

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <malloc.h>
4
5  #define MAX 101
6
7
8  char *substring(char *str, int m, int n)
9  {
10     char *sub;
11     int i;
12
13     if ((m>n) || (m<1) || (n>strlen(str)))
14         return NULL;
15
16     sub = (char *)malloc(n-m+2);
17     for (i=0; i<=n-m; i++)
18         sub[i] = str[i+m-1];
19     sub[i] = '\0';
20
21     return sub;
22 }
23
24
25 int main()
26 {
27     char str[MAX];
28     char *sub;
29     int i, m, n;
30
31     for (i=0; i<MAX-1 && (str[i]=getchar())!='\n'; i++);
32     str[i] = '\0';
33     scanf("%d%d", &m, &n);
34
35     sub = substring(str, m , n);
36     if (sub)
37         puts(sub);
38     else
39         puts("Null pointer");
40
41     return 0;
42 }
43

```

```

1  #include <stdio.h>
2  #include <malloc.h>
3
4
5  struct node {
6      int x;
7      struct node *next;
8  };
9
10
11 struct node *CreateListR(int num)
12 {
13     struct node *head, *rear, *p;
14     int x, i;
15
16     head = NULL;
17     for (i=0; i<num; i++) {
18         scanf("%d", &x);
19         p = (struct node *)malloc(sizeof(struct node));
20         p->next = NULL;
21         p->x = x;
22         if (!head)
23             head = p;
24         else
25             rear->next = p;
26         rear = p;
27     }
28
29     return head;
30 };
31
32
33 void PrintList(struct node *head)
34 {
35     struct node *p;
36
37     p = head;
38     while (p) {
39         printf("%d ", p->x);
40         p = p->next;
41     }
42 }
43
44
45 struct node *ReversePart(struct node *head, int m, int n)
46 {
47     int i;
48     struct node *pm, *pn, *pr;
49
50     if (!head || m<1 || m>=n)
51         return head;
52
53     pm = head;
54     for (i=1; pm && i<m; i++) {
55         pr = pm;
56         pm = pm->next;
57     }
58
59     pn = pm;
60     for (; pn && i<n; i++)
61         pn = pn->next;
62     if (!pn)
63         return head;
64
65     if (m==1)
66         head = head->next;
67     else
68
69
68         pr->next = pm->next;
69     pm->next = pn->next;
70     pn->next = pm;
71
72     return ReversePart(head, m, n-1);
73 };
74
75
76 int main()
77 {
78     struct node *head;
79     int num, m, n;
80
81     scanf("%d", &num);
82     head = CreateListR(num);
83     scanf("%d%d", &m, &n);
84     head = ReversePart(head, m, n);
85     PrintList(head);
86
87     return 0;
88 }
89

```

```

1  #include <stdio.h>
2  #include <malloc.h>
3  #include <stdlib.h>
4
5  #define BLOCK_SIZE 1024
6  #define BLOCK_NUM 10
7  #define POOL_SIZE (BLOCK_SIZE*BLOCK_NUM)
8
9  typedef unsigned char (*Tblkptr) [BLOCK_SIZE];
10
11 typedef struct {
12     void *poolAddress;
13     Tblkptr blockListHead;
14     int nFreeBlocks;
15 } Tpool;
16
17
18 int initMemPool(Tpool *pool, int size)
19 {
20     int i;
21     Tblkptr p;
22
23     pool->poolAddress = malloc(size);
24     if (pool->poolAddress == NULL)
25         return 0;
26
27     pool->blockListHead = pool->poolAddress;
28     pool->nFreeBlocks = size / BLOCK_SIZE;
29
30     p = pool->blockListHead;
31     for (i=0; i<pool->nFreeBlocks-1; i++) {
32         *(Tblkptr *)p = p + 1;
33         p++;
34     }
35     *(Tblkptr *)p = NULL;
36
37     return 1;
38 }
39
40
41 void printBlockList(Tpool *pool)
42 {
43     Tblkptr *pblink;
44     int i=0;
45
46     if (!pool->blockListHead) {
47         printf("Null list.\n\n");
48         return;
49     }
50     pblink = (Tblkptr *)pool->blockListHead;
51     while (pblink) {
52         printf("Block# %2d, address: 0x%p, ", i++, pblink);
53         printf("next->0x%p\n", (void *)(*pblink));
54         pblink = (Tblkptr *)(*pblink);
55     }
56     printf("\n\n");
57 }
58
59
60 void *getBlock(Tpool *pool)
61 {
62     void *p;
63
64     if (pool->nFreeBlocks > 0) {
65         p = pool->blockListHead;
66         pool->nFreeBlocks--;
67         pool->blockListHead = *(Tblkptr *)pool->blockListHead;
68
69         return p;
70     }
71     else
72         return NULL;
73 }
74
75 void putBlock(Tpool *pool, void *p)
76 {
77     if (!p)
78         return;
79
80     *(Tblkptr *)p = pool->blockListHead;
81     pool->blockListHead = p;
82     pool->nFreeBlocks++;
83 }
84
85
86 int main()
87 {
88     Tpool memPool;
89     void *p1, *p2, *p3;
90     int status;
91
92     status = initMemPool(&memPool, POOL_SIZE);
93
94     if (status)
95         printBlockList(&memPool);
96     else {
97         printf("Memory allocation failed.\n");
98         exit(0);
99     }
100
101     p1 = getBlock(&memPool);
102     printf("Allocate memory block 1.\n");
103     printBlockList(&memPool);
104
105     p2 = getBlock(&memPool);
106     printf("Allocate memory block 2.\n");
107     printBlockList(&memPool);
108
109     p3 = getBlock(&memPool);
110     printf("Allocate memory block 3.\n");
111     printBlockList(&memPool);
112
113     putBlock(&memPool, p2);
114     printf("Deallocate memory block 2.\n");
115     printBlockList(&memPool);
116
117     putBlock(&memPool, p3);
118     printf("Deallocate memory block 3.\n");
119     printBlockList(&memPool);
120
121     putBlock(&memPool, p1);
122     printf("Deallocate memory block 1.\n");
123     printBlockList(&memPool);
124
125     free(memPool.poolAddress);
126     return 0;
127 }
128

```



```

1  #include <stdio.h>
2  #include <string.h>
3  #include <malloc.h>
4  #include <ctype.h>
5
6  #define MAX 256
7
8
9  int readlongint(char *x)
10 {
11     int i, j, k;
12
13     for (i=0; i<MAX-1 && (x[i]=getchar())!='\n'; i++);
14     x[i] = '\0';
15
16     while (i>=0 && isspace(x[--i]));
17     x[i+1] = '\0';
18
19     for (j=0; isspace(x[j]); j++);
20     while (x[j]!='0' && isdigit(x[j+1]))
21         j++;
22
23     for (k=0; k<=i-j; k++)
24         if (!isdigit(x[k+j])) {
25             k = 0;
26             break;
27         }
28     else
29         x[k] = x[k+j];
30     x[k] = '\0';
31
32     return k;
33 }
34
35
36 char *addition(char *x, char *y)
37 {
38     char *z;
39     int xlen, ylen, zlen;
40     int xdigit, ydigit, carry, sum;
41     int i, j, k;
42
43     xlen = strlen(x);
44     ylen = strlen(y);
45     zlen = xlen>ylen ? xlen+1 : ylen+1;
46
47     z = (char *)malloc(zlen+1);
48     i = xlen - 1;
49     j = ylen - 1;
50     k = zlen - 1;
51     carry = 0;
52
53     while (i>=0 || j>=0 || carry) {
54         xdigit = i>=0 ? x[i]-'0' : 0;
55         ydigit = j>=0 ? y[j]-'0' : 0;
56         sum = xdigit + ydigit + carry;
57         z[k] = sum % 10 + '0';
58         carry = sum / 10;
59         i--;
60         j--;
61         k--;
62     }
63
64     z[zlen] = '\0';
65     if (k==0)
66         for (i=1; i<=zlen; i++)
67             z[i-1] = z[i];
68
69     return z;
70 }
71
72
73 char *multiplication(char *x, char *y)
74 {
75     char *z, *p, *t;
76     int xlen, ylen, zlen;
77     int xdigit, ydigit, product, carry;
78     int i, j;
79
80     xlen = strlen(x);
81     ylen = strlen(y);
82     zlen = xlen + ylen;
83
84     z = (char *)malloc(zlen+1);
85     z[0] = '0';
86     z[1] = '\0';
87
88     if ((xlen==1 && x[0]=='0') || (ylen==1 && y[0]=='0'))
89         return z;
90
91     p = (char *)malloc(zlen+1);
92
93     for (j=ylen-1; j>=0; j--) {
94         if (y[j] == '0')
95             continue;
96         ydigit = y[j] - '0';
97         carry = 0;
98         for (i=xlen-1; i>=0; i--) {
99             xdigit = x[i] - '0';
100            product = xdigit * ydigit + carry;
101            p[i+1] = product % 10 + '0';
102            carry = product / 10;
103        }
104        p[0] = carry + '0';
105        for (i=xlen+1; i<zlen-j; i++)
106            p[i] = '0';
107        p[i] = '\0';
108
109        if (carry==0)
110            for (i=1; i<=zlen; i++)
111                p[i-1] = p[i];
112        t = z;
113        z = addition(z, p);
114        free(t);
115    }
116
117    free(p);
118
119    return z;
120 }
121
122
123 int main()
124 {
125     char a[MAX], b[MAX];
126     int alen, blen;
127     char *sum, *product;
128
129     alen = readlongint(a);
130     blen = readlongint(b);
131
132     if (alen==0 || blen==0) {
133         puts("Input Error.");
134         return 1;
135     }
136
137     sum = addition(a, b);
138     product = multiplication(a, b);
139
140     printf("%s\n", sum);
141     printf("%s\n", product);
142
143     return 0;
144 }
145

```