МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННО БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебный Центр Информационных Технологий «Информатика»

Лабораторная работа № 2
по дисциплине «Информатика и программирование 2 часть»

Выполнил слушатель: Пешков Е.В.
Вариант: 9
Дата сдачи:
Преподаватель: Юшманов А.А.

Новосибирск, 2019г.

1. **Цель**

   Научиться решать задачи с использованием строк.

2. **Вариант задания**

   Определить структурированный тип, определить набор функций для работы с массивом структур. В структурированной переменной предусмотреть способ отметки ее как не содержащей данных (т.е. "пустой"). Функции должны работать с массивом структур или с отдельной структурой через указатели, а также при необходимости возвращать указатель на структуру. В перечень функций входят:

   - «очистка» структурированных переменных;

   - поиск свободной структурированной переменной;

   - ввод элементов (полей) структуры с клавиатуры;

   - вывод элементов (полей) структуры с клавиатуры;

   - поиск в массиве структуры и минимальным значением заданного поля;

   - сортировка массива структур в порядке возрастания заданного поля (при сортировке можно использовать тот факт, что в Си++ разрешается присваивание структурированных переменных);

   - поиск в массиве структур элемента с заданным значением поля или с наиболее близким к нему по значению.

   - удаление заданного элемента;

   - изменение (редактирование) заданного элемента.

   - вычисление с проверкой и использованием всех элементов массива по заданному условию и формуле (например, общая сумма на всех счетах) - дается индивидуально.

**Вариант 5**

3. Фамилия И.О., количество оценок, оценки, средний балл.

4.

5. **Теория**

**Указатели и ссылки**

Объект, указатель и ссылка

Указатели совместно с адресной арифметикой играют в Си особую роль. Можно сказать, что они определяют лицо языка. Благодаря им Си может считаться одновременно языком высокого и низкого уровня по отношению к памяти.

Если говорить о понятиях указатель, ссылка, объект, то они встречаются не только в языках программирования, но в широком смысле в информационных технологиях. Когда речь идет о доступе к информационным ресурсам, то существуют различные варианты доступа к ним:
**копия (значение, объект) –** пользователь получает точную копию информационного ресурса в момент доступа к ней (например, копию файла, таблицы базы данных и т.п.). Он может как ему угодно изменять его содержимое, что не отражается на оригинале;
**указатель –** адресная информация о расположении информационного ресурса, через которую пользователь может обратиться к нему. При изменении содержимого объекта через указатель на него всегда возникает проблема **синхронизации (разделения)** ресурса между несколькими пользователями, имеющими адресную информацию о нем. Синонимом указателя в информационных технологиях является **ссылка.** Иногда она имеет все внешние признаки объекта, например, ярлык файла на рабочем столе, который внешне выглядит как файл, а на самом деле ссылается на файл-оригинал.

В языках программирования термины объект (значение), указатель и ссылка имеют примерно аналогичный смысл, но касаются способов доступа и передачи значений переменных.

· терминология **ссылка, значение** касается фундаментальных свойств переменных в языках программирования. Имя переменной в различных контекстах может восприниматься как ее значение (содержимое памяти), так и ссылка на нее (адрес памяти, указатель). Например, при присваивании левая часть рассматривается как ссылка, а правая – как значение);

· при передаче формальных параметров при вызове процедур (функций) практически во всех языках программирования реализованы способы передачи **по ссылке** и **по значению;**

· в Паскале и Си определено понятие **указатель** как переменная особого вида, содержащая адрес размещения в памяти другой переменной. Использование указателей позволяется создавать динамические структуры данных, в которых элементы взаимно ссылаются друг на друга;

· и, наконец, в Си существует расширенная интерпретация указателя, именуемая **адресной арифметикой,** которая позволяет интерпретировать значение любого указателя как адрес не отдельной переменной, а памяти в целом, где она размещена.

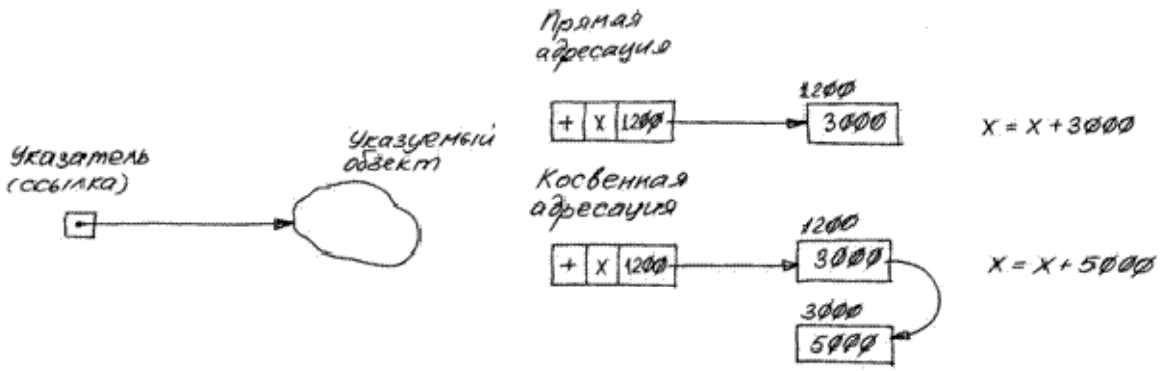
**Указатель в Си**

Передавать данные между программами, данные от одной части программы к другой (например, от вызывающей функции к вызываемой) можно двумя способами:

· создавать в каждой точке программы (например, на входе функции) копию тех данных, которые необходимо обрабатывать;

· передавать информацию о том, где в памяти расположены данные. Такая информация, естественно, является более компактной, чем сами данные, и ее условно можно назвать указателем. Получаем «дилетантское» определение указателя: указатель - переменная, содержащая информацию о расположении в памяти другой переменной.

Наряду с указателем в программировании также используется термин **ссылка.** Ссылка – содержанием ссылки также является адресная информация об объекте (переменной), но внешне она выглядит как переменная (синоним оригинала).




**Указатель как элемент архитектуры компьютера.** Указатели занимают особое место среди типов данных, потому что они проецируют на язык программирования ряд важных принципов организации обработки данных в компьютере. Понятие указателя связано с такими понятиями компьютерной архитектуры как адрес, косвенная адресация, организация внутренней (оперативной) памяти. От них мы и будем отталкиваться**. Внутренняя (оперативная) память** компьютера представляет собой упорядоченную последовательность байтов или машинных слов (ячеек памяти), проще говоря - массив. Номер байта или слова памяти, через который оно доступно как из команд компьютера, так и во всех других случаях, называется **адресом**. Если в команде непосредственно содержится адрес памяти, то такой доступ этому слову памяти называется **прямой адресацией**.

Возможен также случай, когда машинное слово содержит адрес другого машинного слова. Тогда доступ к данным во втором машинном слове через первое называется **косвенной адресацией**. Команды косвенной адресации имеются в любом компьютере и являются основой любого регулярного процесса обработки данных. То же самое можно сказать о языке программирования. Даже если в нем отсутствуют указатели, как таковые, работа с массивами базируется на аналогичных способах адресации данных.

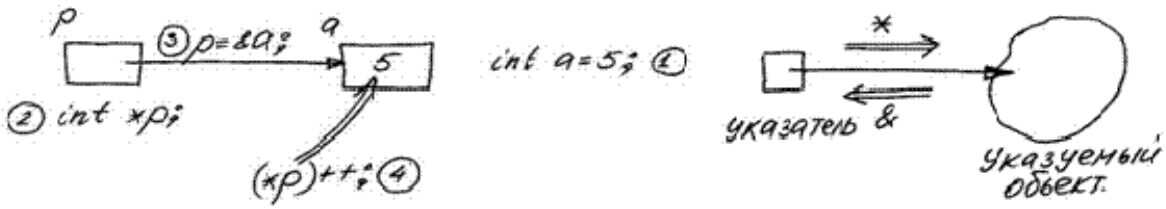
В языках программирования имя переменной ассоциируется с адресом области памяти, в которой транслятор размещает ее в процессе трансляции программы. Все операции над обычными переменными преобразуются в команды с прямой адресацией к соответствующим словам памяти.

Таким образом, в компьютерной архитектуре **указатель - переменная, содержимым которой является адрес другой переменной**.

Соответственно, основная операция для указателя - это косвенное обращение по нему к той переменной, адрес которой он содержит. В Си имеется специальная операция ***** - звездочка, которую называют **косвенным обращением по указателю.** В более широком смысле ее следует понимать как переход от переменной-указателя к той переменной (объекту), на которую он ссылается. В дальнейшем будем пользоваться такими терминами:

· указатель, который содержит адрес переменной, **ссылается** на эту переменную или **назначен** на нее;

· переменная, адрес которой содержится в указателе, называется **указуемой** переменной.




Последовательность действий при работе с указателем включает 3 шага:

1. Определение указуемых переменных и переменной-указателя. Для переменной-указателя это делается особым образом.

```
int     a,x;    // Обычные целые переменнные
int     *p;     // Переменная - указатель на другую  целую переменную
```

В определении указателя присутствует та же самая операция косвенного обращения по указателю. В соответствии с принципами контекстного определения типа переменной (см. 5.5) эту фразу следует понимать так: переменная **p** при косвенном обращении к ней дает переменную типа **int.** То есть свойство ее – быть указателем, определяется в контексте возможного применения к ней операции ***.** Обратите внимание, что в определении присутствует **указуемый тип данных.** Это значит, что указатель может ссылаться не на любые переменные, а только на переменные заданного типа, то есть указатель в Си **типизирован.**

2. Связывание указателя с указуемой переменной. Значением указателя является адрес другой переменной. Следующим шагом указатель должен быть настроен, или **назначен** на переменную, на которую он будет ссылаться.

```
p = &a; // Указатель содержит адрес переменной a
```

Операция **&** понимается буквально как адрес переменной, стоящей справа от нее. В более широкой интерпретации она «превращает» объект в указатель на него (или производит переход от объекта к указателю на него) и является в этом смысле прямой противоположностью опера-

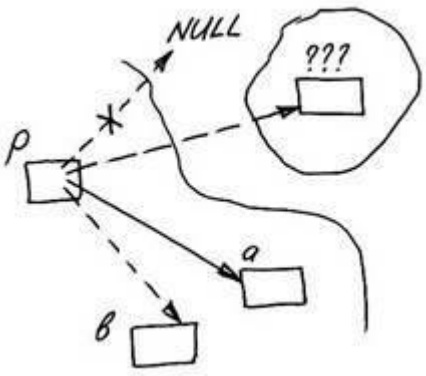
ции ***,** которая «превращает» указатель в указуемый объект. То же самое касается типов данных. Если переменная **a** имеет тип **int,** то выражение **&a** имеет тип – указатель на **int** или **int*.**

3. И наконец, в любом выражении косвенное обращение по указателю интерпретируется как переход от него к указуемой переменной с выполнением над ней всех далее перечисленных в выражении операций.

```
*p=100;          // Эквивалентно a=100
x = x + *p;      // Эквивалентно x=x+a
(*p)++;          // Эквивалентно a++
```

*Замечание:* при обращении через указатель имя указуемой переменной в выражении отсутствует. Поэтому можно считать, что обращение через указатель производится к «безымянной» переменной, а операцию «*» называются также операцией **разыменования указателя.**

Указатель дает «**степень свободы**» или универсальности любому алгоритму обработки данных. Действительно, если некоторый фрагмент программы получает данные непосредственно в некоторой переменной, то он может обрабатывать ее и только ее. Если же данные он получает через указатель, то обработка данных (указуемых переменных) может производиться в любой области памяти компьютера (или программы). При этом сам фрагмент может и «не знать», какие данные он обрабатывает, если значение самого указателя передано программе извне.



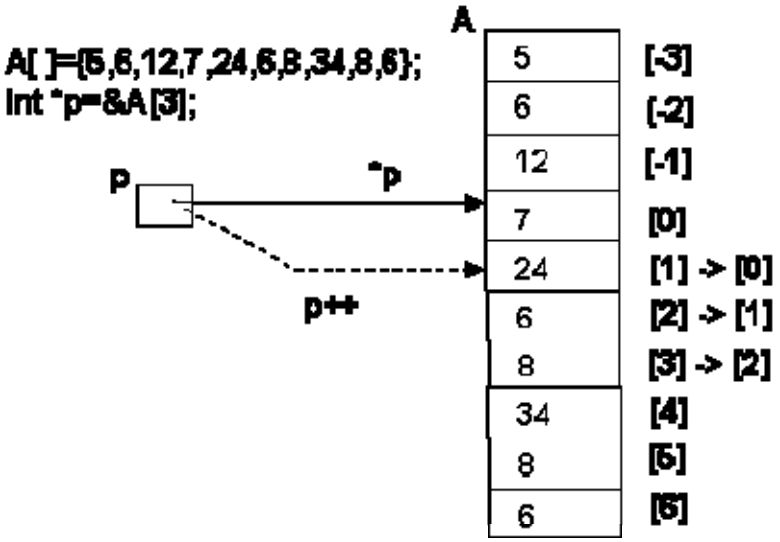

Адресная арифметика и управление памятью

Способность указателя ссылаться на «отдельно стоящие» переменные не меняет качества языка, поскольку нельзя выйти за рамки множества указуемых переменных, определенных в программе. Такая же концепция указателя принята, например, в Паскале. Но в Си существует еще одна, расширенная интерпретация, позволяющая через указатель работать с массивами и с памятью компьютера ни низком (архитектурном) уровне без каких-либо ограничений со стороны транслятора. Это «свобода самовыражения» обеспечивается одной дополнительной операцией **адресной арифметики.** Но сначала определим свойства указателя в соответствии с расширенной интерпретацией.

Любой указатель в Си ссылается на неограниченную в обе стороны область памяти (массив), заполненную переменными указуемого типа с индексацией элементов относительно текущего положения указателя.

Такие свойства указателя обеспечиваются **адресной арифметикой,** которая базируется на нестандартной интерпретации операции **указатель+целое** и других, производных от нее операциях:

- любой указатель потенциально ссылается на неограниченную в обе стороны область памяти, заполненную переменными указуемого типа;
- переменные в области нумеруются от текущей указуемой переменной, которая получает относительный номер 0. Переменные в направлении возрастания адресов памяти нумеруются положительными значениями 1,2,3..., убывания - отрицательными - -1,-2..;
- результатом операции указатель+i является адрес i-ой переменной (значение указателя на i-ую переменную) в этой области относительно текущего положения указателя.




| | Смысл |
|---|---|
| *p | Значение указуемой переменной |
| p+i | Указатель на i-ю переменную после указуемой |
| p-i | Указатель на i-ю переменную перед указуемой |
| *(p+i) | Значение i-й переменной после указуемой |
| p[i] | Значение i-й переменной после указуемой |
| p++ | Переместить указатель на следующую переменную |
| p-- | Переместить указатель на предыдущую переменную |
| p+=i | Переместить указатель на i переменных вперед |
| p-=i | Переместить указатель на i переменных назад |
| *p++ | Получить значение указуемой переменной и переместить указатель к следующей |
| *(--p) | Переместить указатель к переменной, предшествующей указуемой, и получить ее значение |
| p+1 | Указатель на свободную память вслед за указуемой переменной |

В операциях адресной арифметики транслятором автоматически учитывается размер указуемых переменных, то есть **+i** понимается не как смещение на **i** байтов или слов, а как смещение на **i** указуемых переменных. Другая важная особенность: при перемещении указателя нумерация переменных в памяти остается относительной и всегда производится от текущей указуемой переменной.

**Указатели и массивы.** Нетрудно заметить, что указатель в Си имеет много общего с массивом. Наоборот, труднее сформулировать, чем они отличаются друг от друга. Действительно, разница лежит не в принципе работы с указуемыми переменными, а в способе назначения указателя и массива на ту память, с которой они работают. Образно говоря, указателю соответствует массив, «не привязанный» к конкретной памяти, а массиву соответствует указатель, постоянно назначенный на выделенную транслятором область памяти. Это положение вещей поддерживается еще одним правилом: имя массива во всех выражениях воспринимается как указатель на его начало, то есть имя массива **A** эквивалентно выражению **&A[0]** и имеет тип «указатель на тип данных элементов массива». Таким образом, различие между указателем и массивом аналогично различию между переменной и константой: указатель - это ссылочная переменная, а имя массива - ссылочная константа, привязанная к конкретному адресу памяти.

Если МАССИВ=ПАМЯТЬ+УКАЗАТЕЛЬ (начальный адрес), то УКАЗАТЕЛЬ=МАССИВ-ПАМЯТЬ, т.е. указатель это «массив без памяти», «свободно перемещающийся по памяти» массив.

| Массив | Указатель | Различия и сходства |
|---|---|---|
| int A[20] | int *p | |
| A | p | Оба интерпретируются как указатели и оба имеют тип int* |
| --- | p=&A[3] | Указатель требует настройки «на память» |
| A[i]<br>&A[i]<br>A+i<br>*(A+i) | p[i]<br>&p[i]<br>p+i<br>*(p+i) | Работа с областью памяти как с обычным массивом, так и через указатель полностью идентична вплоть до синтаксиса |
| ---- | p++<br>*p++<br>p+=i | Указатель может перемещаться по памяти относительно своего текущего положения |

**Указатели и многомерные массивы.** Двумерный массив реализован как «массив массивов» - одномерный массив с количеством элементов, соответствующих первому индексу, причем каждый элемент представляет собой массив элементов базового типа с количеством, соответствующим второму индексу. Например, **char A[20][80]** определяет массив из 20 массивов по 80 символов в каждом и никак иначе.

Идентификатор массива без скобок интерпретируется как адрес нулевого элемента нулевой строки, или указатель на базовый тип данных. В нашем примере идентификатору **A** будет соответствовать выражение **&A[0][0]** с типом **char*.**

Имя двумерного массива с единственным индексом интерпретируется как начальный адрес соответствующего внутреннего одномерного массива. **A[i]** понимается как **&A[i][0],** то есть начальный адрес **i-го** массива символов.

От такого многообразия возможностей работы с указателями нетрудно прийти в замешательство: как вообще с ними работать, кто за что отвечает? Действительно, при работе с указателями легко выйти «за рамки дозволенного», т.е. определенных самим же программистом структур данных. Поэтому попробуем еще раз обсудить принципиальные моменты адресной арифметики.

**Границы памяти, адресуемой указателем.** Если любой указатель ссылается на неограниченную область памяти, то возникают резонные вопросы: где границы этой памяти, кто и как их определяет, кто и как контролирует нарушение этих границ указателем. Ответ на него неутешителен для начинающего программиста: транслятор принципиально исключает такой контроль как в процессе трансляции программы, так и в процессе ее выполнения. Он не помещает в генерируемый программный код каких-либо дополнительных команд, которые могли бы это сделать. И дело здесь прежде всего в самой концепции языка Си: не включать в программный код ничего, не предусмотренного самой программой, и не вносить ограничений в возможности работы с данными. Следовательно, ответственность ложится целиком на работающую программу (точнее, на программиста, который ее написал).

**На что ссылается указатель?** Синтаксис языка в операциях с указателями не позволяет различить в конкретной точке программы, что подразумевается под этим указателем - указатель на отдельную переменную, массив (начало, середину конец...), какова размерность массива и т.д.. Все эти вопросы целиком находятся в ведении работающей программы. Все же даже поверхностный взгляд на программу позволяет сказать, с чем же работает указатель – с отдельной переменной или массивом.

· наличие операции инкремента или индексации говорит о работе указателя с памятью (массивом);

· использование исключительно операции косвенного обращения по указателю свидетельствует о работе с отдельной переменной.

**Типичные ошибки при работе с указателями.** Основная ошибка, которая периодически возникает даже у опытных программистов – указатель ассоциируется с адресуемой им памятью. Память – это прежде всего ресурс, а указатель – ссылка на него. Здесь же отметим наиболее грубые ошибки:

· неинициализированный указатель. После определения указатель ссылается «в никуда», тем не менее программист работает через него с переменной или массивом, записывая данные по случайным адресам;

· несколько указателей, ссылающихся на общий массив – это все-таки один массив, а не несколько. Если программа работает с несколькими массивами, то они должны либо создаваться динамически, либо браться из двумерного массива;

· выход указателя за границы памяти. Например, конец строки отмечается символов '\0', начало же формально соответствует начальному положению указателя. Если в процессе работы со строкой требуется возвращение на ее начало, то начальный указатель необходимо запоминать, либо дополнительно отсчитывать символы.

Другие операции над указателями

В процессе определения указателей мы рассмотрели основные операции над ними:

· операция присваивания указателей одного типа. Назначение указателю адреса переменной **p=&a** есть одни из вариантов такой операции;

· операция косвенного обращения по указателю (разыменования указателя);

· операция адресной арифметики «указатель+целое» и все производные от нее.

Кроме того, имеется еще ряд операций, понимание которых не выходит за рамки уже имеющейся интерпретации указателя.

**Сравнение указателей на равенство**. Равенство указателей однозначно понимается как совпадение адресов, то есть назначение их на одну и ту же область памяти (переменную).

**Пустой указатель (NULL-указатель)**. Среди множества адресов выделяется такой, который не может быть использован в правильно работающей программе для размещения данных. Это значение адреса называется **NULL-указателем** или «пустым» указателем. Считается, что указатель с таким значением не является корректным (указывает «в никуда»). Обычно такое значение определяется в стандартной библиотеке ввода-вывода в виде **#define NULL 0.**

Значение NULL может быть присвоено любому указателю. Если указатель по логике работы программы может иметь такое значение, то перед косвенным обращением по нему его нужно проверять на достоверность:

```
int        *p,a;
if (…) p=NULL; else p=&a; ...
if (p !=NULL) *p = 5;    ...
```

**Сравнение указателей на «больше-меньше»**: при сравнении указателей производится сравнение соответствующих адресов как беззнаковых переменных. Если оба указателя ссылаются на элементы одного и того же массива, тогда соотношение «больше-меньше» следует понимать как «ближе-дальше» к началу массива:

```
//--- Симметричная перестановка символов строки
 void F(char *p){
 for (char *q=p; *q!=0; q++);                  // Указатель q до конца строки
 for (q--; q>p; p++, q--)                      // Пока p левее q
          { char c; c=*p; *p=*q; *q=c; }     // 3 стакана над переменными под указателями
 }
```

**Разность значений указателей**. В случае, когда указатели ссылаются на один и тот же массив, их разность понимается как «расстояние между ними», выраженную в количестве указуемых переменных.

**Преобразование типа указателя.** Отдельная операция преобразования, связанная с изменением типа указуемых элементов при сохранении значения указателя (адреса), используется при работе с память на низком (архитектурном) уровне (см. **9.2**). Отдельный разговор о преобразовании типов указателей при наследовании (см. **11.3, 11.4**). Сюда же относится **преобразование вида «целое-указатель».**

**Указатель типа void*.** Если фрагмент программы «не должен знать» или не имеет достаточной информации о структуре данных в адресуемой области памяти, если указатель во время работы программы ссылается на данные различных типов, то используется указатель на неопределенный (пустой) тип **void**. Указатель понимается как адрес памяти как таковой, с неопределенной организацией и неизвестной размерностью указуемой переменной. Его можно присваивать, передавать в качестве параметра и результата функции, менять тип указателя, но операции косвенного обращения и адресной арифметики с ним недопустимы.

```
extern int fread(void *, int, int, FILE *);
int        A[20];
fread(A, sizeof(int), 20, fd);
```

Функция **fread** выполняет чтение из двоичного файла **n** записей длиной по **m** байтов, при этом структура записи для функции неизвестна. Поэтому начальный адрес области памяти передается формальным параметром типа **void*.** При подстановке фактического параметра **A** типа **int*** производится неявное преобразование его к типу **void*.**

```
extern void *malloc(int);
int *p = (int*)malloc(sizeof(int)*20);          // Явное преобразование void* к int*
```

Функция **malloc** возвращает адрес зарезервированной области динамической памяти в виде указателя **void*.** Это означает, что функцией выделяется память как таковая, безотносительно к размещаемым в ней переменным. Вызывающая функция явно преобразует тип указателя **void*** в требуемый тип **int*** для работы с этой областью как с массивом целых переменных.

*Вывод*: преобразование указателя **void*** к любому другому типу указателя соответствует «смене точки зрения» программы на адресуемую память от «данные вообще» к «конкретные данные» и наоборот (см. **9.2**) и должно быть сделано явно. Преобразование указателя к типу **void*** не требует явного подтверждения.

Указатель как формальный параметр и результат функции

В Си при передаче параметров в функцию по умолчанию используется **передача по значению (by value)**. Формальные параметры представляют собой аналог собственных локальных переменных функции, которым в момент вызова присваиваются значения фактических параметров. Формальные параметры, представляя собой копии, могут как угодно изменяться - это не затрагивает соответствующих фактических параметров.

Если же фактический параметр должен быть изменен, то формальный параметр можно определить как явный указатель. Тогда фактический параметр должны быть явно передан в виде указателя на ту переменную (с использованием операции **&**).

```
void inc(int *p)
{ (*pi)++; }                    // аналог вызова:  pi = &a
void main()
{ int       a;
inc(&a); }                      // *(pi)++ эквивалентно a++
```

В Си имеется единственное исключение: формальный параметр - массив передается в виде неявного указателя на его начало, то есть **по ссылке**. С помощью адресной арифметики это также можно сделать явно с использованием указателя на его начало.

```
int       sum(int A[],int n)          // Исходная программа
{ int      s,i;
for (i=s=0; i<n; i++) s+= A[i];
return s;}
int       sum(int *p, int n)          // Эквивалент с указателем
{ int      s,i;
for (i=s=0; i<n; i++) s+= p[i];
return s; }

int       x,B[10]={1,4,3,6,3,7,2,5,23,6};
void main()
{ x = sum(B,10); }          // аналог вызова: p = B, n = 10
```

В вызове фигурирует идентификатор массива, который интерпретируется как указатель на начало. Поэтому типы формального и фактического параметров совпадают. Совпадают также оба варианта функций вплоть до генерируемого кода.

**Указатель - результат функции.** Функция в качестве результата может возвращать указатель. Формальная схема функции обязательно включает в себя:

· определение типа результата в заголовке функции как указателя. Это обеспечивается добавлением пресловутой ***** перед именем функции - **int *F(...**;

· оператор **return** возвращает объект (переменную или выражение), являющееся по своей природе (типу данных) указателем. Для этого можно использовать локальную переменную - указатель.

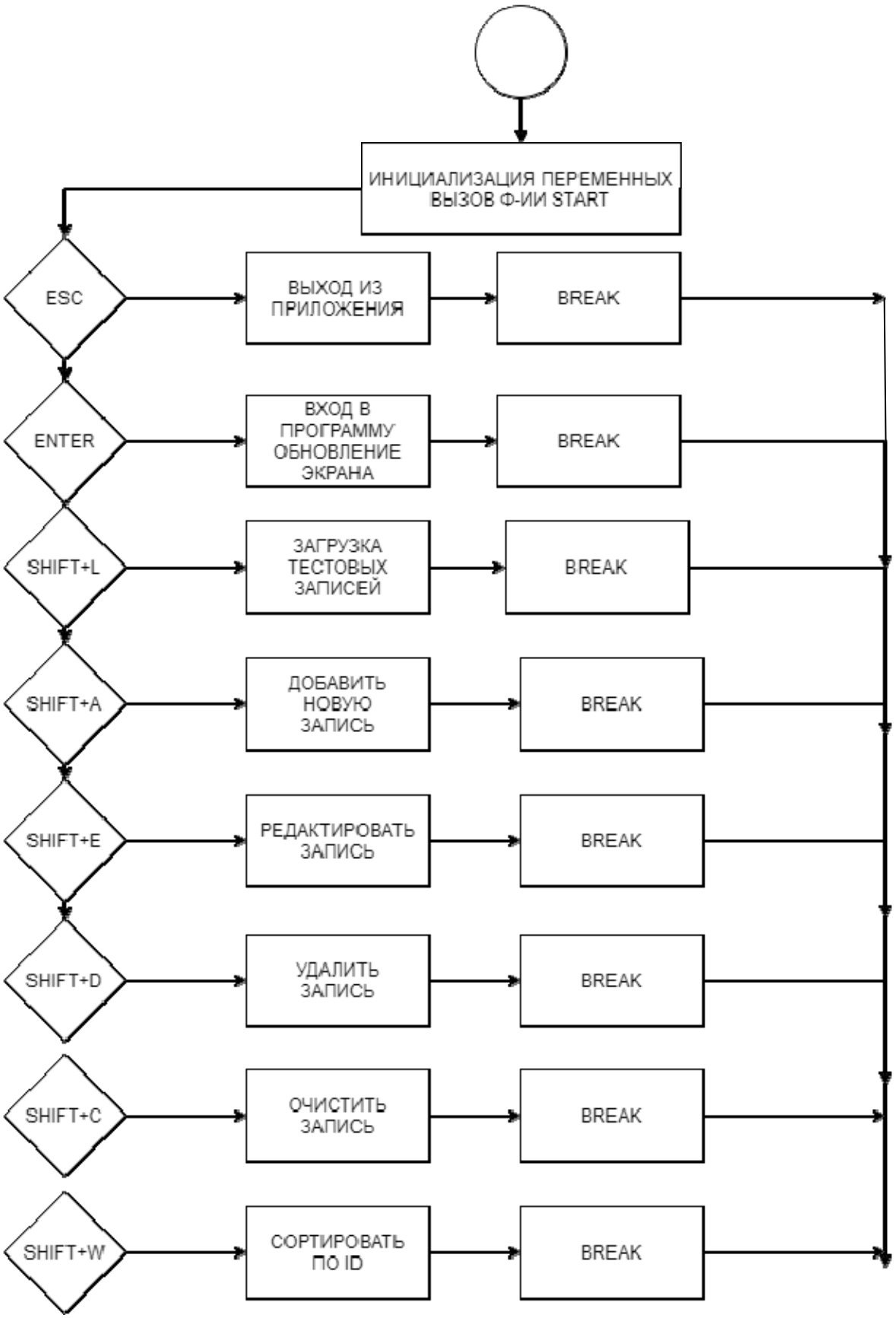
Содержательная сторона проблемы состоит в том, что функция либо **выбирает** один из известных ей объектов (переменных), либо **создает** их в процессе своего выполнения (динамические переменные), возвращая в том и другом случае указатель на нее. Для выбора у нее не так уж много возможностей. Это могут быть:

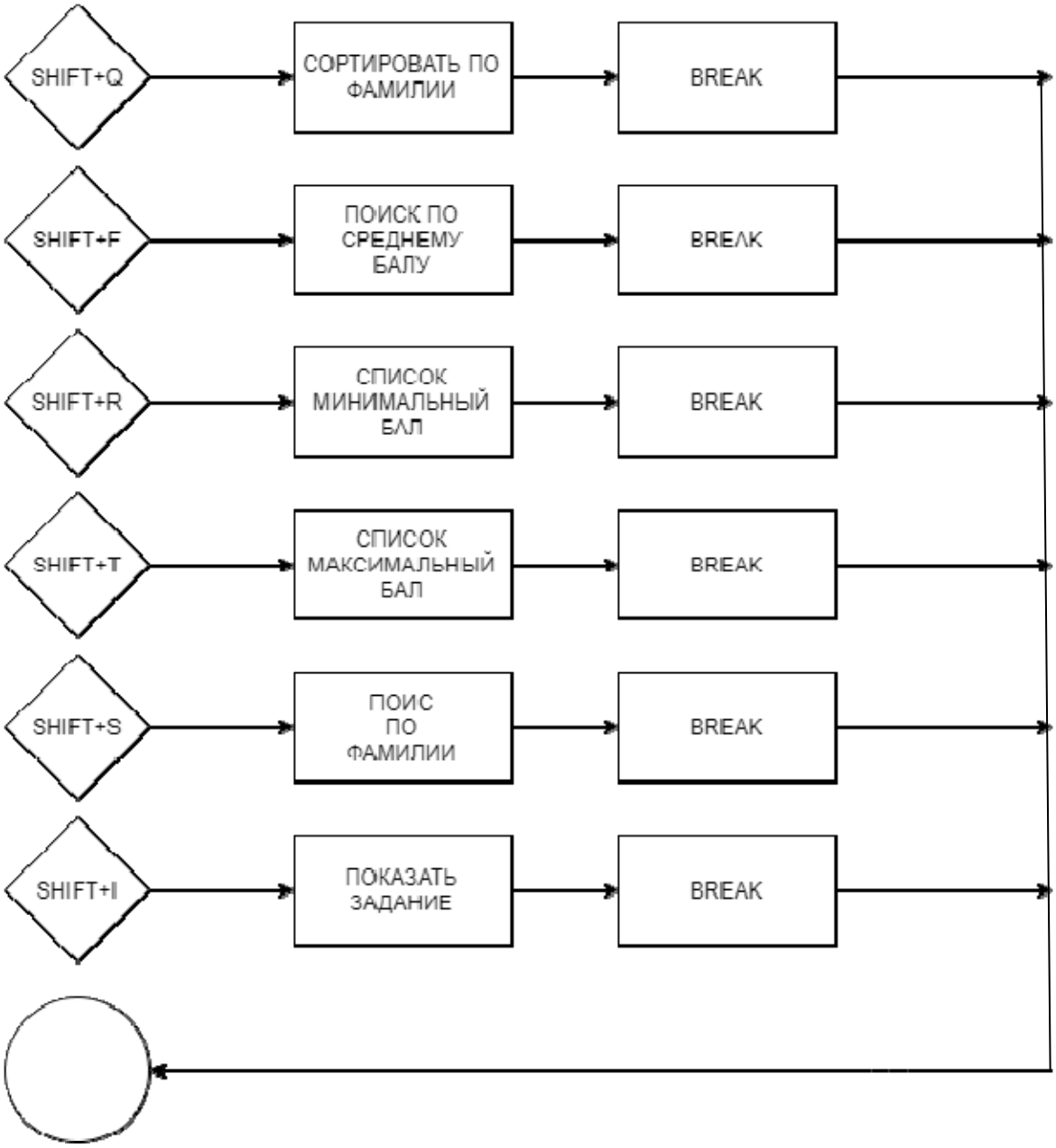
· глобальные переменные программы;

· формальные параметры, если они являются массивами, указателями или ссылками, то есть «за ними стоят» другие переменные.

Функция не может возвратить указатель на локальную переменную или формальный параметр-значение, поскольку они разрушаются при выходе из функции. Это приводит к ошибке времени выполнения, не обнаруживаемой транслятором.

## 6. Алгоритм решения задачи




SHIFT+Q
СОРТИРОВАТЬ ПО ФАМИЛИИ
BREAK
SHIFT+F
ПОИСК ПО СРЕДНЕМУ БАЛУ
BREAK
SHIFT+R
СПИСОК МИНИМАЛЬНЫЙ БАЛ
BREAK
SHIFT+T
СПИСОК МАКСИМАЛЬНЫЙ БАЛ
BREAK
SHIFT+S
ПОИС ПО ФАМИЛИИ
BREAK
SHIFT+I
ПОКАЗАТЬ ЗАДАНИЕ
BREAK

## 7. Описание программной реализации

### 1) Используемые переменные

int last_id = 0; id записи
int* _last_id = &last_id; - указатель на id записи
int count_record = 0; - количество  записей
int* _count_record = &count_record; - указатель на количество записи
int count_free_items = 0; - количество свободных записей
int* _count_free_items = &count_free_items; указатель на количество записей

### 2) Используемые функции

**Функция void info()**
*Аргументы функции*:

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: показывает информацию о задании.

Функция **void start** (student* items, int* _count_record, int* _last_id, int* _count_free_items)
*Аргументы функции*:
student* items, – массив структурированных. перменных.
int* _count_record – количество записей.
int* _last_id – номер id.

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: вызывает необходимые функции для обработки данных.

**Функция void print_menu_header()**
*Аргументы функции*:

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: печатает заголовок меню.

**Функция print_table_header()**
*Аргументы функции*:

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: печатает заголовок таблицы вывода данных.

**Функция void clear_screen()**
*Аргументы функции*:

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: выполняет очистку экрана.

**Функция void print_one_record(student* item)**
*Аргументы функции*:
**student* item – указатель на запись**
*Возвращаемый результат*: ничего не возвращает.

*Принцип работы*: печатает одну строку данных.

**Функция void print_one_record(student* item)**
*Аргументы функции*:
**student* item – указатель на запись**
*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: печатает одну строку данных.

**Функция void print_footer(int* _count_record, int* _count_free_items, student* items)**
*Аргументы функции*:
**int* _count_record – количество записей**
**int* _count_free_items – количество свободных записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: печатает одну строку данных.

**Функция print_line()**
*Аргументы функции*:

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: печатает разделительную линию.

**Функция int count_free_items(student* items, const int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: количество записей.
*Принцип работы*: считает количество записей.

**Функция void print_all_items(student* items, const int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: выводит все записи из массива.

**Функция void print_items_min(student* items, const int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: выводит записи с минимальным средним балом.

**Функция void print_items_max(student* items, const int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: выводит записи с максимальным средним балом.

**Функция void print_items_range_average(student* items, const int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: выводит записи с заданным средним балом.

**Функция double input_average_marks()**
*Аргументы функции*:

*Возвращаемый результат*: введённое значение.
*Принцип работы*: запрашивает ввод данных.

**Функция void print_not_found_error(int _item_id)**
*Аргументы функции*:

*Возвращаемый результат*: введённое значение.
*Принцип работы*: выводит сообщение об ошибке поиска.

**Функция void sort_by_surname(student* items, const int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: сортировка по фамилии.

**Функция void sort_by_id(student* items, const int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: ничего не возвращает.
*Принцип работы*: сортировка по id

**Функция student* delete_item(student* items, int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: указатель на массив.
*Принцип работы*: удаляет запись

**Функция student* clear_item(student* items, int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: указатель на массив.
*Принцип работы*: очищает запись.

**Функция int search_index_free_item(student* items, int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**

*Возвращаемый результат*: индекс записи в массиве.
*Принцип работы*: поиск свободной переменной для записи.

**Функция int search_index_by_id(student* items, const int* _count_record, int item_id)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**
**int item_id – id записи**
*Возвращаемый результат*: индекс записи в массиве.
*Принцип работы*: поиск идекса записи по id

**Функция student* search_by_id(student* items, int item_id, const int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**
**int item_id – id записи**
*Возвращаемый результат*: указатель на найденную запись.
*Принцип работы*: поиск  записи по id

. **Функция void search_by_name(student* items, const int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**
*Возвращаемый результат*: указатель на найденную запись.
*Принцип работы*: поиск  записи по фамилии

. **Функция student* edit_item(student* items, int* _count_record)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**
*Возвращаемый результат*: указатель на начало массива.
*Принцип работы*: редактирование записи

. **Функция student data_add(int* _last_id)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**
*Возвращаемый результат*: структурированная переменная.
*Принцип работы*: создает одну запись

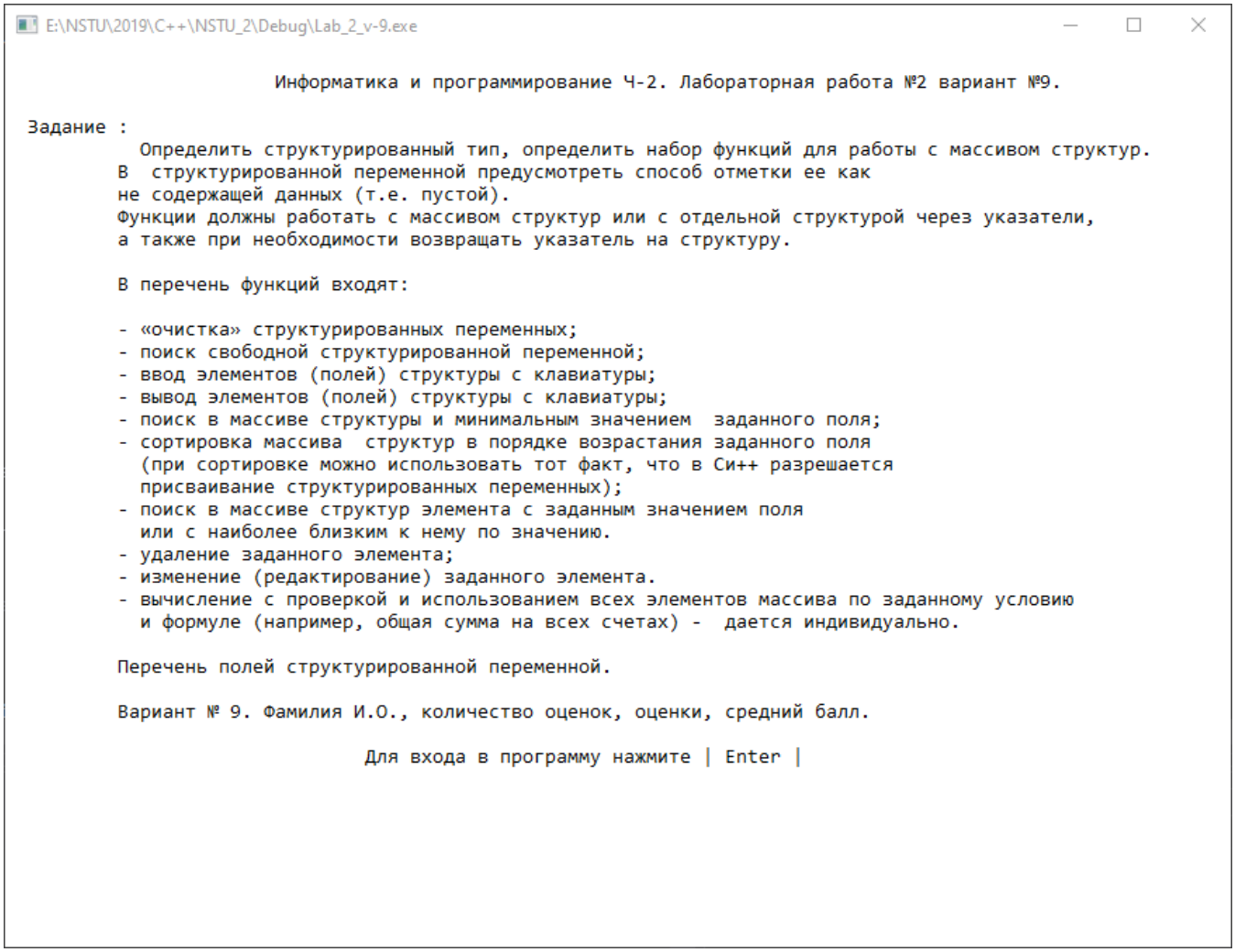
**Функция student* data_add_to_array(student* items, int* _count_record, int* _last_id, const int* _count_free_items)**
*Аргументы функции*:
**int* _count_record – количество записей**
**student* items – указатель на массив записей**
**int* _last_id – последний id**
**const int* _count_free_items – количество свободных записей**
*Возвращаемый результат*: структурированная переменная.
*Принцип работы*: создает одну запись

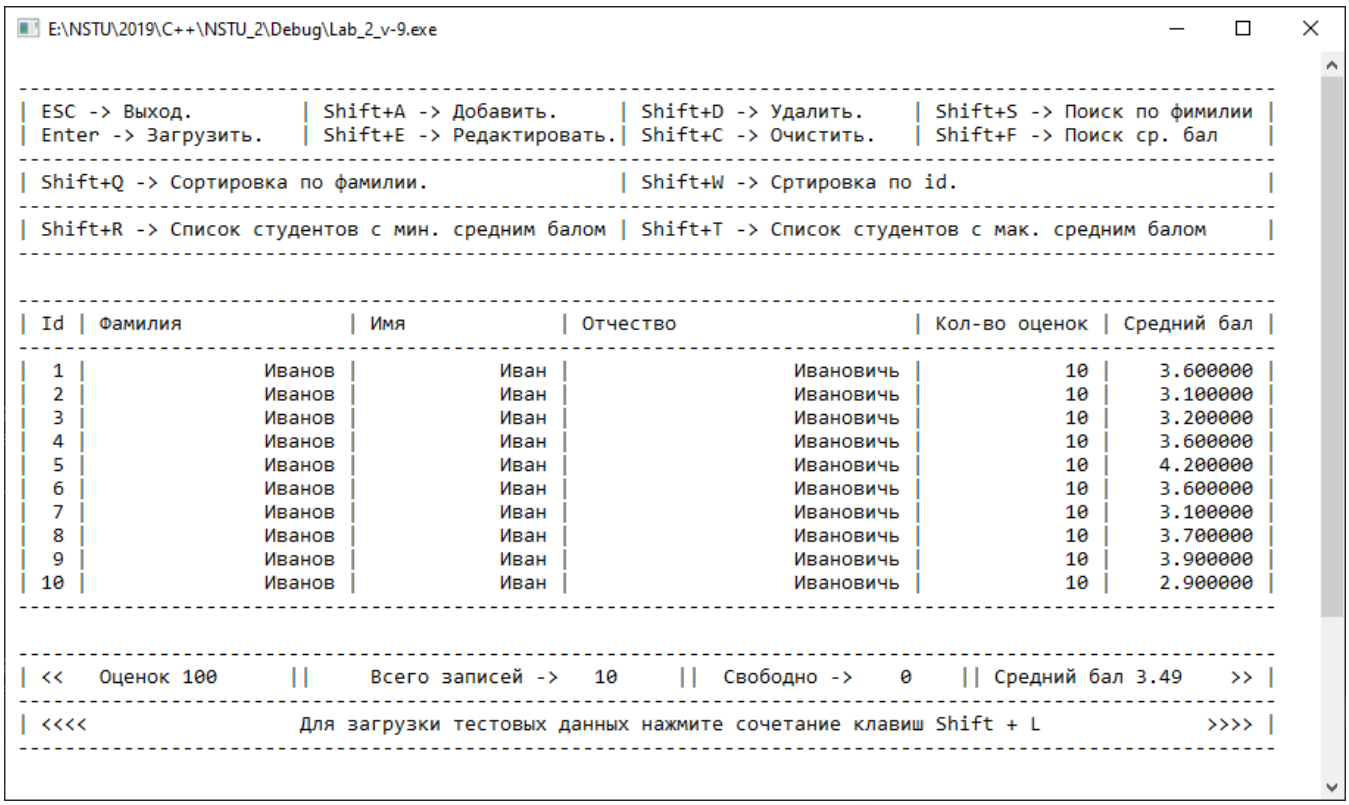
**Функция student add_marks(student item)**
*Аргументы функции*:
**student* items – указатель на массив записей**
*Возвращаемый результат*: структурированная переменная.
*Принцип работы*: добавляет оценки в массив

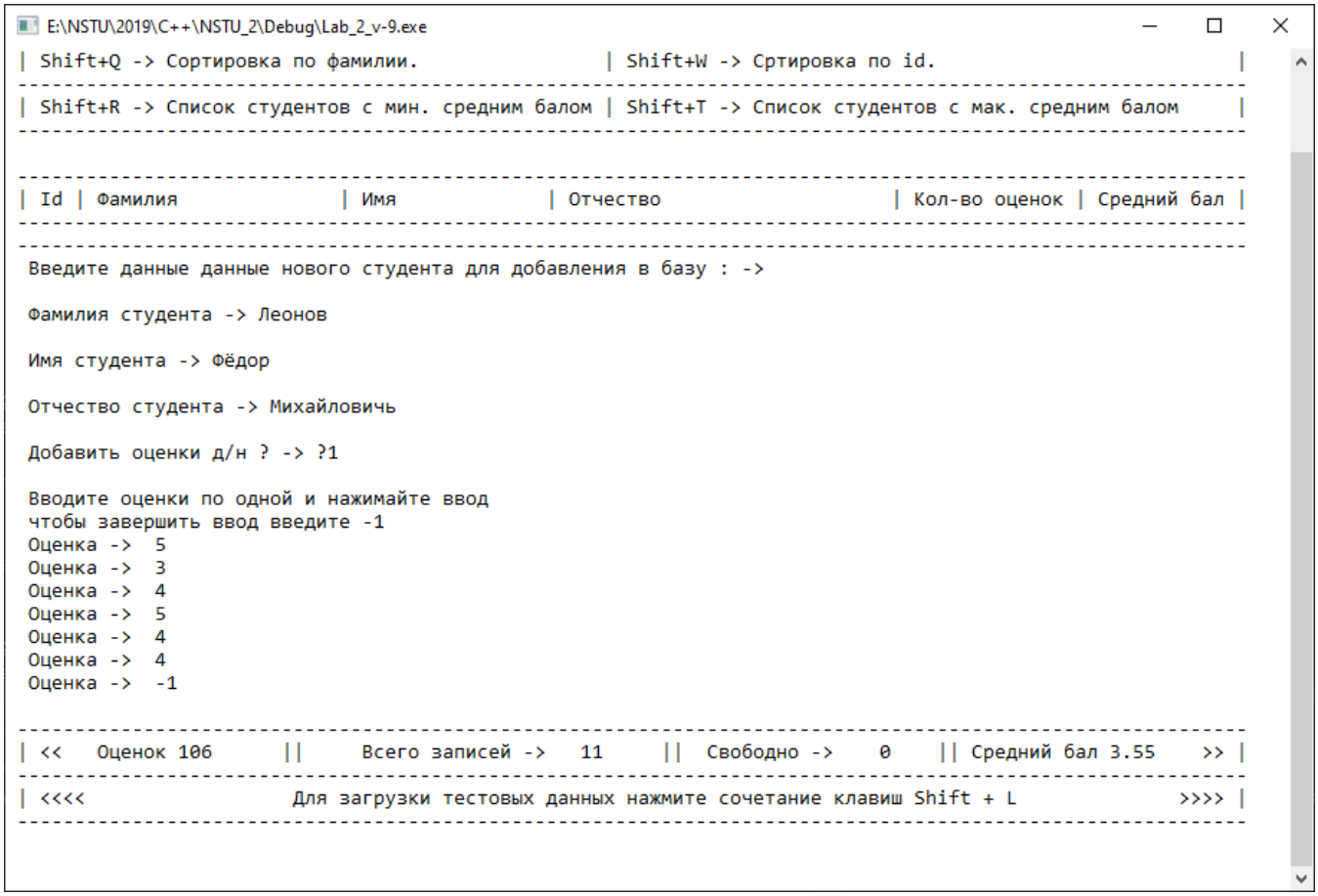
**Функция int input_item_id()**
*Аргументы функции*:

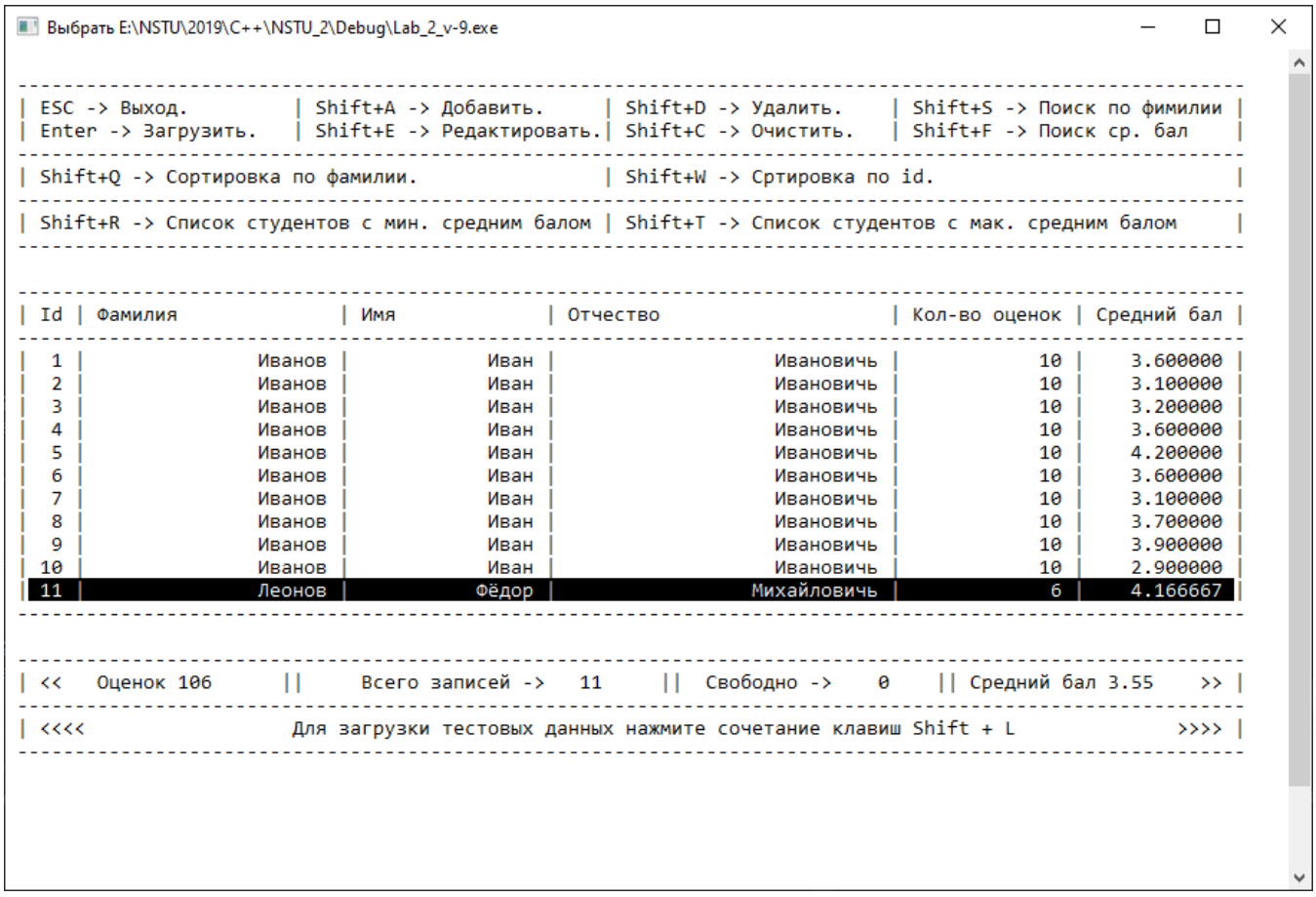
*Возвращаемый результат*: id для поиска.
*Принцип работы*: запращивает ввод от пользователя

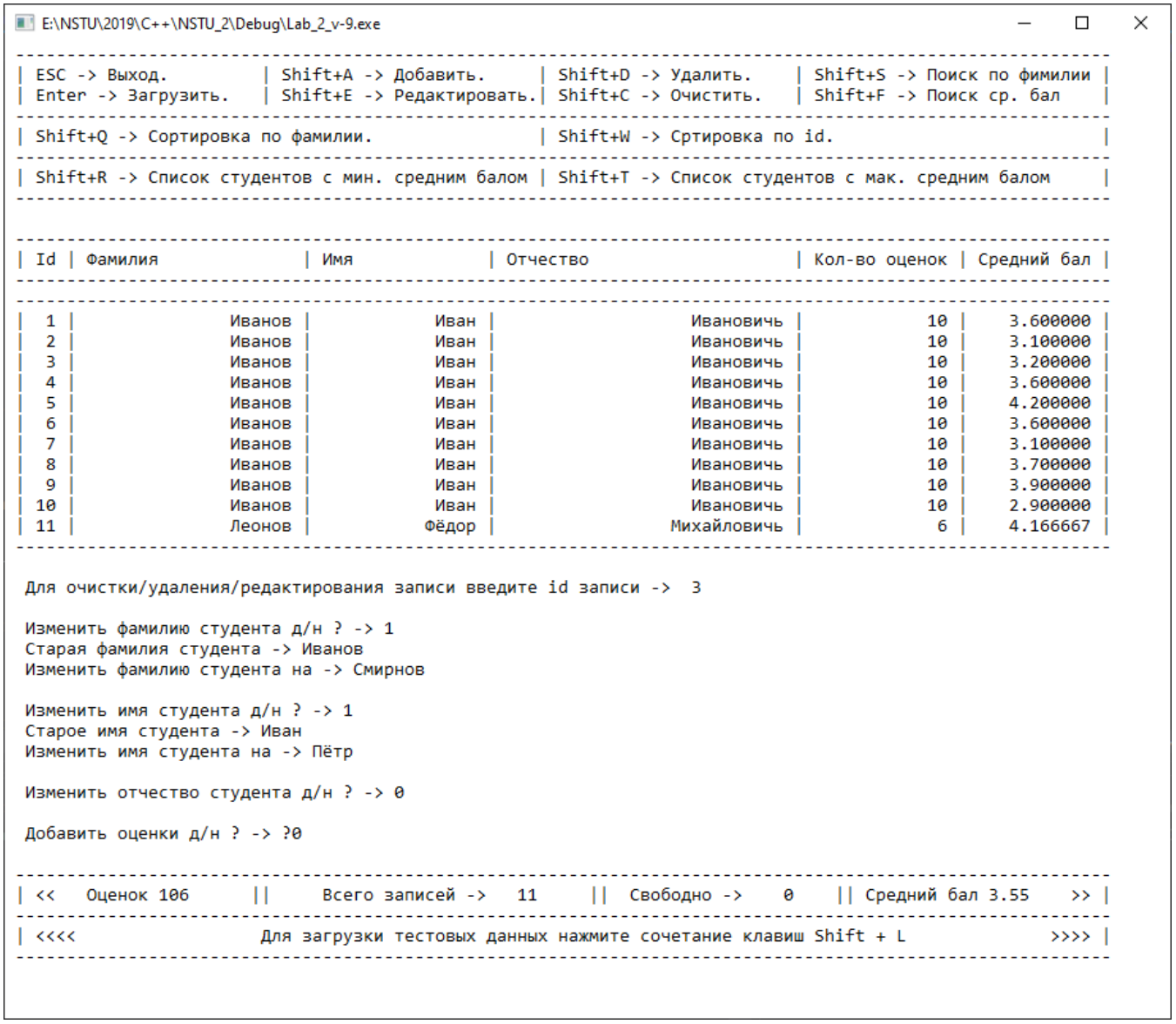
## 8. Пример работы программы

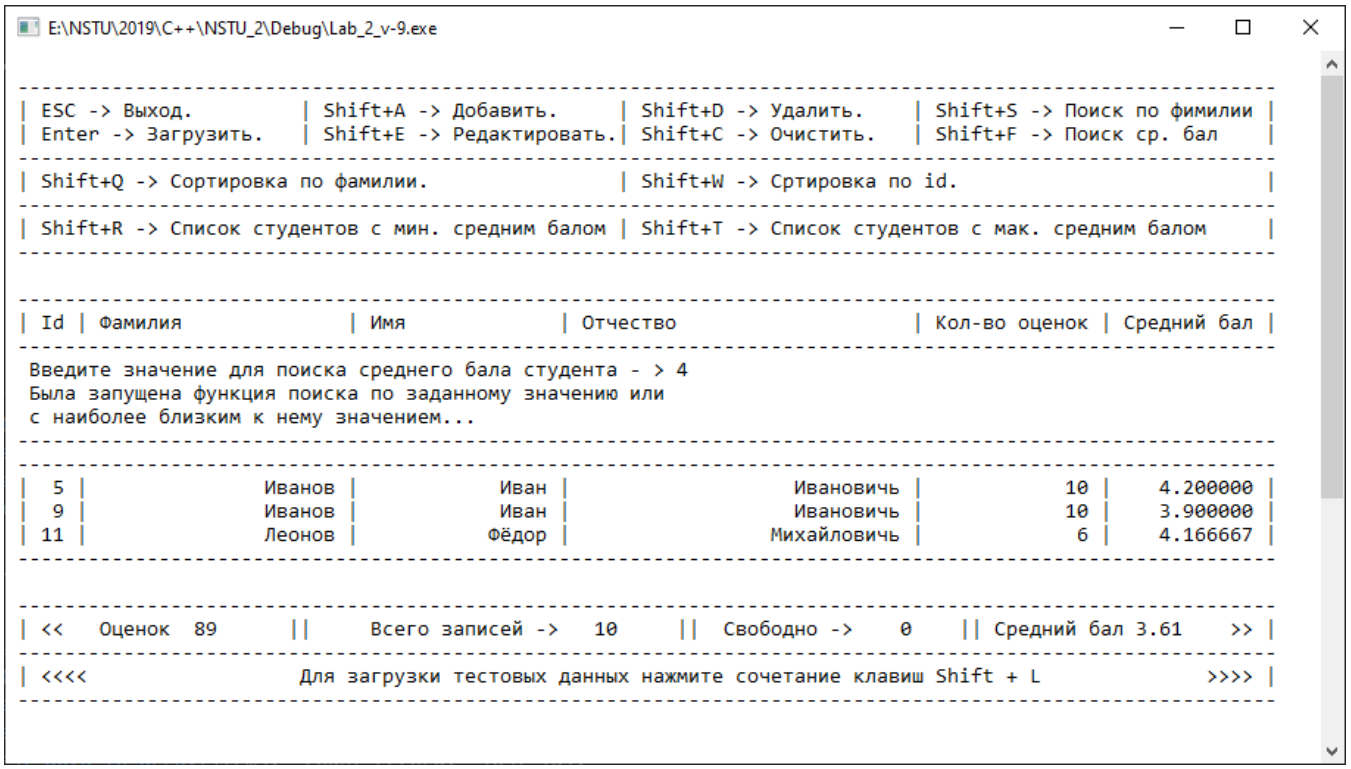



E:\NSTU\2019\C++\NSTU_2\Debug\Lab_2_v-9.exe
| Shift+Q -> Сортировка по фамилии. | Shift+W -> Сртировка по id. |
| Shift+R -> Список студентов с мин. средним балом | Shift+T -> Список студентов с мак. средним балом |
| Id | Фамилия | Имя | Отчество | Кол-во оценок | Средний бал |
Введите данные данные нового студента для добавления в базу : ->
Фамилия студента -> Леонов
Имя студента -> Фёдор
Отчество студента -> Михайловичь
Добавить оценки д/н ? -> ?1
Вводите оценки по одной и нажимайте ввод
чтобы завершить ввод введите -1
Оценка -> 5
Оценка -> 3
Оценка -> 4
Оценка -> 5
Оценка -> 4
Оценка -> 4
Оценка -> -1
| << Оценок 106 || Всего записей -> 11 || Свободно -> 0 || Средний бал 3.55 >> |
| <<<< Для загрузки тестовых данных нажмите сочетание клавиш Shift + L >>>> |


Выбрать E:\NSTU\2019\C++\NSTU_2\Debug\Lab_2_v-9.exe
| ESC -> Выход. | Shift+A -> Добавить. | Shift+D -> Удалить. | Shift+S -> Поиск по фимилии |
| Enter -> Загрузить. | Shift+E -> Редактировать. | Shift+C -> Очистить. | Shift+F -> Поиск ср. бал |
| Shift+Q -> Сортировка по фамилии. | Shift+W -> Сртировка по id. |
| Shift+R -> Список студентов с мин. средним балом | Shift+T -> Список студентов с мак. средним балом |
| Id | Фамилия | Имя | Отчество | Кол-во оценок | Средний бал |
| 1 | Иванов | Иван | Ивановичь | 10 | 3.600000 |
| 2 | Иванов | Иван | Ивановичь | 10 | 3.100000 |
| 3 | Иванов | Иван | Ивановичь | 10 | 3.200000 |
| 4 | Иванов | Иван | Ивановичь | 10 | 3.600000 |
| 5 | Иванов | Иван | Ивановичь | 10 | 4.200000 |
| 6 | Иванов | Иван | Ивановичь | 10 | 3.600000 |
| 7 | Иванов | Иван | Ивановичь | 10 | 3.100000 |
| 8 | Иванов | Иван | Ивановичь | 10 | 3.700000 |
| 9 | Иванов | Иван | Ивановичь | 10 | 3.900000 |
| 10 | Иванов | Иван | Ивановичь | 10 | 2.900000 |
| 11 | Леонов | Фёдор | Михайловичь | 6 | 4.166667 |
| << Оценок 106 || Всего записей -> 11 || Свободно -> 0 || Средний бал 3.55 >> |
| <<<< Для загрузки тестовых данных нажмите сочетание клавиш Shift + L >>>> |


E:\NSTU\2019\C++\NSTU_2\Debug\Lab_2_v-9.exe
| ESC -> Выход. | Shift+A -> Добавить. | Shift+D -> Удалить. | Shift+S -> Поиск по фимилии |
| Enter -> Загрузить. | Shift+E -> Редактировать. | Shift+C -> Очистить. | Shift+F -> Поиск ср. бал |
| Shift+Q -> Сортировка по фамилии. | Shift+W -> Сртировка по id. |
| Shift+R -> Список студентов с мин. средним балом | Shift+T -> Список студентов с мак. средним балом |
| Id | Фамилия | Имя | Отчество | Кол-во оценок | Средний бал |
| 1 | Иванов | Иван | Ивановичь | 10 | 3.600000 |
| 2 | Иванов | Иван | Ивановичь | 10 | 3.100000 |
| 3 | Иванов | Иван | Ивановичь | 10 | 3.200000 |
| 4 | Иванов | Иван | Ивановичь | 10 | 3.600000 |
| 5 | Иванов | Иван | Ивановичь | 10 | 4.200000 |
| 6 | Иванов | Иван | Ивановичь | 10 | 3.600000 |
| 7 | Иванов | Иван | Ивановичь | 10 | 3.100000 |
| 8 | Иванов | Иван | Ивановичь | 10 | 3.700000 |
| 9 | Иванов | Иван | Ивановичь | 10 | 3.900000 |
| 10 | Иванов | Иван | Ивановичь | 10 | 2.900000 |
| 11 | Леонов | Фёдор | Михайловичь | 6 | 4.166667 |
Для очистки/удаления/редактирования записи введите id записи -> 3
Изменить фамилию студента д/н ? -> 1
Старая фамилия студента -> Иванов
Изменить фамилию студента на -> Смирнов
Изменить имя студента д/н ? -> 1
Старое имя студента -> Иван
Изменить имя студента на -> Пётр
Изменить отчество студента д/н ? -> 0
Добавить оценки д/н ? -> ?0
| << Оценок 106 || Всего записей -> 11 || Свободно -> 0 || Средний бал 3.55 >> |
| <<<< Для загрузки тестовых данных нажмите сочетание клавиш Shift + L >>>> |


E:\NSTU\2019\C++\NSTU_2\Debug\Lab_2_v-9.exe
| ESC -> Выход. | Shift+A -> Добавить. | Shift+D -> Удалить. | Shift+S -> Поиск по фимилии |
| Enter -> Загрузить. | Shift+E -> Редактировать. | Shift+C -> Очистить. | Shift+F -> Поиск ср. бал |
| Shift+Q -> Сортировка по фамилии. | Shift+W -> Сртировка по id. |
| Shift+R -> Список студентов с мин. средним балом | Shift+T -> Список студентов с мак. средним балом |
| Id | Фамилия | Имя | Отчество | Кол-во оценок | Средний бал |
Введите значение для поиска среднего бала студента - > 4
Была запущена функция поиска по заданному значению или
с наиболее близким к нему значением...
| 5 | Иванов | Иван | Ивановичь | 10 | 4.200000 |
| 9 | Иванов | Иван | Ивановичь | 10 | 3.900000 |
| 11 | Леонов | Фёдор | Михайловичь | 6 | 4.166667 |
| << Оценок 89 || Всего записей -> 10 || Свободно -> 0 || Средний бал 3.61 >> |
| <<<< Для загрузки тестовых данных нажмите сочетание клавиш Shift + L >>>> |

## 9. Выводы

В ходе выполнения лабораторной работы были изучены строки и указатели и применены в решении задачи.

# Приложение. Текст программы

### Файл header.h

```
#ifndef FIRST_HEADER_H
#define FIRST_HEADER_H

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <stdbool.h>
#include <time.h>
#include<string.h>
#include <conio.h>

#define ENTER 13
#define ESC 27
#define SHIFT_A 65
#define SHIFT_E 69
#define SHIFT_D 68
#define SHIFT_C 67
#define SHIFT_S 83
#define SHIFT_F 70
#define SHIFT_I 73
#define SHIFT_L 76
#define SHIFT_Q 81
#define SHIFT_W 87
#define SHIFT_R 82
#define SHIFT_T 84

#define MARK_COUNT 100

typedef struct {
    int id;
    int is_free;
    char surname[30];
    char name[30];
    char second_name[35];
    int marks_count;
    int marks[MARK_COUNT];
    double marks_average;
} student;

student* students;
// menu
void info();
void clear_screen();
void print_menu_header();
void print_table_header();
void print_one_record(student* item);
void print_footer(int* _count_record, int* _count_free_items, student* items);
void print_line();
void start(student* items, int* _count_record, int* _last_id,
    int* _count_free_items);
int input_item_id();
double input_average_marks();
void print_all_items(student* items, const int* _count_record);
void print_items_min(student* items, const int* _count_record);
void print_items_max(student* items, const int* _count_record);
```

```
void print_items_range_average(student* items, const int* _count_record);
void print_not_found_error(int _item_id);
int count_free_items(student*, const int*);
double all_items_average_marks(student* items, const int* _count_record);
int all_items_count_marks(student* items, const int* _count_record);
int mygetch();
//
// add_item
student* data_add_to_array(student* items, int* _count_record, int* _last_id,
    const int* _count_free_items);
student data_add(int* _last_id);
student add_marks(student item_for_add_mark);
double average_mark(const int* items);
//
// edit_item
student* edit_item(student* items, int* _count_record);
//
// delete_item
student* delete_item(student* items, int* _count_record);
//
// clear_item
student* clear_item(student* items, int* _count_record);
//
// search_items
student* search_by_id(student*, int, const int*);
int search_index_by_id(student*, const int*, int);
int search_index_free_item(student* items, int* _count_record);
double search_min_average_marks(student* items, const int* _count_record);
double search_max_average_marks(student* items, const int* _count_record);
void search_by_name(student* items, const int* _count_record);
//
//sort_items
void sort_by_surname(student* items, const int* _count_record);
void sort_by_id(student* items, const int* _count_record);
//
// add_test_items
student add_test_item(int* _last_id);
student* add_test_data_to_array(student* items, int* _count_record,
    int* _last_id);
int take_random_mark(int left_range, int right_range);
int get_random(int n);
int fill_array_random_mark(int* items);
void fill_array(int*);
int count_marks_in_array(const int*);
//
#endif //FIRST_HEADER_H
```

**Файл main.c**

```
int main()
{
    srand(time(0));

    system("chcp 1251");
    system("mode 109,40");
    // последний сохраненный идентификатор
    int last_id = 0;
    int* _last_id = &last_id;
    // количество записей
    int count_record = 0;
    int* _count_record = &count_record;
    // количество записей
    int count_free_items = 0;
    int* _count_free_items = &count_free_items;
```

```
    //student* item;
    start(students, _count_record, _last_id, _count_free_items);
    //key_search();
    return 0;
}
```

## Файл sort_items.c

```
#include "header.h"

void sort_by_surname(student* items, const int* _count_record)
{
    for (int i = 0; i < (*_count_record); i++)
    {
        for (int j = (*_count_record - 1); j > i; j--)
        {
            if (items[j - 1].surname[0] > items[j].surname[0])
            {
                student tmp = items[j - 1];
                items[j - 1] = items[j];
                items[j] = tmp;
            }
        }
    }
}

void sort_by_id(student* items, const int* _count_record)
{
    for (int i = 0; i < (*_count_record); i++)
    {
        for (int j = (*_count_record - 1); j > i; j--)
        {
            if (items[j - 1].id > items[j].id)
            {
                student tmp = items[j - 1];
                items[j - 1] = items[j];
                items[j] = tmp;
            }
        }
    }
}
```

## Файл delete_items.c

```
#include "header.h"

student* delete_item(student* items, int* _count_record)
{
    int _item_id = input_item_id();
    int _item_index = search_index_by_id(items, _count_record, _item_id);
    if (_item_index != -1)
    {
        for (size_t i = _item_index; i < *_count_record; i++)
        {
            items[i] = items[i + 1];
        }
        (*_count_record)--;
    }
    else {
        print_not_found_error(_item_id);
    }
    return items;
}
```

## Файл clear_items.c

```
#include "header.h"

student* delete_item(student* items, int* _count_record)
{
    int _item_id = input_item_id();
    int _item_index = search_index_by_id(items, _count_record, _item_id);
    if (_item_index != -1)
    {
        for (size_t i = _item_index; i < *_count_record; i++)
        {
            items[i] = items[i + 1];
        }
        (*_count_record)--;
    }
    else {
        print_not_found_error(_item_id);
    }
    return items;
}
```

## Файл sarch_item.c

```
#include "header.h"

int search_index_free_item(student* items, int* _count_record)
{
    student* ptr;
    size_t i;
    for (ptr = items, i = 0; ptr < items + *_count_record; ptr++, i++)
    {
        int _is_free = ptr->is_free;
        if (_is_free == 1) {
            return i;
        }
    }
    return  -1;
}

int search_index_by_id(student* items, const int* _count_record, int item_id)
{
    student* ptr;
    size_t i;
    if (items != NULL) {
        for (ptr = items, i = 0; ptr < items + *_count_record; ptr++, i++)
        {
            int _id = ptr->id;
            if (_id == item_id) {
                return i;
            }
        }
    }
    return -1;
}

student* search_by_id(student* items, int item_id, const int* _count_record)
{
    if (items != NULL) {
        for (student* item = items; item < items + *_count_record; item++)
        {
```

```
            int _id = item->id;
            if (_id == item_id) {
                return item;
            }
        }
    }
    return NULL;
}

double search_min_average_marks(student* items, const int* _count_record)
{
    double _min = items->marks_average;
    double _tmp;
    if (items != NULL)
    {
        for (student* item = ++items; item < items + *_count_record; item++)
        {
            _tmp = item->marks_average;
            if (item->marks_count > 0)
            {
                _min = _tmp < _min ? _tmp : _min;
            }
        }
        return _min;
    }
    return -1.0;
}

double search_max_average_marks(student* items, const int* _count_record)
{
    double _max = items->marks_average;
    double _tmp;
    if (items != NULL)
    {
        for (student* item = ++items; item < items + *_count_record; item++)
        {
            _tmp = item->marks_average;
            if (item->marks_count > 0)
            {
                _max = _tmp > _max ? _tmp : _max;
            }
        }
        return _max;
    }
    return  -1.0;
}

void search_by_name(student* items, const int* _count_record)
{
    char input_name[30];
    char input_surname[30];
    int flag = 0;
    printf("  Введите фамилию студента для поиска - > ");
    scanf("%s", &input_surname);
    printf("  Введите имя студента для поиска - > ");
    scanf("%s", &input_name);

    if (items != 0) {
        for (student* item = items; item < items + *_count_record; item++)
        {
            int surname = strcmp(item->surname, input_surname);
            int name = strcmp(item->name, input_name);

            if (surname == 0 && name == 0)
            {
                print_one_record(item);
                flag = 1;
            }
        }
    }
    if (flag == 0) {
```

```
        printf("  Не найдено записей с введёным именем !!!\n");
    }
}
```

## Файл add_item.c

```
#include "header.h"

student data_add(int* _last_id)
{
    student item;
    int _id = *_last_id + 1;
    char _buffer[35];
    *_last_id = _id;

    printf("  Введите данные данные нового студента для добавления в базу : ->  \n");

    printf("\n  Фамилия студента -> ");
    scanf("%s", &_buffer);
    strcpy(item.surname, _buffer);
    printf("\n  Имя студента -> ");
    scanf("%s", &_buffer);
    strcpy(item.name, _buffer);
    printf("\n  Отчество студента -> ");
    scanf("%s", &_buffer);
    strcpy(item.second_name, _buffer);

    item.id = _id;
    item.is_free = 0;
    fill_array(item.marks);
    item.marks_count = 0;
    item.marks_average = 0;
    item = add_marks(item);

    return item;
}

student* data_add_to_array(student* items, int* _count_record,
    int* _last_id, const int* _count_free_items)
{
    //Поиск свободной переменной
    size_t index_free_item;
    if (items != NULL)
    {
        if (*_count_free_items != 0)
        {
            //поиск индекса свободной переменной
            index_free_item = search_index_free_item(items, _count_record);
            //запись данных по найденному индексу
            items[index_free_item] = data_add(_last_id);
            return items;
        }
        else {
            *_count_record = *_count_record + 1;
            items = (student*)realloc(items, *_count_record * sizeof(student));
            items[*_count_record - 1] = data_add(_last_id);
        }
    }
    else {
        *_count_record = *_count_record + 1;
        items = (student*)realloc(items, *_count_record * sizeof(student));
        items[*_count_record - 1] = data_add(_last_id);
    }
    return items;
}
```

```
student add_marks(student item)
{
    int curent_mark = 0;
    int mark_count = item.marks_count;
    int stop_flag = 0;
    int _yes = 1;
    int _no = 0;
    int _answer;
    printf("\n  Добавить оценки д/н ? -> ?");
    scanf("%d", &_answer);
    if (_answer == _yes) {
        printf("\n");
        printf("  Вводите оценки по одной и нажимайте ввод \n");
        printf("  чтобы завершить ввод введите -1 \n");

        while (stop_flag != 1) {
            printf("  Оценка ->  ");
            scanf("%d", &curent_mark);
            if (curent_mark != -1) {
                if (curent_mark < 2 || curent_mark > 5)
                {
                    curent_mark = 0;
                    mark_count = mark_count;
                }
                else {
                    mark_count = mark_count + 1;
                }
                item.marks[mark_count - 1] = curent_mark;
                item.marks_count = mark_count;
                item.marks_average = average_mark(item.marks);
            }
            else stop_flag = 1;
        }
    }
    return item;
}
//TODO Rewrite this function
double average_mark(const int* items)
{
    double average = 0;
    double sum = 0;
    int i = 0;
    while (items[i] != -1)
    {
        sum = sum + items[i];
        i++;
    }
    return i == 0 ? average : (average = sum / i);
}
//
int input_item_id()
{
    int _item_id = 0;
    printf("\n");
    printf("  Для очистки/удаления/редактирования записи введите id записи ->  ");
    scanf("%d", &_item_id);
    return _item_id;
}
```

## Файл edit_item.c

```
#include "header.h"

student* edit_item(student* items, int* _count_record)
{
    int stop_flag = 0;
```

```
    int mark_count;
    int curent_mark = 0;
    int _yes = 1;
    int _answer = -1;
    char buffer[35] = { 0 };
    int _item_id = input_item_id();
    int _item_index = search_index_by_id(items, _count_record, _item_id);
    // изменение персональных данных
    if (_item_index != -1)
    {
        printf("\n  Изменить фамилию студента д/н ? -> ");
        scanf("%d", &_answer);
        if (_answer == _yes)
        {
            printf("  Старая фамилия студента -> %s", items[_item_index].surname);
            printf("\n  Изменить фамилию студента на -> ");
            scanf("%s", buffer);
            strcpy(items[_item_index].surname, buffer);
        }
        printf("\n  Изменить имя студента д/н ? -> ");
        scanf("%d", &_answer);
        if (_answer == _yes)
        {
            printf("  Старое имя студента -> %s", items[_item_index].name);
            printf("\n  Изменить имя студента на -> ");
            scanf("%s", buffer);
            strcpy(items[_item_index].name, buffer);
        }
        printf("\n  Изменить отчество студента д/н ? -> ");
        scanf("%d", &_answer);
        if (_answer == _yes)
        {
            printf("  Старое отчество студента -> %s", items[_item_index].second_name);
            printf("\n  Изменить отчество студента на -> ");
            scanf("%s", buffer);
            strcpy(items[_item_index].second_name, buffer);
        }
        //
// добавление оценок
        printf("\n  Добавить оценки д/н ? -> ?");
        scanf("%d", &_answer);
        if (_answer == _yes) {
            printf("\n");
            printf("  Вводите оценки по одной и нажимайте ввод \n");
            printf("  чтобы завершить ввод введите -1 \n");
            while (stop_flag != 1)
            {
                printf("  Оценка ->  ");
                scanf("%d", &curent_mark);
                if (curent_mark != -1) {
                    if (curent_mark < 2 || curent_mark > 5)
                    {
                        curent_mark = 0;
                        mark_count = items[_item_index].marks_count;
                    }
                    else {
                        mark_count = items[_item_index].marks_count + 1;// увелич к-во
оценок
                    }
                    items[_item_index].marks_count = mark_count;
                    items[_item_index].marks[mark_count - 1] = curent_mark;// записываем
оценку
                }
                else {
```

```
                    stop_flag = 1;
                }
                items[_item_index].marks_average = average_mark(items[_item_index].marks);
            }
        }
    }
    else {
        print_not_found_error(_item_id);
    }
    return items;
}
```

**Файл menu.c**

```
#include "header.h"

void info()
{
    printf("\n\t\t\tИнформатика и программирование Ч-2. Лабораторная работа №2 вариант
№9.\n\n");
    printf("  Задание : \n");
    printf("\t    Определить структурированный тип, определить набор функций для работы с
массивом структур.\n");
    printf("\t  В  структурированной переменной предусмотреть способ отметки ее как \n\t
не содержащей данных (т.е. ""пустой"").\n");
    printf("\t  Функции должны работать с массивом структур или с отдельной структурой че-
рез указатели,\n");
    printf("\t  а также при необходимости возвращать указатель на структуру. \n\n\t  В пе-
речень функций входят:\n\n");
    printf("\t  - «очистка» структурированных переменных;\n");
    printf("\t  - поиск свободной структурированной переменной;\n");
    printf("\t  - ввод элементов (полей) структуры с клавиатуры;\n");
    printf("\t  - вывод элементов (полей) структуры с клавиатуры;\n");
    printf("\t  - поиск в массиве структуры и минимальным значением  заданного поля;\n");
    printf("\t  - сортировка массива  структур в порядке возрастания заданного поля\n");
    printf("\t    (при сортировке можно использовать тот факт, что в Си++ разрешается\n");
    printf("\t    присваивание структурированных переменных);\n");
    printf("\t  - поиск в массиве структур элемента с заданным значением поля \n");
    printf("\t    или с наиболее близким к нему по значению.\n");
    printf("\t  - удаление заданного элемента;\n");
    printf("\t  - изменение (редактирование) заданного элемента.\n");
    printf("\t  - вычисление с проверкой и использованием всех элементов массива по задан-
ному условию\n");
    printf("\t    и формуле (например, общая сумма на всех счетах) -  дается индивидуаль-
но.\n\n");
    printf("\t  Перечень полей структурированной переменной.\n\n");
    printf("\t  Вариант № 9. Фамилия И.О., количество оценок, оценки, средний балл.\n\n");
    printf("\t\t\t\tДля входа в программу нажмите | Enter |\n");
}

void start(student* items, int* _count_record, int* _last_id, int* _count_free_items)
{
    clear_screen();
    info();
    while (true)
    {
        switch (_getch())
        {
        case ESC:
            free(items);
            exit(0);
```

```
case ENTER: // загрузить записи
    clear_screen();
    print_menu_header();
    print_table_header();
    print_all_items(items, _count_record);
    print_line();
    print_footer(_count_record, _count_free_items, items);
    break;

case SHIFT_L: // загрузка тестовых записей
    clear_screen();
    print_menu_header();
    print_table_header();
    items = add_test_data_to_array(items, _count_record, _last_id);
    print_all_items(items, _count_record);
    print_line();
    print_footer(_count_record, _count_free_items, items);
    break;

case SHIFT_A: // добавить запись
    clear_screen();
    print_menu_header();
    print_table_header();
    print_line();
    items = data_add_to_array(items, _count_record, _last_id, _count_free_items);
    print_footer(_count_record, _count_free_items, items);
    break;

case SHIFT_E: // редактировать запись
    clear_screen();
    print_menu_header();
    print_table_header();
    print_line();
    print_all_items(items, _count_record);
    print_line();
    items = edit_item(items, _count_record);
    print_footer(_count_record, _count_free_items, items);
    break;

case SHIFT_D: // удалить запись
    clear_screen();
    print_menu_header();
    print_table_header();
    print_line();
    print_all_items(items, _count_record);
    print_line();
    items = delete_item(items, _count_record);
    print_footer(_count_record, _count_free_items, items);
    break;

case SHIFT_C: // очистить запись
    clear_screen();
    print_menu_header();
    print_table_header();
    print_line();
    print_all_items(items, _count_record);
    print_line();
    items = clear_item(items, _count_record);
    print_footer(_count_record, _count_free_items, items);
    break;

case SHIFT_W: // запуск сортировки id
    clear_screen();
    clear_screen();
```

```
        print_menu_header();
        print_table_header();
        print_line();
        sort_by_id(items, _count_record);
        print_all_items(items, _count_record);
        print_line();
        print_footer(_count_record, _count_free_items, items);
        break;

    case SHIFT_Q: // запуск сортировки surname
        clear_screen();
        clear_screen();
        print_menu_header();
        print_table_header();
        print_line();
        sort_by_surname(items, _count_record);
        print_all_items(items, _count_record);
        print_line();
        print_footer(_count_record, _count_free_items, items);
        break;

    case SHIFT_F: // запуск поиска средний бал
        clear_screen();
        print_menu_header();
        print_table_header();
        //print_line();
        print_items_range_average(items, _count_record);
        print_line();
        print_footer(_count_record, _count_free_items, items);
        break;

    case SHIFT_R: // спис студентов мин ср бал
        clear_screen();
        print_menu_header();
        print_table_header();
        print_line();
        print_items_min(items, _count_record);
        print_line();
        print_footer(_count_record, _count_free_items, items);
        break;

    case SHIFT_T: // спис студентов макс ср бал
        clear_screen();
        print_menu_header();
        print_table_header();
        print_line();
        print_items_max(items, _count_record);
        print_line();
        print_footer(_count_record, _count_free_items, items);
        break;

    case SHIFT_S: // запуск поиска surname
        clear_screen();
        print_menu_header();
        print_table_header();
        //print_line();
        search_by_name(items, _count_record);
        print_line();
        print_footer(_count_record, _count_free_items, items);
        break;

    case SHIFT_I: // показать задание
        clear_screen();
        info();
```

```
            break;
        }
    }
}

void clear_screen()
{
    system("cls");
}

void print_menu_header()
{
    printf("\n");
    print_line();
    printf(" | ESC -> Выход.         | Shift+A -> Добавить.     | Shift+D -> Удалить.    |
Shift+S -> Поиск по фимилии |\n");
    printf(" | Enter -> Загрузить.   | Shift+E -> Редактировать.| Shift+C -> Очистить.   |
Shift+F -> Поиск ср. бал    |\n");
    print_line();
    printf(" | Shift+Q -> Сортировка по фамилии.               | Shift+W -> Сртировка по
id.                          |\n");
    print_line();
    printf(" | Shift+R -> Список студентов с мин. средним балом | Shift+T -> Список студен-
тов с мак. средним балом     |\n");
    print_line();
    printf("\n");
}

void print_table_header()
{
    //printf("\n");
    print_line();
    // body
    printf(" | Id | Фамилия              | Имя             | Отчество                    |
Кол-во оценок | Средний бал |\n");
    print_line();
    //printf("\n");
}

void print_one_record(student* item)
{
    printf(" |%3d | %20s | %15s | %27s | %13d | %11f |\n",
        item->id, item->surname, item->name, item->second_name, item->marks_count, item-
>marks_average);
}

void print_search_header()
{
    printf("\n");
    print_line();
    // body
    printf(" | Поиск по минимальному среднему балу  ||  |\n");
    print_line();
    printf("\n");
}

void print_footer(int* _count_record, int* _count_free_items, student* items)
{
    *_count_free_items = count_free_items(items, _count_record);
    printf("\n");
    print_line();
    printf(" | <<   Оценок %3d      ||     Всего записей ->  %3d     ||  Свободно ->  %3d
|| Средний бал %.2lf    >> |\n",
```

```
        all_items_count_marks(items, _count_record), *_count_record, *_count_free_items,
all_items_average_marks(items, _count_record));
    print_line();
    printf(" | <<<<                  Для загрузки тестовых данных нажмите сочетание клавиш
Shift + L              >>>> |\n");
    print_line();
    printf("\n");
}

void print_line()
{
    printf(" ");
    for (int i = 0; i < 107; i++) {
        printf("-");
    }
    printf("\n");
}

int count_free_items(student* items, const int* _count_record) {
    int count = 0;
    int _is_free;
    if (items != NULL) {
        for (student* item = items; item < items + *_count_record; item++) {
            _is_free = item->is_free;
            if (_is_free == 1) {
                count++;
            }
        }
    }
    return count;
}

void print_all_items(student* items, const int* _count_record)
{
    if (items != NULL) {
        for (student* item = items; item < items + *_count_record; item++)
        {
            if (item->is_free == 0)
            {
                print_one_record(item);
            }
        }
    }
    else {
        printf("  Отсутствуют записи в базе !\n");
    }
}

void print_items_min(student* items, const int* _count_record)
{
    double _min = search_min_average_marks(items, _count_record);
    if (_min > 0)
    {
        for (student* item = items; item < items + *_count_record; item++)
        {
            if (item->marks_average == _min)
            {
                print_one_record(item);
            }
        }
    }
}

void print_items_max(student* items, const int* _count_record)
```

```
{
    double _max = search_max_average_marks(items, _count_record);
    if (_max > 0)
    {
        for (student* item = items; item < items + *_count_record; item++)
        {
            if (item->marks_average == _max)
            {
                print_one_record(item);
            }
        }
    }
}

void print_items_range_average(student* items, const int* _count_record)
{
    double input_value = input_average_marks();
    double _avr;
    int flag = 0;
    if (items) {
        for (student* item = items; item < items + *_count_record; item++)
        {
            _avr = item->marks_average;
            if (_avr >= input_value - 0.3 &&
                _avr <= input_value + 0.3)
            {
                print_one_record(item);
                flag = 1;
            }
        }
    }
    if (flag == 0) {
        printf("  Не найдено записей с заданным средним балом !!!\n");
    }
}

double input_average_marks()
{
    double input_value;
    printf("  Введите значение для поиска среднего бала студента - > ");
    scanf("%lf", &input_value);
    printf("  Была запущена функция поиска по заданному значению или \n");
    printf("  с наиболее близким к нему значением... \n");
    print_line();
    print_line();
    //_getch();
    return input_value;
}

double all_items_average_marks(student* items, const int* _count_record)
{
    double _all = 0;
    if (items != NULL) {
        for (student* item = items; item < items + *_count_record; item++)
        {
            if (item->is_free == 0)
            {
                _all = _all + item->marks_average;
            }
        }
        return _all / *_count_record;
    }
    return 0;
}
```

```
int all_items_count_marks(student* items, const int* _count_record)
{
    int _all = 0;
    if (items != NULL) {
        for (student* item = items; item < items + *_count_record; item++)
        {
            if (item->is_free == 0)
            {
                _all = _all + item->marks_count;
            }
        }
        return _all;
    }
    return 0;
}

void print_not_found_error(int _item_id)
{
    printf("\n  Не найдено записи с введенным id = %d !!!\n", _item_id);
}

//int mygetch() {
//    struct termios termios, newt;
//    int ch;
//    tcgetattr(STDIN_FILENO, &termios);
//    newt = termios;
//    newt.c_lflag &= ~(ICANON | ECHO);
//    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
//    ch = getchar();
//    tcsetattr(STDIN_FILENO, TCSANOW, &termios);
//    return ch;
//}
```