

Programming Assignment#0 Report

20102122 정효안

From now on, I will represent a polynomial with a form of doubly linked list and implement polynomials multiplication. The overview of the report is as follows.

- Approach to the problem
- Overview of the implementation
- Design
- Code compile and execution
- Conclusion

Approach to the problem

1. In the problem, the inputpoly function, printNode function, and multiple function should be implemented.
2. inputpoly function is a function that receives the coefficient and degree of each term to express polynomials as a doubly linked list.
3. To implement a doubly linked list, each term (node in a double linked list) must be created with the coefficients and degrees input from the inputpoly function, and the nodes must be connected (inserted) in the same way as the doubly linked list.
4. The 'createNode', a function that creates a node, and 'appendNode', a function that connects nodes, should be created. The degree of the polynomial printed with printNode is the ascending order, so it is necessary to implement the ascending order in appendNode.
5. inputpoly and printNode are created using two functions.
6. In the multiply function, which means multiplication of polynomials, the order of multiplication is expressed using conditional statement.

Overview of the implementation

1. struct Node

2. createNode

3. appendNode

4. inputpoly

5. printNode

6. multiply

7. main

Design

In the overview of the implementation, What I already had is a struct Node. And through this and the provided explanation, I made createNode, appendNode, inputpoly, printNode, multiply and main. Here is an explanation of the important parts of the code.

createNode

```
Node *createNode(int d, int c)
{
    Node *newNode = (Node *)malloc(sizeof(Node));

    newNode->degree = d;
    newNode->coefficient = c;

    newNode->next = NULL;
    newNode->prev = NULL;

    return newNode;
}
```

- When input is received from inputpoly, it is a function that generates a node structure(a term)
- A node-sized memory is dynamically allocated to *newNode.
- It returns a node pointer(=address)

appendNode

```
void appendNode(Node **head, Node *newNode)
{
    Node *curr = *head;

    if (*head == NULL)
    {
        *head = newNode;
    }
    else if (curr->degree > newNode->degree)
    {
        newNode->next = curr;
        *head = newNode;
    }
    else
    {
        while (curr->next != NULL && curr->next->degree < newNode->degree)
        {
            curr = curr->next;
        }

        if (curr->degree == newNode->degree)
        {
            curr->coefficient += newNode->coefficient;
            delete (newNode);
        }
        else
        {
            newNode->next = curr->next;
            curr->next = newNode;
        }
    }
}
```

- A function that inserts and connects a newNode into a polynomial.
- Our field is empty (list is empty) or not. Since the head pointer (first node) needs to be updated, a double pointer was used.
- When the list is empty: new node becomes first node.
- When the list is not empty and the degree of the new node is less than the degree of the current node, the new node enters the left side of the current node and becomes the first node. (Ascending order)
- Else, compare the degree of the new node with the degree of the current node, and change the current node to the next node of the current node until the degree of the new node is larger. If the degrees of the current node and the new node are the same, change the coefficients of the current node to the sum of the coefficients and delete the new node. Or add a new node to the right of the current node and to the left of the current node next.

inputpoly

```
Node *inputpoly(void)
{
    int d, c;

    Node *poly = NULL;
    Node *newNode = NULL;

    do
    {
        printf("Input (degree) (coefficient): ");
        scanf("%d %d", &d, &c);

        if ((d >= 0) && (c >= 0))
        {
            if (c == 0)
            {
                d = 0;
            }
            newNode = createNode(d, c);
            appendNode(&poly, newNode);
        }
        else if (d * c < 0)
        {
            printf("Please enter the numbers of the same sign!\n");
        }
        else
        {
            printf("Done!!\n");
            return poly;
        }
    } while (true);
}
```

- Make the node a pointer type to treat as a global variable.
- When the coefficient is 0, consider the degree 0.
- Since the double pointer was used as a parameter, &poly should be used in appendNode.
- When one of the inputs is negative, take input again.
- When both degree and coefficient inputs are negative, the function returns the point to the first node of the polynomial.

printNode

```
void printNode(Node *inp)
{
    Node *curr = inp;

    while (curr != NULL)
    {
        if (curr->degree == 0)
        {
            printf("%d", curr->coefficient);

            if (curr->next == NULL)
            {
                curr = curr->next;
            }
            else
            {
                printf("+");
                curr = curr->next;
            }
        }
        else
        {
            if (curr->coefficient == 0)
            {
                if (curr->next == NULL)
                {
                    curr = curr->next;
                }
                else
                {
                    printf("+");
                    curr = curr->next;
                }
            }
            else
            {
                printf("%dx^%d", curr->coefficient, curr->degree);
                if (curr->next == NULL)
                {
                    curr = curr->next;
                }
                else
                {
                    printf("+");
                    curr = curr->next;
                }
            }
        }
    }

    printf("\n");
}
```

- print until the next node exists
- When it's a constant term, only the coefficient is output.
- End print when there is no next node
- When there is a next node, output '+' (each terms have to be connect with '+') and the next node (=the term with a higher degree) becomes the current node.
- When both coefficients and degrees are not zero

multiply

```
Node *multiply(Node *a, Node *b)
{
    Node *curr1 = a;
    Node *curr2 = b;
    Node *newNode = NULL;
    Node *result = NULL;

    while (curr1 != NULL)
    {
        while (curr2 != NULL)
        {
            int c, d;

            c = curr1->coefficient * curr2->coefficient;
            d = curr1->degree + curr2->degree;

            newNode = createNode(d, c);
            appendNode(&result, newNode);
            curr2 = curr2->next;
        }
        curr2 = b;
        curr1 = curr1->next;
    }
    return (result);
}
```

- a is first polynomial, b is second polynomial
- The order of multiplication is expressed in while block.
- In the multiplication of terms, the coefficient is multiplied, and the degree is added.
- Reset position because of 'curr1->next' case.

main

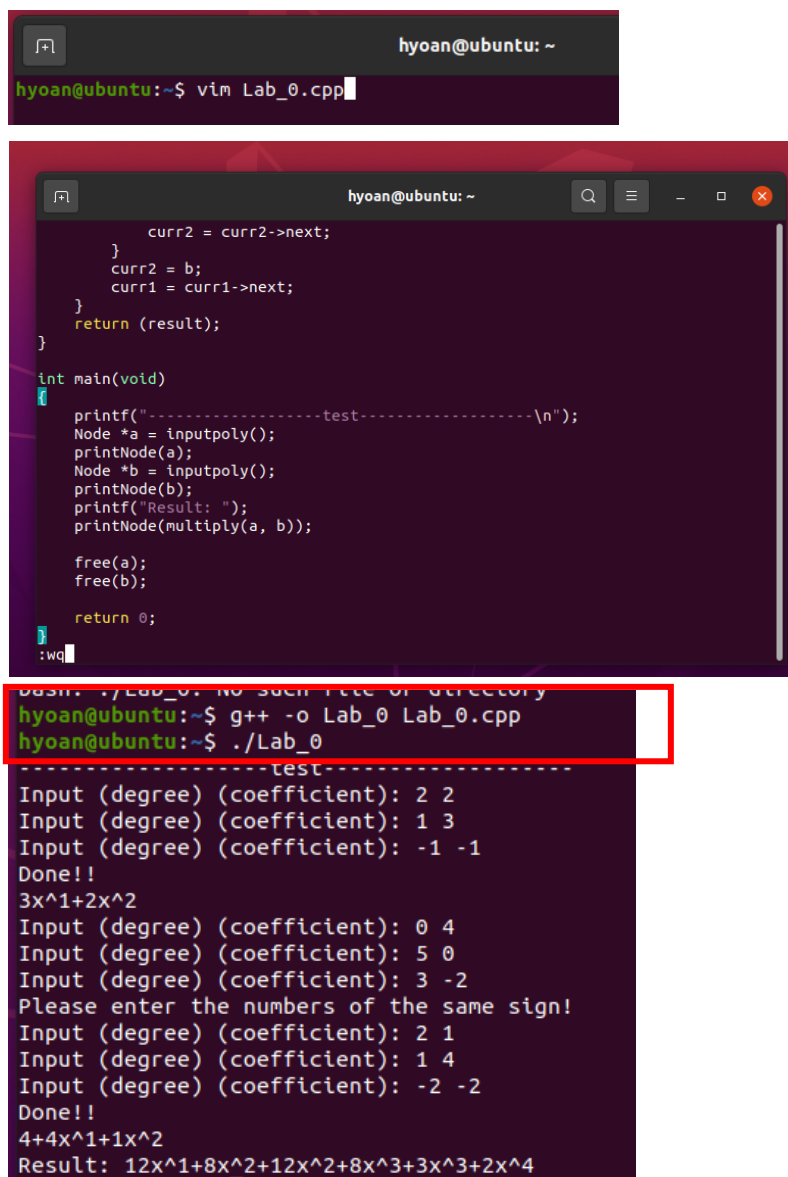
```
int main(void)
{
    printf("-----test-----\n");
    Node *a = inputpoly();
    printNode(a);
    Node *b = inputpoly();
    printNode(b);
    printf("Result: ");
    printNode(multiply(a, b));

    free(a);
    free(b);

    return 0;
}
```

- The first polynomial print.
- The second polynomial print.
- Result polynomial(result of multiplying) output.
- Release dynamic allocation memory.

Code compile and execution



The image shows three terminal windows from an Ubuntu system. The top window shows the command `vim Lab_0.cpp` being executed. The middle window shows the source code for `Lab_0.cpp`, which includes a linked list structure and a `main` function that tests the `inputpoly` and `multiply` functions. The bottom window shows the compilation and execution of the program. The compilation command is `g++ -o Lab_0 Lab_0.cpp`, and the execution command is `./Lab_0`. The output shows the program running a test where it takes polynomial coefficients as input and prints the result of multiplication.

```
hyoan@ubuntu: ~  
hyoan@ubuntu:~$ vim Lab_0.cpp  
  
hyoan@ubuntu:~$ g++ -o Lab_0 Lab_0.cpp  
hyoan@ubuntu:~$ ./Lab_0  
-----test-----  
Input (degree) (coefficient): 2 2  
Input (degree) (coefficient): 1 3  
Input (degree) (coefficient): -1 -1  
Done!!  
3x^1+2x^2  
Input (degree) (coefficient): 0 4  
Input (degree) (coefficient): 5 0  
Input (degree) (coefficient): 3 -2  
Please enter the numbers of the same sign!  
Input (degree) (coefficient): 2 1  
Input (degree) (coefficient): 1 4  
Input (degree) (coefficient): -2 -2  
Done!!  
4+4x^1+1x^2  
Result: 12x^1+8x^2+12x^2+8x^3+3x^3+2x^4
```

Conclusion

I have implemented all the functions required in the problem. However, even if there is no assumption that only numbers are input in the `inputpoly` function, I tried to implement the exception process when inputting characters other than numbers, but failed. Through this assignment, I realized the double linked list directly, and the understanding of the data structure became deeper and it became an opportunity to think deeply about the use of the pointer.