Name: Brihat Ratna Bajracharya

Roll No.: 19/075

Assignment #3 (Principles of Programming Language)

Question 1: Specifications and implementation of vector/array, record, list.

A. Vector/Array

A1. Specification of Vector/Array

A vector is a data structure composed of a fixed number of components of the same type organized as a simple linear sequence

A component of a vector is selected by giving its subscript, an integer (or enumeration value) indicating the position of the components in the sequence. A vector is also termed a one-dimensional array or linear array.

A two-dimensional array, or matrix, has its components organized into a rectangular grid of rows and columns. Both a row subscript and a column subscript are needed to select a component of a matrix. Multidimensional arrays of three or more dimensions are defined in a similar manner.

A2. Implementation of vectors.

The homogeneity of components and fixed size of vector make storage and accessing of individual components straightforward.

Packed and unpacked storage representation. A packed storage representation is one in which components of a vector are packed into storage sequentially without regard for placing a component at the beginning of an addressable word or byte of storage.

B. Records

B1. Specification of Records

A data structure composed of a fixed number of components of different types is usually termed a record. The fundamental difference between a record and an array is that record elements, or fields, are not referenced by indices. Instead, the fields are named with identifiers, and references to the fields are made using these identifiers

B2. Storage Implementation of Records

The fields of records are stored in adjacent memory locations. But because the sizes of the fields are not necessarily the same, the access method used for arrays is not used for records. Instead, the

offset address, relative to the beginning of the record, is associated with each field. Field accesses are all handled using these offsets.

C. List

C1. Specification of Lists

List is an abstract data type that represents a countable number of ordered values and may contain duplicate values. List data structure may provide some of the following operations:

1. **create** an empty list;
2. **test** if the list is empty;
3. **prepend** an element to the list
4. **append** an element to the list
5. **get** the first element (the "**head**") of a list
6. **get** the list consisting of all the elements except the head (called the "**tail**" of the list)
7. **access** the element at the given index

C2. Implementation of Lists

Lists are implemented either as *linked lists* (either singly or doubly linked) or as *arrays*, usually variable length or dynamic arrays.

Implementing lists was originated with the programming language Lisp in which each element of the list contain both its value and a pointer indicating the location of the next element in the list (linked list; also a tree if the list has nested sublists). Some older Lisp implementations (linke in Symbolics 3600) also supported "compressed lists" (using CDR coding) which had a special internal representation (invisible to the user).

Lists can be manipulated using iteration or recursion. The former is often preferred in imperative programming languages, while the latter is the norm in functional languages.

Lists can also be implemented as self-balancing binary search trees holding index-value pairs, providing equal-time access to any element, taking the logarithmic time in the list's size.

Lists form the basis for other abstract data types including ***queue***, ***stack***, and their variations.

Question 2: What is ADT? How is it different from structure data type?

A. ADT (Abstract Data Type)

Abstract data types are user-defined data types which have many of the same characteristics of primitive data types.

1. An abstract data type defines a new type.
2. The type defines allowable values and operations on those values.
3. The implementation of data values and their operations are hidden. The only way to manipulate values of an abstract data type is through the operations defined for the type.

The benefits of implementing an ADT are:

Abstraction means making visible what you want the external world to see and keeping the other details behind the wraps. It hides the complexity in which we decide what information to hide and what to expose.

Information hiding is making inaccessible certain details which would not affect other parts of the system.

Encapsulation is like enclosing in a capsule i.e. enclosing the related operations and data related to an object into that object. Encapsulation is not "the same thing as information hiding". For example, even though information may be encapsulated within record structures and arrays, this information is usually not hidden and is available for use. Encapsulation means just getting a set of operations and data together which belong together and putting them in a capsule.

Abstraction, information hiding, and encapsulation are different but related. Abstraction is a technique that helps us identify which specific information should be visible, and which information should be hidden. Encapsulation can be thought of as the implementation of the abstract class.

B. Structured Data Types

A new data type that is constructed as an aggregate of other data types, called components, is termed a structured data type or data structure. A component may be elementary or it may be another data structure(e.g., a component of an array may be a number or it may be a record, character string, or another array).

The major attributes for specifying data structures includes following:

1. Number of components. A data structure may be of fixed size if the number if components is invariant during its lifetime or of variable size. Variable-sized data structure types usually define

operations that insert and delete components from structures. Arrays and records are common examples of fixed-size data structure types. And stacks, lists, sets, tables and files are examples of variable-size types.

2. Type of each components. A data structure is homogeneous if all its components are of the same type. It is heterogeneous if its components are of different types. Ex. Arrays, sets and files are usually homogeneous, whereas records and lists are usually heterogeneous. A data structure type need a selection mechanism for identifying individual components of the
data structure.

C. ADT v/s Structure Data Type

The ADT defines the logical form of the data type. The data structure implements the physical form of the data type. Structure Data Type is the actual representation of the data during the implementation and the algorithms to manipulate the data elements. Hence, ADT is in the logical level and data structure is in the implementation level. This follows that ADT is implementation independent and Structure Data Type is implementation-dependent.

An Abstract Data Type(ADT) is a mathematical model, a way of logically stating a mathematical concept or data structure which may or may not be fully implemented. It is up to the programmer to define the implementation for ADTs or to import them from other headers or libraries.

For example, some of the ADTs can be implemented as given below:

1. **Stack** can be implemented using an array or a linked list.
2. **Priority Queue** can be implemented using a Binary Heap Data Structure.
3. **Vector** can be implemented using an array
4. **Map** can be implemented using a Red-Black Tree or using a Hash Table.