吴老师，骆学长和孙学长好，以下是我的作业。

上机随堂作业

– 教材

P. 186 题 14: 链表实现集合运算。要求：功能函数 1. void unionSet(struct List *s1, struct List *s2, struct List *result); 2. void differenceSet(struct List *s1, struct List *s2, struct List *result); 3. void intersectionSet(struct List *s1, struct List *s2, struct List *result)。测试要求：S1 = {2, 3, 5, 6}，S2 = {3, 4, 6, 8}。

P. 206 题 8: 自行创建一个.txt 文件，存储整数内容为“1, 2, 3, 3, 4, 4”，按要求编写功能子函数，并将输出结果和文件保存内容进行截图上传。

– 补充题 1：链表奇偶重排

给定一个带头结点的单链表，将链表重新排序，使所有奇数位置的节点排在偶数位置节点之前。要求保持节点的相对顺序不变。位置编号从 1 开始，1 为奇数位，2 为偶数位，以此类推。测试要求：1). 1->2->3->4->5->NULL (预计输出：1->3->5->2->4->NULL); 2). 2->8->3->7->NULL (预计输出：)

– 补充题 2：文本文件单词统计

编写程序统计一个文本文件中的单词数量，并找出出现次数最多的单词。（单词之间用空格或换行分隔，自行新建文件，内容：hello world hello C good good study）。

测试要求：总单词数：7 出现最多的单词：hello，good （次数：2）

Chi-Shan0707/Homework-in-CS10004-Programming-by-yhchi
代码仓库↑

T1.

核心思想：不断比较，不厌其烦地比较

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

// 单向链表节点
struct Node
{
    int val;
    struct Node* next;
};
typedef struct Node Node;
struct List
{
    Node* lsthead;
};
typedef struct List List;
void convert(List* lst, int arr[], int n)
```

```
{
    lst->lsthead = NULL;
    if (n <= 0) return;
    Node *head = ( Node *) malloc ( sizeof (Node) ) ;
    head->val = arr[0];
    head->next = NULL;
    lst->lsthead = head;
    Node* tail = head;
    for (int i = 1; i < n; ++i)
    {
        Node* cur = (Node*)malloc(sizeof(Node));
        cur->val = arr[i];
        cur->next = NULL;
        tail->next = cur;
        tail = cur;
    }
}
void free_list(List* lst)
{
//记得整体清空

    Node* head = lst->lsthead;
    while(head)
    {
        Node *tmp = head;
        head = head->next;
        free(tmp);
    }
    lst->lsthead = NULL;
    //记得置空
}
void print_list(const char* name, List* lst)
{
    printf("set %s  =    {", name);
    Node* head = lst->lsthead;
    int is_firstitem = 1;
    while (head)
    {
        if (!is_firstitem) printf(" , ");
        printf("%d", head->val);
        is_firstitem = 0;
        head = head -> next;
    }
    printf("}\n");
}
void append(List *lst, Node **tail_ptr,int val)
{
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->val = val;
```

```
    new_node->next = NULL;
    if (lst->lsthead == NULL)
    {
        lst->lsthead=new_node;
    }
    else
    {
        (*tail_ptr)->next=new_node;
    }
    *tail_ptr = new_node;
}
// 2. 差集 S1 \ S2
void differenceSet(struct List *s1, struct List *s2, struct List *res) {

    Node* head1 = s1->lsthead;
    Node* head2 = s2->lsthead;
    Node* tail = NULL;
    free_list(res);
    res->lsthead = NULL;
    while(head1!=NULL)
    {
        int exist_in_s2=0;
        for(head2=s2->lsthead; head2!=NULL; head2=head2->next)
        {
            if(head2->val==head1->val)
            {
                exist_in_s2=1;
                break;
            }
        }
        if(!exist_in_s2)append(res, &tail, head1->val);
        head1=head1->next;
    }

}
void intersectionSet(struct List *s1, struct List *s2, struct List *res)
{
    Node* head1 = s1->lsthead;
    Node* head2 = s2->lsthead;
    Node* tail = NULL;
    free_list(res);
    res->lsthead = NULL;
    while(head1!=NULL)
    {
        int exist_in_both=0;
        for(head2=s2->lsthead; head2!=NULL; head2=head2->next)
        {
            if(head2->val==head1->val)
            {
```

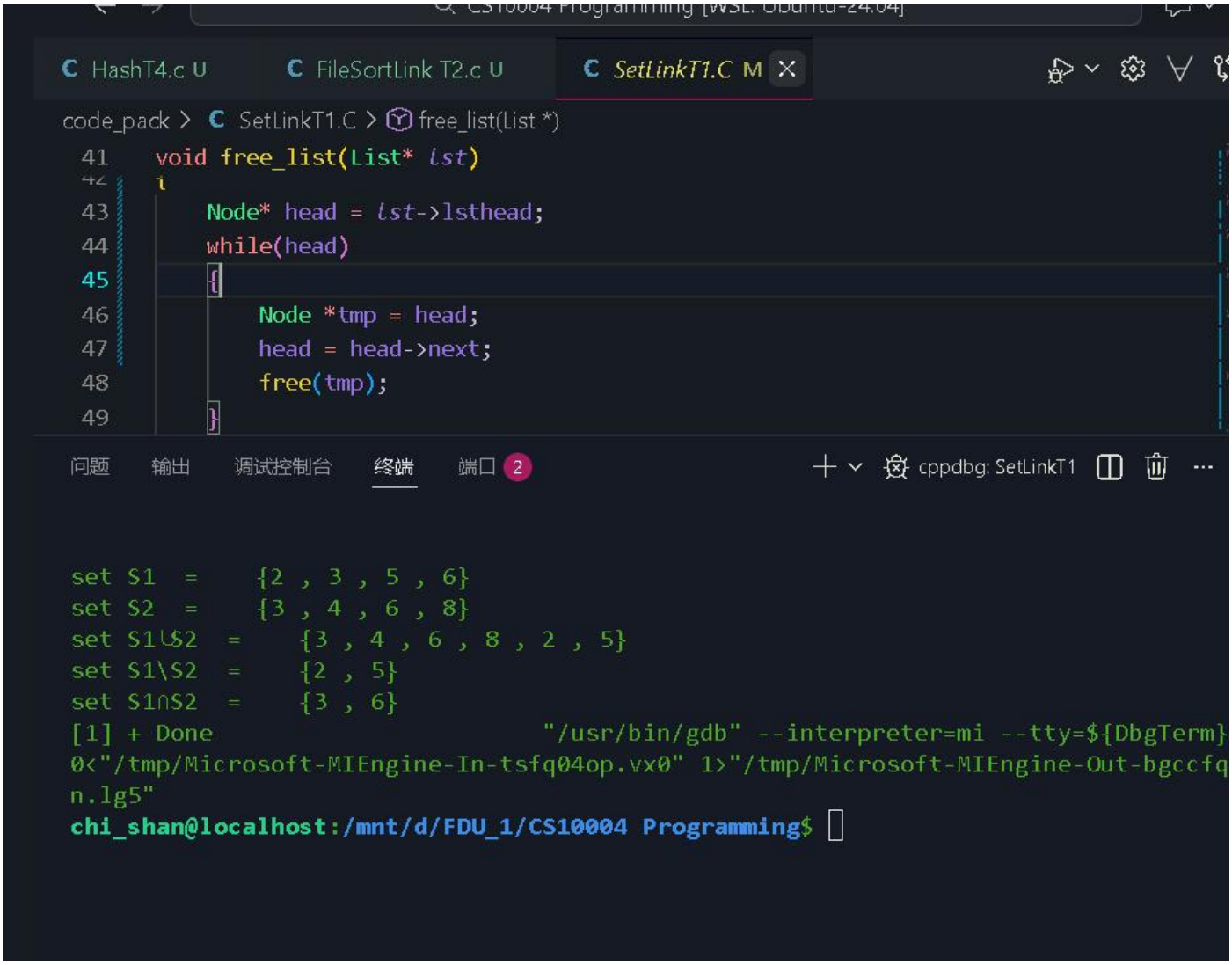
```
                exist_in_both=1;
                break;
            }
        }
        if(exist_in_both)append(res, &tail, head1->val);
        head1=head1->next;
    }
}
```

```
void unionSet(struct List *s1, struct List *s2, struct List *res)
{
    Node* head1 = s1->lsthead;
    Node* head2 = s2->lsthead;
    Node* tail = NULL;
    free_list(res);
    res->lsthead = NULL;
    head2=s2->lsthead;
    while(head2!=NULL)
    {
        append(res, &tail, head2->val);
        head2=head2->next;
    }
    head2=s2->lsthead;
```

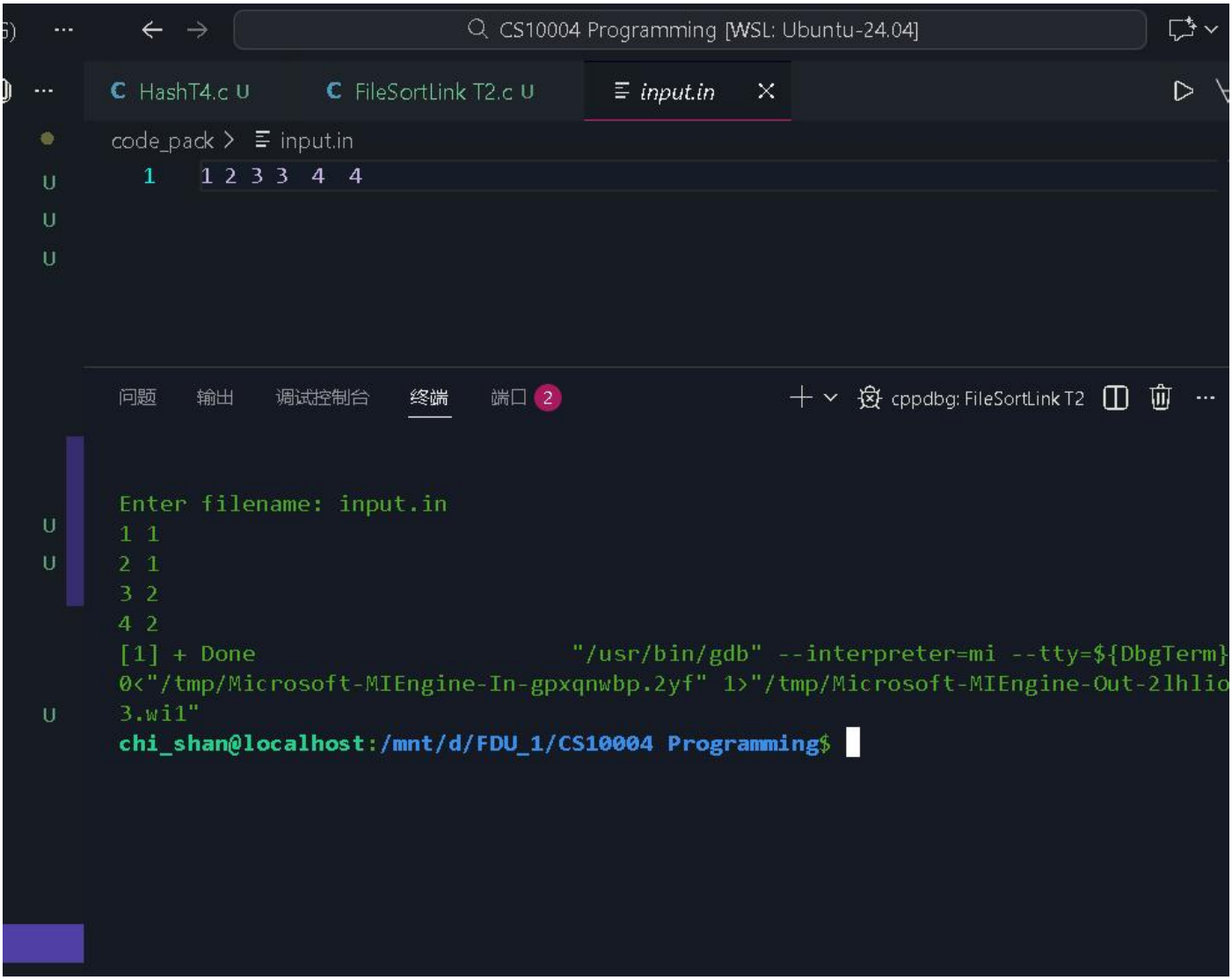
```
    while(head1!=NULL)
    {
        int exist_in_s2=0;
        for(head2=s2->lsthead; head2!=NULL; head2=head2->next)
        {
            if(head2->val==head1->val)
            {
                exist_in_s2=1;
                break;
            }
        }
        if(!exist_in_s2)append(res, &tail, head1->val);
        head1=head1->next;
    }

}
```

```
int main() {
    List S1, S2, U, D, I;
    int a1[] = {2, 3, 5, 6};
    int a2[] = {3, 4, 6, 8};
    convert(&S1, a1, sizeof(a1)/sizeof(a1[0]));
    convert(&S2, a2, sizeof(a2)/sizeof(a2[0]));
    U.lsthead = D.lsthead = I.lsthead = NULL;
```

```
    print_list("S1", &S1);
    print_list("S2", &S2);

    unionSet(&S1, &S2, &U);
    print_list("S1∪S2", &U);

    differenceSet(&S1, &S2, &D);
    print_list("S1\\S2", &D);

    intersectionSet(&S1, &S2, &I);
    print_list("S1∩S2", &I);

    // 释放
    free_list(&S1);
    free_list(&S2);
    free_list(&U);
    free_list(&D);
    free_list(&I);
    return 0;
}
```


HashT4.c U
FileSortLink T2.c U
SetLinkT1.C M
code_pack > SetLinkT1.C > free_list(List *)
41 void free_list(List* lst)
43 Node* head = lst->lsthead;
44 while(head)
45 {
46 Node *tmp = head;
47 head = head->next;
48 free(tmp);
49 }
问题 输出 调试控制台 终端 端口 2
cppdbg: SetLinkT1
set S1 = {2 , 3 , 5 , 6}
set S2 = {3 , 4 , 6 , 8}
set S1∪S2 = {3 , 4 , 6 , 8 , 2 , 5}
set S1\S2 = {2 , 5}
set S1∩S2 = {3 , 6}
[1] + Done "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm}
0<"/tmp/Microsoft-MIEngine-In-tsfq04op.vx0" 1>"/tmp/Microsoft-MIEngine-Out-bgccfq
n.lg5"
chi_shan@localhost:/mnt/d/FDU_1/CS10004 Programming$

T2.




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

struct Node
{
    int val;
    int count;
    struct Node* next;
};
typedef struct Node Node;
```

```c
Node* new_node(int v)
{
    Node* p = (Node*)malloc(sizeof(Node));
    if (!p) return NULL;
    p->val = v;
    p->count = 1;
    p->next = NULL;
    return p;
}
```

```c
void insert(Node** head, int v)
```

```
{
    Node* newn = new_node(v);
    Node* cur = *head;
    if (*head == NULL)
    {
        *head = newn;
        return;
    }
    if ((*head)->val > v)
    {
        newn->next = *head;
        *head = newn;
        return;
    }

    while (cur ->next != NULL && cur ->next->val< v)
    // 不断向后找，知道找到合适的位置
    {
        cur = cur ->next;
    }
    //插入排序√
    if (cur->next != NULL && cur ->next->val == v) {
        cur->next->count++;
        free(newn);
        return;
    }
    newn->next = cur->next;
    cur->next = newn;
}
```

```
void write_list(Node* head)
{
    Node* p = head;
    while (p != NULL)
    {
        printf("%d %d\n", p->val, p->count);
        p = p->next;
    }
}
void free_list(Node* head)
{
    Node* tail = head;
    while (head != NULL)
    {
        Node* temp = head;
        head = head->next;
        free(temp);
    }
}
```

```c
int main(void)
{
    char filename[100];Node* head = NULL;int num;
    printf("Enter filename: ");
    scanf("%s", filename);
    FILE* fp = fopen(filename, "r");
    if(!fp)
    {
        printf("404 NOT FOUND\n");
        return 1;
    }


    while(fscanf(fp, " %d", &num) == 1)
    {
        insert(&head, num);
    }
    fclose(fp);
    write_list(head);
    free_list(head);
    return 0;
}
```

T3

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

// 单向链表节点
struct Node
{
    int val;
    struct Node* next;
};
```

```c
typedef struct Node Node;
```

```c
// 表头结构
struct List
{
    Node* lsthead; // 注意：不使用辅助表元（无哨兵）
};
```

```c
typedef struct List List;
```

```c
// 从数组创建链表（数组应为已排序且不含重复元素以表示集合）
void convert(List* lst, int arr[], int n)
{
```

```
    lst->lsthead = NULL;
    if (n <= 0) return;
    Node *head = ( Node *) malloc ( sizeof (Node) ) ;
    head->val = arr[0];
    head->next = NULL;
    lst->lsthead = head;
    Node* tail = head;
    for (int i = 1; i < n; ++i)
    {
        Node* cur = (Node*)malloc(sizeof(Node));
        cur->val = arr[i];
        cur->next = NULL;
        tail->next = cur;
        tail = cur;
    }
}
```

```
void free_list(List* lst)
{
    Node* head = lst->lsthead;
    while(head)
    {
        Node *tmp = head;
        head = head->next;
        free(tmp);
    }
    lst->lsthead = NULL;
    //记得置空
}
```

```
void print_list(const char* name, List* lst)
{
    printf(" List %s  =    {    ", name);
    Node* head = lst->lsthead;
    int is_firstitem = 1;
    while (head)
    {
        if (!is_firstitem) printf(" -> ");
        printf("%d", head->val);
        is_firstitem = 0;
        head = head -> next;
    }
    printf("    }\n");
}
```

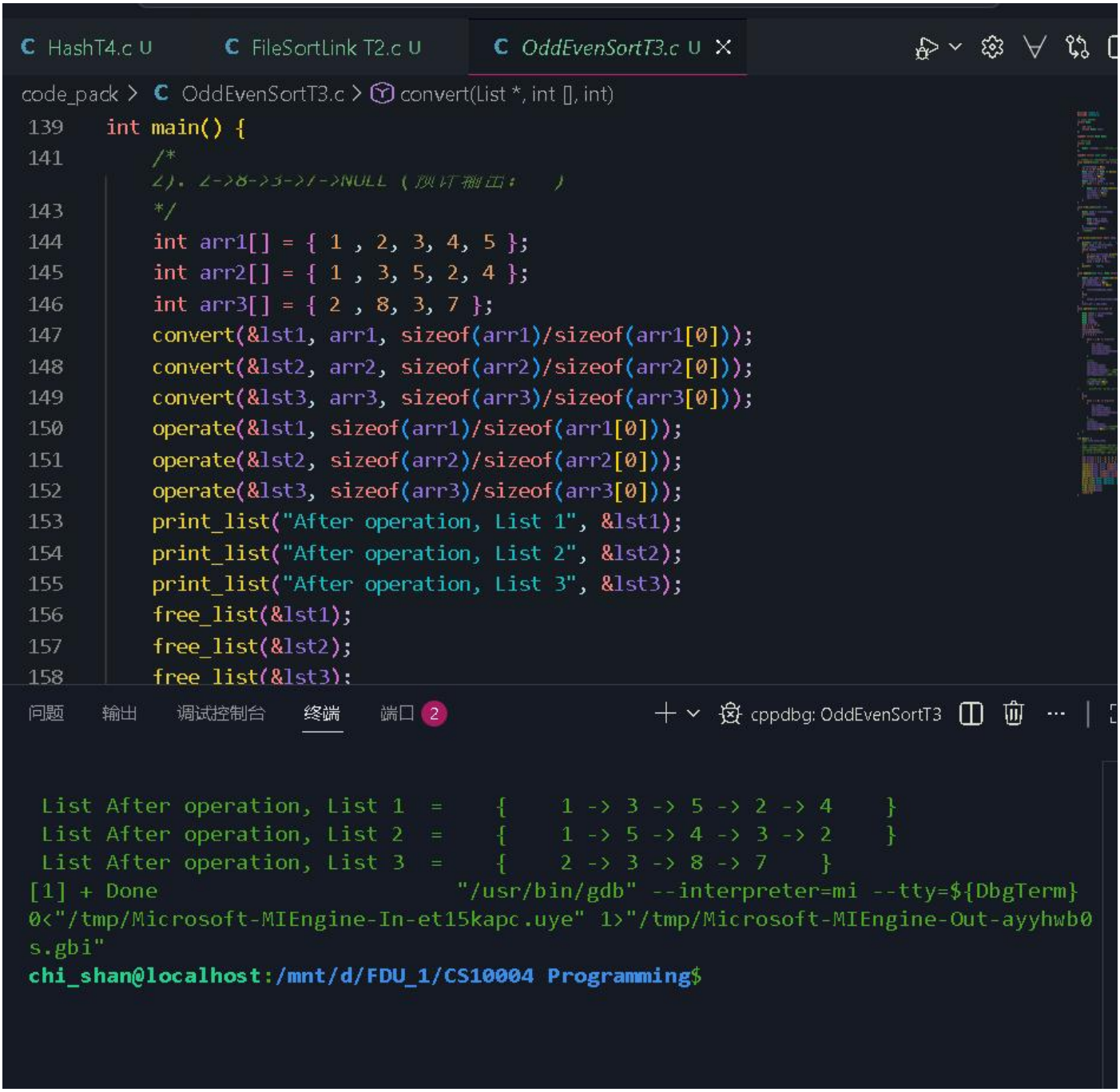
```
void append(List *lst, Node **tail_ptr,int val)
{
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->val = val;
```

```
    new_node->next = NULL;
    if (lst->lsthead == NULL)
    {
        lst->lsthead=new_node;
    }
    else
    {
        (*tail_ptr)->next=new_node;
    }
    *tail_ptr = new_node;
}
void operate(List *lst,int n)
{
    Node *head = lst->lsthead;
    Node *tail = head;
    Node *cur;
    Node *temp;
    Node *second;
    int m = n >> 1;
    int i = 0;
    if(n<=2)return;
    second=head->next;
    if ( n & 1 )
    {
        for( i = 0; i< m-1;++i)
        {
            cur =tail;
            tail=tail->next;
            tail=tail->next;
            (cur->next)->next=tail->next;
            cur->next=tail;
        }

        //剩仨
        cur=tail;
        tail=tail->next;
        tail=tail->next;
        tail->next=second;//原正数第二个
        second=cur->next;//原倒数第二个
        cur->next=tail;//倒数第三个接到最后一个

        //整条奇链的尾巴
        second->next=NULL;
        //重新配置空结点
```

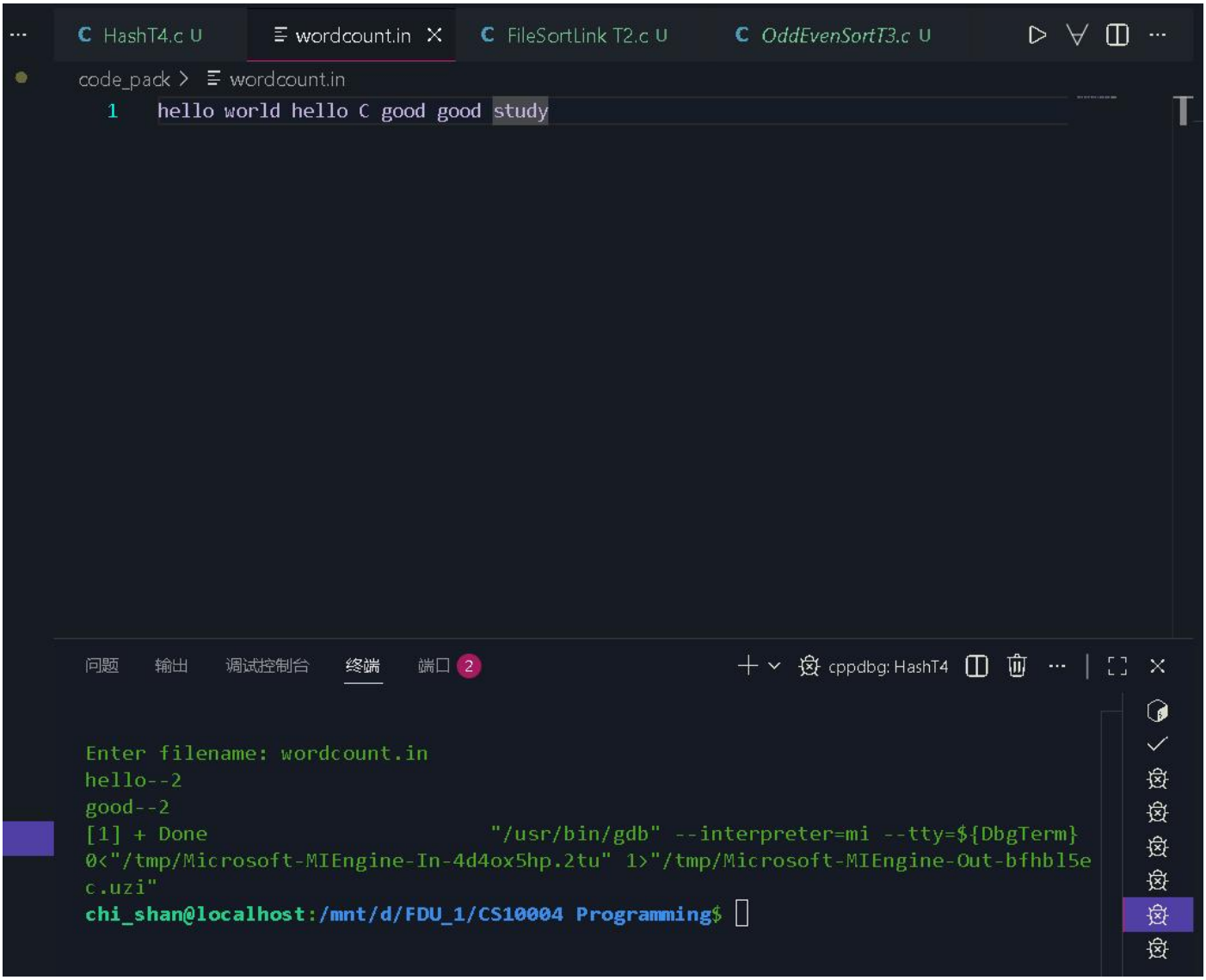
```
//        printf("%d  %d %d",cur->val,cur->next->val,cur->next->next->val);

    }
    else
```

```
    {
        for( i = 0; i< m-1;++i)
        {
            cur =tail;
            tail=tail->next;
            tail=tail->next;
            (cur->next)->next=tail->next;
            cur->next=tail;
        }
        //剩俩
        cur=tail;
        tail=tail->next;
        cur->next=second;//接入正数第二个
        tail->next=NULL;//尾巴变
    }
}
```

```
int main() {
    List lst1,lst2,lst3;
    /*
    给定一个带头结点的单链表，将链表重新排序，使所有奇数位置的节点排在偶数位置节点之前。要求保持节点的相对顺序不变。位置编号从 1 开始，1 为奇数位，2 为偶数位，以此类推。测试要求：1). 1->2->3->4->5->NULL (预计输出：1->3->5->2->4->NULL); 2). 2->8->3->7->NULL (预计输出：  )
    */
    int arr1[] = { 1 , 2, 3, 4, 5 };
    int arr2[] = { 1 , 3, 5, 2, 4 };
    int arr3[] = { 2 , 8, 3, 7 };
    convert(&lst1, arr1, sizeof(arr1)/sizeof(arr1[0]));
    convert(&lst2, arr2, sizeof(arr2)/sizeof(arr2[0]));
    convert(&lst3, arr3, sizeof(arr3)/sizeof(arr3[0]));
    operate(&lst1, sizeof(arr1)/sizeof(arr1[0]));
    operate(&lst2, sizeof(arr2)/sizeof(arr2[0]));
    operate(&lst3, sizeof(arr3)/sizeof(arr3[0]));
    print_list("After operation, List 1", &lst1);
    print_list("After operation, List 2", &lst2);
    print_list("After operation, List 3", &lst3);
    free_list(&lst1);
    free_list(&lst2);
    free_list(&lst3);
    return 0;
}
```


HashT4.c U
FileSortLink T2.c U
OddEvenSortT3.c U
code_pack > OddEvenSortT3.c > convert(List *, int [], int)
int main() {
/*
2). 2->28->23->27->NULL (预计输出:)
*/
int arr1[] = { 1 , 2, 3, 4, 5 };
int arr2[] = { 1 , 3, 5, 2, 4 };
int arr3[] = { 2 , 8, 3, 7 };
convert(&lst1, arr1, sizeof(arr1)/sizeof(arr1[0]));
convert(&lst2, arr2, sizeof(arr2)/sizeof(arr2[0]));
convert(&lst3, arr3, sizeof(arr3)/sizeof(arr3[0]));
operate(&lst1, sizeof(arr1)/sizeof(arr1[0]));
operate(&lst2, sizeof(arr2)/sizeof(arr2[0]));
operate(&lst3, sizeof(arr3)/sizeof(arr3[0]));
print_list("After operation, List 1", &lst1);
print_list("After operation, List 2", &lst2);
print_list("After operation, List 3", &lst3);
free_list(&lst1);
free_list(&lst2);
free_list(&lst3);
问题
输出
调试控制台
终端
端口 2
cppdbg: OddEvenSortT3
List After operation, List 1 = { 1 -> 3 -> 5 -> 2 -> 4 }
List After operation, List 2 = { 1 -> 5 -> 4 -> 3 -> 2 }
List After operation, List 3 = { 2 -> 3 -> 8 -> 7 }
[1] + Done "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm}
0<"/tmp/Microsoft-MIEngine-In-et15kapc.uye" 1>"/tmp/Microsoft-MIEngine-Out-ayyhwb0
s.gbi"
chi_shan@localhost:/mnt/d/FDU_1/CS10004 Programming$

T4




```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define BASE 53
#define MOD1 1013
#define MOD2 101
/*
数组的大小必须是“整数常量表达式”（integer constant expression），而 const int n = 10; 是一个对象定义，不被视作整数常量表达式，
所以不能用作全局数组大小；编译器因此把它当成可变长度（VLA），但 VLA 不能出现在文件作用域，导致“variably modified … at file scope”
错误。
*/
char *words[MOD1][MOD2];
int cnt[MOD1][MOD2];

void insert(char *str)
{
    long encode1=0;
    long encode2=0;
    for(int i=0;str[i];i++)
    {
        if(str[i]>='a'&&str[i]<='z')
```

```
        {
            encode1=encode1*BASE+str[i]-'a'+26;
            encode2=encode2*BASE+str[i]-'a'+26;
            encode1%=MOD1;
            encode2%=MOD2;
        }
        else
        {
            encode1=encode1*BASE+str[i]-'A';
            encode2=encode2*BASE+str[i]-'A';
            encode1%=MOD1;
            encode2%=MOD2;
        }
    }

   words[encode1][encode2] = str;
    //相信自己不会撞

    ++cnt[encode1][encode2];
  //    printf("test: %ld %ld %s \n",encode1,encode2,words[encode1][encode2]);
}
int main()
{
    int i,j,MX;int n=0;
    MX=0;
    char filename[1000];char INPUT[100][100];
    printf("Enter filename: ");
    scanf("%s", filename);
    FILE *fp=fopen(filename,"r");
    /*
    fscanf 的返回值主要有以下几种（假设没有其他错误）：
```

```
1. 正整数：表示成功读取的项数。
```

```
对于 fscanf(fp, " %s", word)，如果成功读取了一个字符串（单词），返回值是 1。
如果格式字符串中有多个 %s 或其他转换符，返回值会是实际读取的项数（例如，fscanf(fp, "%s %d", str, &num) 如果都成功，返回 2）。
2. 0：表示没有成功读取任何项，但文件没有结束。
```

```
这通常发生在格式不匹配时（例如，期望数字但遇到非数字字符）。对于 %s，这种情况很少见，因为 %s 会跳过空白并读取下一个单词，除非文件为
空或格式完全不匹配。
```

```
3. EOF（通常是 -1）：表示文件结束（EOF）或发生读取错误。
```

```
当到达文件末尾时，返回 EOF。
如果文件指针无效或发生 I/O 错误，也返回 EOF。
对于你的代码的具体分析
    */
    if(!fp)
    {
```

```
        printf("404 NOT FOUND\n");
        return 1;
    }
    memset(words, 0, sizeof(words));
    memset(cnt, 0, sizeof(cnt));
    while(fscanf(fp, " %s", INPUT[n])==1)
    {
        insert(INPUT[n]);
        ++n;
    }
    fclose(fp);
    /*
    `fclose(fp);` 用于关闭文件流，刷新缓冲区确保数据写入磁盘，并释放系统资源（如文件描述符）。
不关闭可能导致数据丢失或资源泄漏。
最佳实践：文件操作后及时调用。
    */

    for(i=0;i<MOD1;i++)
    {
        for(j=0;j<MOD2;j++)
        {
            MX=(cnt[i][j]>MX?cnt[i][j]:MX);
//        if(cnt[i][j])    printf("test: %d %d %s--%d\n",i, j, words[i][j],cnt[i][j]);
        }
    }
    for(i=0;i<MOD1;i++)
    {
        for(j=0;j<MOD2;j++)
        {
            if(cnt[i][j]==MX)
            {
                printf("%s--%d\n",words[i][j],MX);
            }
        }
    }
    return 0;
}
```