CHAPTER 3

Data Representation

A computer system provides an environment for the storage, retrieval, and manipulation of data. The data can be viewed differently at the various levels of abstractions. At the lowest level—the hardware and operating system—data is nothing more than a sequence of bits (or bytes). People, however, prefer to view the data not as a sequence of bytes, but as an abstraction of what the data represents.

CHAPTER OBJECTIVES

- To provide an overview of what the nature of digital data.
- To introduce the concepts of elementary data types — those supported by a programming language, and non-elementary data types — those that can not be defined directly within a programming language.
- To introduce the notion of metadata — data about data.
- To introduce the concept of a file.

3.1 The Nature of Data

A computer system, at any point in time, consists of a set of programs and a set of **data items**. A data item is a piece of information that represents either a real-life object or an imaginary object. An integer (e.g., "28") is an example of a data item, and so is an array of integers. Additional examples are a person name (e.g., "Jack"), a car model (e.g, "Ferrari"), and the age of person (e.g, "18").

If your are presented with a data item, say "18", with no additional information, then you have no way of interpreting what the meaning of the data is. That is, the number "18" may be today's temperature as well as the horse power of a car. Similarly, the name "Jack" could be the name of a person or an electrical connector. If the meaning of a particular data item cannot be

easily inferred then it has to be explicitly added either manually by a person, or automatically by a computer system.

Data with an agreed upon meaning is called **information**. The meaning or "semantic" (Greek σεμαντικωσ = belongs to the symbol, significant, or meaningful) of the data can also be expressed as "data about data", called **metadata** (see Section 3.5).

As a side note, the US Federal Standard 1037C (http://www.its.bldrdoc.gov/fs-1037) defines data as "any Representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or by automatic means. Any representations such as characters or analog quantities to which meaning is or might be assigned".

3.1.1 Data Type

Each data item has an associated type, called **data type**, which defines the structure of the data item and the type of operations one may apply on that type. For example, when you declare in C:

```
int A, B
```

you are creating two variables (data items), `A` and `B`, which are of type integer. The only meaningful operations one can apply to integer type data are the arithmetic operations and the assignment of integer constants.

A programming language provides the support for the creation of a number of different data types. As we shall see shortly, the language C allows one to create data types that represent real numbers, single characters, character strings, etc.

An integer variable is an example of a simple data item. Simple data items can be combined to create more complex data items. An array of integers is an example of a more complex data item. As we shall see later, most programming languages allow one to create complex data types.

Some types of data can be created directly within your computer program. That is, the programming language provide the support for the creation of these types of data. For example, the language C (as most other programming language) provides support for the creation of integers type variables (as you have seen in Chapter 2). We refer to these type of data as **elementary**. There are many other data types that the language C support, which we cover piecemeal throughout the text.

Some types of data, however, cannot be created directly within your your computer program. That is, the programming language does not provide a mechanism for the creation of data of that type. An example of such type of data item is JPEG. .... The language C (as most other programming language) provides no support for the creation of data items of type JPEG. Instead, ... We refer to these type of data as **non-elementary**.

3.1.2 Defining new data types

A programming language may allow one to define new data types out of existing data types. For example, one may want to create a new data type that represents a student (let us refer to this new data type as `student`). Such a data type will consists of a number of different fields, including student name,

address, phone number, GPA. We shall see in Section 3.2.3 how one can define such a new data type in the C language.

Once a new data type is defined, one can use that new type in the same way one uses other existing data types. That is, one can create any number of variables of type "student". One can even create an array of students!

3.2 Elementary Data Types

Some of the data types are directly supported by the programming language you use to write programs. In Chapter 2 we introduced two elementary data types—integers and arrays. We now present a few more of the elementary data types supported by the C programming language.

3.2.1 Characters

One very useful data type is the "character", which can take as a values a single alphanumeric value (e.g., "A", "=", ";").

A character variable, say `a`, can be defined in the language C as follows:

```
char a;
```

One can define several character variables using the short hand notation:

```
char a, b, c;
```

The input and output commands for character types are quite similar to those for integer; the only difference is that we use "%c" instead of "%d".

To assign a character, say "?", to a variable `c`, we write:

```
c = '?';
```

```
#include <stdio.h>
main()
{

     int sum;
     char c;

     sum = 0;
     scanf("%c", &c);
     while (c != '\n');
        {sum = sum + 1;
         scanf("%c", &c); }
     printf("%d\n", sum);

}
```

Figure 3.1 C program to input a sequence of characters and print the length of the string.

```
#include <stdio.h>
main()
{

    int i;
    char phone[10];

    i = 0;
    while (i < 10)
        { scanf("%c", &phone[i]); i = i + 1;};
    printf(" ");
    i = 0;
    while (i < 10)
        { printf("%c", phone[i]); i = i + 1;};
    printf("\n");

}
```

Figure 3.2 C program to input a phone number and print it out.

We can now illustrate the use of "char" type variables by writing a a simple C program (Figure 3.1) that inputs a sequence of characters and prints out the total number of character read. Remember that the "end-of-line character" is designated by "\n".

Just as we defined an array of integers, we can define an array of characters. A character array, say `b`, of size "10" is defined in the C language as follows:

```
char b[10];
```

The ability to declare an array of characters allows us to define fixed-length **strings**. For example, suppose that we want to input a 10-digits phone number and store that number in an array of characters "phone". The program to accomplish this is depicted in Figure 3.2

3.2.2 Real Numbers

It is useful to be able to do calculations with not only integers but also real-numbers (e.g., 1.76). The language C provides the data type "float", which can take as a value a real-number. A variable, say `R`, can be defined to be of type `float` as follows:

```
float R;
```

The input and output commands for float types are quite similar to those for integer and character; the only difference is that we use "%f" instead of "%d" (or "%c").

3.2.3 Defining new types

One important aspect of an expressive programming language is the ability to define new data types from currently available data types.

```
#include <stdio.h>
main()
{

     struct point  float x, y;;

     /* declares two variable P1 and P2 of type point */
     struct point P1, P2;

     printf("Please input the first point\n");
     scanf("%f %f", &P1.x, &P1.y);
     printf("Please input the second point\n");
     scanf("%f %f", &P2.x, &P2.y);
     if ((P1.x == P2.x) || (P1.y == P2.y))
         printf("on the same line\n");
         else printf("not on the same line\n");

}
```

Figure 3.3 C program to input two points and decide if they are on the same line.

Suppose we want to define a new data type "point", representing a single point in a two dimensional space (plan) (with coordinates x and y). A point then is simply a two tuple (a,b), where "a" is the location in the x coordinate and "b" is the location in the y coordinate.

The definition of this new data type can be done in the language C via the **struct** construct.

```
struct point {float x, y;};
```

Once the new data type `point` has been defined, we can declare a variable P of that type by using:

```
struct point P;
```

To refer to an individual element in P, say x, we use the "dot" notation, as in:

```
P.x
```

We can now write a simple program (see Figure 3.3) that inputs two points and decides whether they are on the same line in a two dimensional space (horizontally or vertically).

As another more substantive example, suppose that we want to create a small "database" consisting of records of all students in a a department. Each record consists of three fields:

- `name`
- `ID`
- `phone`

```
#include <stdio.h>
main()
{

    int i,j,id;
    struct person {char name[20]; int ID; char phone[10];};
    struct person students[120];

    /* initialize the students strcuture */

    scanf("%d", &id);
    j = 0;
    while (j < 120)
        { if ( students[j].ID == id)
            { i = 0;
              while (i < 20)
                  { printf("%c", students[j].name[i]);
                    i = i + 1;};
              printf(" ");
              i = 0;
              while (i < 10)
                  { printf("%c", students[j].phone[i]);
                    i = i + 1;};
            }
        j = j + 1;
        }
    printf("\n");

}
```

Figure 3.4 C program to input a student ID and print out the student's phone number.

where name and phone are character strings and ID is is an integer.

Each student record is defined as a C-type "struct", which has the following form:

```
struct person {char name[20], int ID, char phone[10];};
```

Let's assume that the maximum number of students we want to keep track of is 120. We then define an array "students" of person-type records as follows:

```
struct person students[120];
```

Next, we write a C program that takes as an input a student ID and prints out the student's phone number. In Figure 3.4, we present that program without bothering to show the initialization code for the dept array (in a more realistic situation that array will be stored somewhere permanently on a computer system and be read from there; more on this later).

3.3 Non-elementary Data Types

There are some data types that are not directly supported by a programming language. An example is a data item of type PDF. Such a data item must be created by a special program – the Adobe ...

- JPEG – for images
- PDF – for documents
- spreadsheet
- Word file
- Web page
- programs (both source code and object code).

Give example Web pages – HTML.

3.4 Realtime Data Types

Although most of the data in a computer system is "conventional data" (e.g, text documents, computer programs, word-processing documents, spreadsheets), a recent trend in technology is the incorporation of **realtime data** (video and audio) into computer systems. These data differ from conventional data in that realtime data—such as frames of video—must be delivered (streamed) according to certain time restrictions (for example, 30 frames per second).

There are a wide range of applications based on realtime data that are in popular use today. These include audio files such as MP3, DVD movies, video conferencing, and short video clips of movie previews or news stories downloaded over the Internet. Other applications may also include live webcasts (broadcasting over the World Wide Web) of speeches or sporting events and even live webcams that allow a viewer in Manhattan to observe customers at a cafe in Paris.

These applications need not be either audio or video; rather, a realtime application often includes a combination of both. For example, a movie may consist of separate audio and video tracks. Realtime data need not be delivered only to desktop personal computers. Increasingly, they are being directed toward smaller devices, including PDAs and cellular telephones. For example, a stock trader may have stock quotes delivered wirelessly and in real time to his PDA.

Some of the realtime data types in use today:

- mpeg – for video
- MP3 – for audio

3.5 Meta Data

As we stated in Section 3.1, the "semantic" (or meaning) of the data can be expressed as "data about data", called **metadata**. Metadata provides the means for both programmers and computer programs to figuring out what a data item represents.

The metadata for a particular data item can be generated in a number of different ways:

- **By the programmer**. XML, database scheme, etc
- **By the computer program generating the data item**. PDF, JPEG

The metadata may be either stored together with the actual data, or separately from the data in a special repository with reference to the corresponding data item.

You can think about the "comments" you provide in the program you write as a rudimentary form of metadata.

As an example, consider today's markup languages used in web-applications, where the metadata is stored as *tags* (annotations) with each occurrence of the data. In Figure 3.5 we present an example of an XML fragment describing a book's information.

As another example pertaining to those readers who are using a Windows operating system. Find an object on your desktop (say a JPEG image or a Powerpoint document) and right click on that object. You will see a drop window with bottom entry termed "properties". When you click on "properties" you will get the metadata information about that object. For a JPEG-type object you get the following information:

- **Type of object**. A JPEG image.
- **Location**. Where the object is stored on your computer.
- **Size**. How big is the object (in bytes).
- **Creation date**. When was the object created.
- **Modification data**. When was the object last modified.

```
<book>
    <authors>
        <name>Silberschatz</name>
        <name>Korth</name>
        <name>Sudarshan</name>
    </authors>
    <title>Database System Concepts</title>
    <publisher>McGraw-Hill</publisher>
    <ISBN>0-07-295886-3</ISBN>
</book>
```

Figure 3.5 XML fragment describing a book's information.

- **Access date**. When was the object last accessed.

[[[Give an example of database scheme??]]] <==

3.6 Files

Between the low-level byte representation of data and the high-level (e.g., spreadsheet) representation of data lies an intermediate level—the **file**.

A **file** is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields). A file is by far the most visible datum accessible to the user without writing programs.

A file is a **permanent** object; it exist on your computer (or other computers) until it is explicitly deleted. A file is usually stored on some **secondary storage** device. It cannot reside in main memory since main memory is usually too small to store all your files, and more importantly, main memory is a *volatile* storage device that loses its contents when power is turned off or when the system crashes.

The most common secondary-storage device is a **magnetic disk**, which provides the bulk of the storage medium in modern computer systems. Disk storage is discussed in Chapter 10.

All data (and remember that program is treated as data) is stored as a collection of files managed by the operating file-system. The file system provides a uniform, logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define the logical storage unit—the "file". The operating system maps files onto physical media and accesses these files via the storage devices.

The operating system implements the abstract concept of a file by managing mass storage (usually disks), and the devices that control them. Also, files are normally organized into directories (folders) to make them easier to use. You can think of a directory as a "metadata" for the collection of files stored in that directory.

To illustrate, consider again the Windows operating system. Find a folder on your desktop and right click on that folder. You will see a drop window with bottom entry termed "properties". When you click on "properties" you will get the metadata information about that folder (and the collection of files/objects it contains), including:

- **Type**. File folder.
- **Location**. Where the folder is stored on your computer.
- **Size**. How big is the folder (in bytes).
- **Contains**. How many files does folder contain.
- **Created**. When was the folder created.

```
#include <stdio.h>
#include <stdlib.h>

main()
{

     FILE *in, *out;

     in = fopen("infile","r");
     out = fopen("outfile","w");

     int a, sum;

     sum = 0;
     fscanf(in, "%d", &a);
     while (a != -1)
        { sum = sum + a;
          fscanf(in, "%d", &a); }
     fprintf(out, "%d\n", sum);

}
```

Figure 3.6 Program to read a set of integers from "infile" and place the sum in "outfile".

Please not that folders (directories) may be nested. That is, a folder may contain not only files but also some other folders.

Finally, in an an environment where multiple users can have access to a file, it may be desirable to control by whom and in what ways (for example, read) that file may be accessed. You can see this access control information by clicking on the "properties" of a file.

==

[[[simultaneous access to a file – refer to chapter in concurrency]]] .

Now that we have a firm grip on what a file is let us us contrast the data residing in a file with data residing in a computer program that you write. The data in a computer program is transient (temporary). It exists for a short period of time and then disappears. For example, in the programs depicted in Chapter 2, all the integer variables are transient—they exist for the duration of the execution of the program. Looking more closely at the sorting program shown in Figure 2.14, the `final` integer array has to be recomputed whenever you need the sorted data. The reason being that once the program completes the `final` array simply disappears. To avoid this, we would have the keep the sorted array in a file, which is kept on your computer until you no longer need it.

The C programming language has mechanisms to allow one to read and write data from files rather than from the keyboard and screen. We will illustrate some of these mechanisms by creating a simple program that inputs a set of positive integers from a file called "infile", sums the integer up, and places the sum in a file "output". The last element in the file is set to the value "-1" to designate the last element in the file. The program to accomplish this is depicted in Figure 3.6.

Let us explain what the various statements in the program do. ....

3.7 View of Data

By now you have a good understanding on what type of data a computer system supports and the type of operations one can apply to the data. We are ready to move on and discuss how the data is actually handled by the various computer components.

Recall that at the the lowest level—the hardware and operating system—data is nothing more than a sequence of bits (or bytes or words). When one looks at a 32-bit word, one cannot, in general, tell if it is data or or an instruction. Even if you were told that this is data, you will not be able to tell whether this is an integer, a real number, a character string, etc.

So the main question is—how does one figure out how to interpret what a sequence of bits represents? The answer is quite simple—the interpretation is done by the program (or hardware) that accesses the data.

For example, when the CPU executes an instruction, which is a 32-bit word, it uses its own decoding mechanism to figure out what the "operand" is (e.g., load), and what the address is. The CPU does not "question" if this is really a valid instruction. It simply executes whatever is being handed to it!! If it handed an integer (or a 4-byte character string) to execute, it will do so as long as the decoding yields a valid operand and address!!

The same holds for any other type of data. For example, If a program expects an input of type integer and it is supplied a character instead, the executions will interpret the character read to be an integer! To illustrate, consider again the simple program depicted in Figure 2.8. Try executing it with input "A". You will see that the the character "A" is being viewed as some integer value and printed out accordingly.

The same holds for output. If you wrote a computer program that outputs an integer, the 32-bit representing the integer will be displayed as an integer (rather the 4-byte alphanumeric characters). Similarly, if your Adobe system generates a PDF document ....

3.8 Other Issues

- Data Independence,
- Seperation of data and code.
- use SQL for declarative languages
- Seperation of policy and Mechanishm
- Where do we talk about DB tables?

3.9 Data Analysis

The term **data mining** refers loosely to the process of semiautomatically analyzing large databases to find useful patterns. Like knowledge discovery in artificial intelligence (also called **machine learning**) or statistical analysis, data mining attempts to discover rules and patterns from data. However, data mining differs from machine learning and statistics in that it deals with large

volumes of data, stored primarily on disk. That is, data mining deals with "knowledge discovery in databases."

Some types of knowledge discovered from a database can be represented by a set of **rules**. The following is an example of a rule, stated informally: "Young women with annual incomes greater than $50,000 are the most likely people to buy small sports cars." Of course such rules are not universally true, but rather have degrees of "support" and "confidence." Other types of knowledge are represented by equations relating different variables to each other, or by other mechanisms for predicting outcomes when the values of some variables are known.

There are a variety of possible types of patterns that may be useful, and different techniques are used to find different types of patterns. In Chapter xxx we study a few examples of patterns and see how they may be automatically derived from a database.

Usually there is a manual component to data mining, consisting of preprocessing data to a form acceptable to the algorithms, and postprocessing of discovered
patterns to find novel ones that could be useful. There may also be more than one type of pattern that can be discovered from a given database, and manual interaction may be needed to pick useful types of patterns. For this reason, data mining is really a semiautomatic process in real life. However, in our description we concentrate on the automatic aspect of mining.

Businesses have begun to exploit the burgeoning data online to make better decisions about their activities, such as what items to stock and how best to target customers to increase sales. Many of their queries are rather complicated, however, and certain types of information cannot be extracted even by using SQL.

Several techniques and tools are available to help with decision support. Several tools for data analysis allow analysts to view data in different ways. Other analysis tools precompute summaries of very large amounts of data, in order to give fast responses to queries. The SQLNN standard now contains additional constructs to support data analysis.

Textual data, too, has grown explosively. Textual data is unstructured, unlike the rigidly structured data in relational databases. Querying of unstructured textual data is referred to as *information retrieval*. Information retrieval systems have much in common with database systems—in particular, the storage and retrieval of data on secondary storage. However, the emphasis in the field of information systems is different from that in database systems, concentrating on issues such as querying based on keywords; the relevance of documents to the query; and the analysis, classification, and indexing of documents.

3.10 Summary

Exercises

3.1 Write a program to input two points and compute the distance between them.

3.2 Write a program to find out the phone number of a person named "Fritz". If the person is not in the database state so; If the person is in the database but has no phone state so.

Bibliographical Notes

Computer Science

- **Data redundancy and inconsistency**. Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In addition, it may lead to **data inconsistency**; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.
- **Integrity problems**. The data values stored in the database must satisfy certain types of **consistency constraints**. For example, the balance of certain types of bank accounts may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

The data values stored in the database must satisfy certain **consistency constraints**. For example, suppose the balance on an account should not fall below $100. The DDL provides facilities to specify such constraints. The database systems check these constraints every time the database is updated. In general, a constraint can be an arbitrary predicate pertaining to the database. However, arbitrary predicates may be costly to test. Thus, database systems concentrate on integrity constraints that can be tested with minimal overhead:

Domain Constraints. A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take. Domain constraints are the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database.

The system/application programs above the OS provide a structure and interpret the data.

We view data differently at the various levels of abstractions

bits, records, information, etc.

It is all about data.

bits, records, information, etc.

A program one writes, which usually resides in a file, is a data item; it is called an object code; The compiled version of that program is also residing in a file, and it called a source code.

source code (program) – represented by a particular programming language.

Object -code – binary

All data in the computer system is stored as a sequence of bytes.

A disk consists, from a logical point of view, of a an array of bytes numbered from 0 to N.

At the OS level we usually look at data as nothing more than a sequence of bytes.

The system/application programs above the OS provide a structure and interpret the data.

We view data differently at the various levels of abstractions

bits, records, information, etc.

need to emphasize the differences between a data item and where/how it is stored – a container

==========

++

Fritz

In computer science we perceive the world as *things* and *actions* on it. These things may exist in reality like a car or a person. They need not to be only artifacts but can be abstract like concepts or unreal like our ideas. Common to all these things is the possibility that it can be acted on it or they can initiate actions itself. All these things have properties which characterize them. The value of a property may change over time like the prize of an "article". This change is caused by an *action* on the *thing* "article". Actions can include complex operations, e.g. create new things by assembly of already existing objects.

In the world of computing we call a thing "object" and describe it by its properties. A property is represented by a name and a value. A value can be anything which represents a fact, something that can be measured or named. It is represented formally as data and algorithms are used to manipulate the data which formalize the actions on the objects. Data and algorithms are formal abstractions for properties of an object and the possible actions. This makes the manipulation of objects mechanizable, e.g. by a computer.

==========