# Importance of Data Flow Diagrams and Entity Relationships Diagrams to Data Structures Systems Design in C++ "A Practical Example"

Mohammed H. S. Al Ashry (Ashry)[1]

[1] Department of Computer Science, the Community College; Department of Computer Science, the College of Sciences and Humanities, Shaqra University, Shaqra, Saudi Arabia

Correspondence: Mohammed H. S. Al Ashry (Ashry), Department of Computer Science, the Community College; Department of Computer Science, the College of Sciences and Humanities, Shaqra University, Shaqra, Saudi Arabia.

Received: August 7, 2017 Accepted: August 17, 2017 Online Published: August 21, 2017

doi:10.5430/jms.v8n4p51 URL: https://doi.org/10.5430/jms.v8n4p51

**Abstract**

Structures have always been employed in designing software applications. In order to specify separate data structures, however, classes should be introduced to facilitate the process. The development of databases, in this case is made easier to specify records, data items, and simpler to identify unique keys within the database. The dataflow and entity relationships diagrams help in the development process. This paper provides a simple set of diagrams emphasizing the importance of these relationships.

**Keywords:** dataflow diagram DFD: the design of the flow of data and records prior to programming, entity relationships diagram ERD: the diagramming design that identifies and specifies the type and category of association, classes: an object or entity types, data structure: a set of related data objects or records

## 1. Introduction

Analysis and design software simplifies the process of building an actual small business application prototype. The characteristics of the application's structure and its data components can be defined prior to the building process. Of course, the relationship between the data within the separate structures in a program is made easier when linked on the bases of both primary keys (PK) and secondary keys (SK), internally, within related routines, and using foreign keys (FK) for external routines (Lipschutz, Seymour. (1986)). No literature has been defined specifically for the purpose of researching the significance of employing data flow diagrams (DFD) and entity relationship diagrams (ERD) in a program. Most literature is immersed in the actual DFD and ERD, within the unified modeling languages UML, for the purpose of simplifying the generation of code for building software or web-based applications.

## 2. Literature Review

DFDs and ERDs are mostly used by systems designers to simplify designing management and decision support systems. Software design processes, such as DFDs and ERDs simplify and reduce time consuming complex coding routines; not to mention business management documentation processes. Research on DFDs and ERDs focuses on new measures and criteria for their manifestations and implementations, in general, not their importance for specifically designing data structures, and other related systems.

There is no literature specifically emphasizing the importance of DFDs and ERDs to the design of data structure. Most related literature stressed the value of Unified Modeling Languages UML, and Systems Engineering Modeling Processes to designing decision and artificial support systems and other management applications, as a whole. Data structure is part of the process that leads to designing prototypes of all types of software applications. It simplifies the process of identifying primary data keys and their relationship to other elements within the data structure as a whole. This segment of the design procedure acts as a guide that relates the elements of a data-structure to the function of that data in handling the specific coding of the related routine.

Available literature, in general, highlighted the whole process of systems design. The specificity of the relationship between data structure systems design and DFDs and ERDs dampened the likelihood of finding literature employing C++ to exemplify the process. This paper goes through the process of relating the DFDs and ERDs to the design of data structure in the coding process utilizing C++.

## 3. The Approach

For this particular C++ data structures example, I chose to design a simple data processing system for a small business, a grocery store chain. In any software, whether it is a web application or a computer processing program the user is basically opening a set of data options. These options range from documentation menu tools – modules or pop up plain data of commodity sale options that change as the user flips through the menu. The main analysis and design process that leads to the final functioning application start with identifying entities and functions and defining the data flow processes and entity relationships. Once the definitions are specified in details the processes and relationships are diagrammed (Martin, James, & McClure, Carma. (1988)). In our grocery example it is followed with structured tabulations of the sub-entities defining the products, employees, costing and their relative output. Functions and their definitions are then planed using simple structured English, algorithm, and the code that links the functions with the data structures is laid out. An important note is that the word functions will be used interchangeably with processes, a function can be a process however a process can be a set of functions.

## 4. The Technical Analysis

The average pap's and mom's grocery store basically sells regular grocery products along with meat and vegetables for medium to large grocery stores. To simplify the process a small to medium store is designated for this example. The following diagram Figure 1 (Schach, Stephen R. (1993)), to be discussed, in the next segment, is a context diagram of a neighborhood grocery store. The assumption is that the store deals with multiple distributers and multiple sale functions through direct sales along with phone and or internet requests.

### *4.1 The Initial Diagramming Analysis*

The grocery store's main- context process, Manages Grocery Store is in the center of the diagram, the rest are external entities that can be linked to internal functions in the DFD's through the ERD's, to be explained in the following section. For example, In-store and off the store customers can purchase or order items and place complaints and requests. Purchased items are either taken directly by the customer or delivered depending on the purchase method.

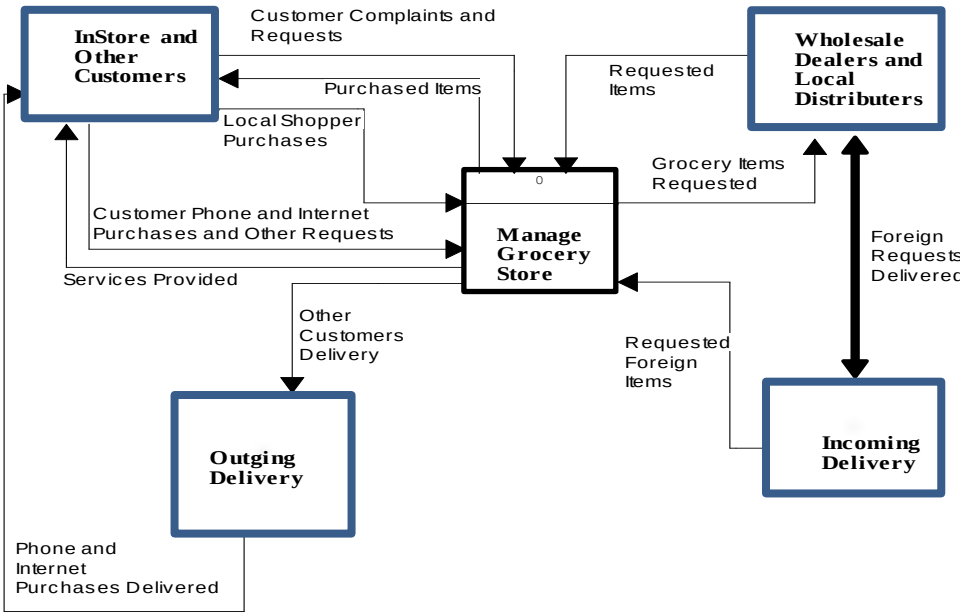


Figure 1. The grocery store financial transaction

The context diagram

The grocery store manager process

The external entities, in this case the peripheral four boxes, are not essential to the entire process, and may not be integrated in the programming code; nevertheless, are encompassed in the data stores for procurement, sales, credit, and liability purposes. For example, dealers and customers change, however, consumers' purchasing habits are essential for procurements.

*4.2 The Data and Entities Analyses*

The following diagram Figure 2 (Martin, James, & Leben, Joe. (1988)), is spawned from the main process of the context diagram. It is the next level in the data flow diagram, level one, and it houses the main processes handling the main functions of the grocery application. Looking at the diagram we see only the main processes and the related data stores. The arrows into and out of the processes are the data flows representing either functional transactions or structured data input or output. Data stores contain financial databases of employees, products, and lists of available employees, products, customers, and distributers.

The databases are divided into two partitions, active databases for ongoing business activities and transactions, and inactive databases for concluded activities and transactions. Open ended arrows are either incoming out of or going into external entities. Each of the processes encompasses a set of functions within the nested procedures in the next lower level of processes. Most actual source codes are written at the lowest level in the diagramming hierarchy, subject to relative entity relationships that may indulge certain functions within higher level processes.

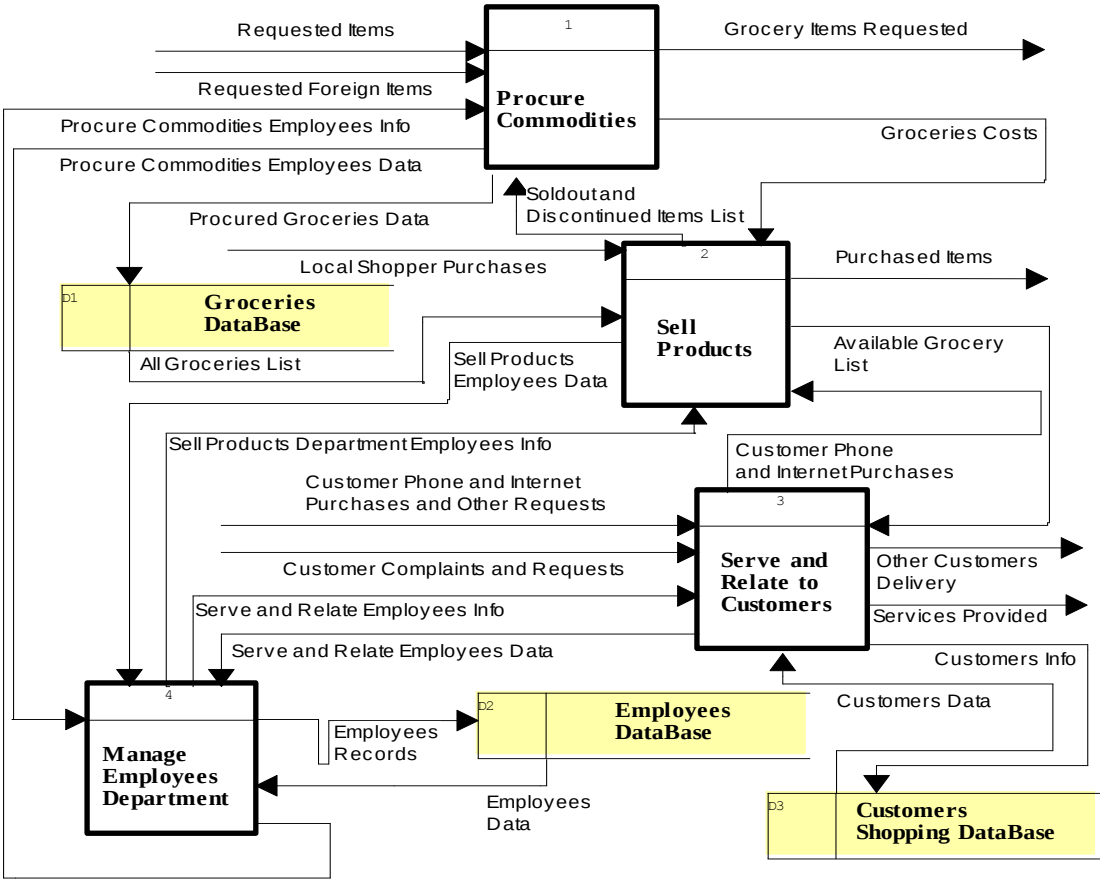


Figure 2. The grocery store financial transaction

The grocery store manager process

Level one diagram

Commodity procurement sales and customer services

There are four processes within this diagram level, procurement, sales, customer service, and employees departments. Each of these processes houses hierarchical nests of processes ending at the level where the activities are specific

enough for structured English and program coding. The following two diagrams are the employee department process and its nested lower level functions.

## 5. The Technical Design

The following set of DFD's and ERD are set up to include the relative structured English (Lipschutz, Seymour. (1986)), and some of the draft source code intended to execute the functions of the software independently or as a web based application. The second level diagram is the child diagram of the Manage Employee Department process. It covers checking employment applicants' resume's, delegating and designating job positions and tasks in accordance to qualifications, experience and rank; which includes employees' wages and financial report. In order to understand the technical aspects of the diagramming the arrows need to be examined methodically.

The 'Check Employees Resume'' process handles employees' assessments, resume' examination, and provide data requisite to employees' financial reports. There are, for example, four arrows leaving the process "Check Employees Resume'". "Procure Commodities Employees Info", which is the data related to the procurement department's employees delivered to the "Setup Employees Position and Functions" department regarding one or more of its employees from the company's employees' management department. The same applies to the other two arrows: "Sell Products Department Employees Info" and "Serve and Relate Employees Info". The fourth arrow relays the data regarding new employment applicants' category and rank assessment. There are also four arrows incoming into this process: "Procure Commodities Employees Data, Sell Products Department Employees Data, and Serve and Relate Employees Data". These arrows are data sent from base departments of employment to the department of employees' management on these employees regarding their conduct, work quality and so forth. A simple example of structured English for the 'Check Employees Resume'' process is included.

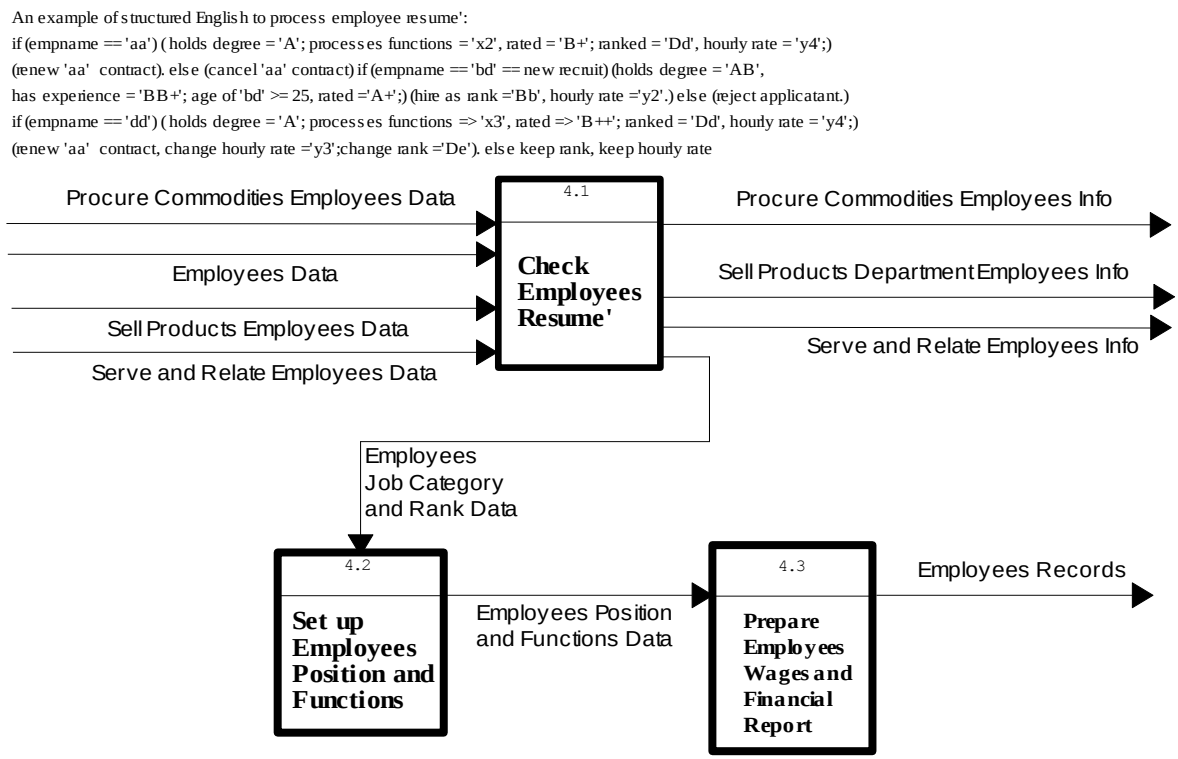


Figure 3. The grocery store financial transaction

The grocery store manager process

Level 2 diagram

Manage employees department

The 'Prepare Employees Wages and Financial Report' process receives employees' positions and functions data to prepare and store employees' records. The spawned offspring of this process provides us with the lowest level

processes that can be employed to create structured language and code to execute the functions of the related set of entities and functions. The following diagram lays out the practical functions related to the Employees Department.

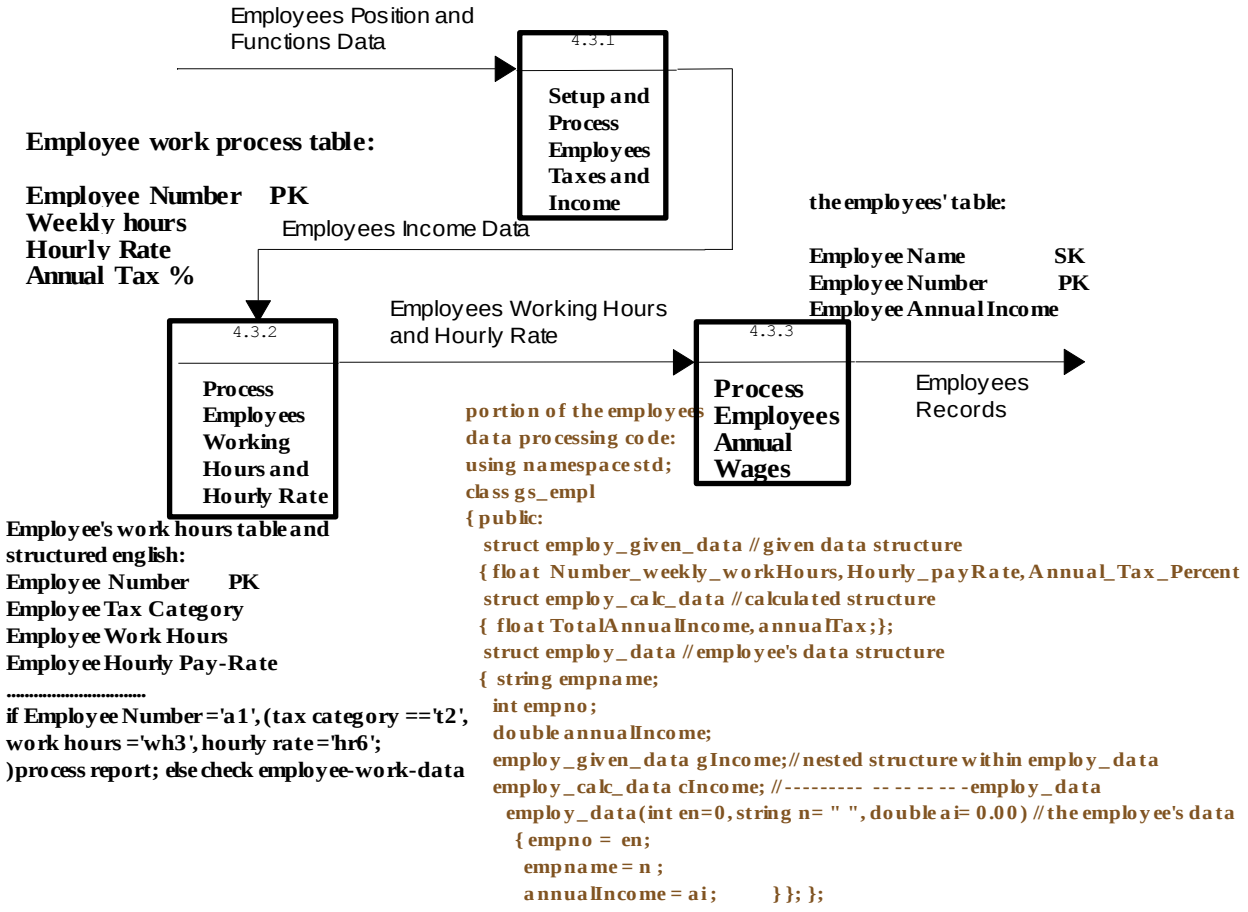


Figure 4. The grocery store financial transaction

The grocery store manager process

Level 3 diagram

Prepare employees wages and finbancial report

In this diagram, examples of the structured data tables for the 'Setup and Process Employees…..', 'Process Employees working ….' and 'Process Employees Annual….' are included to simplify structuring of the data within the databases. Another simple structured English database of the 'Process Employees working ….' is also included. It presents a simple basic code that is used as basis for the source code. A portion of the employee class-structure source code for the employees' wage calculations and records displays the variables within each of the structures and how their nesting simplifies processing the data. In each of the data structured tables the links are usually made up of the components used to relate the entities to each other within the related structured databases.

The object oriented system processes entities-implementations on the bases of their inheritance, multi-operative capacities (polymorphism), and whether they are public or private (encapsulation). These interface and related implementation components default to private for classes, and subclasses, and public for structures, and for their nested structures. For example, the following table I, is a list of simple structures with their primary, foreign and secondary keys, whose methods are defined as public.

Table 1. The main components

| **The main components** | Method protection definition |
|---|---|
| Employee's income table:<br>Employee number PK<br>Employee name SK<br>Employee annual income<br>Annual tax % | Public: Two ways to make the tables private: 1. Replacing the definition within the ***class gs_empl*** to private instead of public. 2. To make a specific database/data table private we add the word private to the constructor such as ***employ_data*** shown in the employee portion of the source code within Figure 4, in the same form applied with the ***class gs_emp***. |
| Employee's work table:<br>Employee number PK<br>Weekly work-hours<br>Hourly pay-rate | Public: Private protects both the data and the code, and either or both accessible only to the object they are created within. Private links the data to the code and either is accessible only within that object through its interface. So, other parts of the program, outside the object, cannot access the data or code |
| Employee's tax table:<br>Employee number PK<br>Tax category<br>Taxed work-hours<br>Taxed hourly pay rate | Public:<br>Can always be protected, data and code with adding protected or private at the appropriate location within the appropriate routine, or structure. |
| Products' departments table:<br>Department name PK<br>Department code SK<br>Number of employees<br>Number of products | Public:<br>Can always be protected, data and code with adding protected or private at the appropriate location within the appropriate routine, or structure. |
| Products departments input:<br>Department code PK<br>Department supervisor FK<br>Department annual product value (DAPV)<br>Department sales<br>Department losses | Public:<br>Can always be protected, data and code with adding protected or private at the appropriate location within the appropriate routine, or structure. |
| Products departments earnings:<br>Department supervisor PK<br>Department name SK<br>Department earnings | Public:<br>Can always be protected, data and code with adding protected or private at the appropriate location within the appropriate routine, or structure. |

The departments' and employees' tables mentioned above are basically entities with special relationships linking each and every one of them (Turban, Efraim, & Aronson, Jaye E. (1988)). ERD is an example of the OOP essential components utilized to simplify and facilitate building the programming code. The importance of a secondary key is its value when the primary key is unknown. In cases where secondary keys are not available, search, locates the primary key of that entity in a different table then extract the primary key. This takes us back to ERDs and their utilization in data structure's design.

The following two ERDs represent a context ERD Figure 5 in which the departments in each of the separate grocery stores are defined by the address of that store. The address, in this case, is a primary key in the department's table however it is a secondary key in the main corporate grocery stores and management table. Here, the stores have their

codes used as the primary code for corporate financial analysis, and the address component is for departmental financial analysis table. The departments' codes are used to examine the departments separately.

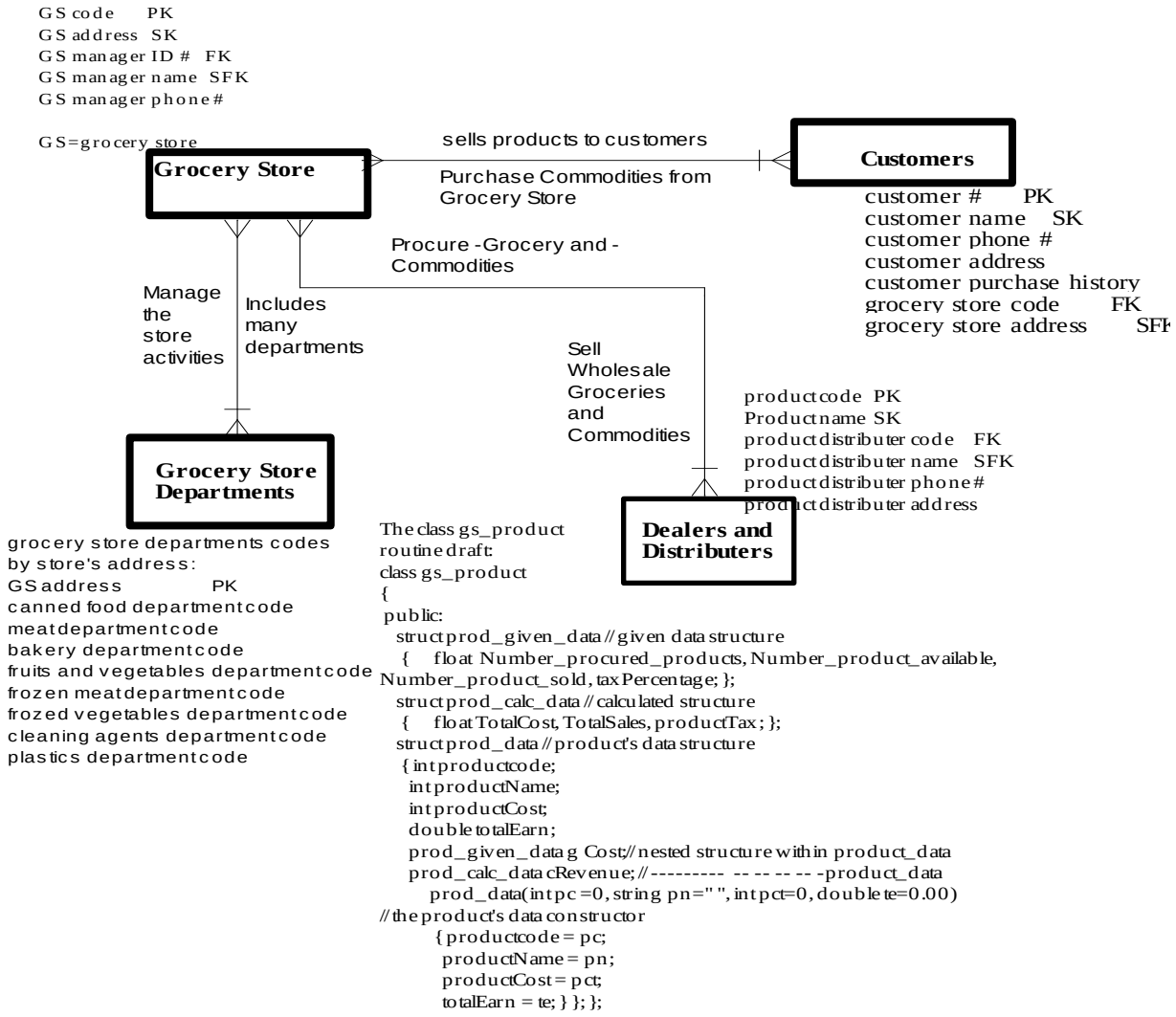


Figure 5. Grocery store ERD departments are defined separate from the main corporate store

The following second ERD diagram, Figure 6, puts more emphasis on departments dealing with both products and customers directly and indirectly. For example, Procurement department procures both customers' special requests along with other marketable and regular store items. Its direct contact with dealers and distributers provides for cost effective consumable items' prices. The General Customers Database keeps track of the customers shopping habits, special requests and their credit data. A simple draft of some of the related entities' tables and the source code's entry to the program's main function is provided.

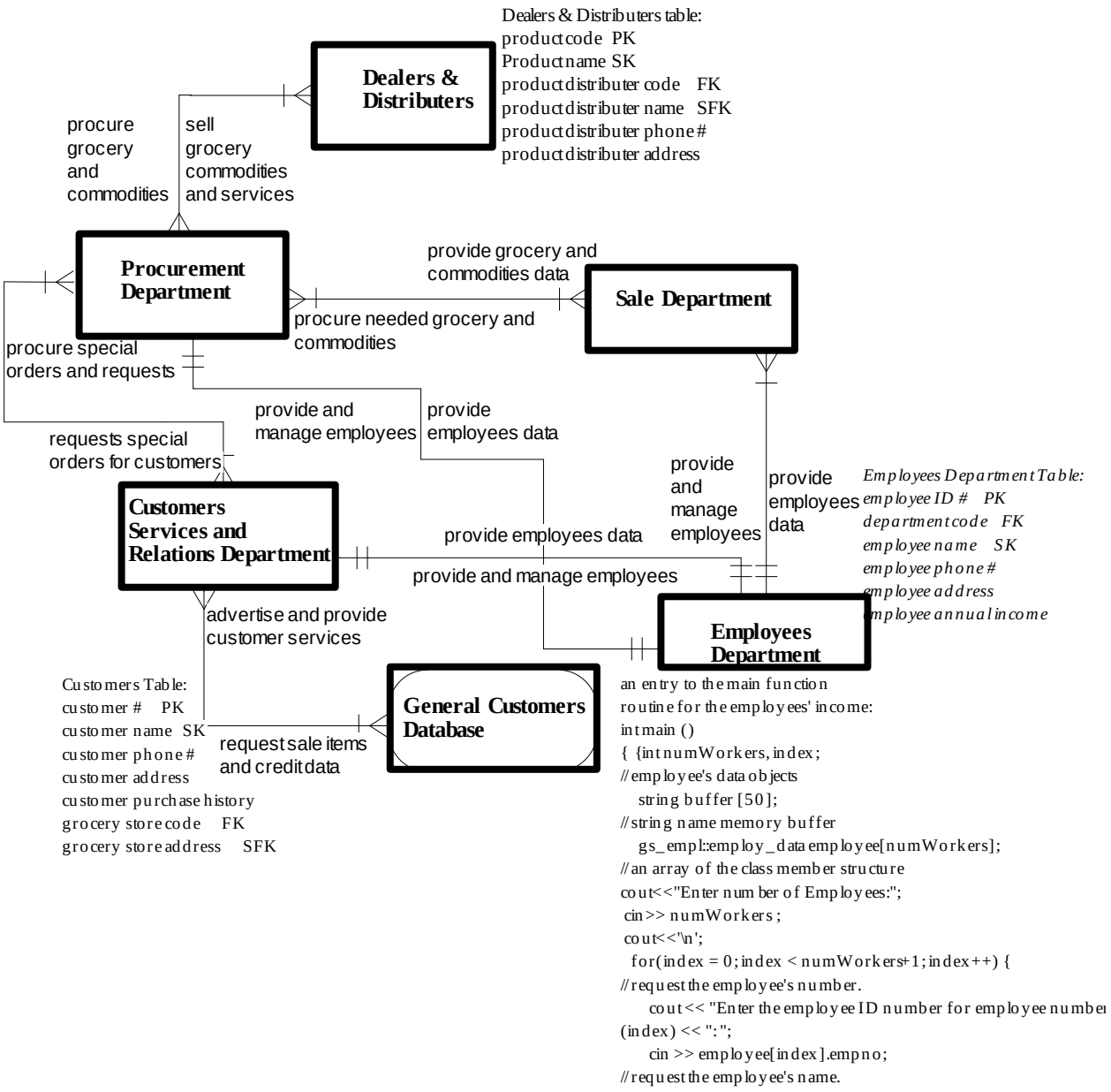


Figure 6. Grocery store entity relationship diagram includes:

1. Some of the related data structures
2. Some of the related simplified source code

For C++ code, some may want to take advantage of its powerful pointer features, where data can be aggregated and have a specific part of the hard disk's memory allocated for specific parts of the database. In this case, the main function can be rewritten to accommodate such need. The following part of the main function's routine, as pointed out in Figure 6, can be done utilizing the pointers. The following is an example of such implementation. The initial code employed structures within classes, and the main function uses structured database indexing method as follows:

***int main ()***

***{ {***

***int numWorkers, index;*** // employee's data objects

***string buffer [50];*** // string name memory buffer

***gs_empl::employ_data employee[numWorkers];*** // an array of the class member structure where ***gs_empl*** //is a class and ***employ_data*** is a structure. The ***employee[numWorkers]*** is an array string for accessing the data.

```
cout<<"Enter number of Employees:";
cin>> numWorkers ;
cout<<'\n';.
for(index = 0; index < numWorkers+1; index++) { // request the employee's number.
    cout << "Enter the employee ID number for employee number " << (index) << ": ";
    cin >> employee[index].empno;                // request the employee's name.
    cout << "Enter the name for employee number " << (index) << ": ";  // get employee's name
    cin >> employee[index].empname;          // Get the weekly hours worked by the employee.
    cout << "How many weekly hours did this employee work? ";
    cin >> employee[index].gIncome.Number_weekly_workHours;       // Get the employee's hourly pay rate.
    cout << "What is the employee's hourly payRate? ";
    cin >> employee[index].gIncome.Hourly_payRate;                // Get the annual tax percentage.
    cout << "What is the employee's annual tax percentage? ";
    cin >> employee[index].gIncome.Annual_Tax_Percentage;
    cout << endl; // get the employee's total income, get the annual tax and income after tax.
```

The previous indexing method can be replaced with indexing using pointer s. The following is an implementation of the pointer features which assigns the addresses of employee's data to pointers and the values at these pointers to facilitate memory allocation. For data memory allocation it is important to point out that addresses are sequential while data are not. It is also important to know that 8 bit and 32 or 64 bit compilers are certainly different for pointers' addressing allocation, subject to data types:

```
int main ()
{  {
int numWorkers, index; // employee's data objects

    string buffer [50]; // string name memory buffer
    gs_empl::employ_data employee[numWorkers], *p, y; // an array of the class member structure where //gs_empl is a class and employ_data is a structure. The employee[numWorkers] is an array string for //accessing the data, *p and y are pointer and structure member, respectively, to the string and data value.

    P = employee; //assigns the address of employee[numWorkers=0] to pointer p.
    y = *p;// assigns the value at p to y.
    cout<<"Enter number of Employees:";
    cin>> numWorkers ;
    cout<<'\n';.
                for(index = 0; p < employee+numWorkers; p++, index++) { // request the employee's number.

    cout << "Enter the employee ID number for employee number " << (index) << ": ";
    cin >> p->empno;                // request the employee's name.
```

***cout << "Enter the name for employee number " << (index) << ": ";*** // get employee's name

***cin >> p->empname;*** // Get the weekly hours worked by the employee.

***cout << "How many weekly hours did this employee work? ";***

***cin >> p->gIncome.Number_weekly_workHours;*** // Get the employee's hourly pay rate.

***cout << "What is the employee's hourly payRate? ";***

***cin >> p->gIncome.Hourly_payRate;*** // Get the annual tax percentage.

***cout << "What is the employee's annual tax percentage? ";***

***cin >> p->gIncome.Annual_Tax_Percentage;***

***cout << endl;*** // get the employee's total income, get the annual tax and income after tax.

In the rest of this routine we just replace the ***employee[index]*** with ***p->*** in the data input of the loop.

Utilizing pointers can add the addressing feature for each of the array's strings by assigning the pointer ***p*** or another variable, such as ***ptr*** to house the address of ***employee[index]***, ***ptr = &employee[index],*** within an empty sector in the HD prior to the assignment of the structures' array of strings (Smith, Richard (2014)). This, of course, encompasses the entire set of departments within the grocery store scheme.

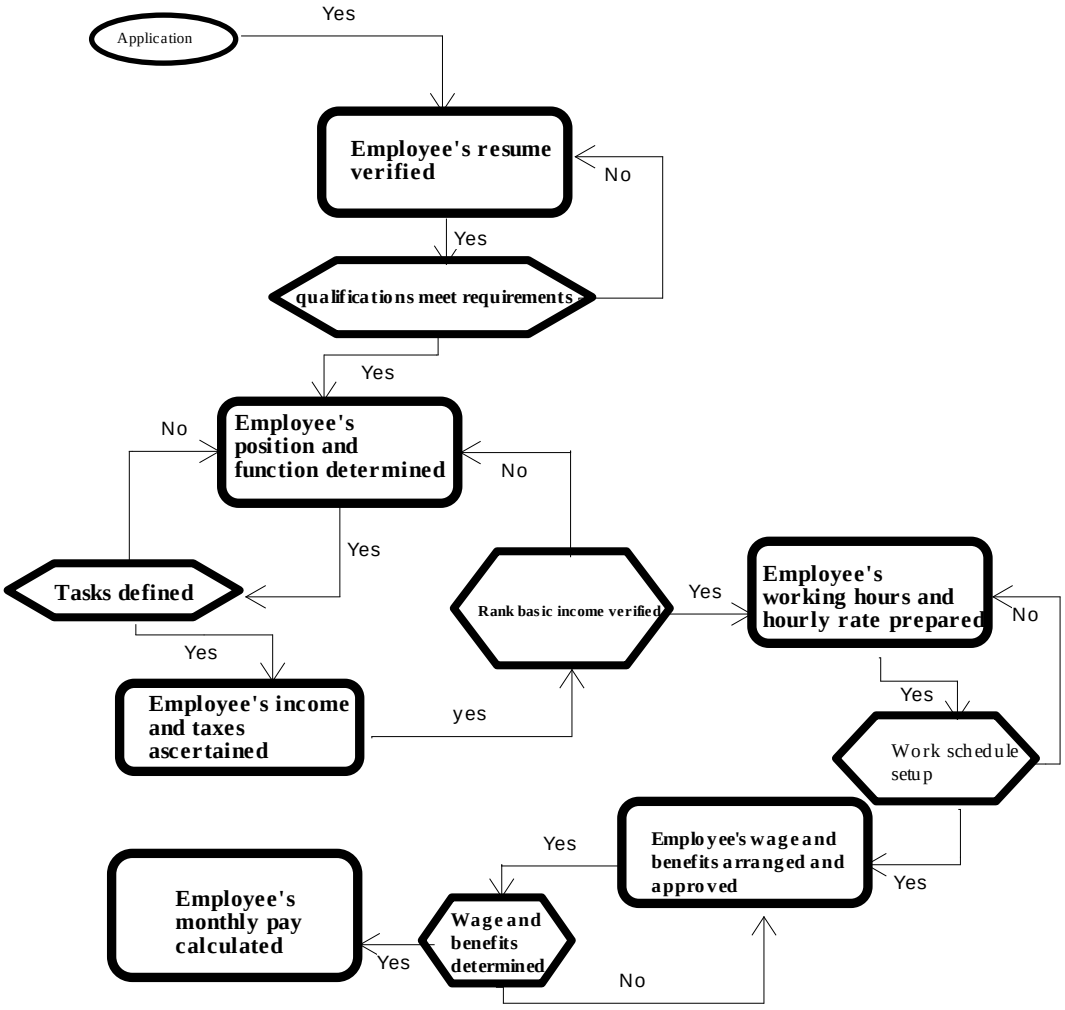


Figure 7. The decision flow chart (DFC)

Employment process at the grocery store

Another related diagramming system used to simplify DFD's and ERD's and in the process help in processing data structures is decision flow charting DFC. The DFC depicts the decision process of employee's recruiting and handling of the recruits' applications within the employees' department. The decision flow, in this chart, emphasizes lists of the employees' department's database which is based on lists of arrays of the data structures. It is undoubtedly obvious that data structures are a lot simpler to design and code when utilizing the DFD, ERD and DFC diagrams.

## 6. Results

It is obvious that DFDs provide both a big picture view and micro-view of the functions-system's designers seek. Some processes represent single functions, others combination of functions subject to the level of the process, or processes and their order within that level. The higher a process is the more likely that multiple functions fall within its sphere; and the lower the more likely it performs a portion of a function. For example, the level-one process # 4: "Manage Employees Department" performs all functions related to the employees within the grocery store, possibly the entire chain, however, the level three, process #4.3.2: "Process Employees Working hours and Hourly Rates" performs functions related to the employees working hours and hourly rates. This means this process houses another level with a set of processes to perform the working and the hourly rates. Entity relationships provide limitations to the intertwining functions and processes interactions. An example is the customer's database interacts with the employees' department subdivisions dealing with customers: customers' services and customers' relations.

## 7. Conclusion

Data flow is the scheme in which data; strings, numbers or arrays, are moved from one department, memory space to another department or memory space, respectively. In the process, these strings, numbers or arrays are either assessed for clarity or accuracy, or just modified. Either way, these structures of data forms are used to provide or improve information between two separate entities. Defining and diagramming the processing of such data-structures simplify and facilitate their manipulation and implementation for designing and/or coding programs and software systems. I recommend using modeling languages that employ the utilization of DFDs and ERDs and other related modeling processes in designing software systems.

## Refrences

Lipschutz, Seymour. (1986). Theory and Problems of Data Structures. *Schaum's outline series.*

Martin, James, & McClure, Carma. (1988). *Structured Techniques: The Basis for Case*. Printice Hall, Englewood Cliffs, New Jersey.

Schach, Stephen R. (1993). *Software Engineering*. Aksen Associates Incorporated Publishers, Richard D. Irwin Inc..

Martin, James, & Leben, Joe. (1988). *The ARBEN group Inc. Data Communication Technology*. Printice Hall, Englewood Cliffs, New Jersey.

Turban, Efraim, & Aronson, Jaye E. (1988). *Decision Support Systems and Intelligent Systems*. Printice Hall Upper Saddle River, New Jersey.

Smith, Richard. (2014). *Working Draft, Standard for Programming Language C++*. Document Number: N4296, Date: 11-19-2014, Revises: N4140, Reply to: Richard Smith, Google Inc.