Data Structure

Lab: #02

Liner Data Structure
Linked List

Asst. Instructor

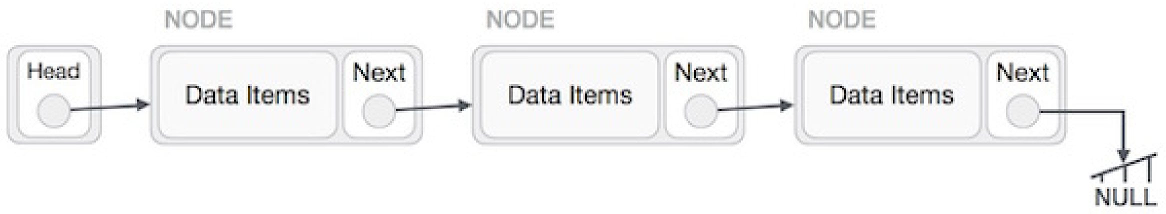
Ali A. Al-Ani

Introduction

A linked list is a sequence of data structures, which are connected together via **links**. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



As per the above illustration, following are the important terms to understand the concept of Linked List.

- Linked List contains a link element called **start or head**.
- Each link carries a **data field**(s) and a link field called **next**.
- Each link is linked with its **next link** using its **next link**.
- Last link carries a link as **null** to mark the end of the list.

Basic Operations

Following are the basic operations supported by a linked list.

- **Insertion**: Adds an element at the beginning, the end and the middle of the list.
- **Deletion**: Deletes an element at the beginning the end and the middle of the list.
- **Display**: Displays the complete list.
- **Search**: Searches an element using the given key.
- **Delete**: Deletes an element using the given key.

C++ Pointers

In computer science, a pointer is a programming language object, whose value refers to (or "points to") another value stored elsewhere in the computer memory using its memory address. Some C++ tasks are performed more easily with pointers, and other C++ tasks, such as **dynamic memory allocation**, cannot be performed without them.

As we know every variable has a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator which denotes an address in memory. Consider the following which will print the address of the variables defined

```
#include <iostream.h>
int main () {
   int  var1;
   char var2[10];
   cout << "Address of var1 variable: ";
   cout << &var1 << endl;
   cout << "Address of var2 variable: ";
   cout << &var2 << endl;
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Address of var1 variable: 0xbfebd5c0
Address of var2 variable: 0xbfebd5b6
```

How to define Pointers?

A pointer is a variable whose value is the address of another variable. Like any variable or constant, we must declare a pointer before we can work with it. The general form of a pointer variable declaration is:

```
type *var-name;
```

Here, type is the pointer's base type; it must be a valid C++ type and **var-name** is the name of the pointer variable. The asterisk(*) we used to declare a pointer is the same asterisk that we use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to character
```

Example: Using Pointers in C++

There are few important operations, which we will do with the pointers very frequently.

(a) We define a pointer variable.
(b) Assign the address of a variable to a pointer.
(c) Finally access the value at the address available in the pointer variable.

This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes use of these operations:

```
#include <iostream.h>
int main () {
   int  var = 20;   // actual variable declaration.
   int  *ip;        // pointer variable
   ip = &var;       // store address of var in pointer variable
   cout << "Value of var variable: ";
   cout << var << endl;
   // print the address stored in ip pointer variable
   cout << "Address stored in ip variable: ";
   cout << ip << endl;
   // access the value at the address available in pointer
   cout << "Value of *ip variable: ";
   cout << *ip << endl;
   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

Structures:

C/C++ arrays allow us to define variables that combine several data items of the same kind, but structure is another user defined data type which allows us to combine data items of different kinds. Structure (the keyword **struct** is used in C++) is use to group variables into a single record. Keyword **struct** is a data-type, like the following C++ data-types (int, float, char, etc...). The format of the struct statement is this:

```
struct [struct-name] {
  member definition;
  member definition;
  ...
  member definition;
} [one or more structure variables];
```

Accessing Structure Members

1. **Use "." dot operator:** To access any member of a structure, we use the member access operator (.). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. We would use struct keyword to define variables of structure type. Following is the example to explain usage of structure.

```
#include <iostream.h>
struct Books {
   string  title;
   int    book_id;
};
 int main() {
   struct Books Book1;        // Declare Book1 of type Book
   // book 1 specification
   Book1.title = "Learn C++ Programming";
   Book1.book_id = 6495407;
  // Print Book1 info
      cout << "Book title : " << Book1.title <<endl;
      cout << "Book id : " << Book1.book_id <<endl;
   return 0;    }
```

When the above code is compiled and executed, it produces the following result –

```
Book title : Learn C++ Programming
Book id : 6495407
```

2. **Use " -> " The arrow operator :** We can define pointers to structures in very similar way as we define pointer to any other variable as follows:

```
struct node {

   int  n;

   struct node *link;

};
```

Now, we can store the address of a structure variable in the above defined pointer variable. To access the members of a structure using a pointer to that structure, you must use the **->** operator as follows:

```
struct node *node1;
node1->n = 3;
node1->link = NULL;
```

Create a Linked list Using structure:

This simple example show how can we create a single linked list using structure.

```
#include <iostream.h>
struct node {
   int  n;
   struct node *link;
}*start;
int main() {
   struct node *node1;
   node1 = new (struct  node);
    start= node1;
    node1->n = 1;
    node1->link = NULL;
         cout << "Start value : " << start<<endl;
         cout << "Node1 value : " << node1->n <<endl;
        cout << "Node1 address : " << node1->link <<endl;
      return 0;
}
```

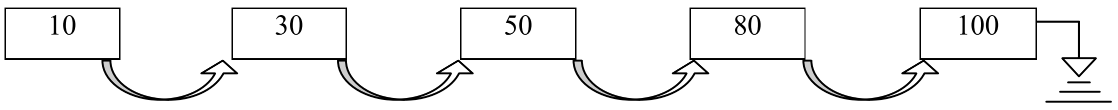
When the above code is compiled and executed, it produces result something as follows:

```
Start value : 0x1678c20
Node1 value : 1
Node1 address : 0
```

Exercises:

1. Write a C++ program for each of the following:
   - **a.** Create a linked list of two nodes.
   - **b.** Delete the first element in the linked list.
   - **c.** Insert one element after position p in a linked list.
   - **d.** Insert one element before position p in a linked list.
   - **e.** Delete all the elements where its data field is odd in any linked list.
   - **f.** Delete the last element of any linked list.
   - **g.** Delete all the elements of even position in the linked list.
   - **h.** Delete any zero value elements in the linked list.
   - **i.** Count the number of elements in the linked list.

2. Write a C++ program to display all elements of any linked list except the element of data value 44.
3. Write a C++ program to delete all elements of the linked list except the first and last elements.
4. Give the following linked list:



Write a C++ program to:

- **a.** increment the value of the fourth and fifth element by 50.
- **b.** Add an element of value 60 before the last element in the list.