A library for linked lists

Dascalu Dorian

Group 10105 B
CE
2016

Contents

Chapter 1

Problem Statement

1.1 A library for linked lists

The library should implement singly and doubly linked lists.

The operations provided by the library should be :

- initialisation of an empty list
- adding a value at the beginning and at the end
- inserting an item at a specified position
- removing an item from a specified position
- computing the length of a list
- appending two lists

This document is typeset using LATEX [1].

Chapter 2

Pseudocode

2.1 Test generator for linked lists

```
1: function
2:     Initialize the length
3:     Initialize the val
4:     Initialize the pos
5:     Create and allocate memory for head_first using init_emptylist function
6:     Create and allocate memory for head_second using init_emptylist function
7:     for i ← 1,length do
8:         val ← random
9:         Push last val in head_first using push_last function
10:        val ← random
11:        Push first val in head_second using push_first function
12:    end for
13:    pos ← random
14:    Print initial list using print_list function
15:    Delete the element from position pos using delete_pos function
16:    Print the new element from position pos using pop_pos funciton
17:    Print list using print_list function
18:
19:    val ← random
20:    pos ← random
21:    Insert val on position pos using push_pos function
22:    Print the new element from position pos using pop_pos function
23:
24:    Print the length of list using length_list
25:    Print list using print_list function
26:
27:    Append the two list using append_lists
28:    Print the appended lists using print_list function
29:    Print the length of appended lists using appended_lists function
30: end function
```

2.2 Print a list

```
1: function
2:     if head->next != NULL then
3:         Create and allocate memory for current node
4:         current ← head
5:
6:         do
7:             current ← current->next
8:             Print current->data
9:         while current->next != NULL
10:    else
11:        Print "The list is empty"
12:    end if
13: end function
```

2.3 Return an element from a given position

```
1: function
2:     Create and allocate memory for current node
3:     current ← head
4:     Initialize the iterator with 0
5:     while iterator < position do
6:         current ← current->next
7:         iterator ← iterator + 1
8:     end while
9: return current->data
10: end function
```

2.4 Return a pointer of linked list

```
1: function
2:     Creates and allocates memory for the head of list
3:     head->data ← NULL
4:     head->next ← NULL
5:     [2] head->prev ← NULL
6: return head
7: end function
```

2.5 Add a new element on the first position of list

```
1: function
2:     Creates and allocates memory for a new node
3:     new_element->data ← val
4:     new_element->next ← head->next
5:     [2]new_element->prev ← head
6:     head->next ← new_element
7: end function
```

2.6 Add a new element on the last position of list

```
1: function
2:     Creates and allocates memory for a new node
3:     Creates and allocates memory for current node
4:     current ← head
5:     while current->next != NULL do
6:         current ← current->next
7:     end while
8:     current->next ← new_element
9:     new_element->data ← val
10:    new_element->next ← NULL
11:    [2]new_element->prev ← current
12: end function
```

2.7 Add a new element on the specified position of list

```
1: function
2:     Create and allocate memory for current node
3:     Initialize the iterator with 0
4:     current ← head
5:     while iterator < position-1 do
6:         current ← current->next
7:         iterator ← iterator + 1
8:     end while
9:     Create and allocate memory for added_element node
10:    added_element->next ← current->next
11:    [2]added_element->prev ← current
12:    current->next ← added_element
13:    added_element->data ← val
14: end function
```

2.8 Delete an element on the specified position of list

```
1: function
2:     Create and allocate memory for current node
3:     Create and allocate memory for deleted node
4:     Initialize the iterator with 0
5:     current ← head
6:     while iterator < position-1 do
7:         current ← current->next
8:         iterator ← iterator + 1
9:     end while
10:    deleted_node ← current->next
11:    current->next ← deleted_node->next
12:    [2]deleted_node->next->prev ← current
13:    free(deleted_node)
14: end function
```

2.9 Return the length of list

```
1: function
2:     Initialize the length with 0
3:     current ← head
4:     while current->next != NULL do
5:         current ← current->next
6:         length ← length + 1
7:     end while
8: return length
9: end function
```

2.10 Append two linked lists

```
1: function
2:     current ← head_list1
3:     while current->next != NULL do
4:         current ← current->next
5:     end while
6:     current->next ← head_list2->next
7:     [2]head_list2->next->prev ← current
8: end function
```

Chapter 3

Aplication design

3.1 The high level architectural overview of the application

The application should implement operations with singly and doubly linked lists.
It is divided into four modules :

- main.c
- singly_functions.h
- doubly_functions.h
- functions.c

The "main.c" module contain the test generator function.
The test generator function call functions from "singly_functions.h" and "doubly_functions.h", defined in "functions.c".
The output is showed in the linked lists output file.

3.2 The specification of the input

The aplication doesn't have specification of the input, because it has an independent test generator function.

3.3 The specification of the output

The output is represented by the linked lists output file.
The test generator function use all functions from modules and print the results in linked lists output file.
The results is showed in a easy way to understand.

3.4 All the modules and their description

3.4.1 Main.c

Libraries 2. : A library for linked lists.
Represent the main modules in the aplication.

3.4.2 Singly_functions.h

C library for operations with singly linked lists.

Implements operations as initialisation of an empty list, adding a value at the beginning and at the end, inserting an item at a specified position, removing an item at a specified position, computing the length of a list and appending two lists for singly linked lists.

3.4.3 Doubly_functions.h

C library for operations with doubly linked lists.

Implements operations as initialisation of an empty list, adding a value at the beginning and at the end, inserting an item at a specified position, removing an item at a specified position, computing the length of a list and appending two lists for doubly linked lists.

3.4.4 Functions.c

C library implementation for operations with singly and doubly linked lists.

Implements operations as initialisation of an empty list, adding a value at the beginning and at the end, inserting an item at a specified position, removing an item at a specified position, computing the length of a list and appending two lists for singly and doubly linked lists.

3.5 Data structure documentation

3.5.1 Doubly struct

Data fields

- int data
  An integer variable for storage the data in linked list.
- struct Doubly *next
  The link to next element.
- struct Doubly *prev
  The link to previous element.

Detailed description

Structure for doubly type of linked lists.

3.5.2 Singly struct

Data fields

- int data
  An integer variable for storage the data in linked list.
- struct Doubly *next
  The link to next element.

Detailed description

Structure for singly type of linked lists.

3.6 The functions grouped by modules

3.6.1 Main.c

Data structures

- struct Singly
- struct Doubly

Functions

void doubly_test (FILE *f)

Parameters
***f** pointer to the file for output results.

Returns
The test results.

Description
length : a variable for length of list, default at 2000 elements.
val : a variable to enter values in list.
pos : a variable for positions in lists.

Create two lists using doubly_push_last and doubly_push_first functions with length value of "length".

Print the initial list and the list after we delete the element.
Print the element from position "pos" (generated with C random function), before and after we delete the element.

Add the a new element "val" on position "pos", both generated with C random function.
Print the element from position "pos".

Print the length of the list and the list.

Append the two list, print the result and lengh of appended lists.

void singly_test (FILE *f)

Parameters
***f** pointer to the file for output results.

Returns
The test results.

Description
length : a variable for length of list, default at 2000 elements.
val : a variable to enter values in list.
pos : a variable for positions in lists.

Create two lists using singly_push_last and singly_push_first functions with length value of "length".

Print the initial list and the list after we delete the element.
Print the element from position "pos" (generated with C random function), before and after we delete the element.

Add the a new element "val" on position "pos", both generated with C random function.
Print the element from position "pos".

Print the length of the list and the list.

Append the two list, print the result and lengh of appended lists.

int main ()

Main function.

Function call tests generator for singly and doubly linked lists giving tests with operation imported from "singly_functions.h" and "doubly_functions.h".

Opens a text file for writing in appending mode. If it does not exist, then a new file is created. The program will start appending content in the existing file content.

Intializes random number generator.

Uses the singly_test and doubly_test, tests generator function.

3.6.2 Singly_functions.h

Data structures

- typedef struct Singly

Functions

void singly_print_list (Singly *head, FILE *f)
Print a singly linked list.

int singly_pop_pos (Singly *head, int pos)
Return an element from a given position.

Singly* singly_init_emptylist ()
Return a pointer of type singly linked list.

void singly_push_first (Singly *head, int val)
Add a new element on the first position of list.

void singly_push_last (Singly *head, int val)
Add a new element on the last position of list.

void singly_push_pos (Singly *head, int pos, int val)
Add a new element on the specified position of list.

void singly_delete_pos (Singly *head, int pos)
Delete an element on the specified position of list.

int singly_length_list (Singly *head)
Return the length of list.

void singly_append_lists (Singly *head_list1, Singly *head_list2)
Append two singly lists.

3.6.3 Doubly_functions.h

Data structures

- typedef struct Doubly

Functions

void doubly_print_list (Doubly *head, FILE *f)
Print a doubly linked list.

int doubly_pop_pos (Doubly *head, int pos)
Return an element from a given position.

Doubly* doubly_init_emptylist ()
Return a pointer of type doubly linked list.

void doubly_push_first (Doubly *head, int val)
Add a new element on the first position of list.

void doubly_push_last (Doubly *head, int val)
Add a new element on the last position of list.

void doubly_push_pos (Doubly *head, int pos, int val)
Add a new element on the specified position of list.

void doubly_delete_pos (Doubly *head, int pos)
Delete an element on the specified position of list.

int doubly_length_list (Doubly *head)
Return the length of list.

void doubly_append_lists (Doubly *head_list1, Doubly *head_list2)
Append two doubly lists.

3.6.4 Functions.c

Functions

void singly_print_list (Singly *head, FILE *f)

Parameters
***head** pointer to the first element of the list.
***f** pointer to the file for output results.

Description
If the list is not empty, create a new node which will go through the list from the beginning.
Print each element until the list ends.

int singly_pop_pos (Singly *head, int pos)

Parameters
***head** pointer to the first element of the list.
pos represent the position from where displays the value.

Returns
The value of element from position "pos".

Description
With a "current" node go through the list to position "pos".
Return the value of "current" node.

Singly* singly_init_emptylist ()

Returns
Return a pointer of type singly linked list.

Description
Creates and allocates memory for a new node.
The list will be empty and the node will point to NULL.

void singly_push_first (Singly *head, int val)

Parameters
***head** pointer to the first element of the list.
val represent the value that will be added.

Description
Creates and allocates memory for a new node.
Gives value to the new node.
The new node will point to second element and the head of list will point to the new node.

void singly_push_last (Singly *head, int val)

Parameters
***head** pointer to the first element of the list.
val represent the value that will be added.

Description
Creates and allocates memory for a new node.
Gives value to the new node.
With a "current" node go through the list until the end.
The "current" node will point to the new node.
The new node will point to NULL.

void singly_push_pos (Singly *head, int pos, int val)

Parameters
***head** pointer to the first element of the list.
val represent the value that will be added.
pos represent the position where the value will be added.

Description
Creates and allocates memory for a new node.
With a "current" node go through the list to position "pos".
The new node will point to the "current" next element.
The "current" node will point to the new node. Gives value to the new node.

void singly_delete_pos (Singly *head, int pos)

Parameters

***head** pointer to the first element of the list.
pos represent the position where the value will be deleted.

Description

Creates and allocates memory for a deleted-node.
With a "current" node go through the list to position "pos".
The deleted-node will be "current" next element.
"Current" node will point to deleted-node next element.
Free the deleted-node memory.

int singly_length_list (Singly *head)

Parameters

***head** pointer to the first element of the list.

Returns

The length of the linked list.

Description

With a "current" node go through the list until the end.
Count each element from list and return the number of elements.

void singly_append_lists (Singly *head_list1, Singly *head_list2)

Parameters

***head_list1** pointer to the first element of the list 1.
***head_list2** pointer to the first element of the list 2.

Description

With a "current" node go to the end of list.
The "current" node will point to the first element of second list.

void doubly_print_list (Doubly *head, FILE *f)

Parameters
***head** pointer to the first element of the list.
***f** pointer to the file for output results.

Description
If the list is not empty, create a new node which will go through the list from the beginning.
Print each element until the list ends.

int doubly_pop_pos (Doubly *head, int pos)

Parameters
***head** pointer to the first element of the list.
pos represent the position from where displays the value.

Returns
The value of element from position "pos".

Description
With a "current" node go through the list to position "pos".
Return the value of "current" node.

Doubly* doubly_init_emptylist ()

Returns
Return a pointer of type doubly linked list.

Description
Creates and allocates memory for a new node.
The list will be empty and the node will point to NULL.

void doubly_push_first (Doubly *head, int val)

Parameters
***head** pointer to the first element of the list.
val represent the value that will be added.

Description
Creates and allocates memory for a new node.
Gives value to the new node.
The new node will point to second element and to the head of list.
The head of list will point to the new node.

void doubly_push_last (Doubly *head, int val)

Parameters
***head** pointer to the first element of the list.
val represent the value that will be added.

Description
Creates and allocates memory for a new node.
Gives value to the new node.
With a "current" node go through the list until the end.
The "current" node will point to the new node.
The new node will point to "current" node and to NULL.

void doubly_push_pos (Doubly *head, int pos, int val)

Parameters
***head** pointer to the first element of the list.
val represent the value that will be added.
pos represent the position where the value will be added.

Description
Creates and allocates memory for a new node.
With a "current" node go through the list to position "pos".
The new node will point to the "current" next element and to "current" node.
The "current" node will point to the new node. Gives value to the new node.

void doubly_delete_pos (Doubly *head, int pos)

Parameters
***head** pointer to the first element of the list.
pos represent the position where the value will be deleted.

Description
Creates and allocates memory for a deleted-node.
With a "current" node go through the list to position "pos".
The deleted-node will be "current" next element.
"Current" node will point to deleted-node next element.
The element after deleted-node will point to current.
Free the deleted-node memory.

int doubly_length_list (Doubly *head)

Parameters

***head** pointer to the first element of the list.

Returns

The length of the linked list.

Description

With a "current" node go through the list until the end.
Count each element from list and return the number of elements.

void doubly_append_lists (Doubly *head_list1, Doubly *head_list2)

Parameters

***head_list1** pointer to the first element of the list 1.
***head_list2** pointer to the first element of the list 2.

Description

With a "current" node go to the end of list.
The "current" node will point to the first element of second list.
The first element of second list will point to the last element of first list.

Chapter 4

Conclusions

Project goal was to create a library for linked lists. The linked lists could be singly or doubly type.

One of the challenging achievement was making the function for generating non-trivial input data. It had to use all functions and print the results in a easy way to read and undestand.

On the other hand, one of the interesting achievement was gather experience with this kind of working and learning [3]LaTeX and [4]Doxygen, for this.

In short term, this project can be used in daily tasks. For example, it can be used to remember certain tasks, the order to resolves and the number of tasks for a day.

In long term, this project can be used to create and administrate a database with multiple fields.

Chapter 5

References

[1] Leslie Lamport, LaTeX: *A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.

[2] Only for doubly linked lists.

[3] LaTeX project site : https://www.latex-project.org/, accessed in May 2016.

[4] Doxygen site : www.doxygen.org, accessed in May 2016.

Chapter 6

Source Code

6.1 Main.c

```
/// \file main.c
/// \brief Libraries 2. : A library for linked lists.

#include <stdio.h> /*for prinf().*/
#include <stdlib.h>
#include <time.h> /* for using function time (get current time) and macro constant NULL.*/

#include "singly_functions.h" /* the header file containing singly linked list functions.*/
#include "doubly_functions.h" /* the header file containing doubly linked list functions.*/

///\brief Singly type of linked list.
///
///Structure for singly type of linked lists.
struct Singly
{
    int data;///< An integer variable for storage the data in linked list.
    struct Singly *next;///< The link to next element.
};

///\brief Doubly type of linked list.
///
///Structure for doubly type of linked lists.
struct Doubly
{
    int data;///< An integer variable for storage the data in linked list.
    struct Doubly *next;///< The link to next element.
    struct Doubly *prev;///< The link to previous element.
};

///\brief Test generator for singly type structure.
///\param *f pointer to the file for output results.
///\return The test results.
void singly_test(FILE *f)
{
    int i;
    int length=2000; /// length : a variable for length of list, default at 2000 elements.\n
    int val; /// val : a variable to enter values in list.\n
    int pos; /// pos : a variable for positions in lists.\n
```

```
    struct Singly *head_first;
    head_first = singly_init_emptylist();

    struct Singly *head_second;
    head_second = singly_init_emptylist();

    /// Create two lists using singly_push_last and singly_push_first functions
    /// with length value of "length".\n
     for(i=1;i<=length;i++)
    {
        val = rand();
        singly_push_last(head_first, val);

        val = rand();
        singly_push_first(head_second, val);
    }

    pos = rand()%2000+1;

    /// Print the initial list and the list after we delete the element.\n
    /// Print the element from position "pos" (generated with C random function),
    /// before and after we delete the element.
    fprintf(f,"Initial singly linked list : \n");
    singly_print_list(head_first, f);
    fprintf(f,"\nThe %d-th element that will be deleted : %d\n", pos, singly_pop_pos(head_first, p
    singly_delete_pos(head_first, pos);
    fprintf(f,"The new element from position %d : %d\n\n", pos, singly_pop_pos(head_first,pos));
    fprintf(f,"List : \n");
    singly_print_list(head_first, f);

    fprintf(f,"-------------------------------------------------------\n");

    pos = rand()%2000+1;
    val = rand();

    /// Add the a new element "val" on position "pos", both generated with C random function.\n
    /// Print the element from position "pos".
    singly_push_pos(head_first, pos, val);
    fprintf(f,"Element %d added on position %d\n", val,pos);
    fprintf(f,"The element from position %d : %d\n",pos, singly_pop_pos(head_first, pos));

    /// Print the length of the list and the list.\n
    fprintf(f,"Length of list : %d\n\n", singly_length_list(head_first));
    fprintf(f,"List : \n");
    singly_print_list(head_first, f);
    fprintf(f,"-------------------------------------------------------\n");

    /// Append the two list, print the result and lengh of appended lists.\n
    singly_append_lists(head_first, head_second);
    fprintf(f,"Length of the appended lists : %d\n\n", singly_length_list(head_first));
    fprintf(f,"Appended Lists : \n");
    singly_print_list(head_first, f);

}

///\brief Test generator for doubly type structure.
///\param *f pointer to the file for output results.
```

```
///\return The test results.
void doubly_test(FILE *f)
{
    int i;
    int length=2000; /// length : a variable for length of list, default at 2000 elements.\n
    int val; /// val : a variable to enter values in list.\n
    int pos; /// pos : a variable for positions in lists.\n

    struct Doubly *head_first;
    head_first = doubly_init_emptylist();

    struct Doubly *head_second;
    head_second = doubly_init_emptylist();

     /// Create two lists using doubly_push_last and doubly_push_first functions
     /// with length value of "length".\n
     for(i=1;i<=length;i++)
    {
        val = rand();
        doubly_push_last(head_first, val);

        val = rand();
        doubly_push_first(head_second, val);
    }

    pos = rand()%2000+1;

    /// Print the initial list and the list after we delete the element.\n
    /// Print the element from position "pos" (generated with C random function),
    /// before and after we delete the element.
    fprintf(f,"Initial doubly linked list : \n");
    doubly_print_list(head_first, f);
    fprintf(f,"\nThe %d-th element that will be deleted : %d\n", pos, doubly_pop_pos(head_first, p
    doubly_delete_pos(head_first, pos);
    fprintf(f,"The new element from position %d : %d\n\n", pos, doubly_pop_pos(head_first,pos));
    fprintf(f,"List : \n");
    doubly_print_list(head_first, f);

    fprintf(f,"-------------------------------------------------------\n");

    pos = rand()%2000+1;
    val = rand();
    /// Add the a new element "val" on position "pos", both generated with C random function.\n
    /// Print the element from position "pos".
    doubly_push_pos(head_first, pos, val);
    fprintf(f,"Element %d added on position %d\n", val, pos);
    fprintf(f,"The element from position %d : %d\n",pos, doubly_pop_pos(head_first, pos));

    /// Print the length of the list and the list.\n
    fprintf(f,"Length of list : %d\n\n", doubly_length_list(head_first));
    fprintf(f,"List : \n");
    doubly_print_list(head_first, f);
    fprintf(f,"-------------------------------------------------------\n");

    /// Append the two list, print the result and lengh of appended lists.\n
    doubly_append_lists(head_first, head_second);
    fprintf(f,"Length of the appended lists : %d\n\n", doubly_length_list(head_first));
    fprintf(f,"Appended Lists : \n");
```

```
    doubly_print_list(head_first, f);

}

///\brief Main function.
///
///Function call tests generator for singly and doubly linked lists giving tests
/// with operation imported from "singly_functions.h" and "doubly_functions.h".\n
int main()
{
    /// Opens a text file for writing in appending mode. If it does not exist, then a new
    ///file is created. The program will start appending content in the existing file content.\n
    FILE *f = fopen("LinkedLists_output.txt","w");

    /// Intializes random number generator.\n
    srand(time(NULL));

    /// Uses the singly_test and doubly_test, tests generator function.\n
    singly_test(f);
    fprintf(f,"\n----------------------------------------------------------------------------\n----------
    doubly_test(f);
    return 0;
}
```

6.2 Singly_functions.h

```
///\file singly_functions.h
///\brief C library for operations with singly linked lists.
///
///Implements operations as initialisation of an empty list, adding a value at
/// the beginning and at the end, inserting an item at a specified position,
///removing an item at a specified position, computing the length of a list
/// and appending two lists for singly linked lists.
#ifndef SINGLY_FUNCTIONS_H_INCLUDED
#define SINGLY_FUNCTIONS_H_INCLUDED

    typedef struct Singly Singly;

    void singly_print_list(Singly *head, FILE *f);
    int singly_pop_pos( Singly *head, int pos );
    Singly* singly_init_emptylist();
    void singly_push_first (Singly *head, int val);
    void singly_push_last(Singly *head, int val);
    void singly_push_pos(Singly *head, int pos, int val);
    void singly_delete_pos (Singly *head, int pos);
    int singly_length_list (Singly *head);
    void singly_append_lists (Singly *head_list1, Singly *head_list2);

#endif // SINGLY_FUNCTIONS_H_INCLUDED
```

6.3 Doubly_functions.h

```
///\file doubly_functions.h
///\brief C library for operations with doubly linked lists.
///
///Implements operations as initialisation of an empty list, adding a value at
```

```
/// the beginning and at the end, inserting an item at a specified position,
/// removing an item at a specified position, computing the length of a list
///and appending two lists for doubly linked lists.
#ifndef DOUBLY_FUNCTIONS_H_INCLUDED
#define DOUBLY_FUNCTIONS_H_INCLUDED

    typedef struct Doubly Doubly;

    void doubly_print_list(Doubly *head, FILE *f);
    int doubly_pop_pos( Doubly *head, int pos );
    Doubly* doubly_init_emptylist();
    void doubly_push_first (Doubly *head, int val);
    void doubly_push_last(Doubly *head, int val);
    void doubly_push_pos(Doubly *head, int pos, int val);
    void doubly_delete_pos (Doubly *head, int pos);
    int doubly_length_list (Doubly *head);
    void doubly_append_lists (Doubly *head_list1, Doubly *head_list2);


#endif // DOUBLY_FUNCTIONS_H_INCLUDED
```

6.4 Functions.c

```
///\file functions.c
///\brief C library implementation for operations with singly and doubly linked lists.
///
///Implements operations as initialisation of an empty list, adding a value at the beginning
///and at the end, inserting an item at a specified position, removing an item at a specified
///position, computing the length of a list and appending two lists for singly and
///doubly linked lists.
#include <stdio.h>
#include <stdlib.h>

#include "singly_functions.h"
#include "doubly_functions.h"

///\brief Singly type of linked list.
///
///Structure for singly type of linked lists.
struct Singly
{
    int data;///< An integer variable for storage the data in linked list.
    struct Singly *next;///< The link to next element.
};

///\brief Print a singly linked list.
///\param *head pointer to the first element of the list.
///\param *f pointer to the file for output results.
void singly_print_list( Singly *head, FILE *f )
{
    /// If the list is not empty, create a new node which will go through the list
    /// from the beginning.\n
    /// Print each element until the list ends.
    if ( head->next != NULL )
    {
        struct Singly *current;
        current = head;
```

```
        do
        {
            current = current->next;
            fprintf(f,"%d ", current->data);

        }while ( current->next != NULL );

        fprintf(f,"\n");
    }
    else
    {
        fprintf(f,"The list is empty\n");
    }
}

///\brief Return an element from a given position
///\param *head pointer to the first element of the list.
///\param pos represent the position from where displays the value.
///\return The value of element from position "pos".
int singly_pop_pos( Singly *head, int pos )
{
    /// With a "current" node go through the list to position "pos".\n
    /// Return the value of "current" node.
    int iterator;
    struct Singly *current;
    iterator = 0;
    current=head;

    while( iterator < pos )
    {
        current = current->next;
        iterator++;
    }
    return current->data;
}

///\brief Return a pointer of type singly linked list.
///\return Return a pointer of type singly linked list.
Singly* singly_init_emptylist()
{
    /// Creates and allocates memory for a new node.\n
    /// The list will be empty and the node will point to NULL.
    struct Singly* head;
    head = (Singly*)malloc(sizeof(Singly));
    head->data = NULL;
    head->next = NULL;

    return head;
}

///\brief Add a new element on the first position of list.
///\param *head pointer to the first element of the list.
///\param val represent the value that will be added.
void singly_push_first ( Singly *head, int val )
{
    /// Creates and allocates memory for a new node.\n
    /// Gives value to the new node.\n
```

```
    /// The new node will point to second element and the head of list
    /// will point to the new node.
    struct Singly *new_Singly;
    new_Singly = (Singly*) malloc(sizeof(Singly));
    new_Singly->data = val;
    new_Singly->next = head->next;
    head->next = new_Singly;
}

///\brief Add a new element on the last position of list.
///\param *head pointer to the first element of the list.
///\param val represent the value that will be added.
void singly_push_last( Singly *head, int val )
{
    /// Creates and allocates memory for a new node.\n
    /// Gives value to the new node.\n
    /// With a "current" node go through the list until the end.\n
    /// The "current" node will point to the new node.\n
    /// The new node will point to NULL.
    struct Singly *current;
    struct Singly *new_Singly;
    current = head;

    while( current->next != NULL )
    {
        current = current->next;
    }

    new_Singly = (Singly*) malloc(sizeof(Singly));
    current->next = new_Singly;
    new_Singly->data = val;
    new_Singly->next = NULL;
}

///\brief Add a new element on the specified position of list.
///\param *head pointer to the first element of the list.
///\param val represent the value that will be added.
///\param pos represent the position where the value will be added.
void singly_push_pos( Singly *head, int pos, int val )
{
    /// Creates and allocates memory for a new node.\n
    /// With a "current" node go through the list to position "pos".\n
    /// The new node will point to the "current" next element.\n
    /// The "current" node will point to the new node.
    /// Gives value to the new node.\n
    int iterator;
    struct Singly *current;
    struct Singly *added_Singly;
    iterator = 0;
    current=head;

    while( iterator < pos-1 )
    {
        current = current->next;
        iterator++;
    }

    added_Singly = (Singly*) malloc(sizeof(Singly));
```

```
    added_Singly->next = current->next;
    current->next = added_Singly;
    added_Singly->data = val;
}

///\brief Delete an element on the specified position of list.
///\param *head pointer to the first element of the list.
///\param pos represent the position where the value will be deleted.
void singly_delete_pos ( Singly *head, int pos )
{
    /// Creates and allocates memory for a deleted-node.\n
    /// With a "current" node go through the list to position "pos".\n
    /// The deleted-node will be "current" next element.\n
    /// "Current" node will point to deleted-node next element.\n
    /// Free the deleted-node memory.
    int iterator;
    struct Singly *current;
    struct Singly *deleted_node;
    iterator = 0;
    current = head;

    while( iterator < pos-1 )
    {
        current = current->next;
        iterator++;
    }

    deleted_node=current->next;
    current->next=deleted_node->next;
    free(deleted_node);

}

///\brief Return the length of list.
///\param *head pointer to the first element of the list.
///\return The length of the linked list.
int singly_length_list ( Singly *head )
{
    /// With a "current" node go through the list until the end.\n
    /// Count each element from list and return the number of elements.
    int length = 0;
    struct Singly *current;
    current = head;

    while ( current->next != NULL )
    {
        current = current->next;
        length++;
    }

    return length;
}

///\brief Append two singly lists.
///\param *head_list1 pointer to the first element of the list 1.
///\param *head_list2 pointer to the first element of the list 2.
void singly_append_lists ( Singly *head_list1, Singly *head_list2 )
{
```

```
    /// With a "current" node go to the end of list.\n
    /// The "current" node will point to the first element of second list.
    struct Singly *current;
    current = head_list1;

    while( current->next != NULL )
    {
        current = current->next;
    }

    current->next = head_list2->next;
}


///\brief Doubly type of linked list.
///
///Structure for doubly type of linked lists.
struct Doubly
{
    int data;///< An integer variable for storage the data in linked list.
    struct Doubly *next;///< The link to next element.
    struct Doubly *prev;///< The link to previous element.
};

///\brief Print a doubly linked list.
///\param *head pointer to the first element of the list.
///\param *f pointer to the file for output results.
void doubly_print_list( Doubly *head, FILE *f )
{
    /// If the list is not empty, create a new node which will go through the list
    /// from the beginning.\n
    /// Print each element until the list ends.
    if ( head->next != NULL )
    {
        struct Doubly *current;
        current = head;

        do
        {
            current = current->next;
            fprintf(f,"%d ", current->data);

        }while ( current->next != NULL );

        fprintf(f,"\n");
    }
    else
    {
        fprintf(f,"The list is empty\n");
    }
}

///\brief Return an element from a given position
///\param *head pointer to the first element of the list.
///\param pos represent the position where displays the value.
///\return The value of element from position "pos".
int doubly_pop_pos( Doubly *head, int pos )
{
```

```
    /// With a "current" node go through the list to position "pos".\n
    /// Return the value of "current" node.
    int iterator;
    struct Doubly *current;
    iterator = 0;
    current=head;

    while( iterator < pos )
    {
        current = current->next;
        iterator++;
    }
    return current->data;
}

///\brief Return a pointer of type doubly linked list.
///\return Return a pointer of type doubly linked list.
Doubly* doubly_init_emptylist()
{
    /// Creates and allocates memory for a new node.\n
    /// The list will be empty and the node will point to NULL.
    struct Doubly* head;
    head = (Doubly*) malloc(sizeof(Doubly));
    head->data = NULL;

    head->next = NULL;
    head->prev = NULL;

    return head;
}

///\brief Add a new element on the first position of list.
///\param *head pointer to the first element of the list.
///\param val represent the value that will be added.
void doubly_push_first ( Doubly *head, int val )
{
    /// Creates and allocates memory for a new node.\n
    /// Gives value to the new node.\n
    /// The new node will point to second element and to the head of list.\n
    /// The head of list will point to the new node.
    struct Doubly *new_Element;
    new_Element = (Doubly*) malloc(sizeof(Doubly));
    new_Element->data = val;
    new_Element->next = head->next;
    new_Element->prev = head;
    head->next = new_Element;
}

///\brief Add a new element on the last position of list.
///\param *head pointer to the first element of the list.
///\param val represent the value that will be added.
void doubly_push_last( Doubly *head, int val )
{
    /// Creates and allocates memory for a new node.\n
    /// Gives value to the new node.\n
    /// With a "current" node go through the list until the end.\n
    /// The "current" node will point to the new node.\n
    /// The new node will point to "current" node and to NULL.
```

```
    struct Doubly *current;
    struct Doubly *new_Element;
    current = head;

    while( current->next != NULL )
    {
        current = current->next;
    }

    new_Element = (Doubly*) malloc(sizeof(Doubly));
    current->next = new_Element;
    new_Element->data = val;
    new_Element->next = NULL;
    new_Element->prev = current;
}

///\brief Add a new element on the specified position of list.
///\param *head pointer to the first element of the list.
///\param val represent the value that will be added.
///\param pos represent the position where the value will be added.
void doubly_push_pos( Doubly *head, int pos, int val )
{
    /// Creates and allocates memory for a new node.\n
    /// With a "current" node go through the list to position "pos".\n
    /// The new node will point to the "current" next element and to "current" node.\n
    /// The "current" node will point to the new node.
    /// Gives value to the new node.\n
    int iterator;
    struct Doubly *current;
    struct Doubly *added_Element;
    iterator = 0;
    current=head;

    while( iterator < pos-1 )
    {
        current = current->next;
        iterator++;
    }

    added_Element = (Doubly*) malloc(sizeof(Doubly));
    added_Element->next = current->next;
    added_Element->prev = current;
    current->next = added_Element;
    added_Element->data = val;
}

///\brief Delete an element on the specified position of list.
///\param *head pointer to the first element of the list.
///\param pos represent the position where the value will be deleted.
void doubly_delete_pos ( Doubly *head, int pos )
{
    /// Creates and allocates memory for a deleted-node.\n
    /// With a "current" node go through the list to position "pos".\n
    /// The deleted-node will be "current" next element.\n
    /// "Current" node will point to deleted-node next element.\n
    /// The element after deleted-node will point to current.\n
    /// Free the deleted-node memory.
    int iterator;
```

```
    struct Doubly *current;
    struct Doubly *deleted_node;
    iterator = 0;
    current = head;

    while( iterator < pos-1 )
    {
        current = current->next;
        iterator++;
    }

    deleted_node=current->next;
    current->next=deleted_node->next;
    deleted_node->next->prev=current;
    free(deleted_node);

}

///\brief Return the length of list.
///\param *head pointer to the first element of the list.
///\return The length of the linked list
int doubly_length_list ( Doubly *head )
{
    /// With a "current" node go through the list until the end.\n
    /// Count each element from list and return the number of elements.
    int length = 0;
    struct Doubly *current;
    current = head;

    while ( current->next != NULL )
    {
        current = current->next;
        length++;
    }

    return length;
}

///\brief Append two doubly lists.
///\param *head_list1 pointer to the first element of the list 1.
///\param *head_list2 pointer to the first element of the list 2.
void doubly_append_lists ( Doubly *head_list1, Doubly *head_list2 )
{
    /// With a "current" node go the end of list.\n
    /// The "current" node will point to the first element of second list.\n
    /// The first element of second list will point to the last element of first list.
    struct Doubly *current;
    current = head_list1;

    while( current->next != NULL )
    {
        current = current->next;
    }

    current->next = head_list2->next;
    head_list2->next->prev = current;
}
```

Chapter 7

Experiments and results

7.1 Description of output

The method used for testing the aplication follow these steps :

1. create two linked lists, one by pushing first and another by pushing last (push first and push last functions)
2. print the initial state of a list (print list function)
3. delete an element and print the new element from that position (delete pos and pop pos functions)
4. insert an element then print the element from that position (push pos and pop pos functions)
5. print the length of list (length list function)
6. append the two lists (append lists functions)
7. print the appended lists and their length (print list and length list functions)

In order to test that the output is correct, i used two more functions : print a list and print an element from a given position.

Once the program used a function we verify the result :

- create lists -> print lists and the length of lists
- delete an element from a position -> print the new element from the deleted element position
- insert an element on a position -> print the element from that position
- append the two lists -> print the appended lists and their length

The output data represent the verifiifcation of linked lists functions. The output file follow these steps :

1. initial state of a list
2. the element from a position in list, before and after we delete him
3. list resulted
4. an element, and a position, which will be inserted in list
5. the element from that position in list
6. length of list
7. list resulted
8. length of appended lists
9. appended lists

7.2 The results

```
Initial singly linked list :
3305 6226 1792 14155 16389 487 12633 945 17170 26254 30796 8620 13764 5302 22179 25418 919 9388 4

The 224-th element that will be deleted : 8161
The new element from position 224 : 1894

List :
3305 6226 1792 14155 16389 487 12633 945 17170 26254 30796 8620 13764 5302 22179 25418 919 9388 4
_____________________________________________

Element 4752 added on position 1742
The element from position 1742 : 4752
Length of list : 2000

List :
3305 6226 1792 14155 16389 487 12633 945 17170 26254 30796 8620 13764 5302 22179 25418 919 9388 4
_____________________________________________

Length of the appended lists : 4000

Appended Lists :
3305 6226 1792 14155 16389 487 12633 945 17170 26254 30796 8620 13764 5302 22179 25418 919 9388 4

_____________________________________________________
_____________________________________________________

Initial doubly linked list :
21503 9015 25902 15962 7915 179 5458 28370 15898 24017 21948 8595 8152 28746 1749 7996 24265 7366

The 1085-th element that will be deleted : 22308
The new element from position 1085 : 71

List :
21503 9015 25902 15962 7915 179 5458 28370 15898 24017 21948 8595 8152 28746 1749 7996 24265 7366
_____________________________________________

Element 1378 added on position 1330
The element from position 1330 : 1378
Length of list : 2000

List :
21503 9015 25902 15962 7915 179 5458 28370 15898 24017 21948 8595 8152 28746 1749 7996 24265 7366
_____________________________________________

Length of the appended lists : 4000

Appended Lists :
21503 9015 25902 15962 7915 179 5458 28370 15898 24017 21948 8595 8152 28746 1749 7996 24265 7366
```