Problem 3 - *(Human Compiler) (Hand-written)

Problem Description

There are four questions related to *pointers* in *C*. In each of the following questions, you will be given an incomplete code segment and its purpose. Without the help of compilers, please fill in blank segments and achieve the purpose. After completing the code segments, you are encouraged to run the codes and verify the correctness on your own.

Compared with *Python*, it is easy to get a *Runtime Error* (RE) when dealing with *pointers* in *C*. For example, the following code segment may lead to a *Segmentation Fault* (depends on the compiler).

```
int *ptr = NULL;
*ptr = 42;
```

This may look stupid, but it somehow occurs when your code grows to have more than 100 lines. Therefore, you need to be very cautious when you allocate, access and free pointers. The problems below are good practices for you; think twice before going to the next one.

Problem 3-(a) Swaps two arrays using pointers.

```
int fake_a[] = {1, 3};
int fake_b[] = {2, 4};
int *real_a = fake_a;
int *real_b = fake_b;

for (int i=0; i<2; i++)
    printf("%d ", *(real_a + i));
for (int i=0; i<2; i++)
    printf("%d ", *(real_b + i));

int *tmp = real_a;
___(1)___ = ___(2)___;    → Ans: real_a = real_b ;
___(3)___ = ___(4)___;    → Ans: real_b = tmp ;

for (int i=0; i<2; i++)
    printf("%d ", *(real_a + i));
for (int i=0; i<2; i++)
    printf("%d ", *(real_b + i));
```

The output should be "1 3 2 4 2 4 1 3".

Problem 3-(b) An array supporting negative indices.

```
#include <stdio.h>
#define MINN -50 // inclusive
#define MAXN 50 // inclusive

int main(){
    int storage[MAXN - MINN + 1]={0};
    int *ary = ___(1)___;    → Ans: &storage[50] ;

    for (int i=MINN; i<=MAXN; i++)
        ary[i] = i;
    for (int i=MINN; i<=MAXN; i++)
        printf("%d ", ary[i]);
    return 0;
}
```

The output should be "-50 -49 ... -1 0 1 ... 49 50 ".

Problem 3-(c) Traverses data nodes in a *linked list*. Please familiarize yourself with linked list in advance. Related topics are covered in Prof. Liu's videos.

```
#include <stdio.h>
#include <stdlib.h> // malloc / free
#include <memory.h> // memset

// Use typedef to define "struct node" as "node".
typedef struct node{
    int data;
    struct node *nxt;
} node;

node *alloc(int data, node *nxt){
    node *tmp = (node *)malloc(sizeof(node));
    tmp->data = data;
    tmp->nxt = nxt;
    return tmp;
}

void destory(node *head){
    if (___(1)___){         Ans: head -> nxt != NULL
        destory(head->nxt);
        // clean sensitive data.
        memset(head, 0, sizeof(node));
        free(head);
    }
}

int main(){
    // create nodes [0, 1, 2, 4]
    node *head = alloc(0, alloc(1, alloc(2, alloc(4, NULL))));
    node *tmp = head;

    // print the nodes subsequently     Ans: tmp -> data
    while (tmp != NULL){
        printf("%d -> ", ___(2)___); // print the data in the node
        tmp = ___(3)___;
    }                                   Ans: tmp -> nxt
    printf("NULL");

    // free the nodes subsequently to avoid memory leak
    destory(head);
    return 0;
}
```

The output should be "0 -> 1 -> 2 -> 4 -> NULL".

Problem 3-(d) Traverses data nodes in a *binary tree*. Please familiarize yourself with binary trees in advance. Related topics are covered in Prof. Liu's videos.

```
#include <stdio.h>
#include <stdlib.h> // malloc / free
#include <memory.h> // memset

// Use typedef to substitute "struct node" with "node".
typedef struct node {
    int data;
    struct node *left, *right;
} node;

// creates a node filled with predefined values
node *alloc(int data, node *left, node *right){
    node *tmp = (node*)malloc(sizeof(node));
    tmp->data = data;
    tmp->left = left;
    tmp->right = right;
    return tmp;
}

// traverses the nodes recursively
void traverse(node *root){
    if (___(1)___){ ————→ Ans: root != NULL
        printf("%d ", root->data);
        traverse(___(2)___); → Ans: root -> left
        traverse(___(3)___); → Ans: root -> right
    }
}

// frees the nodes recursively
void destory(node *root){
    if (___(1)___){ ————→ Ans: root != NULL
        // recursively destory nodes.
        destory(___(2)___); → Ans: root -> left
        destory(___(3)___); → Ans: root -> right
        // clean sensitive data.
        memset(root, 0, sizeof(node));
        free(___(4)___);
    }                       → Ans: root
}

int main(){
    // creates a hierarchical data structure
    node *root = \
        alloc(0,
            alloc(3,
                alloc(7, NULL, NULL),
                alloc(4, NULL, NULL)
            ),
            alloc(2,
                alloc(1, NULL, NULL),
                alloc(9, NULL, NULL)
            )
        );
```