Marmara University - Faulty of Engineering

Department of Computer Engineering

Data Structures (Autumn 2022)

Submit Date: 18/11/2022

Name : Abdelrahman Zahran

Student ID: 150120998

Arithmetic Calculator using Linked Lists

Sections Of the Report: -

- Section (1): Problem Definition.
- Section (2): Implementation Details.
- Section (3): Test Cases.

Section (1): Definition:-

In this program, we will create an arithmetic calculator that can multiply any 2 numbers of any length (numbers can be infinite) of the same base where base is (2 <= base <= 10). The result of the multiplication operation will be displayed in the main base of the system (Input base) and in base 10 (Decimal System).

Throughout the program we will use linked lists data structure to represent the numbers, apply the multiplication process (operation) and the conversion(operation) process using them.

We will use different algorithms in the program to do the desired operations:

- Insertion (Insert nodes (digits of the number) in the linked list one by one at the end).
- Reversal (Reverse nodes (digits of the number) in the linked list one by one).
- Find Length (Obtain the length of the number (number of digits in the number) will give the length of the linked list as well).
- Traversal (moving through the linked list from the beginning to the end node by node – has lots of important implementations and different usages).
- Multiplication (Apply Elementary School Multiplication Algorithm).
- Conversion (Apply Normal Base Conversion for any base ([2,3,4, ..., 10]) to base 10).
- Print (Traverse the linked list (number) and print the data (digits) stored int it one by one).

Section (2): Implementation Details.:-

```c
//------------------------------------------------------------------------------------------------------//
// Define the structure of the node (pointer variable, data).
struct Node {
    int data;
    struct Node *next;
};
// Declare head of the linked list -> Multiplicand.
struct Node *head1;
// Declare head of the linked list -> Multiplier.
struct Node *head2;
// Declare head of the linked list -> Result.
struct Node *head3;
//------------------------------------------------------------------------------------------------------//
```

This is the structure definition used throughout the program to create nodes and linked lists. Each node has 2 blocks (pointer variable, data).

Pointer variable: address of the next node in the linked list.

Data: one digit of the number of type int.

The main () function:

```c
//------------------------------------------------------------------------------------------------------//
// The main() function -> Execution of the program (start + end) in this function.
int main (int argc,char* argv[]) {
    // head of the linked -> NULL
    head1 = NULL;
    head2 = NULL;
    head3 = NULL;
    // Display the output in the Command Prompt.
    if (argc == 1) {
        printf("Error message!");
    } else if (argc >= 2) {
        FILE *InputFilePtr = fopen(argv[1], "r");
        char Input;
        int digit;
        int SystemBase;
        int counter = 0;
        // Read the Input from the .txt file and format it -> store the numbers in linked lists.
        if (InputFilePtr == NULL) {
            puts("File couldn't be opened!");
            exit(1);
        } else {
            while ((Input = getc(InputFilePtr)) != EOF) {
```

```c
                digit = Input - '0';
                // Move to the next line when the current line is finished.
                if (Input == '\n' || Input == ' ') {
                    counter++;
                } else {
                    switch (counter)
                    {
                    // Store the Multiplicand in a linked list (digit by digit) using insert function.
                    case 0:
                        Insert(&head1, digit);
                        break;
                    // Store the Multiplier in a linked list (digit by digit) using insert function.
                    case 1:
                        Insert(&head2, digit);
                        break;
                    // Store the base of the system in an int variable.
                    case 2:
                        SystemBase = digit;
                        // In case base < 2 || > 10 -> terminate.
                        if (!((SystemBase >= 2) && (SystemBase <= 10))) {
                            printf("The Base Number of the system is Invalid!");
                            exit(1);
                        }
                        break;
                    default:
                        break;
                    }
                }
            }
        }
        // Close the Input File.
        fclose(InputFilePtr);
        // Reverse -> Multiplicand.
        Reverse(&head1);
        // Reverse -> Multiplier.
        Reverse(&head2);
        // Length -> Multiplicand. (m)
        int length1 = FindLength(head1);
        // Length -> Multiplier. (n)
        int length2 = FindLength(head2);
        // Length -> Result (m + n + 1)
        int length3 = length1 + length2 + 1;
        int count = 0;
        // Fill the result Linked List with zeros.
        while (count < length3) {
```

```
            Insert(&head3, 0);
            count++;
        }
        // Multiplication -> Multiplicand * Multiplier = Result.
        Multiplication(&head1, &head2, &head3,SystemBase);
        // Print -> Multiplicand.
        Print(head1);
        printf("\n");
        // Print -> Multiplier.
        Print(head2);
        printf("\n");
        // Print -> Result.
        Print(head3);
        printf("\n");
        // Print -> Multiplicand in base (10).
        Conversion(&head1, SystemBase);
        printf("\n");
        // Print -> Multiplier in base (10).
        Conversion(&head2, SystemBase);
        printf("\n");
        // Print -> Result in base (10).
        Conversion(&head3, SystemBase);
    }
}
//-----------------------------------------------------------------------------------------------------------//
```

The main function is built of 2 parts: -

1) The first part is responsible for reading the input from the file and formatting it using file processing and storing the input (multiplicand:: linked list) – (multiplier:: linked list) – (System Base:: int) + checking the value of base between 2 and 10.

2) The second part: Function Calls + printing statements.

The Insert () Function:

```
//-----------------------------------------------------------------------------------------------------------//
// The Insert() function -> Insert new node into a linked list at the end - Time Complexity = O(n).
void Insert(struct Node **head, int data) {
    // Creation of a new node using malloc (data, NULL).
    struct Node *temp1 = (struct Node*)malloc(sizeof(struct Node));
    temp1 -> data = data;
    temp1 -> next = NULL;
    struct Node *temp2 = *head;
    // In case the linked list is empty.
    if (*head == NULL) {
        *head = temp1;
```

```
    } else {
        // Traverse the Linked List till the last node (digit).
        while (temp2 -> next != NULL) {
            temp2 = temp2 -> next;
        }
        // Move to the next node.
        temp2 -> next = temp1;
    }
}
//---------------------------------------------------------------------------------------------------//
```

The insertion function (Insert a node at the end of the linked list).

Algorithm: -

1. Declare the head pointer and make it NULL.

2. Create a new node with the given data. And make the new node => next as NULL.

   (Because the new node is going to be the last node.)

3. If the head node is NULL (Empty Linked List), make the new node as the head.

4. If the head node is not null, (Linked list already has some elements), find the last node. Make the last node => next as the new node.

Complexity Analysis:

- **Time complexity:** O(N), where N is the number of nodes in the linked list. Since there is a loop from head to end, the function does O(n) work.

(This method can also be optimized to work in O (1) by keeping an extra pointer to the tail of the linked list)

- **Auxiliary Space:** O (1)

The Reverse () Function:

```
/---------------------------------------------------------------------------------------------------//
// The Reverse() function -> Reverse a linked list while traversing it - Time complexity = O(n).
void Reverse(struct Node **head) {
    struct Node *current = *head;
    struct Node *next;
    struct Node *prev = NULL;
    // Traverse the linked list till the last node (digit) + change references of each node (pointer
variables).
    while (current != NULL) {
        next = current -> next;
        current -> next = prev;
```

```
        prev = current;
        current = next;
    }
    *head = prev;
}
//---------------------------------------------------------------------------------------------------//
```

The Reverse function (Iterative Approach): -

The idea is to use three pointers current, prev, and next to keep track of nodes to update reverse links.

- Initialize three pointers **prev** as NULL, current as **head**, and **next** as NULL.
- Iterate through the linked list. In a loop, do the following:
  - Before changing the **next** of current, store the **next** node
    - next = current -> next
  - Now update the **next** pointer of current to the **prev**
    - current -> next = prev
  - Update **prev** as current and current as **next**
    - prev = current
    - current = next

Time Complexity: O(N), Traversing over the linked list of size N.

Auxiliary Space: O (1)

The FindLength () Function:

```
//---------------------------------------------------------------------------------------------------//
// The FindLength() function -> Obtain number of nodes (digits of the number) in a linked list (Number).
int FindLength(struct Node *head) {
    int count = 0;
    struct Node *current = head;
    // In case the list is empty.
    if (head == NULL) {
        count = 0;
        printf("The Linked List is Empty!");
        return;
    } else {
        // Traverse the linked list till the last node (digit) + increment the count (Num. of digits in 
the number).
        while (current != NULL) {
            count++;
```

```
            current = current -> next;
        }
    }
    return count;
}
//-----------------------------------------------------------------------------------------------------//
```

The Conversion () Function:

```
//-----------------------------------------------------------------------------------------------------//
// The Conversion() Function -> convert any number in any base system between[2,10] to the base 10
(Decimal System).
void Conversion(struct Node **head, int SystemBase) {
    struct Node *temp1 = *head;
    struct Node *temp2 = NULL;
    struct Node *temp3 = NULL; //result
    struct Node *temp4 = NULL;
    struct Node *tempMultiply1 = NULL;
    struct Node *tempMultiply2 = NULL;
    struct Node *tempMultiply3 = NULL;
    struct Node *tempMultiply4 = NULL;
    int iteration = 0;
    int lengthC =  2 * FindLength(*head);
    int count = 0;
    // Insert in temp4 (double length of the head(number)) 0 -> 0-> 0-> .....
    while (count < lengthC) {
        Insert(&temp4, 0);
        count++;
    }
    // Store base -> tempMultiply1
    Insert(&tempMultiply1, SystemBase);
    // Store base ^ 0 -> tempMultiply2
    Insert(&tempMultiply2, 1);
    // Reverse -> number(input).
    Reverse(&temp1);
    // In case the linked list is empty.
    if (temp1 == NULL) {
        printf("The Linked List is Empty!");
        return;
    } else {
        // Traverse the linked list node by node -> To do conversion.
        while (temp1 != NULL) {
            // Multiply base pow(eg. 2^5) 1 2 3
            if(iteration) {
                tempMultiply3 = NULL;
                // Insert in tempMultiply3 (double length of the head(number)). 0 -> 0-> 0-> .....
```

```
                count = 0;
                while (count < lengthC) {
                    Insert(&tempMultiply3, 0);
                    count++;
                }
                // Multiply Bases (depending on the length of the number): (2) -> 1 (1) -> base (3) ->
result1
                Multiplication(&tempMultiply2, &tempMultiply1, &tempMultiply3, 10);
                // Multiply Bases (depending on the length of the number): (2) -> result1 (1) -> base (3)
-> result2
                tempMultiply2 = tempMultiply3;
                Reverse(&tempMultiply2);
                // tempMultiply2 = 1 initially -> tempMultiply3 = tempMultiply2 * base;
            }
            // Insert digits of the number in tempMultiply4
            tempMultiply4 = NULL;
            Insert(&tempMultiply4, temp1 -> data);
            tempMultiply3 = NULL;
            // Insert in tempMultiply3 (double length of the head(number)). 0 -> 0-> 0-> .....
            count = 0;
            while (count < lengthC) {
                Insert(&tempMultiply3, 0);
                count++;
            }
            // Multiply digits with the corresponding value in terms of multiply bases.
            // (2) -> bases multiplication (4) -> digits of the number (3) -> result
            Multiplication(&tempMultiply2, &tempMultiply4, &tempMultiply3, 10);
            Reverse(&tempMultiply2);
            Reverse(&tempMultiply3);
            // Add the results of each multiplication one by one (node by node).
            AdditionTo(&tempMultiply3, &temp4);
            // Move to the next node
            temp1 = temp1 -> next;
            iteration++;
        }
    }
    // Reverse -> result
    Reverse(&temp4);
    // Remove additional zeros -> in case found.
    while (temp4 -> data == 0) {
        struct Node *temp = temp4;
        // Move to the next node
        temp4 = temp4 -> next;
        free(temp);
        // In case the linked list is empty -> terminate.
```

```
        if (temp4 == NULL)
        break;
    }
    // In case the linked list is empty -> Insert(0).
    if (temp4 == NULL)
    Insert(&temp4,0);

    Print(temp4);
}
//--------------------------------------------------------------------------------------------//
```

The Conversion function is used to convert a number at any base from [2, 10] to the decimal base system which is system base 10.

The logic behind the conversion:

Convert from any Base System to Decimal System

Steps: -

Step 1 − Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

Step 2 − Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.

Step 3 − Sum the products calculated in Step 2. The total is the equivalent value in decimal.

Other Base System to Decimal Conversion

Binary to Decimal:

In this conversion, binary number to a decimal number, we use multiplication method, in such a way that, if a number with base n has to be converted into a number with base 10, then each digit of the given number is multiplied from MSB to LSB with reducing the power of the base. Let us understand this conversion with the help of an example.

Example 1. Convert $(1101)_2$ into a decimal number.

Solution: Given a binary number $(1101)_2$.

Now, multiplying each digit from MSB to LSB with reducing the power of the base number 2.

$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 8 + 4 + 0 + 1$

$= 13$

Therefore, $(1101)_2 = (13)_{10}$

Octal to Decimal:

To convert octal to decimal, we multiply the digits of octal number with decreasing power of the base number 8, starting from MSB to LSB and then add them all together.

Example 2: Convert 22_8 to decimal number.

Solution: Given, 22_8

$2 \times 8^1 + 2 \times 8^0$

$= 16 + 2$

$= 18$

Therefore, $22_8 = 18_{10}$

Hexadecimal to Decimal:

Example 3: Convert 121_{16} to decimal number.

Solution: $1 \times 16^2 + 2 \times 16^{1+} 1 \times 16^0$

$= 16 \times 16 + 2 \times 16 + 1 \times 1$

$= 289$

Therefore, $121_{16} = 289_{10}$

In the Implemented Conversion method: -

We take the linked list that stores the number that we want to convert to decimal and the base of its system.

First, Reverse the linked list (Number), then we traverse that linked list, while traversing we multiply the bases (power) depends on the placement of the digit in the number.

Ex: 5301 = 5->3->0->1------->Reverse ---------> 1->0->3->5 Their placement (0,1,2,3) our base is 6

In the multiplication of the bases using value of their placement, we created 3 linked lists, and we will use the multiplication method to handle the multiplication operation.

- In list one (1) -> 6 we store the base.
- In list two (2) -> 1 we store the value of 6 at placement 0 which is 6^0 = 1
- In list three we store the value obtained from the multiplication (linked list).

After that we update the linked list (1) with the linked list (3) (OVERWRITE!)

Then we apply the multiplication again using a digit from a node of the linked list of the number by the result of the multiplication of the bases and the placement order.

After this we call the additionTo() function to sum the result of each multiplication done in the loop of conversion function.

Time Complexity = O(n^2)

Space Complexity = O(n)

The AdditionTo ():

```
//---------------------------------------------------------------------------------------------------//
// The AdditionTo() function -> add digits (data of the nodes) after the multiplication in the conversion
function is processed.
// -> addition of 2 numbers stored in linked lists to each other (node by node).
void AdditionTo(struct Node**head1, struct Node**head2){
    struct Node* temp1 = *head1;
    struct Node* temp2 = *head2;
    int sum = 0;
    int carry = 0;
    int digit = 0;
    // Traverse the linked list till the last node.
    while (temp1 != NULL) {
        // Fill the second linked list with 0 -> in case it is empty.
        if (temp2 -> next == NULL) {
            Insert(&temp2, 0);
        }
        // First digit of the linked list (1) + First digit of the linked list (2).
        sum = temp1 -> data + temp2 -> data + carry;
        digit = sum % 10; // Obtain digits of base 10
        carry = sum /10;  // Obtain the carry to process the sum operation in a sufficient way.
        temp1 = temp1 -> next;
        temp2 -> data = digit;
        temp2 = temp2 -> next;
    }
    if (carry != 0) {
        temp2 -> data = carry;
    }
}
//---------------------------------------------------------------------------------------------------//
```

The AdditionTo function (Iterative Approach): -

- We will firstly make two linked lists from the given two linked lists.
- Then, we will run a loop till both linked lists become empty.

- in every iteration, we keep the track of the carry.
- In the end, if carry > 0, that means we need extra node at the start of the resultant list to accommodate this carry.

Time Complexity: O (m + n) where m and n are the sizes of given two linked lists.

Auxiliary Space: O (m + n)

The Multiplication ():

```
//------------------------------------------------------------------------------------------//
// The multiplication() function -> Take 2 numbers (2 linked lists) -> input and multiply them using
Elementary School Multiplication Algorithm.
void Multiplication(struct Node **head1, struct Node **head2, struct Node **head3, int SystemBase) {
    struct Node *temp1 = *head1;
    struct Node *temp2 = *head2;
    struct Node *temp3 = *head3;
    struct Node *temp4 = *head3;

    int multiplication = 0;
    int carry = 0;
    while (temp2) {
        carry = 0;
        // Each time we loop or iterate we start from the next node other than the previous one of it that
we used in the previous iteration.
        temp4 = temp3;
        temp1 = *head1;
        while (temp1) {
            // multiply digit by digit + carry in case found from a previous multiplication.
            multiplication = (temp1 -> data) * (temp2 -> data) + carry + (temp4 -> data);
            // Store result (remainder -> mod) in Result linked list.
            temp4 -> data = multiplication % SystemBase;
            // Update carry value.
            carry = multiplication / SystemBase + temp4 -> data / SystemBase;
            // Make each node in the result has only one digit in the result inked list.
            temp4 -> data = temp4 -> data % 10;
            // Move to the next node in temp4 linked list.
            temp4 = temp4 -> next;
            // Move to the next node in temp1 linked list.
            temp1 = temp1 -> next;

        }
        if (carry > 0) {
            temp4 -> data += carry;
        }
```

```
        // Move to the next node in temp3 linked list.
        temp3 = temp3 -> next;
        // Move to the next node in temp2 linked list.
        temp2 = temp2 -> next;
    }
    // Reverse -> To obtain numbers in the correct order.
    Reverse(head1);
    Reverse(head2);
    Reverse(head3);

    // This loop removes -> additional zeros in the beginning of the result linked list in case found.
    while ((*head3) -> data == 0) {
        struct Node *temp = *head3;
        *head3 = (*head3) -> next;
        free(temp);
        if (*head3 == NULL)
        break;
    }
    // In case Result is empty -> Insert (0).
    if (*head3 == NULL)
    Insert(head3,0);
}
//------------------------------------------------------------------------------------------------------//
```

Algorithm: -

Reverse both linked lists

Make a linked list of maximum result size (m + n + 1)

For each node of one list

For each node of second list

a) Multiply nodes

b) Add digit in result LL at corresponding position

c) Now resultant node itself can be higher than one digit

d) Make carry for next node

Leave one last column means next time start

From next node in result list

Reverse the resulted linked list

Output:

1) Step one

First List is: 5->3->0->1

Second List is: 4->1->2->0

2) Step two

First List is: 1->0->3->5

Second List is: 0->2->1->4

3) Step three

First List is: 5->3->0->1

Second List is: 0-…….

4) Step four

First List is: 5->3->0->1

Second List is: …->2…….

And goes on like this following the algorithm rules and obtaining both digits and carry + updating the carry each time.

Resultant list is: 3->5->1->2->4->1->2->0

Time complexity: O (M + N) where M and N are size of given two linked lists respectively

Auxiliary Space: O (1)

The Print ():

```
//--------------------------------------------------------------------------------------------------//
// The Print() function -> Print all the elements (digits -> data of the nodes) stored in a linked list
void Print(struct Node *head) {
    struct Node *temp = head;
    // In case the linked list is empty.
    if (temp == NULL) {
        printf("The Linked List is Empty!");
        return;
    }
    // Traverse the linked list till the last node while printing the values.
```

```
    while (temp != NULL) {
        printf("%d", temp -> data);
        temp = temp -> next;
    }
}
//--------------------------------------------------------------------------------------------------//
```

The Print Function is used to print the data stored in the nodes of a linked list while traversing it (node by node) in case the linked list is not empty.

<u>Algorithm of the print function: -</u>

1. Create a temporary node(temp) and assign the head node's address.

2. While looping, Print the data which is present in the temp node.

3. After printing the data, move the temp pointer to the next node.

4. Do the above process until we reach the end.

Time Complexity = O(n)

Space Complexity = O (1)

Section (3): Test Cases:-

C:\Users\huawei\Desktop\Project\Final\150120998_p1.exe

```
1010111010
1001011000
11001100011111110000
698
600
418800
--------------------------------
Process exited after 0.0251 seconds with return value 0
Press any key to continue . . .
```

C:\Users\huawei\Desktop\Project\Final\150120998_p1.exe

```
1010111010
1001011000
11001100011111110000
698
600
418800
--------------------------------
Process exited after 0.1465 seconds with return value 0
Press any key to continue . . .
```

Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\huawei> cd Desktop
PS C:\Users\huawei\Desktop> cd Project
PS C:\Users\huawei\Desktop\Project> cd Final
PS C:\Users\huawei\Desktop\Project\Final> .\150120998_p1 "150120998_input_1_p1.txt"
5301
4120
35124120
1189
912
1084368
PS C:\Users\huawei\Desktop\Project\Final>
```

Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\huawei> cd Desktop
PS C:\Users\huawei\Desktop> cd Project
PS C:\Users\huawei\Desktop\Project> cd Final
PS C:\Users\huawei\Desktop\Project\Final> .\150120998_p1 "150120998_input_1_p1.txt"
5301
4120
35124120
1189
912
1084368
PS C:\Users\huawei\Desktop\Project\Final> .\150120998_p1 "150120998_input_2_p1.txt"
1010111010
1001011000
110011000111110000
698
600
418800
PS C:\Users\huawei\Desktop\Project\Final>
```