

## Задача

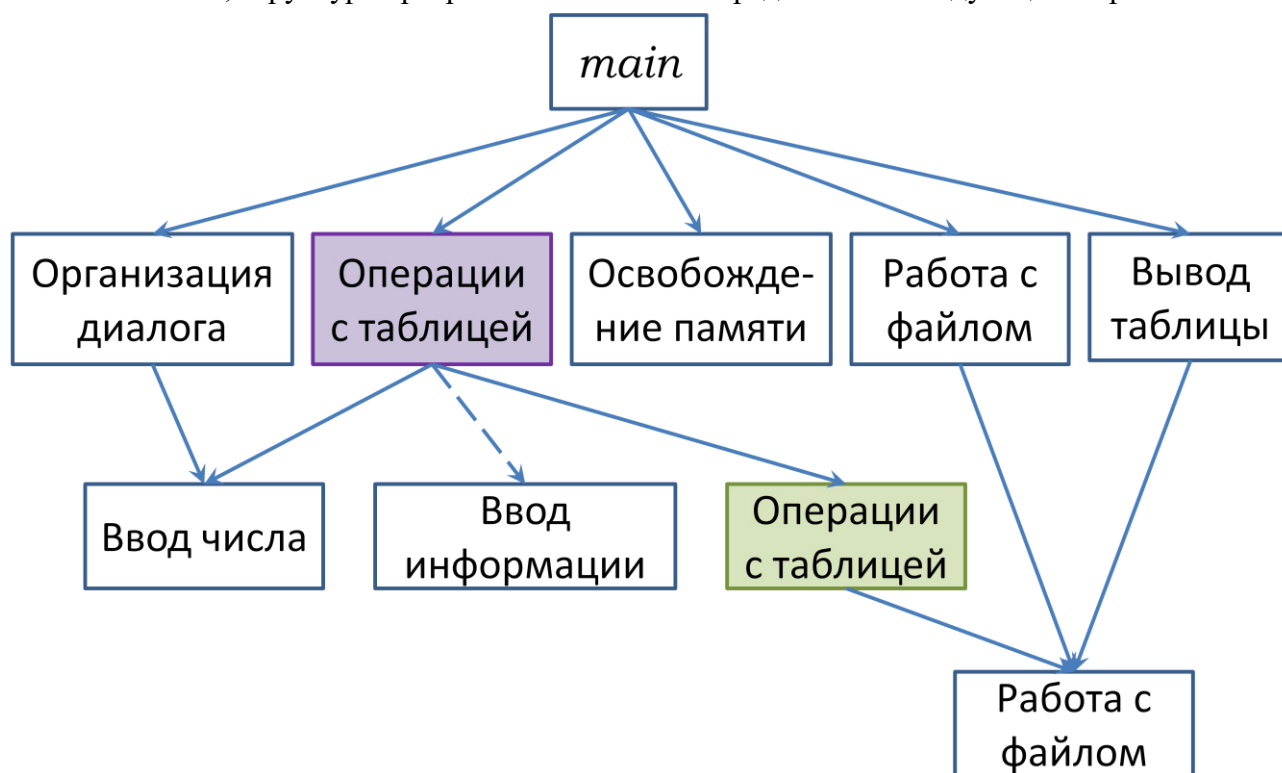
Написать программу для работы с таблицей по запросам оператора.

Вариант б)

- и сама таблица, и информация, относящаяся к элементу таблицы, хранятся во внешней памяти (используется двоичный файл произвольного доступа). Имя файла вводится по запросу из программы;
- все операции выполняются с таблицей, размещенной в основной памяти;
- таблица считывается из файла (или создается в первый раз) в начале сеанса работы и записывается в файл в конце сеанса работы;
- информация, относящаяся к элементу таблицы, записывается в файл сразу же при выполнении операции включения в таблицу. Файл, содержащий информацию, сохраняется открытым в течение всего сеанса работы программы.

## Структура программы

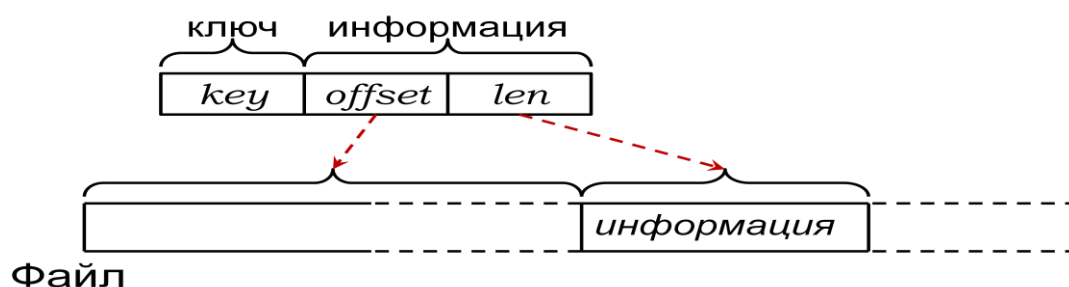
В основном, структура программы сохраняется такой же, как и в варианте а) задачи; добавляются только модули, ответственные за загрузку таблицы из файла и выгрузку таблицы в файл. Соответственно, структура программы может быть представлена следующим образом:



## Структура информации

Прежде всего, необходимо изменить структуру элемента таблицы и самой таблицы. Конечно, эта структура зависит от типа и способа организации таблицы, поэтому сначала рассмотрим простейшую просматриваемую таблицу с уникальными ключами, отображаемую в памяти машины вектором фиксированного размера.

Поскольку информация, ассоциированная с ключом, сразу же записывается в файл, в элементе таблицы надо указать место размещения соответствующей информации в файле.



Следовательно, элемент таблицы можно представить следующим образом:

```
struct Item{
    int key;           // ключ элемента таблицы (уникальный)
    int offset;        // смещение в файле (по отношению к началу файла)
    int len;           // длина информации
};
```

Далее, так как сама таблица (т.е. ключи и ассоциированная с ними информация) размещается в разных местах – и в оперативной памяти, и в файле, желательно всю относящуюся к таблице информацию разместить в одной структуре. При этом желательно организовать работу приложений таким образом, чтобы разные приложения имели возможность работать с таблицами, имеющими разные размеры векторов. Учитывая, что во время работы программы файл с информацией должен оставаться открытым, структура таблицы может быть определена следующим образом:

```
struct Table{
    // тип - просматриваемая таблица - вектор
    int SZ;           // размер вектора
    int n;            // размер таблицы
    Item *first;       // таблица - вектор
    FILE *fd;          // дескриптор файла, чтобы выполнять операции с файлом данных;
};
```

## Организация работы программы

По условиям задачи, сама таблица постоянно находится в оперативной памяти и сохраняется в файле в конце сеанса работы. Соответственно, функция main(), по сравнению с вариантом а), будет выглядеть следующим образом:

```
int main()
{
    Table table = {0, 0, NULL, NULL};
    int rc;

    if (D_Load(&table) == 0) //загружаем таблицу, подготавливаем к работе файл данных
        return 1;

    while(rc = dialog(msgs, NMsgs)) // диалог
        if(!fptr[rc](&table))
            break;

    D_Save(&table); // выгружаем таблицу, закрываем файл данных

    printf("That's all. Bye!\n");
    delTable(&table); // освобождение памяти, занятой таблицей
    return 0;
}
```

Диалоговые функции по сравнению с вариантом а) практически не меняются, но некоторые табличные функции изменятся.

Рассмотрим в качестве примера две функции.

### 1. Функция включения в таблицу нового элемента

А) Диалоговая функция

```
// Диалоговая функция включения в таблицу нового элемента.
// Требуется ввести ключ и информацию, при этом информация должна быть
// введена в новой строке, в ответ на приглашение.
// Если ключ задан неправильно, вся строка игнорируется
int D_Add(Table *ptab)
{
    int k, rc, n;
    char *info = NULL;
    printf("Enter key: -->");
    n = getInt(&k);
```

```

if(n == 0)
    return 0; // обнаружен конец файла

printf("Enter info:\n");
info = getStr(); // вся строка вводится целиком
if (info == NULL)
    return 0; // обнаружен конец файла

rc = insert(ptab, k, info); // вставка элемента в таблицу
free(info); // если элемент вставляется в таблицу - вставляется его копия

printf("%s: %d\n", errmsgs[rc], k);
return 1;
}

```

Б) Табличная функция

```

// Функция включения в таблицу нового элемента.
// В таблицу включается копия передаваемой информации
int insert(Table *ptab, int k, char *str)
{
    if(find(*ptab, k) >= 0)
        return 1; // поиск успешный - дублирование ключей
    if(ptab->n >= ptab->SZ)
        return 2; // переполнение таблицы
    ptab->first[ptab->n].key = k; // запись в таблицу ключа
    ptab->first[ptab->n].len = strlen(str) + 1; // запись в таблицу размера
                                           // информационного блока (включая нуль-байт)
    fseek(ptab->fd, 0, SEEK_END); // позиционирование на конец файла
    ptab->first[ptab->n].offset = ftell(ptab->fd); // запись в таблицу смещения
                                           // информации в файле (по отношению к началу файла)
    // запись информации в файл
    fwrite(str, sizeof(char), ptab->first[ptab->n].len, ptab->fd);
    ++(ptab->n); // изменение размера таблицы
    return 0;
}

```

## 2. Функция поиска в таблице элемента по ключу

А) Диалоговая функция

```

// Диалоговая функция поиска элемента в таблице по ключу
int D_Find(Table *ptab)
{
    char *info = NULL;
    int k, n, i;
    puts("Enter key: -->");
    n = getInt(&k);
    if(n == 0)
        return 0; // обнаружен конец файла

    info = findInfo(*ptab, k);
    if(info){
        printf("key = %d, info = \"%s\"\n", k, info);
        free(info);
    }
    else
        printf("Item %d was not found\n", k);
    return 1;
}

```

Б) Табличная функция 1 – проверка наличия элемента в таблице с указанным ключом

// Функция поиска в таблице элемента, заданного ключом

```

int find(Table ptab, int k)
{
    int i = 0;
    for(; i < ptab.n; ++i)
        if(ptab.first[i].key == k)
            return i;
    return -1;
}

```

В) Табличная функция 2 – поиска информации по заданному ключу

```

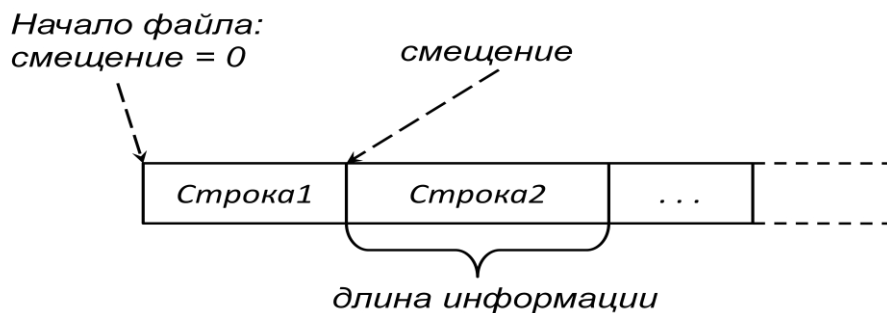
// Функция поиска в таблице информации для элемента, заданного ключом
char * findInfo(Table ptab, int k)
{
    char *info = NULL;
    int i = find(ptab, k);
    if (i >= 0){
        info = (char *)malloc(ptab.first[i].len);
        fseek(ptab.fd, ptab.first[i].offset, SEEK_SET);
        fread(info, sizeof(char), ptab.first[i].len, ptab.fd);
    }
    return info;
}

```

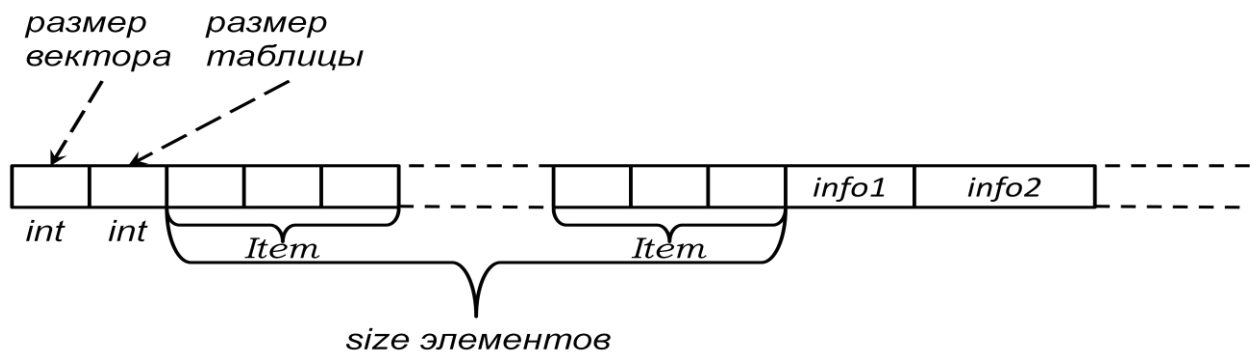
### Структура файла

Так как и сама таблица должна быть размещена в файле, рассмотрим структуру файла.

Структура файла данных может быть представлена следующим образом:



Так как таблица представлена вектором фиксированного размера, целесообразно саму таблицу сохранять в том же файле, где размещается и информация.



Соответственно, в начале сеанса работы необходимо загрузить таблицу из файла (или создать новый файл), а в конце сеанса работы – выгрузить таблицу из оперативной памяти в файл (информацию выгружать не надо – она уже размещена в файле).

### Загрузка таблицы из файла

Поскольку в начале сеанса работы надо либо открыть существующий файл, либо создать новый, целесообразно реализовать две разные табличные функции – загрузить таблицу из существующего файла (и эта функция должна вернуть код ошибки, если указанный файл не существует) и создать новый файл.

А) Табличная функция загрузки из существующего файла

```
// Функция загрузки таблицы из файла в оперативную память.
// Используется один файл, в котором сохраняются и данные и таблица.
// Функция открывает файл на чтение и запись.
// Файл остается открытым в течение всего сеанса работы.
int load (Table *ptab, char *fname)
```

```
{
    // открываем существующий файл таблицы на чтение и запись
    fopen_s(&(ptab->fd), fname, "r+b");
    if (ptab->fd == NULL)
        return 0;
    // файл открыт, можно читать; считываем размер вектора
    fread(&ptab->SZ, sizeof(int), 1, ptab->fd);
    // выделяем память под таблицу
    ptab->first = (Item *)calloc(ptab->SZ, sizeof(Item));
    // считываем размер таблицы
    fread(&ptab->n, sizeof(int), 1, ptab->fd);
    // считываем всю таблицу
    fread(ptab->first, sizeof(Item), ptab->n, ptab->fd);
    return 1;
}
```

Б) Табличная функция создания нового файла

```
// Функция создания новой таблицы. Создается пустая таблица размером sz
// и новый файл, в котором резервируется место для таблицы.
// Используется один файл, в котором сохраняются и данные и таблица.
// Файл остается открытым в течение всего сеанса работы.
```

```
int create(Table *ptab, char *fname, int sz)
{
    // должны создать пустую таблицу размером sz,
    // новый файл и зарезервировать в нем место для таблицы
    ptab->SZ = sz;
    ptab->n = 0;
    if (fopen_s(&(ptab->fd), fname, "w+b") != 0){
        ptab->first = NULL;
        return 0;
    }
    // выделяем память под таблицу, и сразу иницилируем ее нулями
    ptab->first = (Item *)calloc(ptab->SZ, sizeof(Item));
    // записываем в начало файла размер вектора
    fwrite(&ptab->SZ, sizeof(int), 1, ptab->fd);
    // записываем размер таблицы
    fwrite(&ptab->n, sizeof(int), 1, ptab->fd);
    // записываем саму таблицу
    fwrite(ptab->first, sizeof(Item), ptab->SZ, ptab->fd);
    return 1;
}
```

В) Диалоговая функция

```
// диалоговая функция загрузки таблицы из файла
```

```
int D_Load(Table *ptab)
{
    int SZ;
    char *fname = NULL;
    printf("Enter file name: --> ");
    fname = getStr(); // вся строка вводится целиком
    if(fname == NULL)
        return 0; // обнаружен конец файла
    if (load(ptab, fname) == 0){ // указанный файл не существует:
        // Надо создать новый файл и ноую
        таблицу
    }
}
```

```

    printf("Enter possible vector size: -->");
    if (getInt(&SZ) == 0)
        return 0; // обнаружен конец файла
    create(ptab, fname, SZ);
}
free(fname);
return 1;
}

```

### Выгрузка таблицы в файл

Нужно позиционировать на конец файла и сохранить таблицу в файле

```

// Функция выгрузки таблицы в файл.
// Используется один файл, в котором размещаются данные и таблица.
// Функция позиционирует на начало файла, записывает в файл таблицу
// и закрывает файл.
int save(Table *ptab)
{
    // пропускаем в начале файла поле для задания длины вектора
    fseek(ptab->fd, sizeof(int), SEEK_SET);

    // записываем в файл размер таблицы
    fwrite(&ptab->n, sizeof(int), 1, ptab->fd);

    // записываем в файл таблицу
    fwrite(ptab->first, sizeof(Item), ptab->n, ptab->fd);

    // закрываем файл
    fclose(ptab->fd);
    ptab->fd = NULL;
    return 1;
}

```

### Просматриваемая таблица-список

Элемент таблицы в данном варианте имеет следующую структуру:

```

struct Item{
    int key;
    int offset;
    int len;
    struct Item *next;
};

```

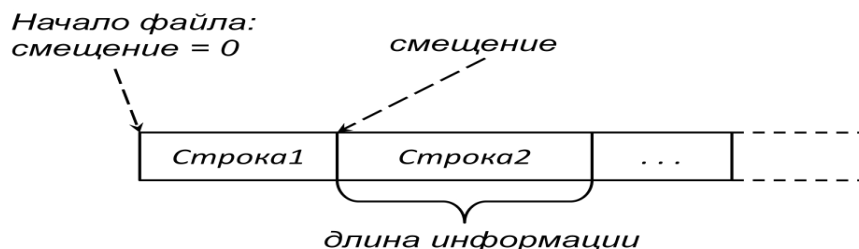
В данном случае потребуется уже два файла, так как невозможно заранее определить размер таблицы и зарезервировать пространство под таблицу. Тем не менее, при определении файла желательно ввести только одно имя (имена конкретных файлов будут определяться программным путем). Поэтому структура таблицы может быть определена следующим образом:

```

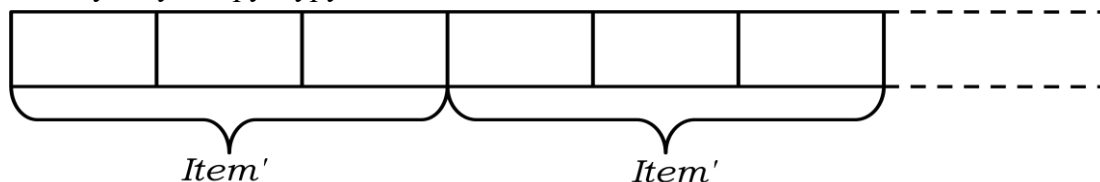
struct Table{
    Item *first; // просматриваемая таблица - список
    char *fname; // имя файла, с которым работает программа
    FILE *fd;    // сохраняем еще и дескриптор файла
};

```

Структура информационного файла (предположим, его имя заканчивается суффиксом .dat) не изменится – в нем только информация:



Файл, содержащий таблицу (табличный файл; имя файла будет заканчиваться суффиксом `.tab`) будет иметь следующую структуру:



Здесь элемент *Item'* отражает тот факт, что в файл заносятся только три поля из элемента списка (кроме указателя на следующий элемент списка).

### Загрузка таблицы из файла

Поскольку в начале сеанса работы надо либо открыть существующий файл (и загрузить из него таблицу), либо создать новый, здесь также можно реализовать две разные табличные функции – загрузить таблицу из существующего файла (и эта функция должна вернуть код ошибки, если указанный файл не существует) и создать новый файл. Но так как при создании нового файла никакая дополнительная информация не требуется, можно обе эти возможности реализовать в одной функции.

А) Табличная функция загрузки таблицы из файла

```
// Функция загрузки таблицы из файла в оперативную память.
// Используются два файла - данных и таблицы. Имена файлов
// отличаются суффиксами.
// Функция открывает оба файла: файл таблицы на чтение,
// файл данных на чтение и запись.
// Если файл таблицы не существует, создается пустая таблица.
// Если файл данных не существует, создается новый файл.
// Если файл таблицы открылся, а соответствующий файл данных – нет, ошибка.
// Файл таблицы закрывается после чтения, файл данных остается открытым.
int load (Table *ptab, char *fname)
{
    FILE *fd = NULL;
    char *fdatname; // имя файла для данных; имя файла для таблицы запишем
                  // в структуру Table
    int len = strlen(fname); // длина основной части имени файла
    Item *cur = NULL, *last = NULL; // вспомогательные переменные для создания списка
    Item item = { 0, 0, 0, NULL }; // вспомогательный элемент для чтения из файла

    // формируем имена файлов для таблицы и данных
    fdatname = _strdup(fname);
    fdatname = (char *)realloc(fdatname, len + 5);
    strcat_s(ptab->fname, len + 5, ".tab");
    fdatname = (char *)realloc(fdatname, len + 5);
    strcat_s(fdatname, len + 5, ".dat");

    // создаем пустую таблицу
    ptab->first = NULL;

    // открываем существующий файл таблицы на чтение
    fopen_s(&fd, ptab->fname, "rb");
    if (fd != 0) {
        // открываем существующий файл данных на чтение и запись;
        // если такого файла нет - ошибка
        fopen_s(&(ptab->fd), fdatname, "r+b");
        if (ptab->fd == NULL) {
            // освобождаем память
            free(fdatname);
            return 0;
        }
    }
}
```

```

// читаем элементы таблицы и заносим их в список
while (fread(&item, sizeof(int), 3, fd)){
    cur = (Item *)calloc(1, sizeof(Item));
    *cur = item;
    if(ptab->first == NULL)
        ptab->first = cur;
    else
        last->next = cur;
    last = cur;
} // конец цикла
// закрываем файл
fclose(fd);
}
else{
    // создаем файл данных
    fopen_s(&(ptab->fd), fdatname, "w+b");
    if (ptab->fd == NULL){
        // освобождаем память
        free(fdatname);
        return 0;
    }
}
Free(fdatname);
return 1;
}

```

Б) Диалоговая функция загрузки таблицы из файла

```

// диалоговая функция загрузки таблицы из файла
int D_Load(Table *ptab)
{
    int rc = 1;
    char *fname = NULL;
    printf("Enter file name: --> ");
    fname = getStr(); // вся строка вводится целиком
    if(fname == NULL)
        return 0; // обнаружен конец файла
    rc = load(ptab, fname);
    if (rc == 0)
        puts("The appropriate data file is not exists");
    free(fname);
    return rc;
}

```

### Выгрузка таблицы в файл

Нужно пересоздать файл и сохранить таблицу в файле; информационный файл просто закрывается.

```

// Функция выгрузки таблицы в файл.
// Используются два файла - данных и таблицы. Имена файлов
// отличаются суффиксами.
// Функция пересоздает файл таблицы, записывает в него элементы таблицы
// и закрывает файл. Файл данных открыт, его надо просто закрыть.
int save(Table *ptab)
{
    FILE *fd = NULL;
    Item *cur = NULL;

    // закрываем файл данных

```



```
fclose(ptab->fd);

// в таблице есть имя файла; работаем с ним
// создаем новый файл таблицы
fopen_s(&fd, ptab->fname, "wb");
if(fd == NULL)
    return 0;    // ошибка при создании файла

// записываем в файл элементы таблицы
for(cur = ptab->first; cur; cur = cur->next){
    ++n;
    fwrite(cur, sizeof(int), 3, fd);
}

// закрываем файл
fclose(fd);
return 1;
}
```