РАБОТА N 3. СОРТИРОВКА С ИСПОЛЬЗОВАНИЕМ ДОПОЛНИТЕЛЬНЫХ МАССИВОВ

Одним из путей повышения эффективности ОС является упорядочение массива в процессе его заполнения, т.е. каждый поступающий вновь ключ находит свое место среди уже поступивших.

Различные методы сортировки с использованием дополнительных массивов применяются в разных ситуациях, возникающих при обработке данных.

Рассмотрим наиболее важные из них.

3.1. Метод вставки

Пусть требуется сформировать упорядоченный массив ключей $B(N)$. Предположим также, что источником является неупорядоченный массив $A(N)$.

Суть метода состоит в том, что каждый новый элемент a_i вставляется в уже упорядоченную совокупность $b_1, b_2, \ldots b_{i-1}$ на подходящее место по следующему алгоритму:

1) $j=1$;

2) сравнить элементы b_j и a_i ; если a_i не соответствует месту j по условию сортировки, то перейти к п.5;

3) сдвинуть вправо все элементы от b_{i-1} до b_j на одну позицию;

4) присвоить значение $b_j = a_i$, т.е. поместить a_i на j-е место;

5) перейти к рассмотрению следующего занятого места: $j=j+1$;

6) если $j<i-1$, то повторять с п.2;

7) поместить a_i на первое незанятое место, $b_j = a_i$;

8) перейти к вводу следующего элемента a_i , т.е. $i=i+1$.

3.2. Метод двухпутевого слияния

Этот метод применяется в случае, когда объединяются два упорядоченных массива. Пусть требуется объединить два массива ключей $A(N)$ и $B(M)$, упорядоченных по возрастанию, в один массив $C(K)$, $K=N+M$, также упорядоченный по возрастанию.

Суть метода состоит в сравнении очередной пары ключей a_i и b_j и занесении минимального из них в массив C. При этом возможны варианты реализации данного метода:

а) занести в очередной элемент массива c_{kl} ключ a_i и сравнить его с ключом b_j, если $b_i < c_{kl}$, то $c_{kl} = b_j$, затем перейти к занесению следующего элемента $c_{kl+1} = b_{i+1}$ и сравнению с a_k и т.д.;

б) сравнить ключи a_i и b_j и минимальный из них занести на место c_{kl}, затем перейти к следующему элементу c_{kl+1} и текущему значению ключа a_{i+1} или b_{j+1}.

Следует отметить, что совпадения ключей a_i и b_j, как правило, не допускаются, в противном случае в массив C ключ вносится однократно.

3.3. Адресная сортировка

Пусть задан массив ключей $X(N)$, представляющих собой целые положительные числа. Адресная сортировка выполняется в два этапа:

1) подсчет числа ключей с одинаковыми значениями в массиве $X(N)$. Результаты подсчета заносятся в массив $A(M)$, где $A(i)$ - число ключей в массиве $X(N)$, имеющих значение i, а M - максимально возможное значение ключа;

2) заполнение массива $Y(N)$ упорядоченными значениями ключей на основе данных, полученных в массиве $A(M)$.

Адресная сортировка упорядочивает ключи по неубыванию, но может использоваться и для упорядочения чисел по невозрастанию.

Рассмотрим подробно алгоритм адресной сортировки:

1) $K=0$, массив $A(M)=0$;

2) при изменении I от 1 до N для массива $X(N)$ сформировать массив $A(M)$ по формуле $A(X(I))=A(X(I))+1$;

3) при изменении I от 1 до M для массива $A(M)$ сформировать массив $Y(N)$: если $A(I)=0$, перейти к п.5;

4) при изменении J от 1 до $A(I)$ присвоить $Y(K)=I$, $K=K+1$;

5) перейти к п.3, если $I<M$.

Таким образом, массив A содержит распределение частот появления ключей в заданной последовательности в порядке их возрастания. Для получения отсортированной последовательности ключей в выходной массив Y текущее значение ключа I записывается столько раз, сколько ключ I встречается в исходной последовательности (внутренний цикл по J).

Следует отметить, что для отсортированной последовательности можно использовать исходный массив $X(N)$. В случае использования языков программирования, позволяющих задавать минимальное значение индекса массива, можно сократить затраты памяти на массив A с соответствующей модификацией алгоритма.

Содержание отчета:

1) структурная схема алгоритма;

2) текст программы;

3) распечатка указанных в задании результатов выполнения программ.

Задание.

Разработать и отладить три программы сортировки, выводя на печать исходные состояния массивов и состояния заполняемого массива после каждого изменения; пусть ключи - целые положительные числа.

Варианты заданий приведены в табл. 3.1 и 3.2, где для каждого метода заданы размерности массивов, критерий упорядочения и вариант реализации.

Для адресной сортировки задано максимальное число повторения ключей, а также имя массива для хранения результата.

Таблица 3.1

Номер варианта	Метод	
	вставки	слияния
1	N=15; упорядочение по убыванию без повторений	N=10, M=8; упорядочение по возрастанию без совпадений ключей
2	N=16; упорядочение по возрастанию без повторений	N=8, M=12; упорядочение по возрастанию без совпадений ключей
3	N=14; упорядочение по убыванию с повторением ключей	N=8, M=8; упорядочение по возрастанию без совпадений ключей
4	N=15; упорядочение по возрастанию с повторением ключей	N=12, M=6; упорядочение по возрастанию с совпадением ключей
5	N=16; упорядочение по убыванию без повторений	N=9, M=9; упорядочение по возрастанию с совпадением ключей
6	N=14; упорядочение по возрастанию без повторений	N=7, M=13; упорядочение по возрастанию с совпадением ключей
7	N=15; упорядочение по неубыванию с повторением ключей	N=9, M=9; упорядочение по возрастанию с совпадением ключей

Таблица 3.2

Номер варианта	Адресная сортировка					
	N	X min	X max	Число повторений	Массив результатов	Порядок сортировки
1	20	1	17	4	Y	по неубыванию
2	20	1	50	3	Y	по неубыванию
3	20	20	70	4	X	по неубыванию
4	25	10	50	5	X	по неубыванию
5	20	10	60	3	X	по неубыванию
6	25	10	60	5	Y	по невозрастанию
7	25	1	50	4	X	по невозрастанию

РАБОТА N 4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СТЕКА

Стеком называется специально организованная память, которая позволяет записывать и считывать информацию только по определенной дисциплине. При этом возможны две дисциплины обслуживания записей:

LIFO (последним пришел - первым ушел) - часто именно эта реализация называется стеком;

FIFO (первым пришел - первым ушел) - в литературе часто называется деком.

Стек должен выполнять следующие функции:

1) запись элемента в стек;

2) чтение элемента из стека, причем прочитанная информация должна стираться из стека;

3) оповещение об экстремальных ситуациях:

- попытка записи в заполненный стек;
- попытка чтения из пустого стека.

Программная реализация стека заключается в выделении одномерного массива $A(N)$, где N - глубина стека, и указателя стека K - простой переменной целого типа, указывающей адрес последнего заполненного элемента или адрес первой свободной ячейки (в зависимости от желания авторов).

Стек типа LIFO находит широкое применение в ОС для организации вложенных циклов, вложенных подпрограмм и в других случаях. Стек типа FIFO лежит в основе организации различного типа очередей.

4.1. Реализация стека LIFO

Возможно несколько вариантов программной реализации стека:

а) стек заполняется, начиная с первого индекса, это наиболее экономичный способ, так как здесь не происходит сдвига информации;

б) заполнение стека со стороны «дна», т.е. с последнего индекса.

4.2. Реализация стека FIFO

Для дека существует много программных реализаций, которые характеризуются следующими особенностями:

а) заполнение стека последовательно, начиная с первого индекса без сдвига информации, и считывание всегда 1-го элемента массива со сдвигом оставшихся элементов влево;

б) заполнение стека, начиная с первого индекса, со сдвигом информации при записи и при считывании;

в) заполнение стека со стороны «дна», т.е. с последнего индекса без сдвига информации, а считывание из последнего индекса со сдвигом информации вправо;

г) «кольцевой» стек отличается тем, что нет сдвига информации ни при записи, ни при считывании, но вводится более сложная система указателей: указатель первого в очереди элемента (начало очереди), указатель конца очереди; здесь возникает также проблема организации «кольцевого» перехода указателей стека.

Содержание отчета:

1) структурная схема алгоритма;

2) текст программы;

3) распечатка указанных в задании результатов выполнения программ.

Задание.

Разработать программу реализации стеков для двух дисциплин с использованием указателя выбора дисциплины обслуживания, значение которого вводится в режиме диалога. Программа должна продемонстрировать все возможные действия, выполняемые стеком, в диалоговом режиме.

Варианты заданий приведены в табл. 4.1, где указана глубина стека и вариант реализации стека FIFO, причем вариант реализации стека LIFO следует выбрать самостоятельно, исходя из соображений совместимости программ для двух видов стека.

Таблица 4.1

Номер варианта	Глубина	Вариант FIFO	Примечания
1	5	а	С подпрограммами
2	5	б	С подпрограммами
3	4	в	Без подпрограмм
4	4	г	Без подпрограмм
5	3	в	С подпрограммами
6	3	г	С подпрограммами
7	4	а	Без подпрограмм

РАБОТА N 5. ХЭШИРОВАНИЕ КЛЮЧЕЙ

Поиск данных в больших расширяющихся таблицах, размеры которых заранее не известны, приходится вести другим способом. Линейный поиск здесь неприменим из-за размеров таблицы, а двоичный потребовал бы при добавлении каждой новой записи проводить повторное упорядочение (сортировку) таблицы по возрастанию ключа, что выливалось бы в дополнительные временные потери. Поэтому таблицы организуются по способу перемешанных (разбросанных, рэндомизированных, с преобразованием ключа, хэш-таблиц - аналогичные названия) таблиц.

В перемешанных таблицах, как и в таблицах с прямым доступом, используется адресная функция $I(K)$, которая по заданному значению ключа K позволяет определить значение адреса $I(K)$. Однако здесь заранее допускается, что $I(K)$ не обеспечивает однозначности, т.е. могут возникать ситуации (называемые коллизиями), при которых

$$I(K_i) = I(K_j), \quad K_i \neq K_j .$$

Вектор хранения, куда отображаются записи, имеет длину $N > M$, где M — число табличных записей. Предполагается, что M сверху ограничено.

Простейший алгоритм открытого перемешивания при занесении в таблицу:

1) по заданному K вычислить $i=I(K)$;

2) если запись с номером i в таблице пустая, то занести в нее новую запись;

3) если нет, то положить $i=(i+1) mod\ N$ и вернуться к п.2.

Очевидно, что поиск записи по заданному ключу аналогичен.

Таким образом, последовательность процедур 2 и 3 обеспечивает линейный поиск свободной позиции в таблице для помещения в нее коллизийной записи. Неравномерность размещения записей, а следовательно, скопление коллизийных записей в некоторых областях таблицы увеличивает линейный поиск. Поэтому в ряде случаев улучшить общую картину можно, изменив процедуру 3 данного алгоритма. В этом случае полагают $i=(i+P) \bmod N$.

Пусть $\sigma = M/N$, где M - текущее значение для числа записей, уже включенных в таблицу. Тогда приближенное значение для длины поиска в таблице

$$S(\sigma) = \frac{2-\sigma}{2-2\sigma},$$

т.е. длина поиска зависит от коэффициента заполнения таблицы. Достаточно интересным фактом для перемешанных таблиц является то, что средняя длина поиска зависит не от размера таблицы, а только от ее заполнения.

В целях сокращения длины поиска при размещении коллизийных записей в процессе расширения таблицы часто все переполняющие записи выносят в другую таблицу, «сцепленную» с основной. Простейший вариант состоит в том, что переполняющие записи выносят во вспомогательную таблицу, где их размещают по методу линейного поиска. В организации перемешанной таблицы каждая запись имеет усложнение в своей структуре. В ней вводится поле, где размещается

адрес первой коллизийной записи, расположенной в «сцепленной» таблице. Точно так же усложняется структура записей «сцепленной» таблицы, поскольку в них хранят адрес размещения следующей коллизийной записи, и т.д.

Средняя длина поиска в этом случае оценивается приблизительно:

$$S(M,N) = 1 + \frac{(M-1)}{2N}.$$

Очень часто после того как расширение таблицы завершено и места в таблице достаточно для размещения коллизийных записей, содержимое сцепленной таблицы переносится в основную при сохранении их сцепления по адресам размещения.

При выборе хэш-функции h, отображающей множество ключей $k \in K$ в адреса $a \in [\varnothing, M-1]$, требуется, чтобы она не только сокращала число коллизий, но и исключала скучивание ключей в отдельных частях таблицы.

5.1. Выбор хэш-функции

Выбор хэш-функции производится, как правило, из двух основных.

а) *Метод деления.* В качестве значения хэш-функции используется остаток от деления ключа на некоторое целое M, т.е.

$$L(K) = K \bmod M.$$

Если M является степенью основания системы счисления, то значением хэш-функции являются младшие разряды ключа. Для предотвращения скучивания ключей лучше брать M простым числом. Желательно связать M с длиной таблицы размещения.

б) *Метод умножения.* В качестве хэш-функции используют $L(k) = [M(k\alpha)]$, где $[x]$- наибольшее целое, не превосходящее x.

Константе α, как правило, присваивают значение, называемое золотым сечением: $\alpha = (\sqrt{5}-1)/2 \approx 0.648$. Здесь M может быть кратно 2, т.е. $M = 2^m$, т.е. M может быть длиной таблицы размещения.

5.2. Разрешение коллизии

а) *Метод открытой адресации*

Алгоритм можно представить в форме:

1) $i=0$;

2) $a = h(k)+i$;

3) если a свободно, конец алгоритма (если это процедура вставки, то занесение по адресу a;если процедура поиска, то ключа нет).

4) если ключ, связанный с адресом a, равен k, то при поиске алгоритм заканчивается;

5) $i=i+1$ и переход к шагу 2.

б) *Линейное апробирование.* В этом случае в схеме открытой адресации используется $a=h(k)+ci$, где c - константа; c и M выбираются взаимно простыми, c - не очень малым.

в) *Квадратичное апробирование.* Это схема открытой адресации, где $a = h(k) + ci + di^2$, где c,d - константы.

г) *Двойное хэширование.* Это схема открытой адресации, $a=h(k)+iq(k)$, где $q(k)$ - хэш-функция, почти такая, как $h(k)$, но не эквивалентная ей. Так, например, используют следующие представления:

$$a = (k \bmod M) + i(1 + k \bmod (M-2))$$

или

$$a = h(k) + i(M - h(k)).$$

Теоретическая оценка числа коллизий при случайном равномерном распределении по M адресам дается следующими соотношениями:

математическое ожидание: $Ml^{-N/M} - (M-N)$;

дисперсия: $Ml^{-N/M}(1-(M-N)l^{-N/M}/M$. Распределение при $1 << N << M$ приближается к распределению Пуассона, при $1 << N \approx M$ - к нормальному распределению.

Содержание отчета:

1) структурная схема алгоритма;

2) текст программы;

3) распечатка указанных в задании результатов выполнения программ.

Задание.

Взять множество целых двузначных ключей $A(N)$. Разработать две программы, демонстрирующие заполнение хэш-таблицы и поиск хотя бы одного ключа в таблице для заданной хэш-функции (одна - без «сцепленной» таблицы, $M > N$, другая - с использованием «сцепленной» таблицы $M = N$).

В табл. 5.1 приведены варианты заданий.

Таблица 5.1

Номер варианта	Число ключей	Метод хэширования	Метод разрешения коллизий
1	18	а	б
2	19	а	в
3	18	б	а
4	19	б	в
5	18	а	г
6	18	б	а
7	19	б	г

РАБОТА N 6. ОДНОСВЯЗНЫЕ И ДВУСВЯЗНЫЕ СПИСКИ

Одно- и двусвязные списки относятся к линейным динамическим структурам данных, которые характеризуются переменным размером структуры и отсутствием последовательного расположения элементов структуры в памяти.

Число элементов динамической структуры может изменяться от нуля до некоторой величины, определяемой доступным объемом машинной памяти. Логическая последовательность элементов задается в явном виде с помощью одного или двух указателей, хранящихся в самих элементах. Таким образом, элементы динамической структуры могут быть разбросаны в памяти хаотическим образом, в результате чего усложняется процедура доступа к элементам. Для доступа к требуемому элементу необходимо просматривать список с его начала.

Связный список - это структура, элементами которой служат записи с одним и тем же форматом.

6.1. Односвязные списки

В односвязном списке каждый элемент состоит из двух различных по назначению полей: информационного поля и поля указателя. В информационном поле хранятся данные, которые могут размещаться в нескольких логически связанных подполях. Кроме того, некоторые подполя могут содержать указатели на поля дополнительной информации, напрямую не относящиеся к связным спискам. В поле указателя хранится адрес следующего элемента списка. Для каждого элемента (кроме первого и последнего) существует единственный предыдущий и единственный последующий элемент. Между элементами одного списка в памяти могут находиться элементы других списков.

Как правило, для каждого списка существует особая запись, называемая дескриптором и содержащая указатель (адрес) начала списка, текущее число элементов в списке, имя списка и другую вспомогательную информацию (рис.6.1).

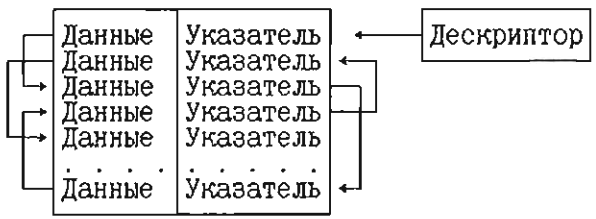

Рис.6.1

Последний элемент списка в поле указателя содержит специальный признак, свидетельствующий о конце списка. Если в области памяти, отведенной под структуры, содержатся элементы, не входящие ни в один список, то, как правило, их поля указателей содержат признаки пустой записи.

6.2. Двусвязные списки

Нередко при просмотре линейного списка возникает необходимость продвижения в любом из двух направлений: от начала к концу и от конца к началу. В этом случае используют линейный двусвязный список, который отличается от односвязного тем, что содержит два указателя, один из которых адресует следующий элемент, а другой - предыдущий элемент списка. Дескриптор двусвязного списка содержит указатель как начала, так и конца списка. В конце каждого пути (по прямым и обратным указателям) находятся признаки конца списка. К основным операциям, выполняемым над связными списками, относятся операции создания нового списка, удаления существующего списка, включения элемента в список и исключения элемента из списка. Кроме того, часто используются операции определения свободных элементов в области памяти линейных структур, упорядочения элементов по какому-либо признаку в информационном поле.

6.3. Общие алгоритмы выполнения основных операций над односвязными списками

а) *Создание нового списка*

Входные данные: массив значений для информационных полей элементов списка.

1. Найти свободный элемент списка.
2. Записать его адрес в дескриптор.
3. Заполнить информационное поле найденного элемента требуемыми данными.
4. Если данные исчерпаны, записать в поле указателя признак конца списка и закончить работу.
5. Запомнить адрес текущего элемента.
6. Найти свободный элемент.
7. Записать его адрес в поле указателя предыдущего элемента.
8. Перейти к п.3.

б) *Удаление существующего списка*

Входные данные: дескриптор удаляемого списка.

1. Считать из дескриптора адрес начала списка.
2. Запомнить содержимое поля указателя текущего элемента.
3. Записать в поле указателя признак свободного элемента.

4. Если в поле указателя был не признак конца списка, перейти к п. 2.

в) *Добавление элемента в список*

Входные данные: дескриптор списка, порядковый номер элемента списка, после которого надо вставить элемент в список, добавляемые данные.

1. Найти свободный элемент.

2. Если добавляется первый элемент в список, то запомнить адрес списка, записать в дескриптор адрес найденного свободного элемента, перейти к п. 6.

3. Найти элемент списка, после которого необходимо вставить новый элемент.

4. Запомнить содержимое поля указателя (адрес элемента, следующего за новым).

5. Записать в поле указателя адрес свободного элемента.

6. Заполнить данными информационное поле свободного элемента.

7. Записать в поле указателя сохраненный адрес следующего элемента.

г) *Исключение элемента из списка*

Входные данные: порядковый номер удаляемого элемента списка, дескриптор списка.

1. Если исключается первый элемент списка, то изменить содержимое дескриптора, перейти к п. 4.

2. Найти элемент, после которого находится исключаемый элемент списка, и запомнить его адрес.

3. Переписать информацию из поля указателя удаляемого элемента в поле указателя предыдущего элемента.

4. Пометить удаляемый элемент как свободный.

Содержание отчета:

1) структурная схема алгоритма;

2) текст программы;

3) распечатка указанных в задании результатов выполнения программ.

Задание 1. Составить программы работы с односвязными списками, пользуясь следующими соглашениями. В качестве области памяти односвязных списков использовать массив размерности [2,50], начальное содержимое которого считать из файла SPIS1.DAT. Поле указателя элемента списка содержит номер столбца массива со следующим элементом списка. Последний элемент содержит в поле указателя значение 0, а свободный элемент - значение -1. Вывод информационных полей осуществляется в виде строки символов форматным оператором вывода через спецификацию 'A1' (FORTRAN-IV). Восьмеричные и десятичные коды символов ASCII и содержимое файла SPIS1.DAT даны в приложениях 1 и 2.

Варианты задания 1:

1. Создать новый односвязный список, информационные поля которого содержат символы слова «ДРАЙВЕР». Вывести содержимое сформированного массива и содержимое сформированного списка (информационных полей и полей указателей).

2. Удалить односвязный список, дескриптор которого указывает на адрес 25. Вывести содержимое сформированного массива.

3. Дополнить односвязный список, дескриптор которого указывает на адрес 2, элементом с символом «– » между 3-м и 4-м элементами списка. Вывести содержимое информационных полей полученного списка в виде строки символов.

4. Удалить из односвязного списка, дескриптор которого указывает на адрес 30, пятый элемент списка. Вывести содержимое списка в виде строки символов.

5. Определить все элементы, не относящиеся ни к одному списку (дескрипторы списков содержат адреса 1, 2, 25, 30) и не помеченные как свободные. Перевести эти элементы в разряд свободных и вывести их адреса.

6. Дополнить односвязный список, дескриптор которого указывает на адрес 25, элементом с символом « . », стоящим после последнего элемента списка. Вывести содержимое информационных полей полученного списка в виде строки символов.

7. Скопировать односвязный список, дескриптор которого указывает на адрес 30, сформировав в свободных элементах новый список. Вывести содержимое сформированного массива и адрес начала сформированного списка.

Задание 2. Составить программы работы с двусвязными списками, пользуясь следующими соглашениями. В качестве области памяти двусвязных списков использовать массив размерности [3,50], начальное содержимое которого читать из файла SPIS2.DAT. Поля указателей элемента списка содержат номера столбцов массива со следующим и предыдущим элементами списка соответственно. Последний элемент содержит в поле указателя значение 0, а свободный элемент - значение -1. Вывод информационных полей осуществляется в виде строки символов форматным оператором вывода через спецификацию 'A1' (FORTRAN-IV). Восьмеричные и десятичные коды символов ASCII и содержимое файла SPIS2.DAT даны в приложениях 1 и 2.

Варианты задания 2:

1. Дополнить двусвязный список, дескриптор которого указывает на адрес 2, элементом с символом «-» между 3-м и 4-м элементами списка. Вывести содержимое информационных полей полученного списка в виде строки символов, начиная с последнего элемента.

2. Создать новый двусвязный список, информационные поля которого содержат символы слова «ДРАЙВЕР». Вывести содержимое сформированного массива и содержимое сформированного списка (информационных полей и полей указателей), начиная с последнего элемента.

3. Удалить двусвязный список, дескриптор которого указывает на адрес 25. Вывести содержимое сформированного массива.

4. Определить все элементы, не относящиеся ни к одному списку (дескрипторы списков содержат прямые адреса 1,2,25,30) и не помеченные как свободные. Перевести эти элементы в разряд свободных и вывести их адреса.

5. Удалить из двусвязного списка, дескриптор которого указывает на обратный адрес 39, пятый элемент списка. Вывести содержимое списка в виде строки символов, начиная с последнего элемента.

6. Определить все элементы, не относящиеся ни к одному списку (дескрипторы списков содержат обратные адреса 20,15,39, 33) и не помеченные как свободные. Перевести эти элементы в разряд свободных и вывести их адреса.

7. Дополнить двусвязный список, дескриптор которого указывает на обратный адрес 20, элементом с символом «-» между 3-м и 4-м элементами списка. Вывести содержимое информационных полей полученного списка в виде строки символов.

6.4. Пример реализации функций ввода-вывода на языке FORTRAN-IV (ЭВМ «Электроника МС0585»)

а) Для односвязных списков . Скопировать файл данных SPIS1.DAT в рабочий файл данных:

.COPY SPIS1.DAT S03519.DAT

где S03519 - имя программы пользователя на языке FORTRAN-IV. Перед запуском программы необходимо дать назначение:

.AS S03519.DAT 1

Затем оттранслировать и запустить программу, содержащую фрагменты, аналогичные следующим:

```
      BYTE A(2,50),B(50),D(4)    !Массив, резервирующий область
C памяти структур, выходной массив, массив дескрипторов
      DATA D/1,2,25,30/          !Значения дескрипторов
      READ (1,1) A               !Загрузить начальное содержимое
1     FORMAT (20O3)              ! области списков
      TYPE 2,A                   !Вывести на экран начальное со-
2     FORMAT (10(X,A1,X,I2))     !держимое области списков
      . . . . . .
      A(1,10)='Д'                !Занести в информационные поля
      A(1,11)=«103               !элементов списка коды символов
                                 !«Д» и «С» (103₈)
      . . . . . .
C Здесь в массиве «B» формируется последовательность информаци-
C онных полей требуемого списка. N - длина списка
      . . . . . .
      WRITE (6,3) (B(I),I=1,N)   !Вывод содержимого информационного
3     FORMAT ('+',A1,$)          !поля списка длиной N
      WRITE (6,*)                !Перевести строку для след. списка
      . . . . . .
```

б) Для двусвязных списков. Скопировать файл данных SPIS2.DAT в рабочий файл данных:

.COPY SPIS2.DAT S03519.DAT

где S03519 - имя программы пользователя на языке FORTRAN-IV. Перед запуском программы необходимо дать назначение

.AS S03519.DAT 1

Затем оттранслировать и запустить программу, содержащую фрагменты, аналогичные следующим:

```
      BYTE A(3,50),B(50),D(4)    !Массив, резервирующий область
C памяти структур, выходной массив, массив дескрипторов
      DATA D/1,2,25,30/          !Значения дескрипторов
      READ (1,1) A               !Загрузить начальное содержимое
1     FORMAT (20O3)              ! области списков
      TYPE 2,A                   !Вывести на экран начальное со-
2     FORMAT (10(X,A1,X,I2,X,I2) !держимое области списков
      . . . . . .
      A(1,10)='Д'                !Занести в информационные поля
      A(1,11)=«103               ! элементов списка коды символов
                                 ! «Д» и «С» (103₈)
      . . . . . .
C Здесь в массиве «B» формируется последовательность информаци-
C онных полей требуемого списка. N - длина списка.
      . . . . . .
      WRITE (6,3) (B(I),I=1,N)   !Вывод содержимого информационного
3     FORMAT ('+',A1,$)          !поля списка длиной N
      WRITE (6,*)                !Перевести строку для след. списка
      . . . . . .
```

Содержимое файла данных SPIS1.DAT

Г 6	К 7	Б 13	Е 11	К 10	Д 4	А 5	И 18	Д 15	Н 16
_ 14	П 8	У 9	С 12	Ь 0	И 3	О 19	С 17	К 20	? 0
Т 36	О 31	_ -1	И 38	В 22	Т 21	_ -1	_ -1	_ -1	С 34
Т 32	_ 26	К 0	П 24	_ -1	А 39	_ -1	С 40	К 0	О 33
_ -1	! 41	_ -1	В 42	_ 44	_ -1	К 48	_ -1	_ -1	_ -1

Содержимое файла данных SPIS2.DAT

Г 6 0	К 7 0	Б 13 16	Е 11 6	К 10 7	Д 4 1	А 5 2
И 18 12	Д 15 13	Н 16 5	_ 14 4	П 8 14	У 9 3	С 12 11
Ь 0 9	И 3 10	О 19 18	С 17 8	К 20 17	? 0 19	Т 36 26
О 31 25	_ -1 -1	И 38 34	В 22 0	Т 21 32	_ -1 -1	_ -1 -1
_ -1 -1	С 34 0	Т 32 22	_ 26 31	К 0 40	П 24 30	_ -1 -1
А 39 21	_ -1 -1	С 40 24	К 0 36	О 33 38	_ -1 -1	! 41 44
_ -1 -1	В 42 41	_ 44 46	_ -1 -1	К 48 42	_ -1 -1	_ -1 -1
_ -1 -1						

Примечание. Значения числовых полей даны в десятичной системе счисления. Символ «_» обозначает пробел.