```c
#include <stdio.h>
#include <string.h>
#include <malloc.h>

#define MAX 101


char *substring(char *str, int m, int n)
{
    char *sub;
    int i;

    if ((m>n) || (m<1) || (n>strlen(str)))
        return NULL;

    sub = (char *)malloc(n-m+2);
    for (i=0; i<=n-m; i++)
        sub[i] = str[i+m-1];
    sub[i] = '\0';

    return sub;
}


int main()
{
    char str[MAX];
    char *sub;
    int i, m, n;

    for (i=0; i<MAX-1 && (str[i]=getchar())!='\n'; i++);
    str[i] = '\0';
    scanf("%d%d", &m, &n);

    sub = substring(str, m , n);
    if (sub)
        puts(sub);
    else
        puts("Null pointer");

    return 0;
}

```

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


struct node {
    int x;
    struct node *next;
};


struct node *CreateListR(int num)
{
    struct node *head, *rear, *p;
    int x, i;

    head = NULL;
    for (i=0; i<num; i++) {
        scanf("%d", &x);
        p = (struct node *)malloc(sizeof(struct node));
        p->next = NULL;
        p->x = x;
        if (!head)
            head = p;
        else
            rear->next = p;
        rear = p;
    }

    return head;
};


void PrintList(struct node *head)
{
    struct node *p;

    p = head;
    while (p) {
        printf("%d ", p->x);
        p = p->next;
    }
}


struct node *ReversePart(struct node *head, int m, int n)
{
    int i;
    struct node *pm, *pn, *pr;

    if (!head || m<1 || m>=n)
        return head;

    pm = head;
    for (i=1; pm && i<m; i++) {
        pr = pm;
        pm = pm->next;
    }

    pn = pm;
    for (; pn && i<n; i++)
        pn = pn->next;
    if (!pn)
        return head;

    if (m==1)
        head = head->next;
    else
        pr->next = pm->next;
    pm->next = pn->next;
    pn->next = pm;

    return ReversePart(head, m, n-1);
};


int main()
{
    struct node *head;
    int num, m, n;

    scanf("%d", &num);
    head = CreateListR(num);
    scanf("%d%d", &m, &n);
    head = ReversePart(head, m, n);
    PrintList(head);

    return 0;
}
```

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

#define BLOCK_SIZE 1024
#define BLOCK_NUM  10
#define POOL_SIZE  (BLOCK_SIZE*BLOCK_NUM)

typedef unsigned char (*Tblkptr)[BLOCK_SIZE];

typedef struct {
    void   *poolAddress;
    Tblkptr blockListHead;
    int     nFreeBlocks;
} Tpool;


int initMemPool(Tpool *pool, int size)
{
    int i;
    Tblkptr p;

    pool->poolAddress = malloc(size);
    if (pool->poolAddress == NULL)
        return 0;

    pool->blockListHead = pool->poolAddress;
    pool->nFreeBlocks   = size / BLOCK_SIZE;

    p = pool->blockListHead;
    for (i=0; i<pool->nFreeBlocks-1; i++) {
        *(Tblkptr *)p = p + 1;
        p++;
    }
    *(Tblkptr *)p = NULL;

    return 1;
}


void printBlockList(Tpool *pool)
{
    Tblkptr *pblink;
    int i=0;

    if (!pool->blockListHead) {
        printf("Null list.\n\n\n");
        return;
    }
    pblink = (Tblkptr *)pool->blockListHead;
    while (pblink) {
        printf("Block# %2d, address: 0x%p, ", i++, pblink);
        printf("next->0x%p\n", (void *)(*pblink));
        pblink = (Tblkptr *)(*pblink);
    }
    printf("\n\n");
}


void *getBlock(Tpool *pool)
{
    void *p;

    if (pool->nFreeBlocks > 0) {
        p = pool->blockListHead;
        pool->nFreeBlocks--;
        pool->blockListHead = *(Tblkptr *)pool->blockListHead;
        return p;
    }
    else
        return NULL;
}


void putBlock(Tpool *pool, void *p)
{
    if (!p)
        return;

    *(Tblkptr *)p = pool->blockListHead;
    pool->blockListHead = p;
    pool->nFreeBlocks++;
}


int main()
{
    Tpool memPool;
    void *p1, *p2, *p3;
    int status;

    status = initMemPool(&memPool, POOL_SIZE);

    if (status)
        printBlockList(&memPool);
    else {
        printf("Memory allocation failed.\n");
        exit(0);
    }

    p1 = getBlock(&memPool);
    printf("Allocate memory block 1.\n");
    printBlockList(&memPool);

    p2 = getBlock(&memPool);
    printf("Allocate memory block 2.\n");
    printBlockList(&memPool);

    p3 = getBlock(&memPool);
    printf("Allocate memory block 3.\n");
    printBlockList(&memPool);

    putBlock(&memPool, p2);
    printf("Deallocate memory block 2.\n");
    printBlockList(&memPool);

    putBlock(&memPool, p3);
    printf("Deallocate memory block 3.\n");
    printBlockList(&memPool);

    putBlock(&memPool, p1);
    printf("Deallocate memory block 1.\n");
    printBlockList(&memPool);

    free(memPool.poolAddress);
    return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>

#define MAX 256


int readlongint(char *x)
{
    int  i, j, k;

    for (i=0; i<MAX-1 && (x[i]=getchar())!='\n'; i++);
    x[i] = '\0';

    while (i>=0 && isspace(x[--i]));
    x[i+1] = '\0';

    for (j=0; isspace(x[j]); j++);
    while (x[j]=='0' && isdigit(x[j+1]))
        j++;

    for (k=0; k<=i-j; k++)
        if (!isdigit(x[k+j])) {
            k = 0;
            break;
        }
        else
            x[k] = x[k+j];
    x[k] = '\0';

    return k;
}


char *addition(char *x, char *y)
{
    char *z;
    int  xlen, ylen, zlen;
    int  xdigit, ydigit, carry, sum;
    int  i, j, k;

    xlen = strlen(x);
    ylen = strlen(y);
    zlen = xlen>ylen ? xlen+1 : ylen+1;

    z = (char *)malloc(zlen+1);
    i = xlen - 1;
    j = ylen - 1;
    k = zlen - 1;
    carry = 0;

    while (i>=0 || j>=0 || carry) {
        xdigit = i>=0 ? x[i]-'0' : 0;
        ydigit = j>=0 ? y[j]-'0' : 0;
        sum = xdigit + ydigit + carry;
        z[k] = sum % 10 + '0';
        carry = sum / 10;
        i--;
        j--;
        k--;
    }

    z[zlen] = '\0';
    if (k==0)
        for (i=1; i<=zlen; i++)
            z[i-1] = z[i];

    return z;
}


char *multiplication(char *x, char *y)
{
    char *z, *p, *t;
    int  xlen, ylen, zlen;
    int  xdigit, ydigit, product, carry;
    int  i, j;

    xlen = strlen(x);
    ylen = strlen(y);
    zlen = xlen + ylen;

    z = (char *)malloc(zlen+1);
    z[0] = '0';
    z[1] = '\0';

    if ((xlen==1 && x[0]=='0') || (ylen==1 && y[0]=='0'))
        return z;

    p = (char *)malloc(zlen+1);

    for (j=ylen-1; j>=0; j--) {
        if (y[j] == '0')
            continue;
        ydigit = y[j] - '0';
        carry = 0;
        for (i=xlen-1; i>=0; i--) {
            xdigit = x[i] - '0';
            product = xdigit * ydigit + carry;
            p[i+1] = product % 10 + '0';
            carry = product / 10;
        }
        p[0] = carry + '0';
        for (i=xlen+1; i<zlen-j; i++)
            p[i] = '0';
        p[i] = '\0';

        if (carry==0)
            for (i=1; i<=zlen; i++)
                p[i-1] = p[i];
        t = z;
        z = addition(z, p);
        free(t);
    }

    free(p);

    return z;
}


int main()
{
    char a[MAX], b[MAX];
    int  alen, blen;
    char *sum, *product;

    alen = readlongint(a);
    blen = readlongint(b);

    if (alen==0 || blen==0) {
        puts("Input Error.");
        return 1;
    }

    sum = addition(a, b);
    product = multiplication(a, b);

    printf("%s\n", sum);
    printf("%s\n", product);

    return 0;
}
```