                     1
            1         2         3         4         5         6         7         8         9         Ø
  1234567890123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø1234567890
1      XX/XX/XX                    Compusell Inventory Listing                    Page XXØX
2
3 Prod.                                     Weight    Qty. on     Average     Current     Selling
4 Num.        Description                 lbs.  Ozs.  Hand          Cost        Cost       Price
5
6 XXXXXX    XXXXXXXXXXXXXXXXXXXXXXXXX     XØ    XX      XXØX    X,XXØ.XX    X,XXØ.XX   $X,XX$.XX
7 XXXXXX    XXXXXXXXXXXXXXXXXXXXXXXXX     XØ    XX      XXØX    X,XXØ.XX    X,XXØ.XX   $X,XX$.XX
8
```

3. CompuSell wants to send out two separate mailings to each of its customers contained in file CSCSTP (see Appendix F for the record layout). Accordingly, the company asks you to write a label-printing program that will print two-across labels. Each of the labels reading across should represent the same customer. The printer will be loaded with continuous label stock when this program is run. Each label is five print lines long. The desired format for the labels is shown below. Note that the information in the parentheses is included to let you know what should appear on the label; it should not appear within your output.

```
                                                                                                       1
            1         2         3         4         5         6         7         8         9         Ø
   123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø
 1      XXXXXXXXXX XXXXXXXXXXXXXXX             XXXXXXXXXX XXXXXXXXXXXXXXX        (first, last name)
 2      XXXXXXXXXXXXXXXXXXXX                   XXXXXXXXXXXXXXXXXXXX              (street address)
 3      XXXXXXXXXXXXXXX  XX XXXXX-XXXX         XXXXXXXXXXXXXXX  XX XXXXX-XXXX    (city, state, zip)
 4
 5
 6      XXXXXXXXXX XXXXXXXXXXXXXXX             XXXXXXXXXX XXXXXXXXXXXXXXX
 7      XXXXXXXXXXXXXXXXXXXX                   XXXXXXXXXXXXXXXXXXXX
 8      XXXXXXXXXXXXXXX  XX XXXXX-XXXX         XXXXXXXXXXXXXXX  XX XXXXX-XXXX
 9
1Ø
11
12
```

4. CompuSell wants a telephone and address listing of all its suppliers. Write a program to produce this listing. Your input file, CSSUPP, is described in Appendix F.

```
                                                                                                       1
            1         2         3         4         5         6         7         8         9         Ø
   123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø
 1        Compusell Supplier List as of XX/XX/XX               Page XXØX
 2
 3   Name/Address                    Phone             Contact Person
 4
 5 XXXXXXXXXXXXXXXXXXXXXXXXX     (XXX) XXX-XXXX   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 6 XXXXXXXXXXXXXXXXXXXX
 7 XXXXXXXXXXXXXXX XX XXXXX-XXXX
 8
 9
1Ø XXXXXXXXXXXXXXXXXXXXXXXXX     (XXX) XXX-XXXX   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
11 XXXXXXXXXXXXXXXXXXXX
12 XXXXXXXXXXXXXXX XX XXXXX-XXXX
13
14
15 XXXXXXXXXXXXXXXXXXXXXXXXX     (XXX) XXX-XXXX   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
16 XXXXXXXXXXXXXXXXXXXX
17 XXXXXXXXXXXXXXX XX XXXXX-XXXX
18
```

5. CompuSell wants a listing of all its employees. Write a program to produce this listing. Your input file, CSSEMP, is described in Appendix F.

```
                                                                                                       1
            1         2         3         4         5         6         7         8         9         Ø
   123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø123456789Ø
 1 XX/XX/XX                                                                                   Page XXØX
 2                                      CompuSell Employee Listing By Employee Number
 3
 4 Employee First       Last             Street                                                Phone
 5 Number   Name        Name             Address              City            St  Zip Code     Number
 6 999999  XXXXXXXXXX  XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XX 99999-9999 999-999-9999
 7 999999  XXXXXXXXXX  XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XX 99999-9999 999-999-9999
 8 999999  XXXXXXXXXX  XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX XX 99999-9999 999-999-9999
 9
1Ø
11
```