Data abstraction

According to the Merriam-Webster online dictionary, the word abstract is defined as “ disassociated from any specific instance” or “ expressing a quality apart from an object”, or “ having only intrinsic form with little or no attempt at pictorial representation or narrative content”.

From these definitions, it can be possible to get an idea that to abstract an object implies something ethereal and nebulous, completely disjoint from a concrete instance of that object. In an idea reminiscent of Plato, the world can be separated into two things – the abstract idea and the concrete instance.

Understanding the concept of the abstract data type or ADT is easier knowing the definitions of abstraction. An ADT is a representation of a concrete instance. Computers can only process ones or zeros and can only store long ones and zeros.

However, in building programs a programmer might want to develop code that interacts or models real world objects or processes. ADTs are “ invented” data types – data types that are modeled after the abstract idea of the concrete instance. An example is the string data type found in some programming languages.

A computer cannot store a string (only ones and zeros) yet programmers can do operations on a string like concatenation (using the + operator) effortlessly as if the computer or compiler understands that the user is working with sentences.

This brings to light an important concept when dealing with ADTs – the concept of information hiding. A compiler designer might engineer a

programming language to handle strings in many ways. He may choose to use ASCII or EBCDIC, use 8 bits per character or a full 32 bit word, use little endian or big endian storage.

All these choices are invisible to the user. All the developer needs to understand is that to concatenate strings uses a “+” operator. Indeed, for an abstracted data type to be functional the functionality of that data type should reflect that what that ADT represents independent of the implementation.

The nitty gritty of its workings is hidden behind a wall called the interface. The interface (associated operations, properties, etc) is all that the programmer needs and should need to know. A good wall is a prerequisite of good ADT design.

So far the paper has discussed about ADTs as data types that represent an idea (such as a string) that is not natively supported by the hardware. A developer might also make his or her own ADTs through the use of data structures.

A data structure is basically just an ordered way of organizing data. An example of a data structure is the struct in C, linked lists, and trees. A developer may choose to create one of these data structures in order to represent an abstract idea. He may choose to use a tree in order to represent afamilytree.

In designing user created ADTs, the concept of information hiding should still be remembered. The ADT should provide a constant standard interface for every method or subroutine that chooses to call it. Additionally, it goes

without saying that the data structure of choice should efficiently model the abstract idea it represents. Using a tree to represent genealogy is easier and makes more sense compared to using linked lists.

A soda vending machine, even though it is quite simple is a good illustration of the many aspects of ADT design. The developer might need to store the types of sodas the machine is selling. As there is no “ soda” data type, the programmer might use strings. When the machine vends, the machine should also know that there is one less soda in its storage.

A programmer might then choose to implement the sodas as a stuct composed of one string (for the soda name) and an integer representing the number of soda cans left. When the customer presses a button corresponding to a soda, the soda name is displayed on the screen and the machine checks if there are still soda cans left.

If there are cans left, the vend process continues through with the customer getting his soda (after payment of course) and the integer counter for the soda is decremented by one. However if the counter is of value zero already, the machine halts the operation and tells the customer to pick another soda.

Bibliography

Carrano, Frank, and Janet Prichard. Data Abstraction and Problem Solving with C++ Walls and Mirrors. 3rd ed. Boston: Addison-Wesley, 2001.

Sedgewick, Robert. Algorithms in C. 3rd ed. Boston: Addison-Weslet, 1998.

Sun Developer Network [Website], java. sun. com