

Работа с файл

1зад. В двоичен файл имате цяло число N и след него N цели числа

- Да се напише програма, която създава файла
- Да се напише програма, която изкарва на екрана броя на четните и броя на нечетните от така дадените N числа
- Да се напише програма, която прочита така подадения масив от N елемента и изкарва в текстов файл сортирания във възходящ ред масив


```
FILE *fp;  
if((fp=fopen("fileN","wb"))==NULL)  
{  
    printf("Error opening file!!!\n");  
    exit(1);  
}  
fwrite(&n,sizeof(int),1,fp);
```

.....

```
fclose(fp);
```

```
-----  
if((fp=fopen("fileN","rb"))==NULL)  
{  
    printf("Error opening file for reading!!!\n");  
    exit(1);  
}  
fread(&n,sizeof(int),1,fp);
```

```
if((fp2=fopen("fileText","w"))==NULL)  
{  
    printf("Error opening file for reading!!!\n");  
    exit(1);  
}  
for(i=0;i<n;i++)  
{  
    fprintf(fp2,"%d ",mas[i]);  
}
```



2зад. Създайте структура служител съдържаща информация за име, брутна заплата и пол. Създайте елемент на списък със същата информация.

Напишете функция , която записва в двоичен файл информационната част на списъка. Името на файла е параметър на функцията.

Напишете функция, която връща сумата от заплатите на жените , имената ,на които започват с буква по-голяма от М.

Дефинирайте функция, която създава списък от служители с брутна заплата по-голяма от $\frac{1}{2} \max$ (\max е максималната заплата в списъка).

Напишете функция , която изтрива служителите , чието име започва с А.

```
struct Employee
```

```
{  
    char name[30];  
    float salary;  
    char gender;  
};
```

```
struct List
```

```
{  
    struct Employee employee;  
    struct List *next;  
};  
typedef struct List List;
```

```
void writeToFile(List *root, char fname[])
```

```
{  
    List *current;  
    FILE *file;  
    current = root;  
    if ((file = fopen(fname, "wb")) == NULL){  
        printf("Error!!");  
        exit(1);  
    }  
    else{  
        while (current != NULL)  
        {  
            fwrite(&current->employee, sizeof(struct Employee), 1, file);  
            current = current->next;  
        }  
        fclose(file);  
    }  
}
```

```
float Sum(List *root);
```

```
while (curr!= NULL)
```

```
{
```

```
    if (((curr->employee.name[0]) > 'M') && (curr->employee.gender == 'f'))
```

```
    {
```

```
        sum = sum + curr->employee.salary;
```

```
    }
```

```
    curr = curr->next;
```

```
}
```

```
List *New_stack(List* head);
```

```
List *curr_item = head;
```

```
List *item, *head2;
```

```
head2 = NULL;
```

```
item = NULL;
```

```
while (curr_item != NULL)
```

```
{
```

```
    if (curr_item->employee.salary > maxS/2)
```

```
    {
```

```
        item = (List *) malloc(sizeof(List));
```

```
        item->employee = curr_item->employee;
```

```
        item->next = head2;
```

```
        head2 = item;
```

```
    }
```

```
    curr_item = curr_item->next;
```

```
}
```

List *delete_item(List *root, char val)

```
{
List* prev_item=root;
List* curr_item=root;
while(curr_item!=NULL)
{
    if(curr_item->employee.name[0]==val)
    {
        if(curr_item == root) {
            root = root->next;
            prev_item = root;
            free(curr_item);
            curr_item=prev_item;
        }
        else{
            prev_item->next = curr_item->next;//
            free(curr_item);
            curr_item=prev_item->next; }
    }
    else{
        prev_item = curr_item;
        curr_item=curr_item->next;
    }
}
return root;
}
```