ICS'12 Spring Homework 11

1) What is a **false fragmentation**? And how to solve this phenomenon? Please list at least two ways and make a simple introduction about them.

   False fragmentation is a phenomenon where there is a lot of available free memory chopped up into small, unusable free blocks.

   The common ways are immediate coalescing and deferred coalescing. Immediate coalescing merges any adjacent blocks each time a block is freed. While deferred coalescing waits to coalesce free blocks at some later time.

2) This problem tests your understanding of memory bugs. Each of the code sequences below may or may not contain memory bugs. The code all compiles without warnings or errors. If you think there is a bug, please write **YES** and indicate the type of bug from the list below of memory bugs. Otherwise, if you think there are no memory bugs in the code, please write **NO**.

   Bugs:
   1. buffer overflow error
   2. memory leak
   3. potential for dereferencing a bad pointer
   4. incorrect use of free

A:

```
/*
* An attempt to write a safe version of strndup() function
* Note: For this problem, assume that if the function returns a
* non-NULL pointer to dest, then the caller eventually frees the dest buffer.
*/

char *strndup(char *src, int max)
{
    char *dest;
    int i;

    if (!src || max <= 0)
```

```
        return NULL;

    dest = (char*)malloc(max);
    for (i=0; i < max && src[i] != '\0'; i++)
    {
        dest[i] = src[i];
    }
    dest[i] = '\0';
    return dest;
     }
```

___YES__Type of bug: __2,3

B:

```
/*
* Note: For this problem, the function below reverses a single linked list, and
* returns the head node of the new list.
*/

typedef int ElemType;
typedef struct Node
{
    ElemType data;
    struct Node *next;
}Node,*LinkList;

LinkList ListReverse(LinkList L)
{
    Node *temp;
    temp = L;
    LinkList nList;
   nList = (LinkList)malloc(sizeof(LinkList))

    nList->data = L->data;
    nList->next = NULL;

    while(L->next != NULL)
    {
        temp = nList->next;
        nList->next = L->next;
        L->next = L->next->next;
        nList->next->next = temp;

    }
    free(L);
```

return nList;

}

YES Type of bug: 2

3) Show the type of locality (temporal or spatial) for each object.

```
Code:
int a[10][20];
int b[10];
for (int i = 0; i < 10; ++i)
{
    for (int j = 0; j < 20; ++j)
    {
        a[i][j] += b[i];
    }
}
```

Object	Type of Locality
a	spatial
b	spatial
i	temporal
j	temporal

4) Do you think placement strategy and replacement strategy affect the miss rate of a cache? Give your reasons.

Both yes.

Placement strategy: If it's not properly selected, blocks being put into the cache may easily clash with each other. For example, let hash function be h(N)=N%8, then access sequence such as 0,8,16,24 will occur at the same place, which results clashes. But if we choose h(N)=N%7(a prime number), the situation will be better.

Replacement strategy: If not properly selected, thrashing will occur. Assume a cache with four blocks and takes FIFO replacement strategy. Now consider the following sequence of accesses 0, 1, 2, 3, 0, 2, 1, 0, 4, 0, 1, 2, 3.You will find that none of these accesses hits! If the cache takes LRU replacement strategy, the miss rate can be reduced because it uses locality better.