```c
/**
 * Lab Sheet - 4: Linked list representation of the characters array  in C.
 *
 * 1. Conversion of Character Array to Linked List.
 * 2. Conversion of Character Linked List to Array.
 * 3. Finding sub-string of the linked list represented string.
 * 4. Perform Zip operation over the two linked lists of strings.
 *
 * By: Santa Basnet
 * Date: 10/01/2022
 * Everest Engineering College.
 */

#include <malloc.h>

/**
 * Data Representation.
 */

struct CharacterNode {
    /**
     * Data Part.
     */
    char data;
    /**
     * Self Reference Part.
     */
    struct CharacterNode *next;
};

/**
 * Zipped data representation.
 */
struct ZippedNode {
    char data[3];
    struct ZippedNode *next;
};

/**
 * Input
 * @return characterPointer
 */
const char *input() {
    return "HOUSE";
}

/**
 * Second String.
 * @return characterPointer
 */
const char *inputSecond() {
    return "schools";
}
```

```
/**
 * Create a new node and initialize given
 * character data ch.
 */
struct CharacterNode *createNode(char ch) {
    struct CharacterNode *newNode = (struct CharacterNode *)
malloc(sizeof(struct CharacterNode));
    if (newNode != nullptr) {
        newNode->data = ch;
        newNode->next = nullptr;
    }
    return newNode;
}

/**
 * Create a new node of type ZippedNode from the two characters.
 */
struct ZippedNode *createZippedNode(char first, char second) {
    struct ZippedNode *newNode = (struct ZippedNode *) malloc(sizeof(struct
ZippedNode));
    newNode->data[0] = first;
    newNode->data[1] = second;
    return newNode;
}

/**
 * Display Linked List.
 */
void displayList(struct CharacterNode *head) {
    struct CharacterNode *traverse = head;
    while (traverse != nullptr) {
        printf("[ %c {%X}] -> ", traverse->data, traverse->next);
        traverse = traverse->next;
    }
}

/**
 * Insert operation of the character list.
 */
struct CharacterNode *InsertCharacter(struct CharacterNode *head, char ch) {
    /**
     * 1. Head is NULL, create a node and make it head.
     */
    if (head == nullptr) {
        head = createNode(ch);
        return head;
    }
    /**
     * Insert in the last position.
     */
    struct CharacterNode *traverse = head;
```

```
    for (; traverse->next != nullptr;) traverse = traverse->next;
    traverse->next = createNode(ch);
    return head;
}

/**
 * Create an array from linked list.
 */

char *createFromList(struct CharacterNode *head) {
    char *arrData = (char *) malloc(100);
    int index = 0;
    struct CharacterNode *traverse = head;
    for (; traverse != nullptr;) {
        arrData[index++] = traverse->data;
        arrData[index + 1] = '\0';
        traverse = traverse->next;
    }
    return arrData;
}

/**
 * Build a linked list from an array of characters.
 */
struct CharacterNode *buildLinkedList(const char *data) {
    struct CharacterNode *head = nullptr;
    int i = 0;
    char ch = *(data + i++);
    while (ch != '\0') {
        printf("\nInput: %c", ch);
        head = InsertCharacter(head, ch);
        ch = *(data + i++);
    }
    return head;
}

/**
 * Sub string from given string represented linked list.
 */
struct CharacterNode *SubStringOf(struct CharacterNode *head, int from, int len)
{
    /**
     * Verify if the linked list is empty or not.
     */
    if (head == nullptr) return nullptr;

    /**
     * Find the start pointer to accumulate the substring part.
     */
    int start;
    struct CharacterNode *begin = head;
    for (start = 0; start < from && begin != nullptr; start++)
```

```
            begin = begin->next;

    /**
     * Accumulate the sub-string from the linked list.
     */
    struct CharacterNode *subStrData = nullptr;
    for (start = 0; begin != nullptr && start < len; start++) {
        subStrData = InsertCharacter(subStrData, begin->data);
        head = head->next;
    }
    return subStrData;
}

/**
 * Display Zipped List.
 */
void displayZippedList(struct ZippedNode *head) {
    struct ZippedNode *traverse = head;
    while (traverse != nullptr) {
        printf("[ %s {%X}] -> ", traverse->data, traverse->next);
        traverse = traverse->next;
    }
}

/**
 * Perform Insertion in the zipped list.
 */
struct ZippedNode *InsertZippedData(struct ZippedNode *head, char first, char
second) {
    if (head == nullptr) head = createZippedNode(first, second);
    else {
        struct ZippedNode *traverse = head;
        while (traverse->next != nullptr) traverse = traverse->next;
        traverse->next = createZippedNode(first, second);
    }
    return head;
}

/**
 * Perform Zip operation on Two Linked Lists.
 */
struct ZippedNode *ZipLL(struct CharacterNode *first, struct CharacterNode
*second) {
    struct ZippedNode *zippedHead = nullptr;
    if (first == nullptr || second == nullptr) return zippedHead;
    while (first->next != nullptr && second->next != nullptr) {
        zippedHead = InsertZippedData(zippedHead, first->data, second->data);
        first = first->next;
        second = second->next;
    }
    zippedHead = InsertZippedData(zippedHead, first->data, second->data);
    return zippedHead;
}
```

```
/**
 * Program starts here with successful
 * output 1.
 * @return
 */
int main() {
    printf("Characters List");

    /**
     * 1. Build Linked List from Character Array.
     */
    printf("\n\nSolution 1: Created Linked List: \n");
    struct CharacterNode *head = buildLinkedList(input());
    displayList(head);
    printf("\n");

    /**
     * 2. Create a character array from a Linked list.
     */
    char *arr = createFromList(head);
    printf("\nSolution 2: Converted Array from Linked List: %s", arr);
    printf("\n");

    /**
     * 3. Substring of the given link list represented string.
     */
    printf("\nSolution 3: Substring extracted in a Linked List: \n");
    struct CharacterNode *subStr = SubStringOf(head, 2, 2);
    displayList(subStr);
    printf("\n");

    /**
     * 4. Zipll function part.
     */
    printf("\nSolution 4: Zip Operation over two linked lists.");
    printf("\n\bFirst List: \n");
    struct CharacterNode *firstList = buildLinkedList(input());
    printf("\n\nSecondList: \n");
    struct CharacterNode *secondList = buildLinkedList(inputSecond());
    struct ZippedNode *zippedList = ZipLL(firstList, secondList);
    printf("\n\nZipped List: \n");
    displayZippedList(zippedList);
    printf("\n");

    return 1;
}
```

Output

```
-----------------------------------------------------------------
Solution 1: Created Linked List:

Input: H
```

```
Input: O
Input: U
Input: S
Input: E

[ H {E0AC86D0}] -> [ O {E0AC86F0}] -> [ U {E0AC8710}] -> [ S {E0AC8730}] -> [ E
{0}] ->

Solution 2: Converted Array from Linked List: HOUSE

Solution 3: Substring extracted in a Linked List:
[ U {E0AC87E0}] -> [ U {0}] ->

Solution 4: Zip Operation over two linked lists.
First List:

Input: H
Input: O
Input: U
Input: S
Input: E

SecondList:

Input: s
Input: c
Input: h
Input: o
Input: o
Input: l
Input: s

Zipped List:
[ Hs {E0AC89A0}] -> [ Oc {E0AC89C0}] -> [ Uh {E0AC89E0}] -> [ So {E0AC8A00}] ->
[ Eo {0}] ->


                                        ***
```