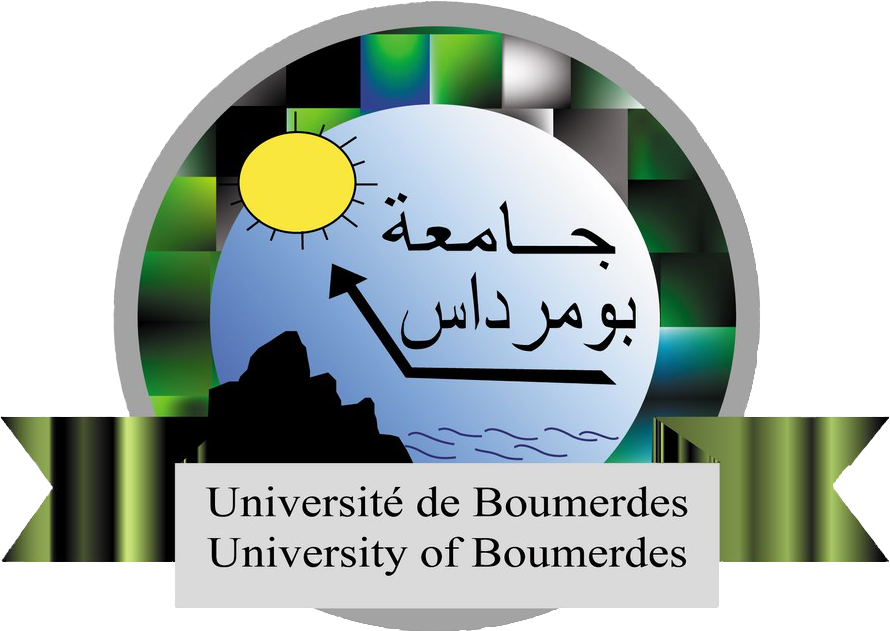
**University M’hamed Bougara Boumerdes**

**Institute of Electrical and Electronics Engineering (ex: INELEC)**



**Algorithms and Data structures laboratory**

LAB N°2

Saturday, March 5, 2022

**Superviser:**  **Students:**

Dr. A. Zitouni Madaoui Zakaria G3 Computer

Maallem Nassim G3 Computer

**Objectives:**

* Write a C++ List of names data structure using array implementation
* Write a C++ List of names data structure using pointer implementation

**Task 1:** ArrayList implementation

In order to implement an array list of names we wrote a struct that represents the list. The struct keeps track of the list size and contains a fixed array for storing the names. After that we implemented few functions to add functionality to insert, delete, search, traverse forward and backward. Finally, we wrote a simple logic in the main() function to display a menu that allows the user to use all the previously built functionality. This was the resulting code:

#include <iostream>

#include <cstring>

using namespace std;

#define MAX\_ELEMENTS 100

#define MAX\_NAME\_SIZE 20

#define DIRECTION\_FORWARD 1

#define DIRECTION\_BACKWARD 0

#include <iostream>

#include <cstring>

using namespace std;

#define MAX\_ELEMENTS 100

#define MAX\_NAME\_SIZE 20

#define DIRECTION\_FORWARD 1

#define DIRECTION\_BACKWARD 0

*//Arraylist struct*

typedef struct

{

unsigned size = 0;

char data[MAX\_ELEMENTS][MAX\_NAME\_SIZE];

} ArrayList;

*//Arraylist prototypes*

void list\_insert(ArrayList &list, char item[], unsigned position);

void list\_remove(ArrayList &list, unsigned position);

int list\_search(const ArrayList &list, const char \*item);

void list\_traverse(const ArrayList &list, unsigned direction);

int main()

{

char user\_option = '\0';

char temp\_str[MAX\_NAME\_SIZE];

unsigned temp\_position = 0;

ArrayList l;

while (user\_option != 'q')

{

cout << "> Please select an option to proceed: \n"

<< " i: insert new element\n"

<< " d: delete an element \n"

<< " s: search for index of an element \n"

<< " f: traverse list forward \n"

<< " b: traverse list backward \n"

<< " q: to quit the program \n";

cin >> user\_option;

switch (user\_option)

{

case 'i': */\*insert\*/*

cout << "> type the position & name to insert: ";

cin >> temp\_position >> temp\_str;

list\_insert(l, temp\_str, temp\_position);

break;

case 'd': */\*delete\*/*

cout << "> type item position to delete: ";

cin >> temp\_position;

list\_remove(l, temp\_position);

break;

case 's': */\*serach and get index\*/*

cout << "> type the name to search for: ";

cin >> temp\_str;

cout << temp\_str << " index is : " << list\_search(l, temp\_str) << "\n";

break;

case 'f': */\*traverse forward\*/*

list\_traverse(l, DIRECTION\_FORWARD);

break;

case 'b': */\*traverse backward\*/*

list\_traverse(l, DIRECTION\_BACKWARD);

break;

case 'q': */\*quit\*/*

cout << "Quitting program ... ";

break;

default:

cout << "Wrong input ! try again !\n";

break;

}

cout << "\n------------------------------------------\n\n";

}

return 0;

}

*//prototypes implementation*

void list\_insert(ArrayList &list, char item[], unsigned position)

{

if (position >= MAX\_ELEMENTS)

cout << "Index out of bound ! position must be less than " << MAX\_ELEMENTS << "\n";

else if (list.size == MAX\_ELEMENTS)

cout << "List is full !\n";

else

{

for (int i = list.size; i > position; i--)

{

strcpy(list.data[i], list.data[i - 1]);

}

strcpy(list.data[position], item);

list.size++; *//incerement size*

}

}

void list\_remove(ArrayList &list, unsigned position)

{

if (position >= list.size)

cout << "element in position " << position << " doesn't exist!\n";

else

{

for (int i = position; i < list.size - 1; i++)

{

strcpy(list.data[i], list.data[i + 1]);

}

list.size--; *//decerement size*

}

}

void list\_traverse(const ArrayList &list, unsigned direction)

{

if (list.size == 0)

cout << "List is empty\n";

else if (direction == DIRECTION\_FORWARD)

{

for (int i = 0; i < list.size; i++)

{

cout << "name[" << i << "]= " << list.data[i] << "\n";

}

}

else

{

for (int i = list.size - 1; i >= 0; i--)

{

cout << "name[" << i << "]= " << list.data[i] << "\n";

}

}

}

int list\_search(const ArrayList &list, const char \*item)

{

for (int i = 0; i < list.size; i++)

{

if (strcmp(item, list.data[i]) == 0)

return i;

}

return -1;

}

**Task 2:** LinkedList (pointer) implementation

In order to implement a linked list of names we wrote a struct that represents a node. Each node contains a **c-string** , a pointer to the next node in the list , and a pointer to the previous node in the list. This way we have what is called a **Doubly Linked List** and that makes traversing the list in both directions very efficient. We also wrote another stuct called **LinkedList** which acts as a wrapper for the list and it keeps track of the list size, first and last node. After that we implemented similar functions as before to add the functionality to insert, delete, search, traverse forward and backward. Finally, we used the same logic in the main() function to display a menu that allows the user to use all the previously built functionality. This was the resulting code:

#include <iostream>

#include <cstring>

using namespace std;

#define MAX\_ELEMENTS 100

#define MAX\_NAME\_SIZE 20

#define DIRECTION\_FORWARD 1

#define DIRECTION\_BACKWARD 0

struct node{

char data[MAX\_NAME\_SIZE];

struct node\* next = nullptr;

struct node\* prev = nullptr;

};

typedef struct node node;

*//LinkedList wrapper struct*

typedef struct

{

unsigned size = 0;

node\* first = nullptr;

node\* last = nullptr;

} LinkedList;

*//Arraylist prototypes*

void list\_insert(LinkedList &list, char item[], unsigned position);

void list\_remove(LinkedList &list, unsigned position);

int list\_search(const LinkedList &list, const char \*item);

void list\_traverse(const LinkedList &list, unsigned direction);

int main()

{

*//same as previous main*

}

*//prototypes implementation*

void list\_insert(LinkedList &list, char item[], unsigned position)

{

node \*temp\_1 = list.first;

if( position > list.size || position < 0){ */\*Wrong input\*/*

cout << "no such position \n";

return; *//do not increment, retuern directly*

}

else if (list.first == nullptr && position == 0){ */\*inserting for first time\*/*

list.first = new node; *// create new node*

strcpy(list.first->data, item); *// put data in new node*

list.first->next = list.first->prev = nullptr;

}

else if(position == 0){ */\*inserting at index 0\*/*

node \*temp\_2 = new node; *// create new node*

strcpy(temp\_2->data, item); *// put data in new node*

temp\_2->prev = nullptr;

temp\_2->next = temp\_1;

temp\_1->prev = temp\_2;

list.first = temp\_2; *//keep track of first element*

}

else{ */\*inserting at index greater than 0 \*/*

for (int i = 1; i < position ; i++) */\*get node[position-1]\*/*

{

temp\_1 = temp\_1->next;

}

node \*temp\_2 = new node; *// create new node*

strcpy(temp\_2->data, item); *// put data in new node*

temp\_2->next = temp\_1->next;

temp\_2->prev = temp\_1;

temp\_1->next = temp\_2;

if (position == list.size) list.last = temp\_2; *//track last element aswell*

else temp\_2->next->prev = temp\_2; *// if next element is not null correct it's backward link*

}

list.size++; *//incerement size*

}

void list\_remove(LinkedList &list, unsigned position)

{

node \*temp\_1 = list.first;

if( position >= list.size || position < 0){ */\*Wrong input\*/*

cout << "no such position \n";

return; *//do not decrement, retuern directly*

}

else if(position == 0){ */\* removing element at index 0\*/*

node \*temp\_2 = temp\_1->next;

list.first = temp\_2; *//keep track of first element when deleting*

delete temp\_1;

if(list.size != 1) temp\_2->prev = nullptr; *//correct back link if first element is not the last*

}

else{ */\*deleting element at index greater than 0 \*/*

for (int i = 0; i < position ; i++) */\*get node[position]\*/*

{

temp\_1 = temp\_1->next;

}

node \*temp\_0 = temp\_1->prev; *// get prev node pointer*

node \*temp\_2 = temp\_1->next; *// get next node pointer*

temp\_0->next = temp\_2;

delete temp\_1;

if (position == (list.size-1)) list.last = temp\_0; *//keep track of last when deleting*

else temp\_2->prev = temp\_0; *//if next element isn't null correct it's back link*

}

list.size--; *//decerement size*

}

void list\_traverse(const LinkedList &list, unsigned direction)

{

int index;

if(list.size == 0) cout << "List is empty\n";

else if (direction == DIRECTION\_FORWARD)

{

node\* temp = list.first;

index = 0;

while (temp != nullptr)

{

cout << "element[" << index++ << "]= " << temp->data << "\n";

temp = temp->next;

}

}

else

{

node\* temp = list.last;

index = list.size;

while (temp != nullptr)

{

cout << "element[" << --index << "]= " << temp->data << "\n";

temp = temp->prev;

}

}

}

int list\_search(const LinkedList &list, const char \*item)

{

node\* temp = list.first;

int index = 0;

while (temp != nullptr)

{

if(strcmp(temp->data, item) == 0) return index;

temp = temp->next; *//traverse forward*

index++;

}

return -1;

}

**Conclusion:**

By the end of this experiment, we come to the conclusion that an array implementation of a list is very simple to write (required less logic), but it imposes the restriction of fixed size. On the other hand, the pointer implementation is more complex and takes more memory (next pointer and name) but it allows for a flexible size and allocates memory only when needed.