**Generic Linked List**

**Linked List is Linear Data Structures that store values in nodes. As we do know here each Node possesses two properties namely the value of the node and link to the next node if present so. Linked List can not only be of Integer data type but String, boolean, Float, Character, etc.**

#### What are generic linked lists?

Generic linked lists are implementations of linked lists which can store data of any data type.

**For Example:**

**A picture containing diagram

Description automatically generated**

**A picture containing chart

Description automatically generated**

**The above two linked lists are having different types of data. The first one stores integer whereas the second one stores float. To implement the above mentioned linked lists generally, we would define two different nodes individually for both the linked lists.**

**We can implement such a “generic” Linked List Data Type that can store values of any data type.**

There are 6 primary member functions of a linked list: E

* add (data): It adds an element at the end of the linked list
* add (position, data): It adds an element to any valid position in the linked list
* remove(key): It removes node that contains key from the linked list
* clear() : it clears the entire linked list
* empty(): It checks if the linked list is empty or not
* length(): It returns the length of the linked list

***Note: Time complexity is of order N for adding and removing operations and of order 1 for other operations.***

Unlike [C++](https://www.geeksforgeeks.org/c-plus-plus/) and [Java](https://www.geeksforgeeks.org/java/), [C](https://www.geeksforgeeks.org/c/) doesn’t support generics. How to create a linked list in C that can be used for any data type?

In this article, we are going to discuss how to implement such a data structure in the C programming language.

**Approach**

Let’s look at the structure of a node in a singly linked list:

struct Node {

int data;

Node\* next;

};

It can store two values:

* the data
* the pointer to the next node.

According to the definition of a generic linked list, it should be able to store data of any type.

It can be implemented using **template** in C++. But in a language like C, there is no support for generics. So how to tackle this limitation?

We have something called a ‘void pointer’ in C. The void pointer can store the address of any data type. It will allow the program to store the address of any data type as per there the user requirements and hence can be used to implement a generic linked list in C.  
We can now change the structure of the node as:

struct Node {

void \*data;

struct Node \*next;

};

So, now we have something called void pointer in our node using which we can create a generic linked list in C.

Now let us look at the implementation of two functions **push\_front()** and **printIt()** on our generic linked list.

**push\_front()**

**push\_front()** function will take 'x' as one of its argument and will add this element ‘x’ at the beginning of the linked list.

Now, how to implement such a function for the generic linked list?  
This function will require 2 information about the data being added:

* size of new data
* the data

Since the size of data is not predefined, we need to pass the data as a void pointer and also provide its size.

**Algorithm**

* Allocate memory for new\_node.
* Allocate memory for the value of new\_node using the size provided.
* Set the next of this new\_node to point to the current head.
* Copy the new data 'x', to the new\_node’s value byte by byte.
* Change the head pointer to point to new\_node.

**printIt()**

**printIt()** function will print the contents of the linked list.

To print the contents of a linked list, this function will require the pointer to the head of the linked list, along with the suitable function to print according to the type of data stored in it. For that, we have defined 2 helper functions to print integer and float type data. We can pass these functions to the printIt() function as a void pointer.

Following is a sample C code to demonstrate working of generic linked list.

// C program for generic linked list

#include<stdio.h>

#include<stdlib.h>

/\* A linked list node \*/

**struct** Node

{

    // Any data type can be stored in this node

**void**  \*data;

**struct** Node \*next;

};

/\* Function to add a node at the beginning of Linked List.

   This function expects a pointer to the data to be added

   and size of the data type \*/

**void** push(**struct** Node\*\* head\_ref, **void** \*new\_data, **size\_t** data\_size)

{

    // Allocate memory for node

**struct** Node\* new\_node = (**struct** Node\*)**malloc**(**sizeof**(**struct** Node));

    new\_node->data  = **malloc**(data\_size);

    new\_node->next = (\*head\_ref);

    // Copy contents of new\_data to newly allocated memory.

    // Assumption: char takes 1 byte.

**int** i;

**for** (i=0; i<data\_size; i++)

        \*(**char** \*)(new\_node->data + i) = \*(**char** \*)(new\_data + i);

    // Change head pointer as new node is added at the beginning

    (\*head\_ref)    = new\_node;

}

/\* Function to print nodes in a given linked list. fpitr is used

   to access the function to be used for printing current node data.

   Note that different data types need different specifier in printf() \*/

**void** printList(**struct** Node \*node, **void** (\*fptr)(**void** \*))

{

**while** (node != NULL)

    {

        (\*fptr)(node->data);

        node = node->next;

    }

}

// Function to print an integer

**void** printInt(**void** \*n)

{

**printf**(" %d", \*(**int** \*)n);

}

// Function to print a float

**void** printFloat(**void** \*f)

{

**printf**(" %f", \*(**float** \*)f);

}

/\* Driver program to test above function \*/

**int** main()

{

**struct** Node \*start = NULL;

    // Create and print an int linked list

    unsigned int\_size = **sizeof**(**int**);

**int** arr[] = {10, 20, 30, 40, 50}, i;

**for** (i=4; i>=0; i--)

       push(&start, &arr[i], int\_size);

**printf**("Created integer linked list is \n");

    printList(start, printInt);

    // Create and print a float linked list

    unsigned float\_size = **sizeof**(**float**);

    start = NULL;

**float** arr2[] = {10.1, 20.2, 30.3, 40.4, 50.5};

**for** (i=4; i>=0; i--)

       push(&start, &arr2[i], float\_size);

**printf**("\n\nCreated float linked list is \n");

    printList(start, printFloat);

**return** 0;

}

Output:

Created integer linked list is

10 20 30 40 50

Created float linked list is

10.100000 20.200001 30.299999 40.400002 50.500000

#### Time complexity

**push():** O(1)  
**printIt():** O(n), where n is the number of nodes in the linked list.