TOPIC: LIST AND LINKED LIST

Introduction

In computer science, a list or sequence is an abstract data structure that implements an ordered collection of values, where the same value may occur more than once. Each instance of a value is often called an item, entry, or element of the list; if the same value occurs multiple times, each occurrence is considered a distinct item.

List is a collection of data, element, component or objects with similar data type. List always represented as a record. Generally, a collection of data items that can be selected by indices computed at run-time, including array data structure, an arrangement of items at equally spaced addresses in computer memory. List can be implemented using array that contains sequence of data/record.

A list of number using array

<table>
<thead>
<tr>
<th>listArray</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
<td>[3]</td>
<td>[4]</td>
</tr>
</tbody>
</table>

In computer science, a linked list is data structure that consists of a sequence of data records such that in each record there is a field that contains a reference (i.e., a link) to the next record in the sequence.

A linked list whose nodes contain two fields: a string value and a link to the next node

The principal benefit of a linked list over a conventional array is that the order of the linked items may be different from the order that the data items are stored in memory or on disk. For that reason, linked lists allow insertion and removal of nodes at any point in the list, with a constant number of operations.

Concept

List Characteristic

- maximum size
- array for storing entries
- number of elements/entries
- current position
List Operations

Operations involved in implementing list using array are createList, insertItem, and DeleteItem. For createList operation, just declare an array.

Example, if want to create list of 10 number, declared as

```
#define maxSize 10 // used to declare maximum size of array
int listNumber[maxSize]; // declaration of an array
int NoOfItem; // used to store no. of item in an array
```

To verify whether an array are empty or not, just check the NoOfItem variable. If NoOfItem is equal to zero (0), it shows that the list is empty. Looping statement can be used to traverse the list of array; that loop accordingly to the NoOfItem in the list. For insertion and deletion operation, the implementation is more complicated based on the data type of the list.

Problem with array

- Array implementations of lists use a static data structure. Often defined at compile-time. Cannot be altered while program is running.
- This means we usually waste space rather than have program run out.
- It also means that it is difficult to construct ordered lists. In our implementation, data must be added to the end. If inserted before the end, all others beneath it must shuffle down. This is slow and inefficient.

Insertion shuffle

```
insert 1 here
```

```
2 4 5 8 11 13 16
1 2 4 5 8 11 13 16
0 1 2 3 4 5 6 7 8 9 10 11 12 13
```

Limitation of array

- An array has a limited number of elements
  - routines inserting a new value have to check that there is room
- Can partially solve this problem by reallocating the array as needed (how much memory to add?)
  - adding one element at a time could be costly
  - one approach - double the current size of the array
- A better approach: use a Linked List
  - and dynamically allocate memory

Why Linked Lists?

Linked lists and arrays are similar since they both store collections of data. The terminology is that arrays and linked lists store "elements" on behalf of "client" code. The specific type of element is not important since essentially the same structure works to store elements of any type. One way to think about linked lists is to look at how arrays work and think about alternate approaches.
Linked List: Basic Ideas

- A linked list is an ordered series of connected data / nodes
- Each element of the linked list has
  - Some data
  - A link to the next element
- The link is used to chain the data
- Example: A linked list of integers

```
Data  Link
20   45   75   85
```

- The linked list can grow and shrink

```
add(75), add(85)
20   45   75   85
```
```
delete(85), delete(45), delete(20)
75
```

Linked List Structure

Before writing the code to build the above list, we need two data types...

- **Node** The type for the nodes which will make up the body of the list. Each node contains a single client data element and a pointer to the next node in the list. Type: `struct node`
  ```c
  struct node
  {
    int data;
    struct node* next;
  };
  ```

- **Node Pointer** The type for pointers to nodes. This will be the type of the head pointer and the `.next` fields inside each node. In C and C++, no separate type declaration is required since the pointer type is just the node type followed by a `*`. Type: `struct node*`
## Memory Drawing

The best way to design and think about linked list code is to use a drawing to see how the pointer operations are setting up memory.

<table>
<thead>
<tr>
<th>Code segment</th>
<th>Memory Drawing</th>
</tr>
</thead>
</table>
| • Declare structure for a node  
  ```c
  struct Node  
  {  
    int data;  
    Node *next;  
  };  
  ``` |
| ![Memory Drawing](image1) |
| • Declare pointer  
  ```c
  typedef Node *NodePtr;  
  NodePtr Head;  
  ``` |
| ![Memory Drawing](image2) |

### Manipulation / Operation of Linked List

- Start the first node from scratch

<table>
<thead>
<tr>
<th>Code segment</th>
<th>Memory Drawing</th>
</tr>
</thead>
</table>
| • Set the Head pointer to NULL value  
  ```c
  Head = NULL;  
  ``` |
| ![Memory Drawing](image3) |
| • Create new pointer to pointer to a new node  
  ```c
  NodePtr newPtr;  
  ``` |
| ![Memory Drawing](image4) |

<table>
<thead>
<tr>
<th>Code segment</th>
<th>Memory Drawing</th>
</tr>
</thead>
</table>
| • Create new node  
  ```c
  newPtr = new Node;  
  newPtr->data = 20;  
  newPtr->next = NULL;  
  Head = newPtr;  
  ``` |
| ![Memory Drawing](image5) |
• Inserting / adding new node

<table>
<thead>
<tr>
<th>Code segment</th>
<th>Memory Drawing</th>
</tr>
</thead>
</table>
| • Inserting a node at the beginning  
  newPtr = new Node;  
  newPtr->data = 13;  
  newPtr->next = Head;  
  Head = newPtr; | ![Memory Drawing](image) |

• Deleting / removing node

<table>
<thead>
<tr>
<th>Code segment</th>
<th>Memory Drawing</th>
</tr>
</thead>
</table>
| • Deleting a head node  
  NodePtr delPtr;  
  delPtr = head;  
  Head = Head->next;  
  Delete delPtr; | ![Memory Drawing](image) |

Effectiveness of Linked List

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamically in size</td>
<td>Complex programming processed</td>
</tr>
<tr>
<td>Insert and delete operation doesn’t need to shuffle an existing item</td>
<td>High in computerizing time and memory management</td>
</tr>
<tr>
<td>Less time used in insert and delete operation</td>
<td>Not suitable for simple list with minimal size</td>
</tr>
<tr>
<td>Used for large size of element (unknown size)</td>
<td>Complex in updating of program</td>
</tr>
</tbody>
</table>

Differences between list and linked list

<table>
<thead>
<tr>
<th>List</th>
<th>Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Static size of list</td>
<td>• Dynamic size of list</td>
</tr>
<tr>
<td>• Add &amp; delete item required much steps</td>
<td>• Add &amp; delete item required less steps</td>
</tr>
<tr>
<td>• Suitable for less of list</td>
<td>• Suitable for larger of list</td>
</tr>
<tr>
<td>• Easy programming processes</td>
<td>• Complex programming processes</td>
</tr>
<tr>
<td>• Easy program update</td>
<td>• Complicated program update</td>
</tr>
<tr>
<td>• Low of memory and computerizing time</td>
<td>• High of memory and computerizing time</td>
</tr>
</tbody>
</table>
Activity

1. Based on the diagram below, draw new diagram for each statement given:

   [Diagram of linked list: 32 -> 2 -> 56 -> 99]

   a) New element ‘15’ inserted between second and third element in linked list
   b) Element 2 removed from the linked list

2. Illustrate deleting operation from linked list

Assessment

Q1. Linked list is
   A. A data structure that consists of a sequence of data records such that in each record there is a field that contains a reference
   B. An arrangement of items at equally spaced addresses in computer memory
   C. An abstract data structure that implements an ordered collection of values, where the same value may occur more than once
   D. A data structure that consists of a parallels of data records such that in each record there is a field that contains a reference

Q2. Linked list consists of at least
   A. Two fields: Data, link
   B. Two fields: Number, pointer
   C. Two fields: Integer, link
   D. Two fields: Number, link

Q3. Data structure for a node in linked list?
   A. struct Node
      {
         int data;
         int *next;
      }
   B. struct Node
      {
         char data;
         Node *link;
      }
   C. struct Node
      {
         int data;
         Node next;
      }
   D. struct Node
      {
         char data;
         char *link;
      }

Q4. Based on the diagram below, draw the changes that happen after `pqr = pqr->next->next` statement being executed:

   [Diagram of linked list: pqr -> B -> C -> D -> E]

© prepared by: nurizah@ict.puo.edu.my
Q5. Write down code statement to show an operation for inserting node in front of the linked list based on the diagram given.

Before

After

Q6. TRUE or FALSE?

A. In a linked list, components are only logically next to each other whereas in an array they are also physically next to each other.
B. Nodes in a linked list structure must contain a link member.
C. In deleting an item from a linked list, we need to keep track of the previous node.

Summary

- A data structure in which each element contains a pointer to the next element, thus forming a linear list.
- Linked lists are a way to store data with structures so that the programmer can automatically create a new place to store data whenever necessary.
- The linked list is relocatable, meaning it can be moved about in memory at will, and it can also be quickly and directly serialized for storage on disk or transfer over a network.
- A linked list is a dynamic data structure and therefore the size of the linked list can grow or shrink in size during execution of the program. A linked list does not require any extra space therefore it does not waste extra memory. It provides flexibility in rearranging the items efficiently.
- The limitation of linked list is that it consumes extra space when compared to a array since each node must also contain the address of the next item in the list to search for a single item in a linked list is cumbersome and time consuming.