Задача

Написать программу для работы с таблицей по запросам оператора.

Предусмотреть следующие операции:

- вставка нового элемента,
- удаление элемента,
- поиск элемента,
- вывод содержимого таблицы.

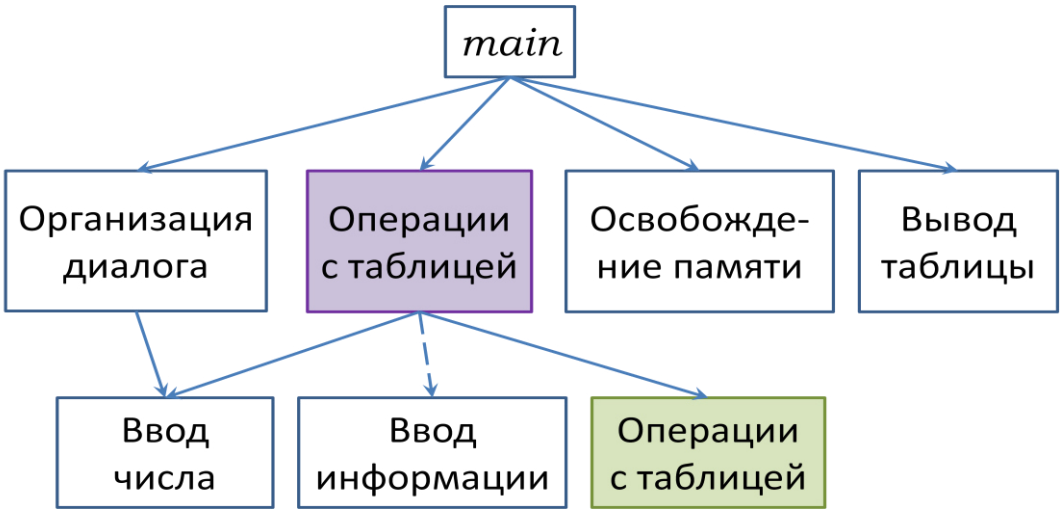
Вариант а)

– 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.

Структура программы

- функция *main()*,
- функции ввода данных с необходимыми проверками,
- функции организации диалога,
- функции для работы с таблицами.

Соответственно, структура программы может быть представлена следующим образом:



Структура информации

Прежде всего, необходимо определить 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.

Соответственно, элемент таблицы можно представить следующим образом:

```
struct Item{
      int key;          // ключ элемента таблицы (уникальный)
      char * info;      // информация, ассоциированная с данным ключом
};
```

Далее, так как сама таблица может изменяться при выполнении различных операций с ней, желательно всю относящуюся к таблице информацию разместить в одной структуре. Следовательно, структура таблицы может быть определена следующим образом:

```
const int SZ = 10;      // размер вектора

struct Table{           // тип - просматриваемая таблица - вектор
      int n;            // размер таблицы
      Item first[SZ];   // таблица – вектор фиксированного размера
```

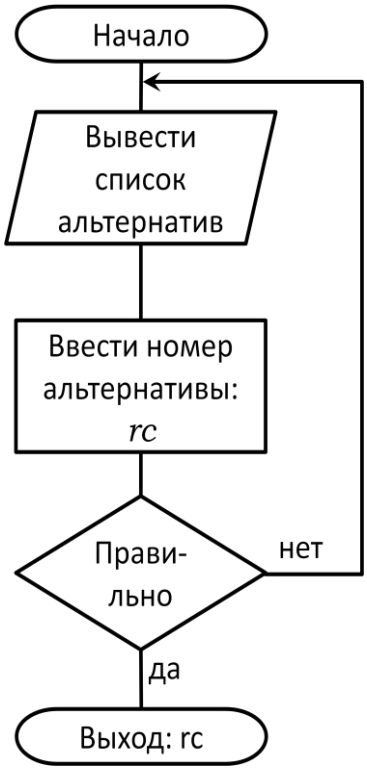
```
};
```

Значение константы SZ определяется по желанию (например, 10 или 100), но желательно иметь возможность изменять это значение (между запусками программы), чтобы проверить разные ситуации, которые могут возникнуть при выполнении операций с таблицей.

Организация работы программы

По условиям задачи, работа с таблицей осуществляется по запросам оператора. Это означает, что оператор может выполнять интересующие его операции в произвольном порядке. Следовательно, необходимо организовать диалог. Поскольку все программы в курсе реализуются на основе консольного приложения, диалог целесообразно организовать в виде альтернатив: на экран выводится список альтернатив, определяющих доступные операции, и пользователь (оператор) выбирает нужную операцию, вводя номер альтернативы. Организация диалога должна контролировать возможные ошибки при вводе и должна отрабатывать состояние «конец файла», которое всегда может возникнуть при вводе.

Алгоритм организации диалога может иметь следующий вид:

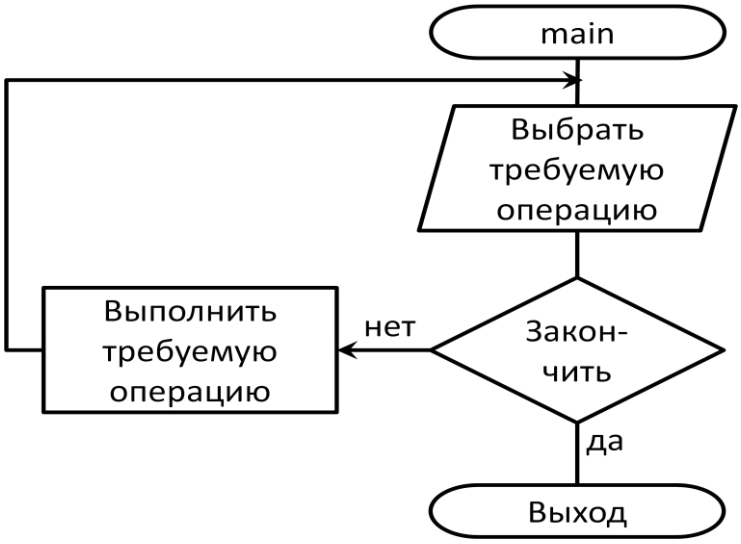


Определение функции приведено ниже:

```
// Альтернативы меню для организации диалога
// список альтернатив
const char *msgs[] = {"0. Quit", "1. Add", "2. Find", "3. Delete", "4. Show"};
// количество альтернатив
const int NMsgs = sizeof(msgs) / sizeof(msgs[0]);

// выбор номера альтернативы
int dialog(const char *msgs[], int);

// Функция для выбора номера альтернативы.
// Возвращает правильный номер альтернативы.
// В строке может быть указан только номер альтернативы;
// если в строке после номера альтернативы есть что-то еще,
// весь остаток строки удаляется из буфера

int dialog(const char *msgs[], int N)
{
```

```
        char *errmsg = "";
        int rc;
        int i, n;

        do{
                puts(errmsg);
                errmsg = "You are wrong. Repeate, please!";

                // вывод списка альтернатив
                for(i = 0; i < N; ++i)
                        puts(msgs[i]);
                puts("Make your choice: --> ");

                n = getInt(&rc); // ввод номера альтернативы
                if(n == 0) // конец файла - конец работы
                        rc = 0;
        } while(rc < 0 || rc >= N);

        return rc;
}
```

В этом случае в функции `main()`, на основании введенного номера альтернативы, выполняется соответствующая функция из группы функций организации диалога, которая вводит необходимые данные и вызывает соответствующую функцию работы с таблицей. Схема алгоритма функции `main()` приведена ниже:



Следует внимательно отнестись к тому, как определить требуемую операцию. В зависимости от введенного номера альтернативы, необходимо вызывать на исполнение разные функции. Первым (достаточно очевидным) решением является использование предложения `if`:

`if` (выбрана альтернатива 1)

вызвать диалоговую функцию вставки элемента в таблицу;

`else`

`if`(выбрана альтернатива 2)

вызвать диалоговую функцию поиска элемента в таблице;

. . .

Очевидным недостатком такого подхода является то, что при изменении набора альтернатив придется перепрограммировать функцию `main()`, что, конечно, является не очень хорошим решением.

Поэтому здесь следует воспользоваться теми возможностями, которые предоставляет язык Си в организации работы с функциями. В языке Си функция представляет собой такой же объект, как и другие (например, массив). Для этого объекта так же определены некоторый тип данных и допустимые операции. Так, имя функции ассоциируется с адресом её размещения в памяти машины, а адреса можно сохранять в каких-либо других объектах программы. Так, например, если имеется некоторая функция с прототипом `int f(int)`, тогда имя функции определяется как адрес размещения в памяти машины функции с одним аргументом типа `int`, возвращающей результат типа `int`. Адреса же в языке Си определяются типом указателя.

Следовательно, можно определить переменную, значением которой может быть адрес функции: `int (*fptr)(int)`. Такое определение, с учетом приоритетов операторов языка и круглых скобок, читается так: `fptr` есть указатель (так как использованы круглые скобки – `(*fptr)`) на функцию (следующая пара круглых скобок интерпретируется как оператор вызова функции), имеющую один параметр типа `int` и возвращающую результат типа `int`. Такую переменную можно проинициализировать: `fptr = f;` (обратите внимание на то, что справа от присваивания стоит только имя функции, а не её вызов). Далее можно вызвать на исполнение ту функцию, которая определена в переменной `fptr`: `fptr(5);`

Объекты типа указателей на функции можно объединять в массивы, при этом в массив должны включаться указатели на функции с одинаковыми прототипами. Соответственно, в данной задаче мы можем определить необходимые диалоговые функции по работе с таблицами и объединить указатели на эти функции в массив:

```
// функции для организации диалога;
// при обнаружении конца файла возвращают 0
int D_Add(Table *),     // вставка элемента в таблицу
    D_Find(Table *),    // поиск элемента в таблице
    D_Delete(Table *), // удаление элемента из таблицы
    D_Show(Table *),    // вывод содержимого таблицы
```

Прототипы всех функций должны быть одинаковыми. Функция вставки элемента в таблицу может изменить данные таблицы, поэтому требуется передавать в функцию указатель на таблицу. Отсюда, и все остальные функции используют параметр типа указатель.

Теперь можно определить и проинициализировать массив указателей на функцию с таким прототипом:

```
// массив указателей на функции - для реализации выбора функции
// порядок перечисления функций в списке инициализации
// должен соответствовать порядку указания альтернатив в списке альтернатив
int (*fptr[])(Table *) = {NULL, D_Add, D_Find, D_Delete, D_Show};
```

Тогда функция `main()` будет иметь следующую реализацию:

```
int main()
{
  Table table = ...; // создается пустая таблица соответствующего типа
  int rc;

  while(rc = dialog(msgs, NMsgs))
       if(!fptr[rc](&table)) // вызывается диалоговая функция,
                             // соответствующая введенному номеру альтернативы
            break;

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

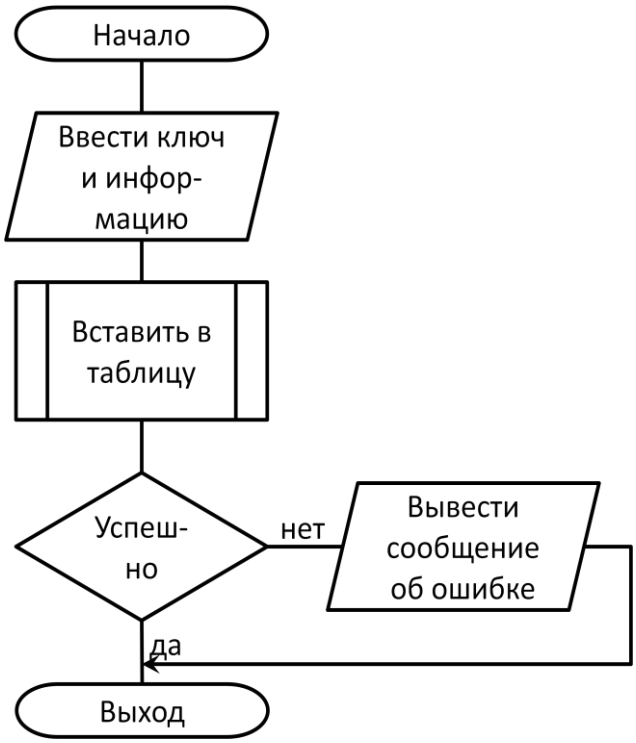
```
                                // если использовалась динамическая память
    return 0;
}
```

Очень важно разделять выполняемые действия: функции обработки таблиц не должны использовать ввод/вывод, так как они могут вызываться из любого, в том числе, не консольного, приложения. Ввод необходимых для выполнения данных и вывод результатов должны выполняться в диалоговых функциях. Кроме того, необходимо учитывать, как представляется информация, которая должна сохраняться в таблице (обычно соответствующие данные передаются в хранилище по значению, и в хранилище заносится копия данных). В соответствии с этим требованием разрабатываются необходимые диалоговые и табличные функции.

Реализация табличных функций здесь не рассматривается – соответствующие алгоритмы были подробно рассмотрены на лекциях.

1. Диалоговая функция включения в таблицу нового элемента

А) Алгоритм функции



Б) Реализация функции

Для формирования и вывода диагностических сообщений о тех или иных ошибках, встретившихся при выполнении операции вставки нового элемента в таблицу, целесообразно использовать массив диагностических сообщений. Функция, обнаружившая в своей работе соответствующую ошибку, возвращает её код завершения (т.е. индекс сообщения в массиве сообщений).

```
// Массив диагностических сообщений об ошибках
const char *errmsgs[] = {"Ok", "Duplicate key", "Table overflow"};

// Диалоговая функция включения в таблицу нового элемента.
// Требует ввести ключ и информацию, при этом информация должна быть введена
// в отдельной строке, в ответ на приглашение.
// Если ключ задан неправильно, вся строка игнорируется.
// Функция возвращает:     1, если операция была завершена успешно,
```

```
//                    0, если обнаружен конец файла.
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;
}
```

Остальные диалоговые функции оформляются по такой же схеме.