Data Structures for Virtual Reality Applications

February 13th 2009

MAE 574, Virtual Reality Applications and Research

Instructor: Govindarajan Srimathveeravalli

Abstract Data Structures (ADT)

- What is Abstraction ?
- What is an Abstract Data Structure ?
- What is the difference between an ADT and a C++ data type ?
- What is the relation between ADT and Object Oriented Programming ?

ADT

- ADT's
  - Means of aggregating data
  - Provide methods to operate on collected data
  - Similar to classes but has no polymorphism or inheritance

  ADT's are needed for VR apps to
  - Serve as data collections and collection objects
  - Ease implementation of algorithms
  - Search methods
  - Support graphics and rendering

Types of ADT

- Sequence collections and Associative collections
- Common ADT's
  - Arrays
  - List and List Variants
  - Stacks
  - Queues and DeQueues
  - Trees
  - Binary Tree
  - (Others) Maps, Sets, Graphs, Hash tables, Heaps etc..
- The STL !

What makes a ADT ?

- Basic requirements of an ADT would be :-
  - Create new collection
  - Add item(s) to collection
  - Delete item(s) from collection
  - Find item(s) in collection
  - Destroy the collection

```
class Container
{
        public:
        container( “Enter Storage Object Type” );   /* constructor */
        ~container(); /* destructor */
        /* add methods to provide various actions */
        Container* createContainer( “pass parameters for creation” );
        void        addToContainer( Container C, void* data );
        void        deleteFromContainer( Container C, void* data );
        void*       findInContainer( Container C, int key );
};
```

Code courtesy: John Morris, the CS Library at Stanford and Michael Goodrich

Arrays

- The most rudimentary data container
- Significant disadvantage is size and memory management.
- However it satisfies all of ADT's basic requirements
- The class we have described can be modified as follows to support creation of an array :-

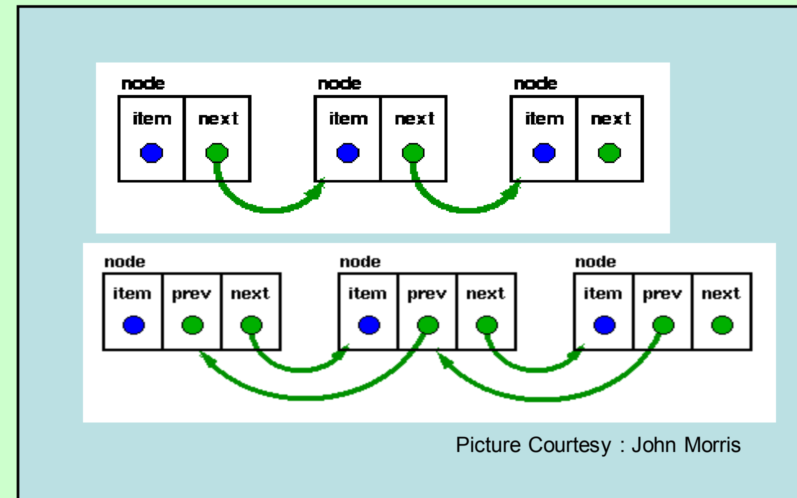
```
class Container
{
            public:
            container( "Enter Storage Object Type" );   /* constructor */
            ~container(); /* destructor */
            /* add methods to provide various actions */
            Container* createContainer( int max_items, int item_size );
            void        addToContainer( Container C, void* data );
            void        deleteFromContainer( Container C, void* data );
            void*       findInContainer( Container C, int key );
            public:   /* data members */
            int item_cnt;
            int max_cnt;   /* Not strictly necessary */
            int item_size;
            void* items[];
};
```

Code courtesy: John Morris, the CS Library at Stanford and Michael Goodrich

Lists

- Lists
  - A sequential collection of data
  - In a way can be considered an extension of arrays
  - Provide the advantage that size is not a limitation
- Types include :-
  - Ordered
  - Sorted
  - Singly linked
  - Doubly linked
  - Circularly linked

```
class Container
{
        public:
        /* All Methods are similar to previous implementation, minor changes
however exists */
        public:     /* data members */
        Node*       headNode;
};
struct Node
{
        void*       data;
        /* additional parameters can be:- */
        struct Node*        previousNode;
        structNode*         nextNode;
}
```



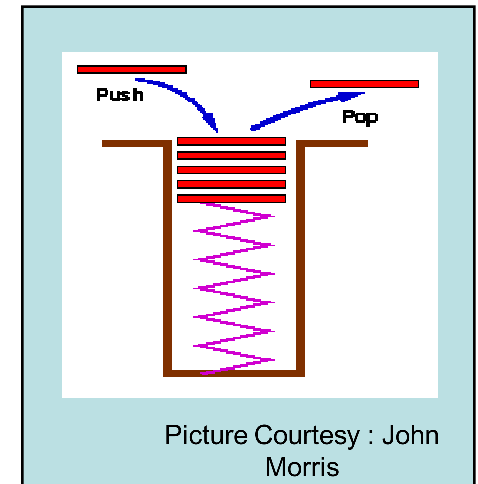
Code courtesy: John Morris, the CS Library at Stanford and Michael Goodrich

More on Lists ……..

- Vector, provided by STL can also be discussed as a special case of list.
- Applications in VR include :-
  - File storage
  - Sequential callbacks

Stacks

- Lists that are implemented with access (and deletion) only through the head node represent another ADT called stacks
- Stacks characterize
  - Sequential access
  - LIFO
- The OpenGL context !!
  - glPushMatrix() and glPopMatrix
  - Maintains a stack of Matrices (for object transformations)
  - The order matrices in stack access influences object hierarchy and positioning
- Use a stack based representation in your own home grown code !
- Stack frames and debugging



Picture Courtesy : John Morris

Queues

- Queues
  - Similar concept as a stack but follows FIFO
  - Insertion are at the rear and removal from the head
  - Push() and Pop()
- Types of Queues
  - DeQueues
    - Double ended Queue where removal and addition can happen from both sides
  - Priority Queue
    - Addition and Removal based on indexing

Note ……

- Queues are used in VE’s to handle calls from various interfaces and functions in an orderly fashion
- We have looked at the most standard ADT’s
- All of them are Implemented in STL
  - Vector
  - List
  - Queue
  - Stack
- So it is best to utilize the STL if we ever have to implement on of these ADT in a program
- We shall next look at how to use a STL vector and a List

STL - Basics

- Terminology
  - Container :- Class templates that can be used to creates holder for user defined or standard data types. Examples would include vectors, lists and deques.
  - Algorithms :- Functions for carrying out various actions on the data stored in containers
  - Iterator :- Can be called a “top level” pointer to any container
  - Allocator :- Memory managers for the containers

Vector

- Basically a dynamic array
- Provides random access, can also use array indexing to provide access
- Provides quickest (compared to stacks, deques and lists) access to data
- Template specification

  ```
  template<class T, class Allocator=allocator<T>>class vector
  ```

- “T” will specify the data stored and the allocator takes care of memory allocation and management

Vector declarations, (To use a vector , have to include <vector> !!)

```
vector<int>          counter;
vector<char>         cv(20); // length
vector<char>         cv(20, 'X'); // initialization
vector<int>          counter (counter);
vector<myClass>      vectorOfObjects;
```

// Note : In most case we do not specify the allocator for the vector, leaving it to the default one to manage memory

A Vector supports the following operators, == , <, <=, !=, >, >=

A vector also supports the [ ] subscript.

Vectors ….

- Most commonly used member functions of a vector are:
  - **size()** – returns the current size of a vector
  - **begin()** - returns an iterator to the beginning of a vector
  - **end() -** returns iterator to one past the last element
  - **push_back()** – adds an element to the end of a vector
  - **pop_back()** - deletes the last element in a vector
  - **insert()** - inserts “n” copies of specified “element” before a given element
  - **erase()** - erases the element specified by the index and returns the iterator to the element after it
  - **clear()** - deletes the elements in vector, makes its size zero

```
vector< int >              counterVec;              // declare a vector for holding a list of in objects

for(int i = 0; i < 5; i++)                          // add a series of values to the vector list
{ counterVec.push_back( i ); }

// The values entered in the vector can then be accessed sequentially for display
cout << "First Run **************************************" << endl;
// The size of the vector can be found using the size() function
cout << "The number of elements stored in the vector is   " << counterVec.size() << endl;

for( i = 0; i < counterVec.size(); i++)
{ cout << "The value stored in the vector is  " << counterVec[i] << endl; }
cout << endl;

// remove the last value from the vector
counterVec.pop_back();

cout << "Second Run **************************************" << endl;
// The size of the vector can be found using the size() function
cout << "The number of elements stored in the vector is   " << counterVec.size() << endl;
// now printout the values again
for( i = 0; i < counterVec.size(); i++)
{ cout << "The value stored in the vector is  " << counterVec[i] << endl; }
cout << endl;

// clear the vector
counterVec.clear();
// now its size is zero
cout << "The number of elements stored in the vector is   " << counterVec.size() << endl;
```

STL and Iterators

- Array subscripting based access is not allowed/supported in all STL containers
- Iterators are provided for element access in all STL containers
- A random access iterator supports operations ++ , -- , + and –
- Other types include bi-directional, forward, input and output

```
vector< int >::iterator      p_counter;
```

```
vector<int>  counterVec; // declare a vector for holding a list of int objects
vector<int>::iterator   pint;
pint = counterVec.begin();

for( i = 0; i < 6; i++)      // add a series of values to the vector list
{
                *pint = i;
                pint++;
}

for( i = 0; i < counterVec.size(); i++)
{
                cout << "The value stored in the vector is  " << *pint << endl;
                pint++;
}
cout << endl;

// insert a bunch of new values to the vector
pint = counterVec.begin();
pint += 3;
counterVec.insert(pint,4,6);

// erase values from the given vector
counterVec.erase( pint, pint+2 );
```

Lists

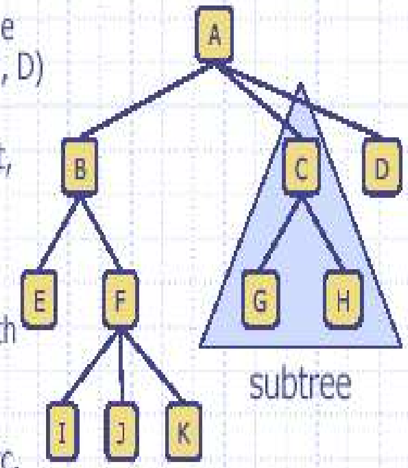
- Bi directional linear list
- Dynamic size
- Sequential access
- Additional functions:
  - merge() - merges two given lists, the merged list becomes empty at the end of the operation
  - pop_back(), pop_front(), push_front()
  - size_type() - returns the number of items in the list
  - sort() - sorts the list

Trees

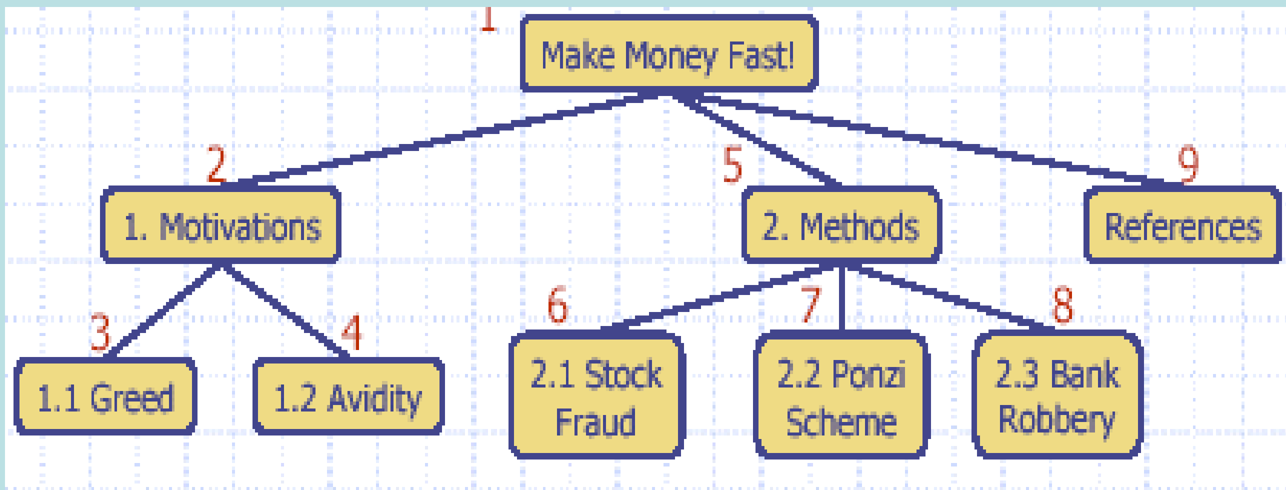
- Trees is a hierarchical data structure that is widely used in the graphics and visualization
- Also consists of nodes , but with a parent-child relationship
- Traversal :- Is the order/methodology followed when “reading” the contents (nodes) of a tree

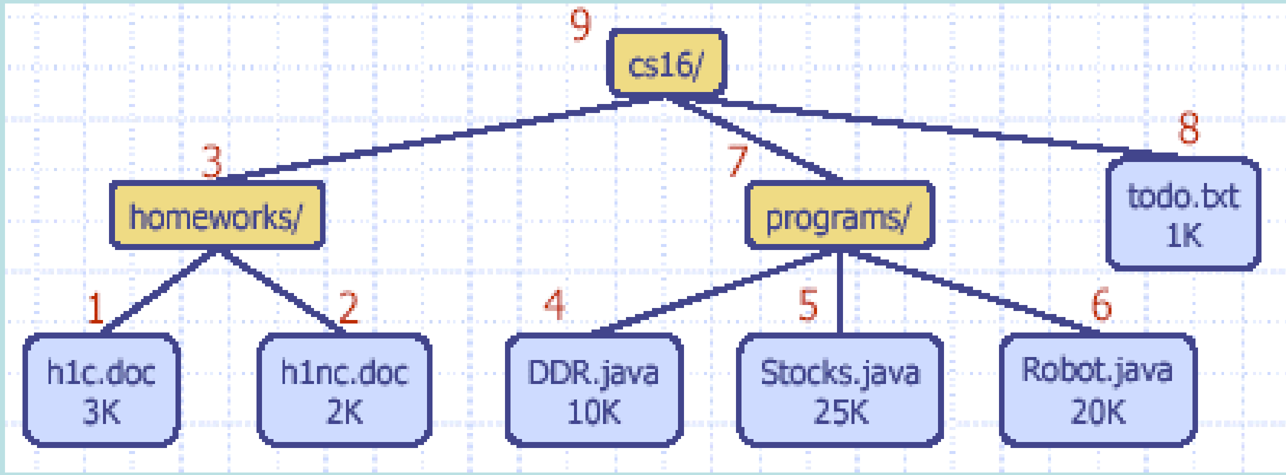
Tree Terminology

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node (3)
- Descendant of a node: child, grandchild, grand-grandchild, etc.
- Subtree: tree consisting of a node and its descendants



- Pictures courtesy : Michael Goodrich


1
Make Money Fast!
2
1. Motivations
5
2. Methods
9
References
3
1.1 Greed
4
1.2 Avidity
6
2.1 Stock Fraud
7
2.2 Ponzi Scheme
8
2.3 Bank Robbery


9
cs16/
3
homeworks/
7
programs/
8
todo.txt 1K
1
h1c.doc 3K
2
h1nc.doc 2K
4
DDR.java 10K
5
Stocks.java 25K
6
Robot.java 20K

Binary Trees ….

- Specific case of a tree where each node has only two other nodes connected to it.
- Each node of the tree is a binary tree
- Widely used in graphic applications to store data and for searching
- Also called a binary search tree (BST)
- It is a specific case where the nodes are arranged in a fashion where lesser value goes left and greater values go right