BEINGZERO PHASE-2

# VECTORS USING LINKED LIST OF ARRAYS

MINI PROJECT


TEAM MEMBERS:

- 19H51A04F2 – B. RAJEEV KUMAR
- 19H51A04F3 – B. GOUTHAM
- 19H51A04F8 – G. PAWAN KALYAN


**Date** : June 2021

**MENTORS:** Ashutosh

**SEMESTER:** II SEMESTER 2nd YEAR.

# ABSTRACT:

Implementation of vectors using linked list of arrays. The main purpose of our project is to implement vectors such that memory allocation will be dynamic.

The Primary goal of the project is to implement vectors and its functions by merging the arrays using linked list in C++.

The secondary goal of the project is to implement the vector functions with time complexity of O(1).

Table of contents:

# 1 PROJECT DESCRIPTION:

## 1.1 Purpose of the project:

The main purpose of our project is to implement vectors such that memory allocation will be dynamic.

## 1.2 Goals:

- The Primary goal of the project is to implement vectors and its functions by merging the arrays using linked list in C++.
- The secondary goal of the project is to implement the vector functions with time complexity of O(1).

## 1.3 METHODOLOGY:

### 1.3.1Alternative approaches:

- Implementation of vectors using arrays:
  - We can implement vectors by resizing array by reallocating the memory.
  - If an extra element is inserted, then the size of the array can be increased by using realloc function.
  - If an element is deleted, we can shrink the size of the array using realloc and thus we can achieve the goal of implementing vectors

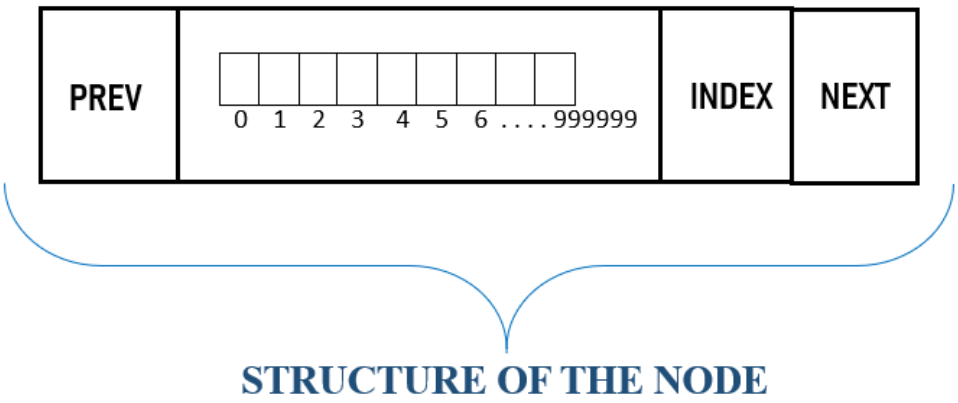
- Implementation of vectors using singly linked list of arrays:
  - We can implement vectors also by using singly linked list of arrays by merging the arrays using singly linked list.
  - The major drawback with this approach is, deletion at the end takes time complexity of O(n).

## 1.3.2Current approach chosen:

- Implementing vectors using doubly linked list.

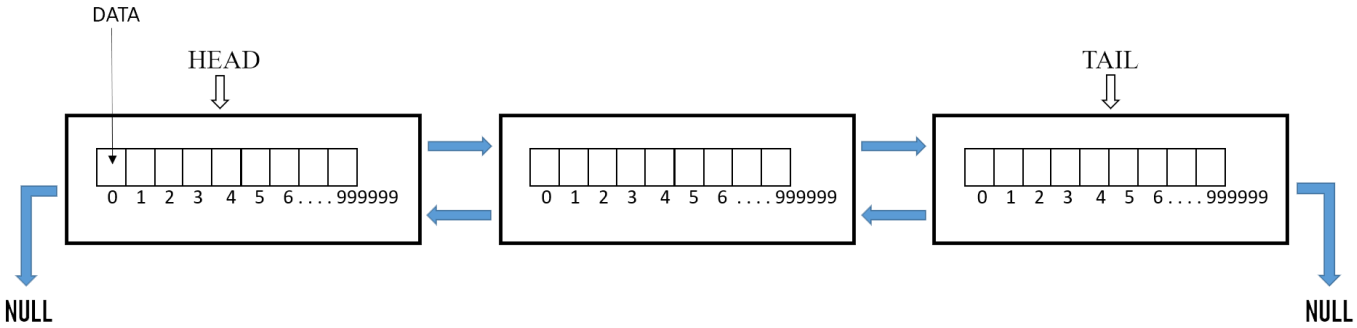
## 1.3.3 Detailed description of current approach:

- In the current approach we merge the max sized arrays using doubly linked list.




STRUCTURE OF THE NODE

- We created a structure with structure variables as an array of max size, a variable named index, and two pointers previous and next.




- We created the following functions:
  *pushBack()*
  - During pushback operation initially a node is created with an array of max size, as the node's array gets filled, we move to the next node and store the remaining elements and this process continues until all the elements are stored.

*popBack()*

- In the popBack operation we delete last element present in the array of the node which is pointed to the tail pointer.
- If the array contains only one element then during the popback operation the node which is pointed to tail is deleted and tail to previous is made as tail.

*get_element()*

- In the get_element function it takes index as input and returns the element present in that index value.

*get_size()*

- The get size function return the size of the array.

*is_empty()*

- The is_empty function returns true if the array is empty and false if the array is not empty.

*display_vector()*

- It prints all the elements in the array, by traversing through all the nodes of the doubly linked list.

## 1.4 MESUREMENTS TO BE DONE:

We've to measure the execution time of all the functions by using clock function from time library and we plotted the graphs for all the functions by varying the inputs from 0 to 1000000.

## 1.5 CONSTRAINTS:

Values to be pushBack:

$-2147483648 \leq$ value $\leq 2147483647$.

# 2 CODE:

```cpp
#include <bits/stdc++.h>
using namespace std;

#define ll long long int
#define ull unsigned long long int

struct Lv
{
    int a[1000000]; //
    int idx = 0;
    Lv * next;
    Lv * prev;
};

Lv *head = NULL, *tail = NULL;
ll arr_size = -1;

void pushBack(int x)
{
    if (arr_size == -1)
    {
        Lv *nn = (Lv*) malloc(sizeof(Lv));
        nn->next = NULL;
        nn->prev = NULL;
        head = nn;
        tail = nn;
    }
    else if (tail->idx > 999999)
    {
        Lv *nn = (Lv*) malloc(sizeof(Lv));
        nn->next = NULL;
        tail->idx--;
        nn->prev = tail;
        tail->next = nn;
        tail = nn;
    }

    arr_size++;
    tail->a[tail->idx] = x;
    tail->idx++;
}

void popBack()
{
    if (arr_size == -1)
    {
        cout << "Vector is empty cannot delete node!" << "\n";
    }
    else if (tail->idx - 1 == 0)
    {
        if (tail->prev == NULL)
        {
            tail = NULL;
```

```
            head = NULL;
            arr_size--;
        }
        else
        {
            Lv *temp = tail;
            tail = tail->prev;
            free(temp);
            arr_size--;
        }
    }
    else
    {
        arr_size--;
        tail->idx--;

    }
}

int getSize()
{
    return arr_size + 1;

}

void getEle(int z)
{
    if (head == NULL || z > arr_size)
    {
        cout << "No element found\n";
        return;
    }
    else
    {
        Lv *temp = head;
        int j = 0;
        for (int i = 0; i <= arr_size; i++)
        {
            if (i == z)
            {
                cout << "element at index " << z << " is " << temp->a[j] << "\n";
                break;
            }

            if (j < 1000000)
            {
                j++;
            }
            else
            {
                temp = temp->next;
                j = 0;
                j++;
```

```
107             }
108         }
109     }
110 }
111
112 void display_vector()
113 {
114     if (head == NULL)
115     {
116         cout << "Vector is empty!\n";
117     }
118     else
119     {
120         cout << "Vector elements are: ";
121         Lv *temp = head;
122         int j = 0;
123         for (int i = 0; i <= arr_size; i++)
124         {
125             if (j < 1000000)
126             {
127                 cout << temp->a[j] << " ";
128                 j++;
129             }
130             else
131             {
132                 temp = temp->next;
133                 j = 0;
134                 cout << temp->a[j] << " ";
135                 j++;
136             }
137         }
138
139         cout << "\n";
140     }
141 }
142
143 bool isEmpty()
144 {
145     if (head == NULL)
146     {
147         return true;
148     }
149     else
150     {
151         return false;
152     }
153 }
154
155 int main()
156 {
157     clock_t begin = clock();
158     ifstream fin;
159     fin.open("input_1.txt");
```

```
    int q;
    while (fin >> q)
    {
        switch (q)
        {
            case 1:
                int x;
                fin >> x;
                pushBack(x);
                break;
            case 2:
                popBack();
                break;
            case 3:
                cout << getSize() << "\n";
                break;
            case 4:
                int z;
                fin >> z;
                getEle(z);
                break;
            case 5:
                display_vector();
                break;
            case 6:
                cout << "does ector is Empty : ";
                cout << isEmpty() << "\n";
                break;
            default:
                exit(0);
        }
    }

    clock_t end = clock();
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
    cout << time_spent;

    return 0;
}
```

# 3 TEST PLANS:

## 3.1 Approach:

By merging the arrays using linked list.

## 3.2 Features to be tested/not tested:

- pushBack()
- popBack()
- get_element()
- display_vector()
- isEmpty()
- get_size()

Testing this functions when array is empty, contains one element and contains more than one element.

### 3.3 PASS/FAIL CRITERIA:

When no element is inserted:

https://ideone.com/vKfpdu

| INPUT: | OUTPUT: |
|---|---|
| 2 - (when delete operation is Performed) | Vector is empty cannot popBack! |
| 3 - (get size) | 0 |
| 4 - (get element at index 1) | Invalid index |
| 5 - (display vector) | Vector is empty! |
| 6 - (isEmpty) | 1 |

When one element is inserted:

https://ideone.com/ml7y0j

| INPUT | OUTPUT |
|---|---|
| 1 1 - (inserting 1 using pushBack) | |
| 3 - (get_size) | 1 |
| 4 0 - (get element at index 1) | Element at index 0 is 1 |
| 4 2 - (get element at index 2) | Invalid index |
| 5 - (display_vector) | Vector elements are: 1 |
| 6 - (is_empty) | 0 |
| 2 - (popBack) | |
| 5 - (display_vector) | Vector is empty! |

When multiple elements are inserted:

https://ideone.com/ByEXO8

| INPUT | OUTPUT |
|---|---|
| 1 1 - (inserting 1 using pushBack) | |
| 1 2 - (inserting 2 using pushBack) | |
| 1 3 - (inserting 3 using pushBack) | |
| 1 4 - (inserting 4 using pushBack) | |
| 5 - (display_vector) | Vector elements are: 1 2 3 4 |
| 2 - (popBack) | |
| 1 5 - (inserting 5 using pushBack) | |
| 3 - (get_size) | 4 |
| 6 - (is_empty) | 0 |

| | |
|---|---|
| 2 - (popBack) | |
| 1 6 - (inserting 6 using pushBack) | |
| 1 7 - (inserting 1 using pb) | |
| 1 8 - (inserting 8 using pushBack) | |
| 5 - (display_vector) | Vector elements are:1 2 3 6 7 8 |
| 2 - (popBack) | |
| 2 - (popBack) | |
| 2 - (popBack) | |
| 2 - (popBack) | |
| 2 - (popBack) | |
| 5 - (display_vector) | Vector elements are: 1 |
| 2 - (popBack) | |
| 5 - (display_vector) | Vector is empty! |

List of test cases:

TC_1:performing popBack() when size of vector is 0.
https://ideone.com/sWTKK4

TC_2:performing isEmpty() when size of vector is 0.
https://ideone.com/j5pbYS

TC_3:performing getSize() when size of vector is 0.
https://ideone.com/gbW8HY

TC_4:performing getEle() when size of vector is 0.
https://ideone.com/XkTLD0

TC_5:performing display_vector() when size of vector is 0.
https://ideone.com/vssqhD

TC_6:performing pushBack() when size of vector is 0.
https://ideone.com/6UMN2b

TC_7:performing pushBack() multiple times.
https://ideone.com/aYdu59

TC_8:performing popBack() multiple times.

https://ideone.com/nn7L05

TC_9:performing getEle() when elements are present.
https://ideone.com/Mgfma0

TC_10:performing getEle() and popBack().

https://ideone.com/XGUlhn

TC_11:performing display_vector() and popBack().
https://ideone.com/wtSpkh

TC_12:performing display_vector() and getEle().
https://ideone.com/TzhjXg

TC_13:performing pushback() and isempty().

https://ideone.com/Hnsz5x

TC_14:performing pushback() and get_size().

https://ideone.com/3nrbjU

TC_15:performing popBack() and get_size().

https://ideone.com/Tki0rB

TC_16:getting last element using get_size() and getEle().
https://ideone.com/wDnEuW

TC_17:performing isempty() and pop_back() and getSize().

https://ideone.com/7P3Gum

TC_18:performing isempty() and getEle().

https://ideone.com/quOsWk

TC_19:performing isempty() and getSize().

https://ideone.com/fork/WCOTNc

TC_20:performing all the operations when one element is inserted.

https://ideone.com/ml7y0j

TC_21:performing all the operations when zero elements are inserted.

https://ideone.com/vKfpdu

TC_22:performing all the operations when multiple elements are inserted. https://ideone.com/ByEXO8

TC_23:when 2267349 inputs are given and getSize() fuction called.

```
2267349
execution time:0.943
Process returned 0 (0x0)   execution time : 5.135 s
Press any key to continue.
```

TC_24:when 2267349 inputs are given and calling getSize() after a popBack operation is performed.

```
2267349
2267348
execution time:0.879
Process returned 0 (0x0)   execution time : 4.416 s
Press any key to continue.
```

TC_25:when 2267349 inputs are given and performing get elements after with index above 1000000.

```
element at index 1111111 is 1923452
execution time:0.993
Process returned 0 (0x0)   execution time : 5.221 s
Press any key to continue.
```

# 4 MESUREMENT AND ANALYSIS:

## 4.1 THEORETICAL TIME COMPLEXITYANALYSIS:

| FUNCTION | TIME COMPLEXITY |
|---|---|
| pushBack() | Constant – O(1) |
| popBack() | Constant – O(1) |
| getSize() | Constant – O(1) |
| getEle() | Linear – O(n) |
| isEmpty() | Constant – O(1) |
| display_vector() | Linear – O(n) |

## 4.2 TABULAR DATA FOR MEASURED TIME-TAKEN VS N:

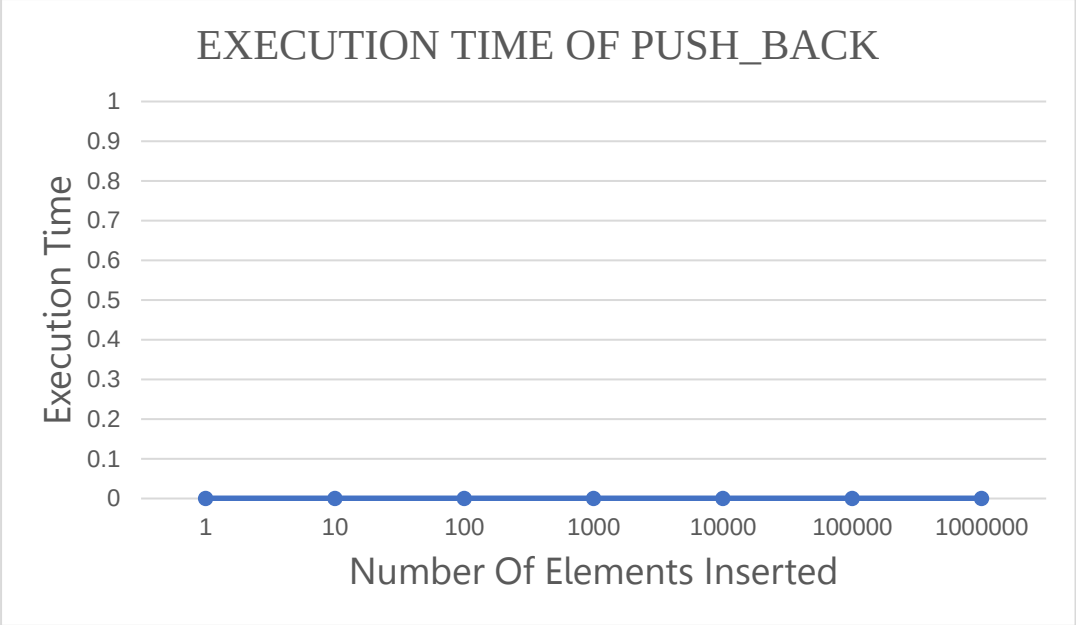
For pushback:

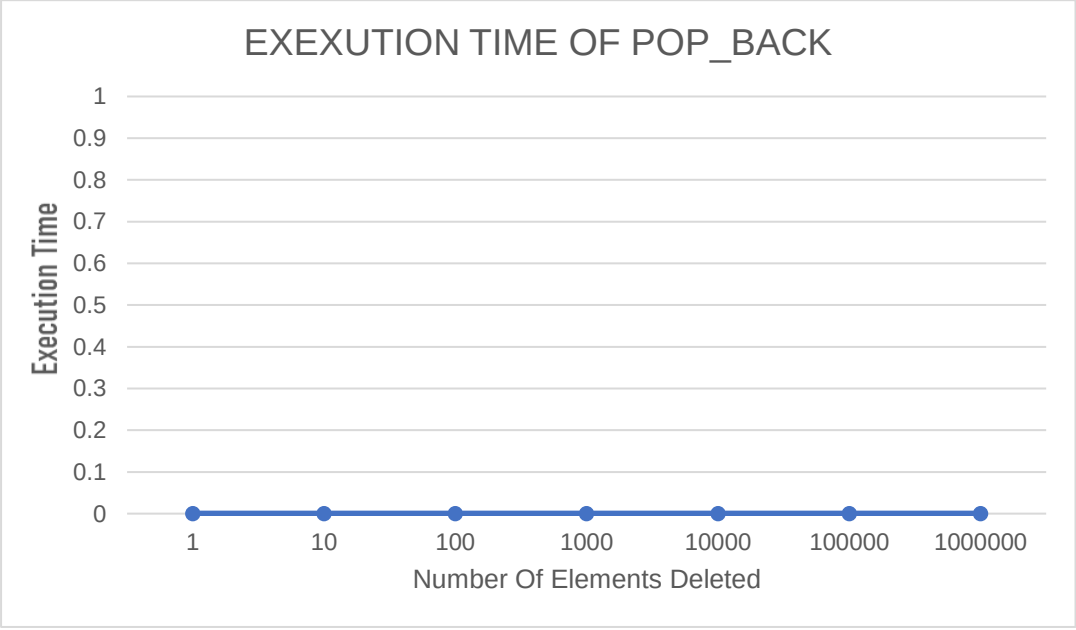
| No of elements pushed | Execution time(ms) |
|---|---|
| 1 | 0 |
| 10 | 0 |
| 100 | 0 |
| 1000 | 0 |
| 10000 | 0 |
| 100000 | 0 |
| 1000000 | 0 |

For popBack:

| No of elements popbacked | Execution time(ms) |
|---|---|
| 1 | 0 |
| 10 | 0 |
| 100 | 0 |
| 1000 | 0 |
| 10000 | 0 |
| 100000 | 0 |
| 1000000 | 0 |

## 4.3 Graph plotting:



## 5 Conclusions:

- We implemented vector using linked list of array
- We've concluded that accessing  an element takes O(n) time complexity,which is one of the drawback of implementing vectors using linked list of arrays.

## 6 Future enhancements:

- In the future, this Data structure can be modified so that it can support different data types.

## 7 Difficulties faced:

- Resolving corner cases!
- Plotting graphs and calculating time complexities

## 8 Reference links:


- https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/
- https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_algorithm.htm
- https://www.cplusplus.com/reference/vector/vector/
- https://guides.lib.berkeley.edu/how-to-write-good-documentation