Records and Files

Learning Outcomes

After reading this section, you will be able to:

- Stream data using standard library functions to access persistent text

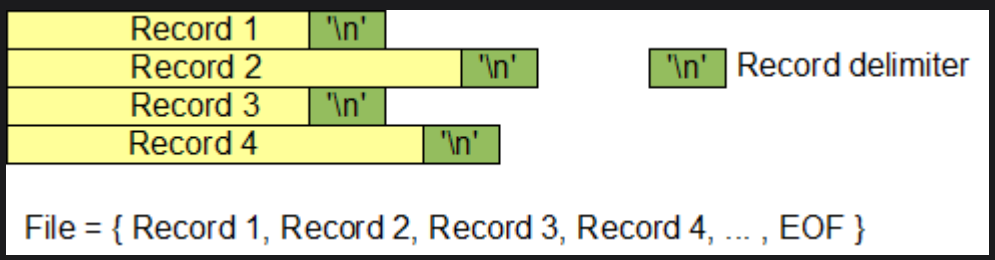
Introduction

Persistent data hierarchies typically consist of databases composed of files, which consist of records, which consist of fields, which consist of bytes, which are stored in bits. In files that hold a tabular structure, each record contains the same number of fields.

This chapter describes how to identify records and fields in a text file and how to retrieve tabular data.

Records

A **record** occupies a single line in a text file and holds all of the data associated with one chunk of information. The record is a sequence of characters that ends with a record delimiter. The typical record delimiter is the newline character (`\n`).



Consider a text file named `produce.txt` containing information about items of produce in a grocery store. Each record consists of the **SKU** for a product and its **unit price**.

```
4664 1.49
4419 1.29
```

```
4011 0.59
```

To determine the number of records in this file, we count the number of newline (`'\n'`) characters:

```c
// Number of Records
// records.c

#include <stdio.h>

int main(void)
{
    FILE *fp = NULL;
    int c, nrecs;

    fp = fopen("produce.txt", "r");

    if (fp != NULL)
    {
        nrecs = 0;
        do {
            c = fgetc(fp);

            if (c != EOF)
            {
                if ((char)c == '\n')
                nrecs++;
            }
        } while (feof(fp) == 0);

        printf("%d records on file\n", nrecs);
        fclose(fp);
    }

    return 0;
}
```

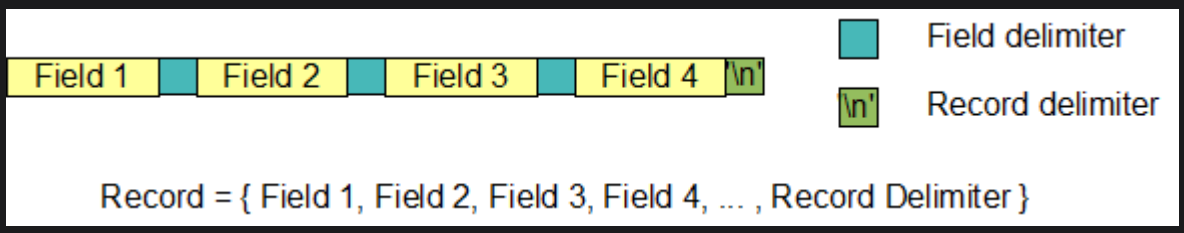
The above program produces the following output:

```
3 records on file
```

> **NOTE**
>
> Since this program determines the number of records in the file by counting the newline characters, to report the correct number of records, the last record in the file must end with a newline character. If the last record does not end with a newline character, the count will be off by one.

Fields

A **field** holds one element of information within a single record. We separate adjacent fields within a record by a **field delimiter**.



Consider the file named `produce.txt` (see above). Each record contains two fields: the first field holds the **SKU** and the second field holds the **unit price**. The field delimiter is a blank character.

The following program reads the fields of each record in the file and displays their contents:

```c
// Record and Fields
// recordFields.c

#include <stdio.h>

int main(void)
{
    FILE *fp = NULL;
    int sku;
    double price;

    fp = fopen("produce.txt", "r");

    if (fp != NULL)
    {
        printf(" Produce Items\n"
        " =============\n\n"
```

```
        "SKU       Price\n"
        "--------------\n");

        while (fscanf(fp,"%d%lf\n", &sku, &price) == 2)
        {
            printf("%4d %10.2lf\n", sku, price);
        }

        fclose(fp);
    }

    return 0;
}
```

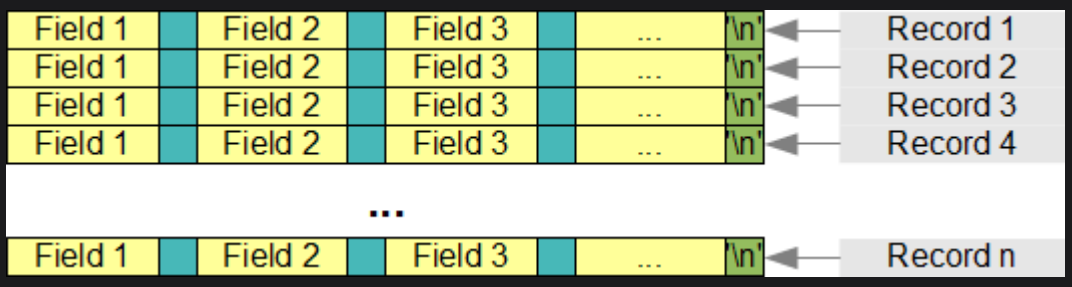
The above program produces the following output:

```
 Produce Items
 =============

SKU       Price
--------------
4664       1.49
4419       1.29
4011       0.59
```

Tables

A **table** is a set of records in which each record contains the same number of fields.



DANGER

> If one of the fields in a record is a character field, the blank character might not be suitable as a field delimiter and we select a special character for that purpose.

Consider the file named `sale.txt` (its contents are listed below). Each record in this file contains three fields:

1. **SKU**
2. **price status** (a single character where a blank character represents the regular price and `*` represents a sale)
3. **unit price**

The field delimiter is the semi-colon character (`;`):

The following program reads each record from the file and displays the fields in a tabular format:

```
// Tabular Data
// table.c

#include <stdio.h>

int main(void)
{
    FILE *fp = NULL;
    int sku;
    char status;
    double price;

    fp = fopen("sale.txt","r");

    if (fp != NULL)
    {
        printf(" Produce Items\n"
        " =============\n\n"
        "SKU  Sale  Price\n"
        "----------------\n");

        while (fscanf(fp, "%d;%c;%lf", &sku, &status, &price) == 3)
        {
            printf("%4d %c %8.2lf\n", sku, status, price);
        }
```

```
        fclose(fp);
    }

    return 0;
}
```

The above program produces the following output:

```
 Produce Items
 =============

SKU  Sale  Price
---------------
4664 *      1.49
4419 *      1.29
4011        0.59
```

> **NOTE**
>
> We have included the field delimiters within `fscanf()`'s format string and these delimiter characters are discarded and not assigned to any variables.