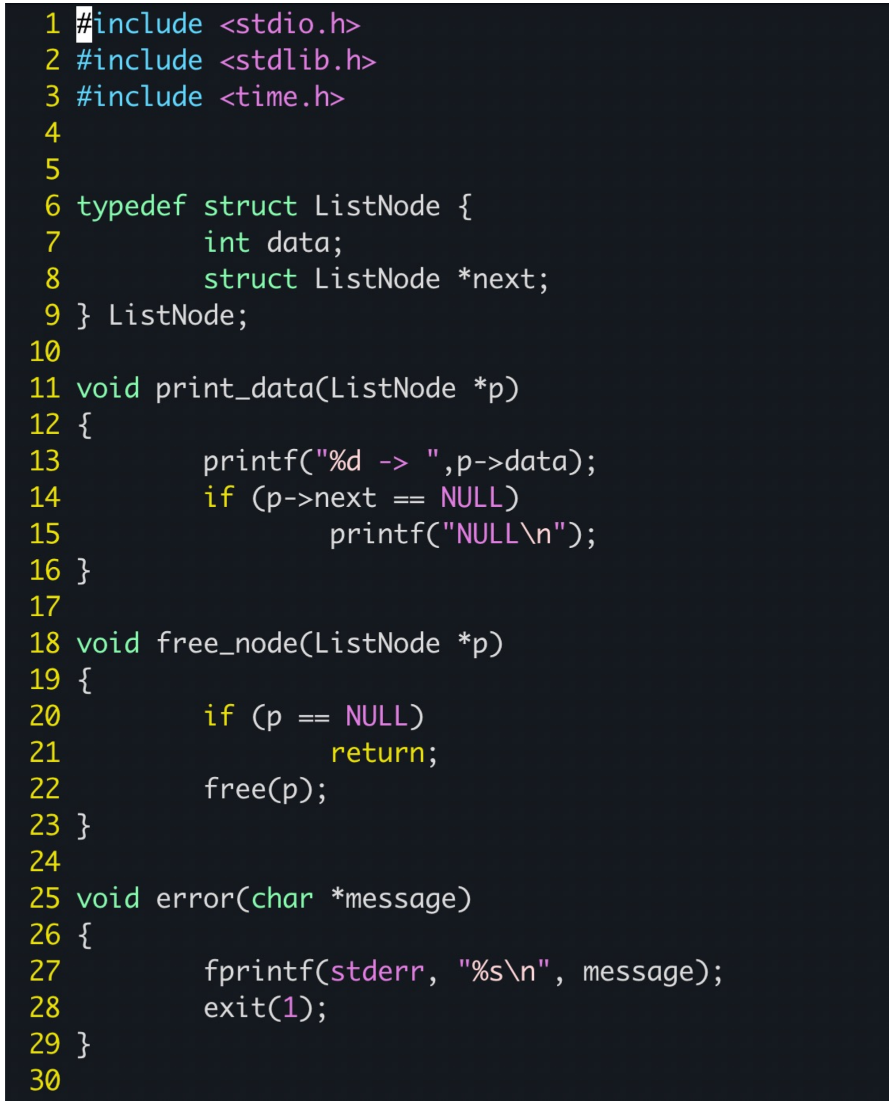
Algorithm

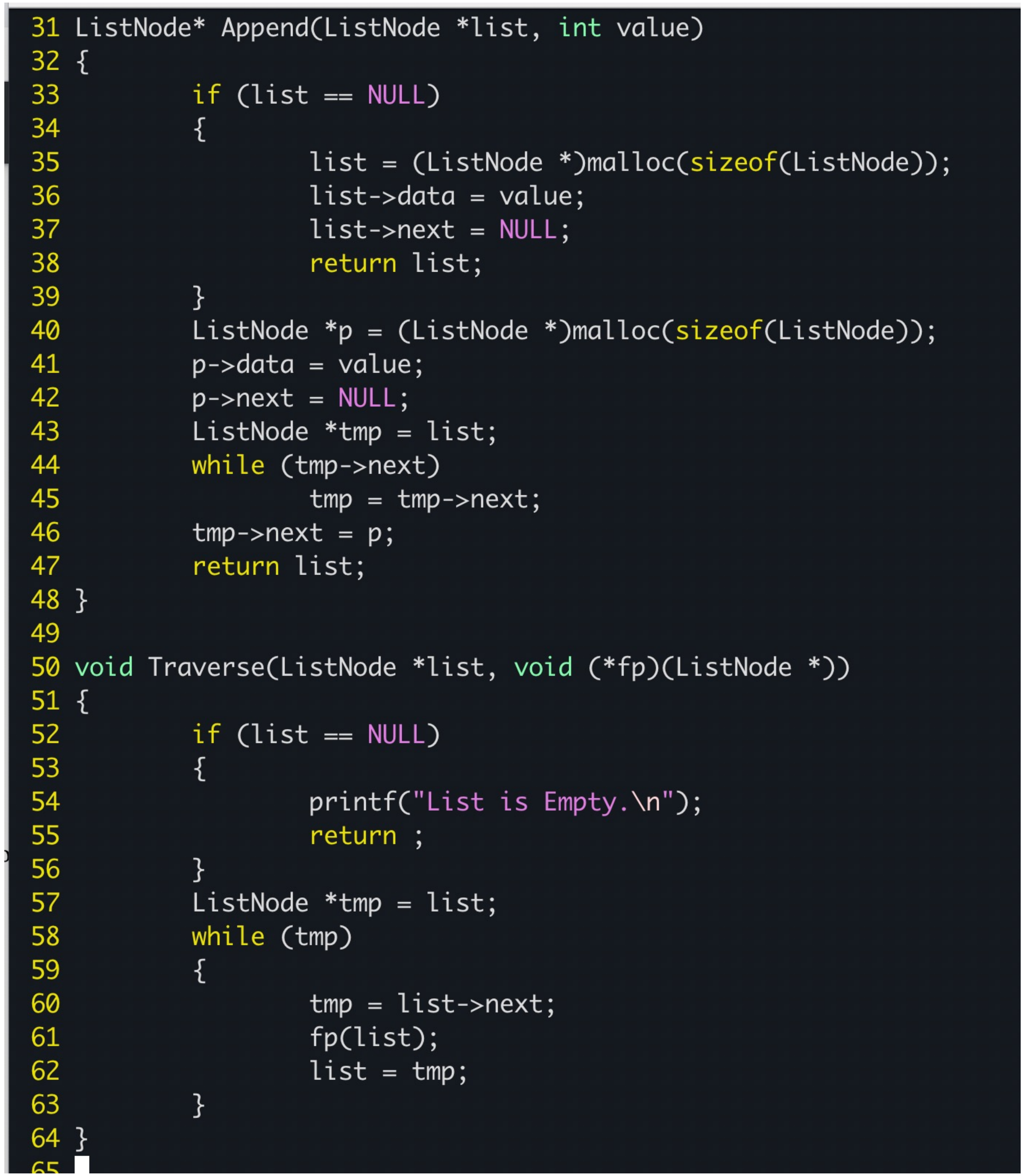
Assignment 3

물리학과 20182326 이선민

Problem 1 - code

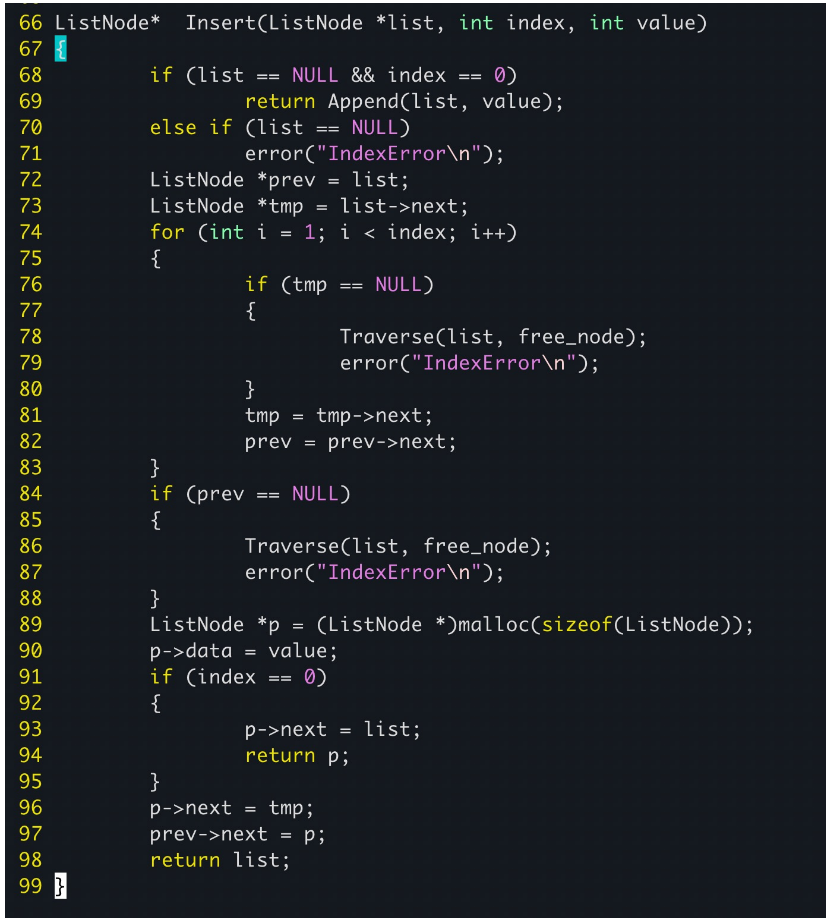


```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


typedef struct ListNode {
        int data;
        struct ListNode *next;
} ListNode;

void print_data(ListNode *p)
{
        printf("%d -> ",p->data);
        if (p->next == NULL)
                printf("NULL\n");
}

void free_node(ListNode *p)
{
        if (p == NULL)
                return;
        free(p);
}

void error(char *message)
{
        fprintf(stderr, "%s\n", message);
        exit(1);
}

```

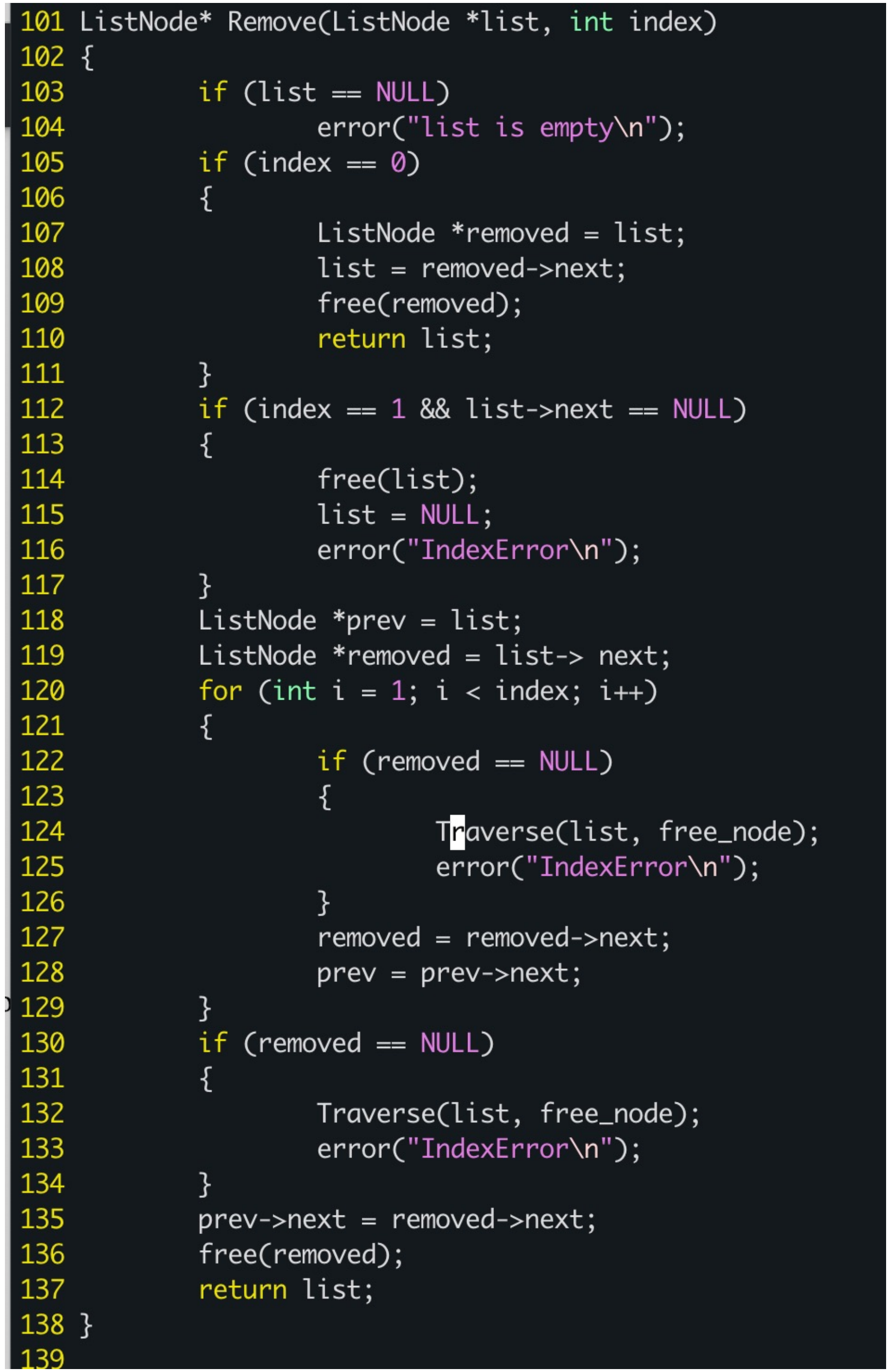


```c
ListNode* Append(ListNode *list, int value)
{
        if (list == NULL)
        {
                list = (ListNode *)malloc(sizeof(ListNode));
                list->data = value;
                list->next = NULL;
                return list;
        }
        ListNode *p = (ListNode *)malloc(sizeof(ListNode));
        p->data = value;
        p->next = NULL;
        ListNode *tmp = list;
        while (tmp->next)
                tmp = tmp->next;
        tmp->next = p;
        return list;
}

void Traverse(ListNode *list, void (*fp)(ListNode *))
{
        if (list == NULL)
        {
                printf("List is Empty.\n");
                return ;
        }
        ListNode *tmp = list;
        while (tmp)
        {
                tmp = list->next;
                fp(list);
                list = tmp;
        }
}
```

Problem1 - code

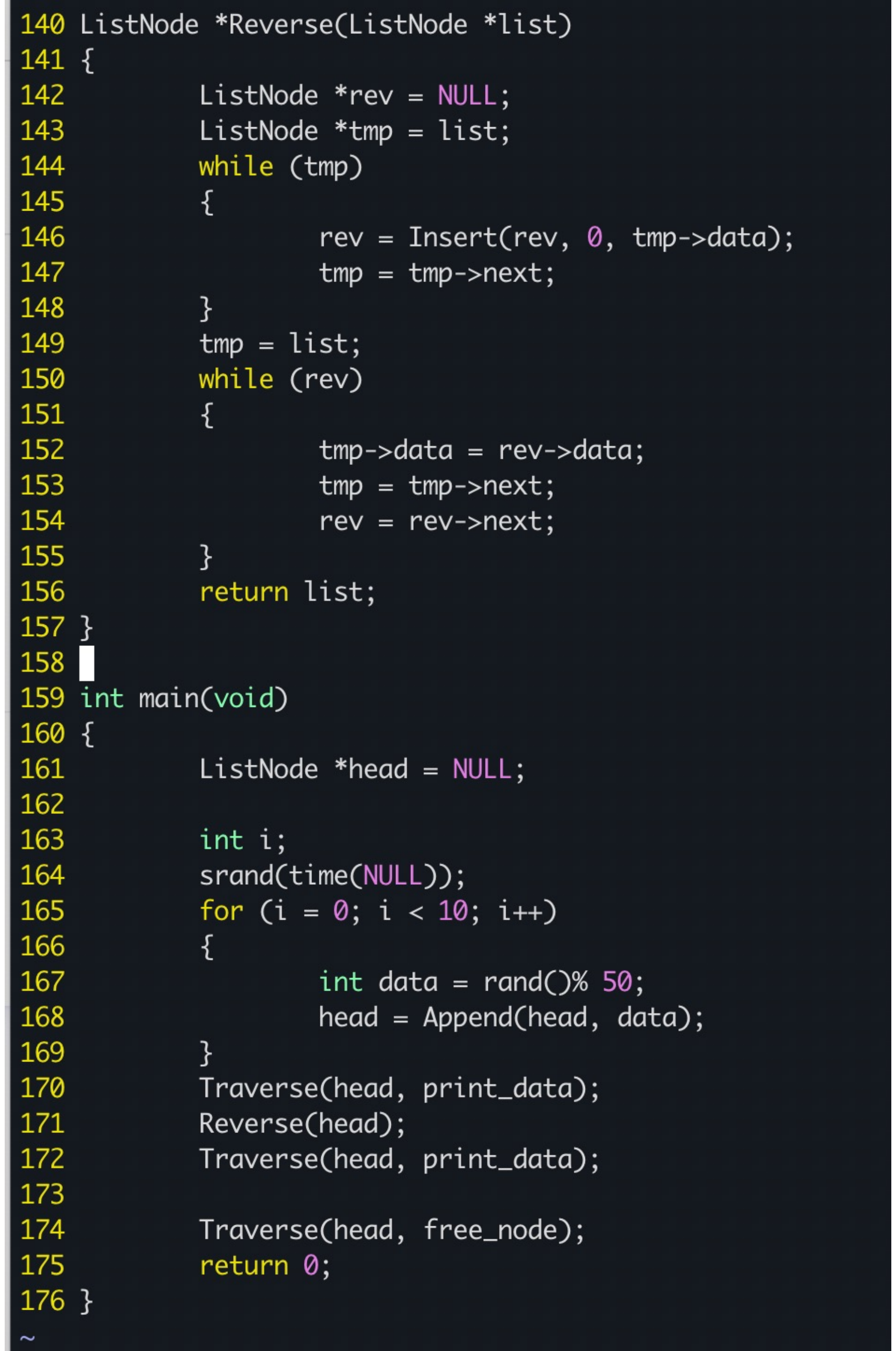


```c
ListNode*  Insert(ListNode *list, int index, int value)
{
        if (list == NULL && index == 0)
                return Append(list, value);
        else if (list == NULL)
                error("IndexError\n");
        ListNode *prev = list;
        ListNode *tmp = list->next;
        for (int i = 1; i < index; i++)
        {
                if (tmp == NULL)
                {
                        Traverse(list, free_node);
                        error("IndexError\n");
                }
                tmp = tmp->next;
                prev = prev->next;
        }
        if (prev == NULL)
        {
                Traverse(list, free_node);
                error("IndexError\n");
        }
        ListNode *p = (ListNode *)malloc(sizeof(ListNode));
        p->data = value;
        if (index == 0)
        {
                p->next = list;
                return p;
        }
        p->next = tmp;
        prev->next = p;
        return list;
}
```



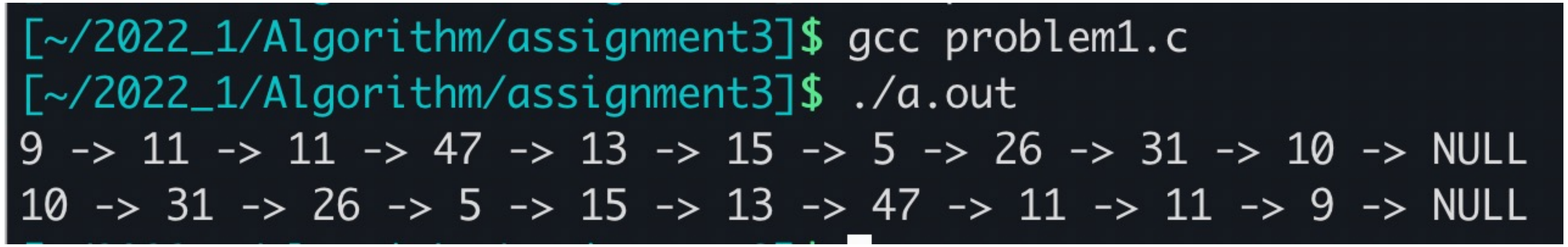
```c
ListNode* Remove(ListNode *list, int index)
{
        if (list == NULL)
                error("list is empty\n");
        if (index == 0)
        {
                ListNode *removed = list;
                list = removed->next;
                free(removed);
                return list;
        }
        if (index == 1 && list->next == NULL)
        {
                free(list);
                list = NULL;
                error("IndexError\n");
        }
        ListNode *prev = list;
        ListNode *removed = list-> next;
        for (int i = 1; i < index; i++)
        {
                if (removed == NULL)
                {
                        Traverse(list, free_node);
                        error("IndexError\n");
                }
                removed = removed->next;
                prev = prev->next;
        }
        if (removed == NULL)
        {
                Traverse(list, free_node);
                error("IndexError\n");
        }
        prev->next = removed->next;
        free(removed);
        return list;
}
```

Problem1 - code



```
ListNode *Reverse(ListNode *list)
{
        ListNode *rev = NULL;
        ListNode *tmp = list;
        while (tmp)
        {
                rev = Insert(rev, 0, tmp->data);
                tmp = tmp->next;
        }
        tmp = list;
        while (rev)
        {
                tmp->data = rev->data;
                tmp = tmp->next;
                rev = rev->next;
        }
        return list;
}

int main(void)
{
        ListNode *head = NULL;

        int i;
        srand(time(NULL));
        for (i = 0; i < 10; i++)
        {
                int data = rand()% 50;
                head = Append(head, data);
        }
        Traverse(head, print_data);
        Reverse(head);
        Traverse(head, print_data);

        Traverse(head, free_node);
        return 0;
}
~
```

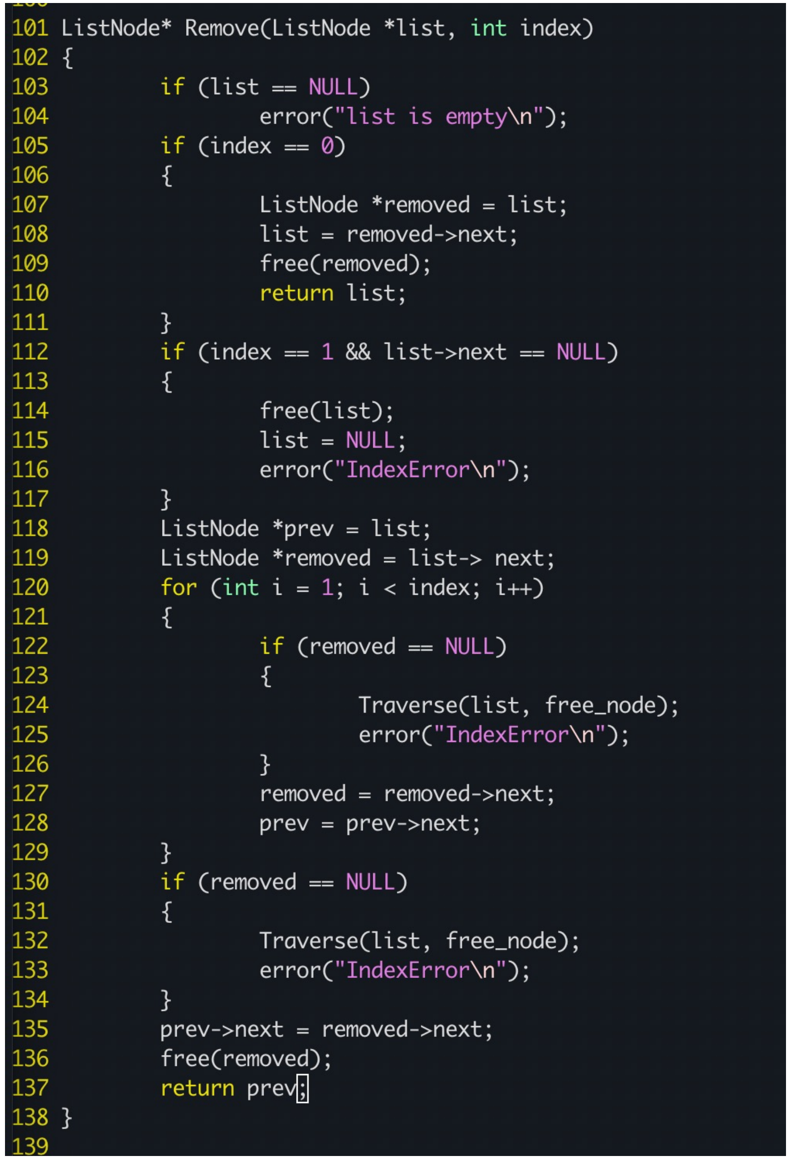
Problem 1 - result



```
[~/2022_1/Algorithm/assignment3]$ gcc problem1.c
[~/2022_1/Algorithm/assignment3]$ ./a.out
9 -> 11 -> 11 -> 47 -> 13 -> 15 -> 5 -> 26 -> 31 -> 10 -> NULL
10 -> 31 -> 26 -> 5 -> 15 -> 13 -> 47 -> 11 -> 11 -> 9 -> NULL
```

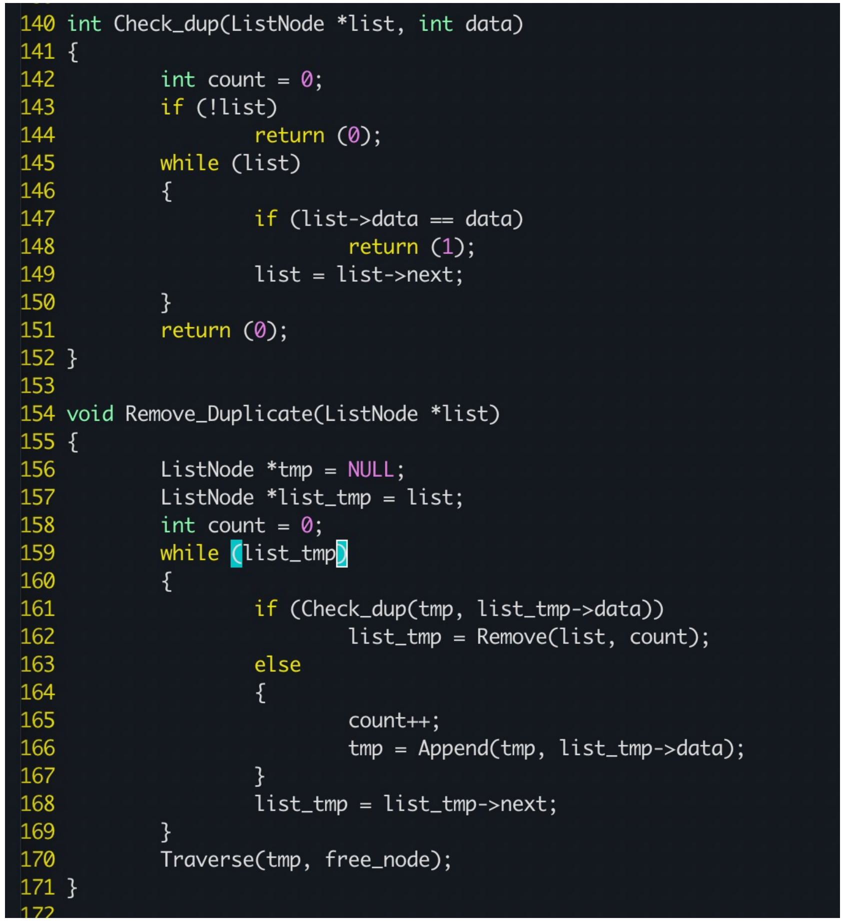
Problem2 - code

- The linked list code was used in the same way as Problem 1, but only the return value of the Remove function was changed. The return value was set as the node immediately before the deleted node.

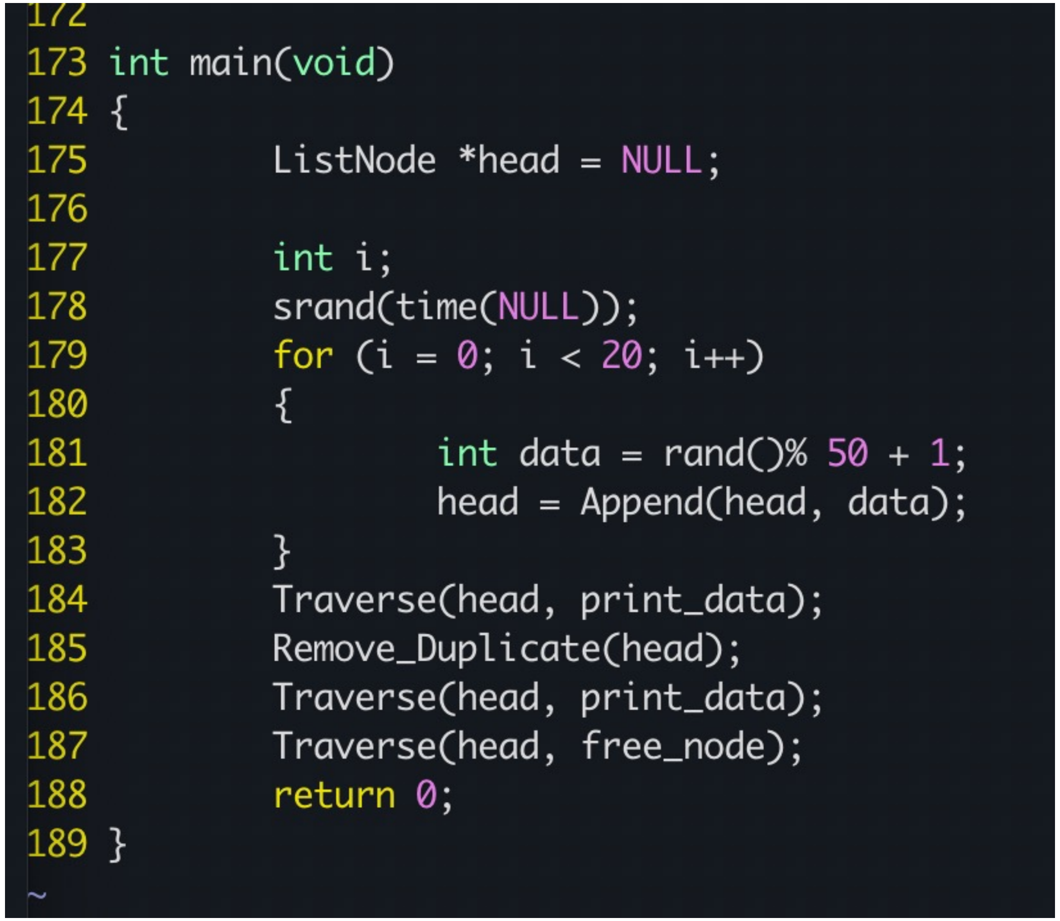


```c
ListNode* Remove(ListNode *list, int index)
{
        if (list == NULL)
                error("list is empty\n");
        if (index == 0)
        {
                ListNode *removed = list;
                list = removed->next;
                free(removed);
                return list;
        }
        if (index == 1 && list->next == NULL)
        {
                free(list);
                list = NULL;
                error("IndexError\n");
        }
        ListNode *prev = list;
        ListNode *removed = list-> next;
        for (int i = 1; i < index; i++)
        {
                if (removed == NULL)
                {
                        Traverse(list, free_node);
                        error("IndexError\n");
                }
                removed = removed->next;
                prev = prev->next;
        }
        if (removed == NULL)
        {
                Traverse(list, free_node);
                error("IndexError\n");
        }
        prev->next = removed->next;
        free(removed);
        return prev;
}
```

Problem2
- newly added function, modifying main function

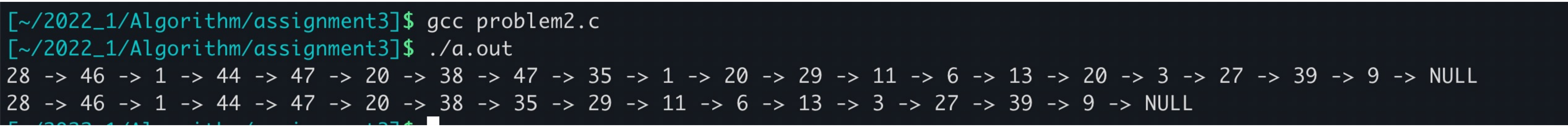


```c
int Check_dup(ListNode *list, int data)
{
        int count = 0;
        if (!list)
                return (0);
        while (list)
        {
                if (list->data == data)
                        return (1);
                list = list->next;
        }
        return (0);
}

void Remove_Duplicate(ListNode *list)
{
        ListNode *tmp = NULL;
        ListNode *list_tmp = list;
        int count = 0;
        while (list_tmp)
        {
                if (Check_dup(tmp, list_tmp->data))
                        list_tmp = Remove(list, count);
                else
                {
                        count++;
                        tmp = Append(tmp, list_tmp->data);
                }
                list_tmp = list_tmp->next;
        }
        Traverse(tmp, free_node);
}
```



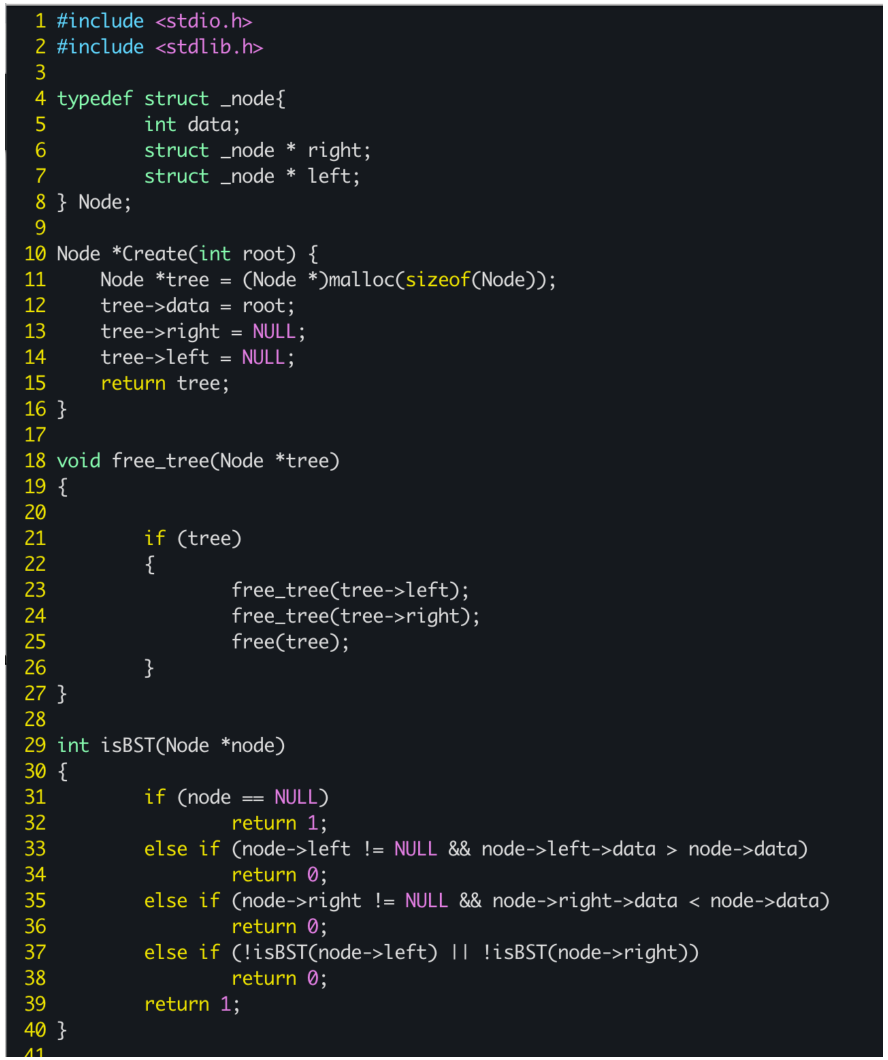
```c
int main(void)
{
        ListNode *head = NULL;

        int i;
        srand(time(NULL));
        for (i = 0; i < 20; i++)
        {
                int data = rand()% 50 + 1;
                head = Append(head, data);
        }
        Traverse(head, print_data);
        Remove_Duplicate(head);
        Traverse(head, print_data);
        Traverse(head, free_node);
        return 0;
}
```

Problem2 - result

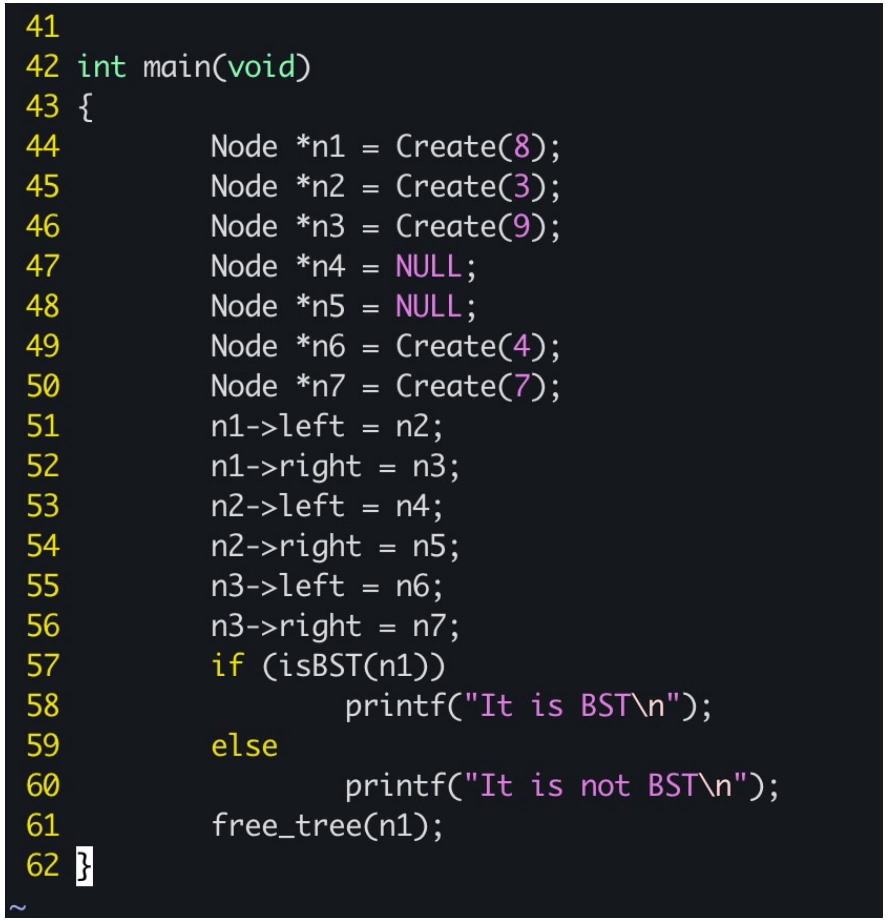


```
[~/2022_1/Algorithm/assignment3]$ gcc problem2.c
[~/2022_1/Algorithm/assignment3]$ ./a.out
28 -> 46 -> 1 -> 44 -> 47 -> 20 -> 38 -> 47 -> 35 -> 1 -> 20 -> 29 -> 11 -> 6 -> 13 -> 20 -> 3 -> 27 -> 39 -> 9 -> NULL
28 -> 46 -> 1 -> 44 -> 47 -> 20 -> 38 -> 35 -> 29 -> 11 -> 6 -> 13 -> 3 -> 27 -> 39 -> 9 -> NULL
[~/2022_1/Algorithm/assignment3]$
```

Problem3 - code

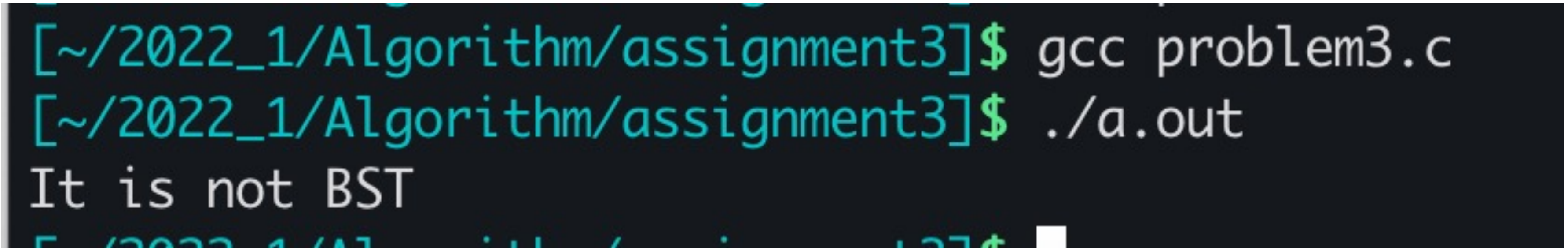


```c
#include <stdio.h>
#include <stdlib.h>

typedef struct _node{
        int data;
        struct _node * right;
        struct _node * left;
} Node;

Node *Create(int root) {
    Node *tree = (Node *)malloc(sizeof(Node));
    tree->data = root;
    tree->right = NULL;
    tree->left = NULL;
    return tree;
}

void free_tree(Node *tree)
{

        if (tree)
        {
                free_tree(tree->left);
                free_tree(tree->right);
                free(tree);
        }
}

int isBST(Node *node)
{
        if (node == NULL)
                return 1;
        else if (node->left != NULL && node->left->data > node->data)
                return 0;
        else if (node->right != NULL && node->right->data < node->data)
                return 0;
        else if (!isBST(node->left) || !isBST(node->right))
                return 0;
        return 1;
}
```



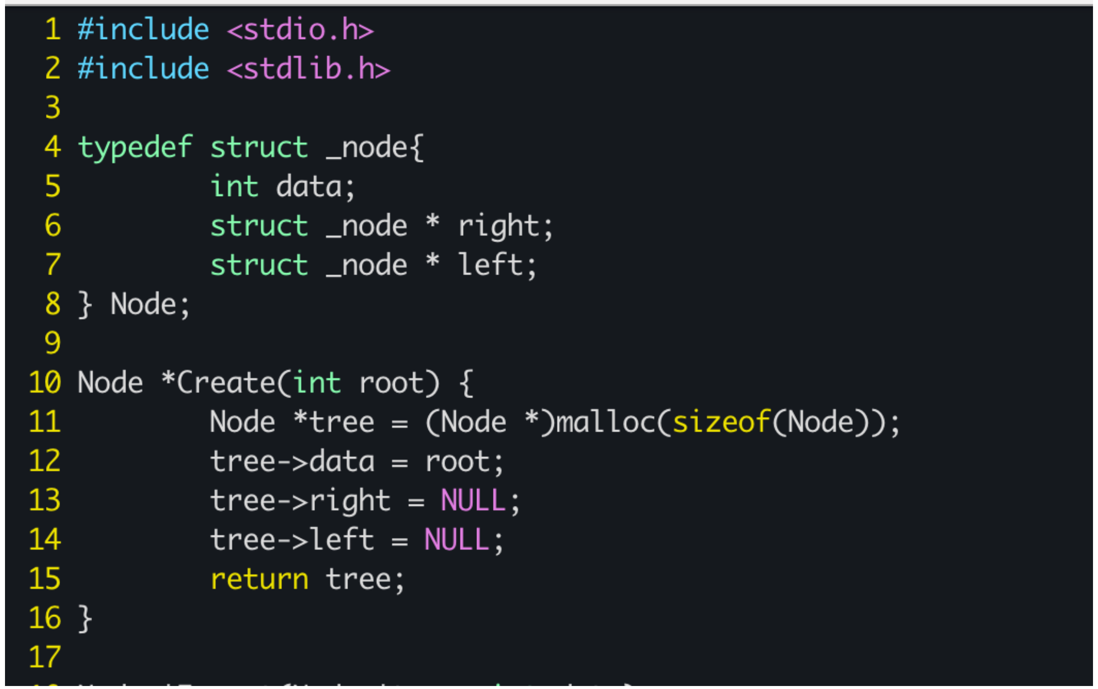
```c

int main(void)
{
        Node *n1 = Create(8);
        Node *n2 = Create(3);
        Node *n3 = Create(9);
        Node *n4 = NULL;
        Node *n5 = NULL;
        Node *n6 = Create(4);
        Node *n7 = Create(7);
        n1->left = n2;
        n1->right = n3;
        n2->left = n4;
        n2->right = n5;
        n3->left = n6;
        n3->right = n7;
        if (isBST(n1))
                printf("It is BST\n");
        else
                printf("It is not BST\n");
        free_tree(n1);
}
~
```

Problem3 - result

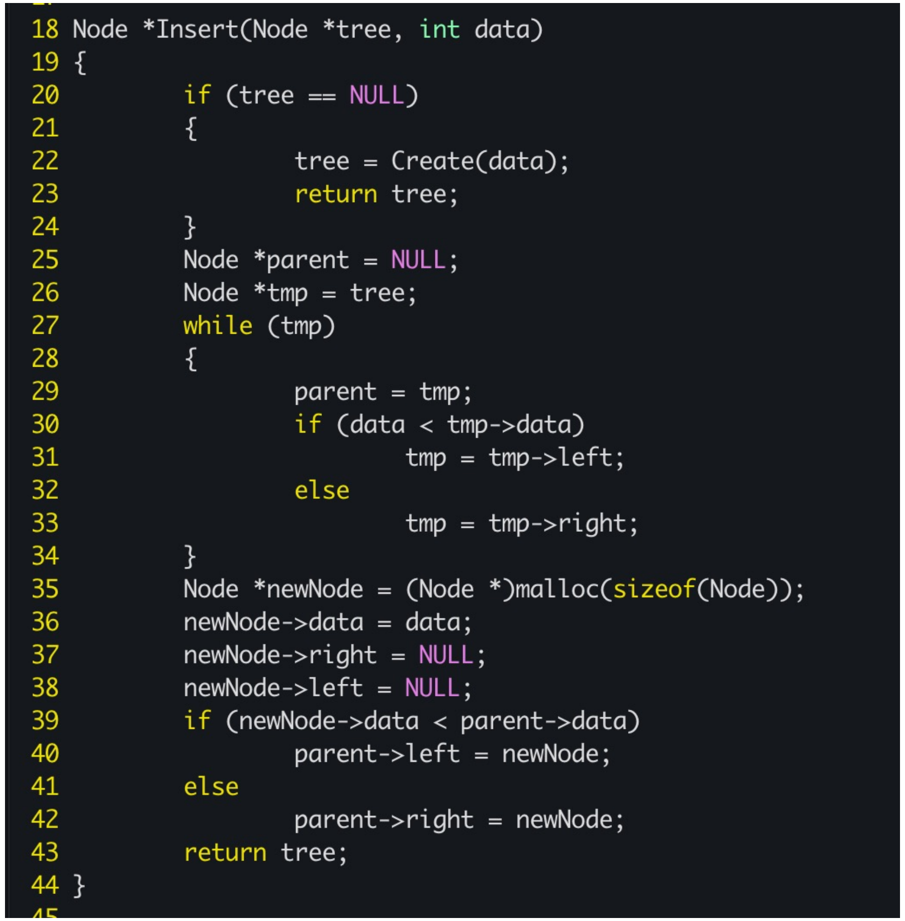


```
[~/2022_1/Algorithm/assignment3]$ gcc problem3.c
[~/2022_1/Algorithm/assignment3]$ ./a.out
It is not BST
[~/2022_1/Algorithm/assignment3]$
```

Problem4 - code

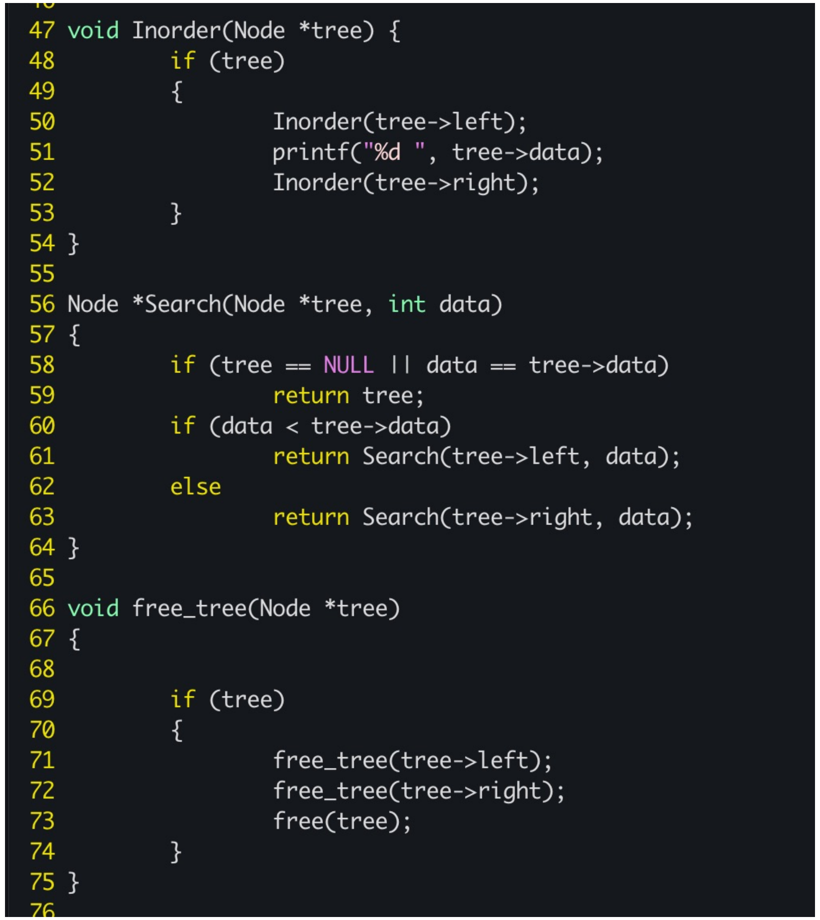


```c
#include <stdio.h>
#include <stdlib.h>

typedef struct _node{
        int data;
        struct _node * right;
        struct _node * left;
} Node;

Node *Create(int root) {
        Node *tree = (Node *)malloc(sizeof(Node));
        tree->data = root;
        tree->right = NULL;
        tree->left = NULL;
        return tree;
}

```

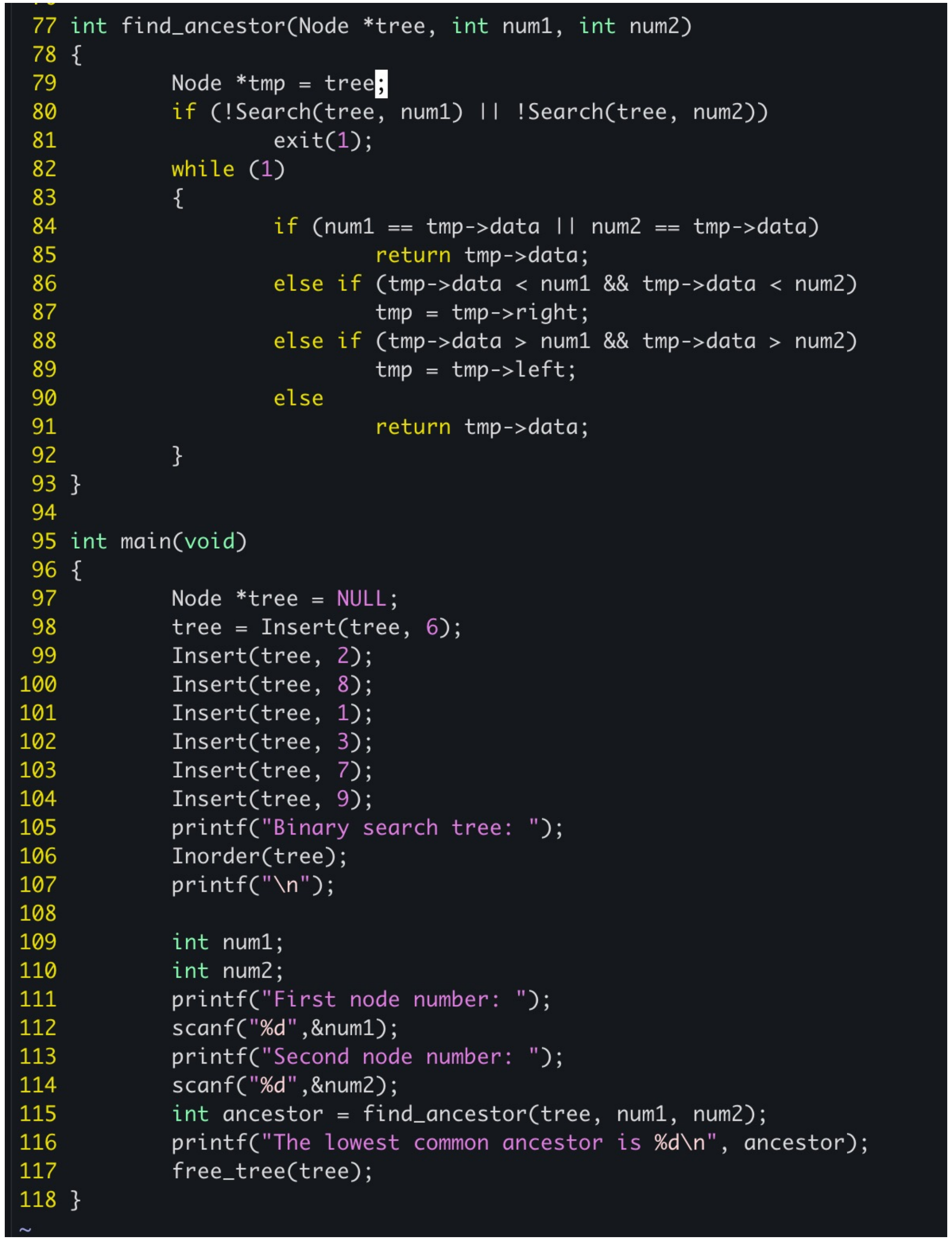


```c
Node *Insert(Node *tree, int data)
{
        if (tree == NULL)
        {
                tree = Create(data);
                return tree;
        }
        Node *parent = NULL;
        Node *tmp = tree;
        while (tmp)
        {
                parent = tmp;
                if (data < tmp->data)
                        tmp = tmp->left;
                else
                        tmp = tmp->right;
        }
        Node *newNode = (Node *)malloc(sizeof(Node));
        newNode->data = data;
        newNode->right = NULL;
        newNode->left = NULL;
        if (newNode->data < parent->data)
                parent->left = newNode;
        else
                parent->right = newNode;
        return tree;
}
```

Problem4 - code

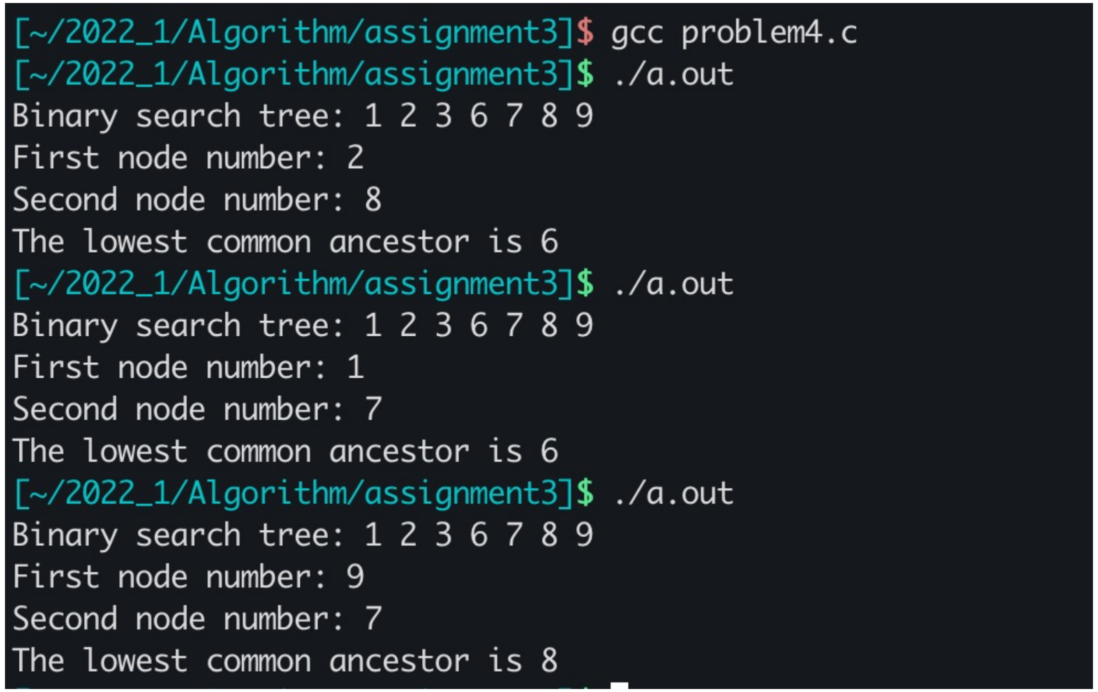


```c
void Inorder(Node *tree) {
        if (tree)
        {
                Inorder(tree->left);
                printf("%d ", tree->data);
                Inorder(tree->right);
        }
}

Node *Search(Node *tree, int data)
{
        if (tree == NULL || data == tree->data)
                return tree;
        if (data < tree->data)
                return Search(tree->left, data);
        else
                return Search(tree->right, data);
}

void free_tree(Node *tree)
{

        if (tree)
        {
                free_tree(tree->left);
                free_tree(tree->right);
                free(tree);
        }
}
```



```c
int find_ancestor(Node *tree, int num1, int num2)
{
        Node *tmp = tree;
        if (!Search(tree, num1) || !Search(tree, num2))
                exit(1);
        while (1)
        {
                if (num1 == tmp->data || num2 == tmp->data)
                        return tmp->data;
                else if (tmp->data < num1 && tmp->data < num2)
                        tmp = tmp->right;
                else if (tmp->data > num1 && tmp->data > num2)
                        tmp = tmp->left;
                else
                        return tmp->data;
        }
}

int main(void)
{
        Node *tree = NULL;
        tree = Insert(tree, 6);
        Insert(tree, 2);
        Insert(tree, 8);
        Insert(tree, 1);
        Insert(tree, 3);
        Insert(tree, 7);
        Insert(tree, 9);
        printf("Binary search tree: ");
        Inorder(tree);
        printf("\n");

        int num1;
        int num2;
        printf("First node number: ");
        scanf("%d",&num1);
        printf("Second node number: ");
        scanf("%d",&num2);
        int ancestor = find_ancestor(tree, num1, num2);
        printf("The lowest common ancestor is %d\n", ancestor);
        free_tree(tree);
}
~
```

Problem4 - result



```
[~/2022_1/Algorithm/assignment3]$ gcc problem4.c
[~/2022_1/Algorithm/assignment3]$ ./a.out
Binary search tree: 1 2 3 6 7 8 9
First node number: 2
Second node number: 8
The lowest common ancestor is 6
[~/2022_1/Algorithm/assignment3]$ ./a.out
Binary search tree: 1 2 3 6 7 8 9
First node number: 1
Second node number: 7
The lowest common ancestor is 6
[~/2022_1/Algorithm/assignment3]$ ./a.out
Binary search tree: 1 2 3 6 7 8 9
First node number: 9
Second node number: 7
The lowest common ancestor is 8
```