МЕЖДУВЕДОМСТВЕННЫЙ ГЕОФИЗИЧЕСКИЙ КОМИТЕТ
ПРИ ПРЕЗИДИУМЕ АН СССР

Т.Е. Кроли II, С.В. Ферронский

ВЛИЯНИЕ ИЗМЕНЕНИЯ КЛИМАТА НА ГИДРОЛОГИЮ КАСПИЙСКОГО МОРЯ

Вып. 2

Пространственное усреднение точечных метеорологических измерений

Москва 1990

УДК 551.521

Данная работа является второй из серии, посвященной результатам выполнения проекта "Влияние изменения климата на гидрологию Каспийского моря". Проектом намечается построение гидрологических моделей бассейна Каспийского моря и исследование влияния изменений климата, оцениваемых по сценариям моделей общей циркуляции атмосферы на гидрологические процессы бассейна.

В этом разделе описывается метод преобразования точечных измерений, которые получены на отдельных метеорологических станциях гидрологического бассейна, в пространственно усредненные для использования в гидрологических моделях стока.

Ответственный редактор:
Проф. Г.Н. Голубев

© Междуведомственный геофизический комитет АН СССР, М., 1990 г.

SOVIET GEOPHYSICAL COMMITTEE OF THE USSR ACADEMY OF SCIENCES

T.E. Croley II, S.V. Ferronsky

CLIMATE CHANGE IMPACTS ON THE HYDROLOGY OF THE CASPIAN SEA

Issue 2

Areal averaging of point meteorological measurements

Moscow 1990

This paper is the second in a series describing work on the project: "Climatic Change Impact upon the Hydrology of the Caspian Sea." The project anticipates construction of hydrological models of the Caspian Sea and study of climatic change impacts, taken from scenarios of atmospheric global circulation models, upon hydrological processes.

Here is described the method of converting point measurements, available from individual meteorological stations throughout the area containing a hydrological basin, to spatial averages for use with hydrological runoff models.

Editor:
Prof. G.N. Golubev

1. Введение

Для исследования влияния изменений климата на гидрологию Каспийского моря и соседних гидрологических бассейнов планируется построение гидрологических моделей суточного разрешения, описывающих сток и испарение. Данные модели являются точечными и используют суточные ряды таких метеорологических параметров, как средняя, минимальная и максимальная температуры воздуха, осадки, инсоляция, облачность, влажность и скорость ветра. При построении моделей используются наблюдательные данные со всех метеорологических станций в бассейне Каспийского моря. Эти данные подвергаются процедуре пространственного усреднения таким образом, что учитывалась переменная плотность сети станций во времени.

В данном применяется метод усреднения Тиссена, который является эффективным при наличии большого количества точек наблюдения. При его использовании среднесуточные метеорологические данные с сети станций обрабатываются по методу средневзвешенных площадей влияния станции. В конечном итоге получаем ряды усредненных по площади среднесуточных метеорологических данных для каждого гидрологического района бассейна Каспийского моря. Процедура преобразования является сложной, поскольку сеть станции не является статической. Станции добавляются, закрываются, меняют свое положение, временно выбрасываются из сети, когда они не могут предоставить данные в течении некоторых промежутков времени. Особая запись данных и используемый алгоритм определения "тиссеновых" площадей обеспечивает эффективное выполнение этой части работы. Все процедуры, которые здесь примеряются, написаны специально для использования на персональных компьютерах с ограниченной памятью и языком фортран. Так что при выполнении работы можно свободно переходить с одного компьютера на другой.

Процесс обработки данных состоит из следующих этапов: 1) за-

писи точечных измерений в отдельные файлы станций, 2) автоматической обработки основной базы данных, и 3) автоматической обработки усредненных данных. В разделе 2 приведена информация об исходных данных и требованиях к их использованию для бассейнов рек Урал и Эмба. В разделе 3 описан эффективный алгоритм определения весов Тиссена. Описание пакета программ и отдельные структуры алгоритма обработки данных даны в разделе 4.

2. Сбор данных

Базы гидрометеорологических данных, которые используются авторами для построения моделей гидрологических бассейнов Урала и Эмбы, а также самого Каспийского моря включают: среднесуточные величины осадков, минимальные и максимальные и среднесуточные величины температур воздуха, среднесуточные значения облачности, влажности воздуха и максимальные величины скорости ветра приблизительно с 258 станций. Среди них были отобраны 58 станций в бассейне реки Урал и 31 станция в бассейне реки Эмба со среднесуточными данными за период с 1955 по 1976 гг. Эти данные были получены во ВНИИГМИ-МЦД (МЦД-81).

Как отмечено во введении, метеорологическая сеть не является статической. Она меняется по мере того, как станции добавляются, смещаются из рассмотрения или временно не функционируют. Изменчивость метеорологической сети станций приводит к возникновению проблемы обработки данных. Процедуры, которые учитывают неизменную сеть, содержащую все используемые станции, и предусматривают экстраполяцию отсутствующих данных, вносят ошибки. В то же время процедуры, которые учитывают данные со всех станций, должны быть достаточно оперативными для обработки большой информации. Например, в бассейне реки Эмба было обработано 371225 станций-дней, содержащих информацию об осадках, минимальной и максимальной температурах с 31 станции за период с 1 января 1955 по 31 декабря 1976 (8036 дней). В бассейне реки Урал соответственно было использовано 693872 станций-дней за тот же

период времени с 58 станций. Анализ показал, что для бассейна Эмбы число различных конфигураций станций равняется 452. Однако поскольку некоторые из конфигураций станций повторяются, то фактически сеть менялась 5410 раз. Для бассейна реки Урал число таких изменений оказалось равным 6659.

3., Усреднение Тиссена

Тиссеново усреднение используется для преобразования точечных измерений в почти реальном масштабе времени в усредненные по площади величины для каждого бассейна. Оно является более точным по сравнению с простым арифметическим усреднением, поскольку учитывает площадь влияния каждой станции. Сеть наблюдения в почти реальном масштабе времени является слишком редкой для использования в достаточно сложных методах обработки данных, позволяющих определить непрерывное распределение соответствующих величин. Метод Тиссена состоит в определнии границы вокруг каждой станции, причем граница располагается посредине между данной и близлежащими станциями. Граница образует многоугольник вокруг каждой станции и площадь, ограниченная этим многоугольником, принимается за площадь влияния кажой станции. Отношение данной площади к площади всего подбассейна определяет долю площади подбассейна, которая представляется данной станцией. Вычисляя также отношения для всех станций, получаем ряд средневзвешанных величин, которые отражают относительное влияние каждой станции в бассейне. В сумме эти веса дают единицу и используются при суммировании измерений с соответствующих станций для получения усредненных по площади величин для всего подбассейна.

Построение многоугольников Тиссена вручную не представляется возможным при достаточно частом изменении сети станций. Поэтому для аналитического представления тиссеновой сети станций (Прин и Сибсон, 1978; Шамос, 1978) были разработаны вычислительные алгоритмы. Трудности, связанные с аналитической аппроксимацией бассейна приводят к большим ошибкам (Флорест, 1974). Другой подход

к определению величин отношений многоугольников к бассейнам может быть реализован посредством методов случайного выбора точек бассейна (Дискин, 1969, 1970; Григ, 1972; Ших, 1973; Ших и Хамрик, 1975). Однако эти подходы являются слишком грубыми и могут потребовать большого количества пробных точек для получения хорошего приближения. Требования к количесву производимых вычислений зависят от формы бассейна и его размеров по отношению к размерам всей карты. Для эффективного и точного определения весов Тиссена в данной работе используется алгоритм, разработанный Кроли и Хартман (1985) В нем используется факт выпуклости многоугольников Тиссена а также преимущества памяти современных компьютеров, которые позволяют использовать цифровые карты высокого разрешения.

Цифровая карта представляет собой двумерный массив, описывающий басейн. Он образуется посредотвом совмещения проекции карты с декартовой сеткой. Индексы массива представляют собой положение координат ячеек сетки на карте. Для каждого набора координат (индексов массива) массив содержит число, свидетельствующее о нахождении ячейки внутри или вне бассейна. Для оцифровки карты бассейна реки Эмбы (масштаб карты равнялся 1:500000) использовалась сторона ячейки в 1 км, а для оцифровки карты бассейна реки Урал сторона ячейки принята равной 2 км (Кроли и Ферронский, 1990). Разрешение на карте бассейна до одного квадратного километра в бассейне реки Эмба и до 4 км^2 в бассейне реки Урал приводит к точности, которая является достаточной при точности имеющихся карт. Практически для вычисления весов Тиссена перебирают все ячейки карты и определяют, какая из станций расположена ближе всего к ячейке путем сравнения расстояний. После нахождения ближайшей станции принадлежность каждой клетки бассейна определяется путем прочтения цифровой карты. Окончательно все точки, лежащие внутри каждого многоугольника и внутри бассейна, пересчитываются одновременно, а веса Тиссена определяются путем определения отношений этих величин к общей площади бассейна. Поскольку при этом рассматриваются все точки, то ошибка выборки

точек исключается. Однако, этот метод требует большого количества времени на вычисления, поскольку поиск ближайшей станции проводится для каждой отдельной станции на цифровой карте. Значительный выигрыш в вычислениях получается, если определять многоугольники по их соседям.

Многоугольники Тиссена и их пересечения с любой прямоугольной площадью карты (последняя также является многоугольником) будут выпуклыми областями, т.е. любой отрезок прямой линии, соединяющий две точки на границе области содержит все точки между этими двумя внутри области. Дискретное представление многоугольников Тиссена как набора точек, которые которые последовательно перенумерованы параллельно линии сетки, подобно параллелограмму, может быть расположено для получения проектов в любом из двух указанных направлений. Это представление может быть полностью определено по координатам граничных ячеек. В таком случае применима идея построения контуров (Дрисколл, 1969; Коттафава и Ло Моли, 1969; Хиап и Нинк, 1969; МакЛоин, 1974) для определения граничных ячеек тиссеновских многоугольников. Однако, свойство выпуклости многоугольников позволяет использовать более эффективный алгоритм и относительно более короткую процедуру. Он также использует структуру ячеек многоугольника и избегает интерполяцию, используемую в методе контуров.

Алгоритм проиллюстрирован простым примером, показанным на рис. 1. Граничные ячейки определяются следующим образом. Сначала определяется первая ячейка на границе многоугольника и затем проводится поиск вдоль границы для нахождения других граничных ячеек. Ячейка прилегающая к исходной ячейке в первоначально выбранном направлении должна находиться вне многоугольника. Исходный путь поиска выбирается произвольно вне многоугольника. На рис. 1 он выбран в направлении север-восток-юг-запад (СВЮЗ), приводящий к поиску по часовой стрелке. Для того чтобы найти другую ячейку на границе многоугольника, производим поиск в указаных направлениях до тех пор, пока следующая наиденная ячейка

многоугольника не окажется внутри многоугольника. На рис. 1 поиск продолжается на север от ячейки 1 (исходной ячейки) и следующая найденная ячейка лежит вне многоугольника. Таким образом, далее проиизводится поиск на восток от этой ячейки до тех пор, пока не найдена ячейка 2, которая оказывается внутри многоугольника. Затем поиск следующей ячейки начинается снова с ячейки 2 в том же самом порядке. Если при поиске в предложенном порядке мы возвращаемся к той же точке с которой начали (вырожденный случай), произведем одно обращение направления поиска и начнем сначала. Например в ячейке 4 на рис. 1 поиск в направлении СВЮЗ привел к возвращению в точку 4. Тогда направление поиска было изменено на восток-юг-запад-север (ВЮЗС), и поиск был возобновлен. Поиск продолжался до тех пор пока граница не была определена полностью, что соответствует возвращению к первым двум ячейкам в их исходном порядке или после изменения направления поиска более чем пять раз. Поскольку исходная точка может лежать на двух границах многоугольника, то желательно закончить определение границы, как только исходная ячейка будет найдена снова. Для каждой исследуемой ячейки определяется ее местонахождение внутри многоугольника или вне его по расстоянию до ближайшей станции. Если эта станция является той, для которой строится многоугольник, то ячейка лежит внутри многоугольника. Сокращение объема вычислений появляется только при исследовании ячеек, ограничивающих границу многоугольника, но не для каждой ячейки многоугольника. Заметим, что рис. 1 только иллюстрирует алгоритм определения границ. Поскольку здесь показано слишком мало ячеек в многоугольнике, преимущества процедуры не видны.

Определение исходной ячейки требует специального обсуждения, нужна гарантия того, что в соответствии с описанной процедурой поиск будет работать правильно. В противном случае процедура поиска приведет к положению точек, которые лежат вне многоугольника, а не на его границе. Первоначальная точка может быть легко найдена в случае, если станция многоугольника Тиссена находится внутри границ цифровой карты. Начинать надо с проверки

положения станции, которая должна находиться внутри многоугольника. Далее, двигаясь в направлении на север и проверяя, что мы находимся внутри многоугольника, остановимся, когда найдем первую точку вне многоугольника или карты. Тогда последняя ячейка, найденная внутри многоугольника и карты, будет первой ячейкой, граничащей на севере с внешней ячейкой. Эта внешняя ячейка может быть использована как исходная ячейка для построения границы многоугольника.

Если станция в многоугольнике Тиссена находится вне цифровой карты, то должна быть найдена исходная точка (если она существует), которая лежит на краю части Тиссенового многоугольника, принадлежащего карте. Причем с севера должна быть ячейка, не принадлежащая этому многоугольнику. В этом случае поиск приведет к нахождению границы многоугольника также как и ранее. Оценим восемь секторов, лежащих вне карты. 1. Если станция расположена на северовостоке вне цифровой карты, то поиск следует начать с северозападного угла цифровой карты и вести его на восток, пока не будет обнаружена ячейка, принадлежащая многоугольнику. Если на северном краю карты ячейки не найдено, то работу следует начать сначала с северовосточного края карты и вести поиск на юг, разыскивая первую точку. Если опять ничего не удалось найти, то многоугольник не имеет пересечения с цифровой картой и для этой станции тиссеновский вес равняется нулю. 2. Если станция расположена на востоке от цифровой карты, то поиск следует начать с северовосточного угла и вести его на юг. 3. Если станция расположена на юговостоке карты, то поиск следует начать с северовосточного края карты и вести его в сторону юга. Далее продолжить поиск, начиная с юговосточного края карты и вести его на запад, если необходимо. 4. Если станция расположена на юге карты, то поиск следует начать произвольно с югозападного угла и вести его на восток, пока не будет найдена ячейка внутри многоугольника. Затем следует вести поиск на север, пока не будет найдена ячейка вне многоугольника или вне карты. Ячейка, найденная перед этим будет считаться первой. 5. Если станция расположена на

юговостоке карты, поиск следует начать с северозападного угла и вести его на юг. Затем начать с юговосточного угла и вести поиск на восток, если необходимо. 6. Если станция расположена на западе карты, поиск следует начать с северозападного угла и вести его на юг. 7. Если станция расположена на северозападе карты, поиск следует начать с северозападного угла и вести его на восток. Затем снова начать с северозападного угла и вести поиск на юг, если необходимо. 8. Если станция расположена на севере карты, поиск следует начать с северозападного угла и вести его на восток.

В каждом из этих восьми случаев описанная процедура определит ячейку, которая принадлежит тиссенову многоугольнику карты (если такая ячейка существует) с прилегающей с севера ячейкой, которая либо расположена вне карты, либо вне многоугольника. Тогда с помощью алгоритма можно использовать эту исходную ячейку в качестве края многоугольника, расположенного на карте.

Вместо того, чтобы находить координаты всех границ многоугольника с последующей интерпретацией их для систематического использования, процедура непосредственно использует свойство выпуклости каждого многоугольника, как описано выше, посредством описания многоугольника в терминах левых и правых границ для каждого ряда ячеек внутри многоугольника. Первоначально размеры многоугольника определяются как нулевые путем определения верхнего и бокового рядов, а также левой и правой абсцисс, определяющих пределы пересечения. Исходная ячейка на краю или внутри многоугольника и на краю карты определяется затем, как описано раньше. Если такая точка не существует, то многоугольник остается пустым и станция имеет тиссенов вес равный нулю. В противном случае границы многоугольника расширяются для того, чтобы включить исходную точку. По мере того, как определяется каждая новая точка на краю многоугольника, последний расширяется для ее включения. После того как многоугольник описан, новая информация заносится в таблицу. Ряды этой таблицы соответствуют станциям измерения, а колонки описывают их положение внутри или вне бассейна. Для

каждого ряда внутри многоугольника цифровая карта читается внутри левой и правой границ многоугольника с целью определения положения ячеек и затем определяется положение в счетной таблице. По мере того, как окно многоугольника выражается в виде ряда значений абсциссы для каждого ряда карты, доступ к карте облегчается. Абсциссы многоугольников записываются в таблицу с целью освобождения памяти машины. Поскольку счетная таблица может быть записана непосредственно по номерам станций и бассейнов, счет может проводиться непосредственно без проверки того, какое приращение должно быть использовано. Такая процедура приведет к существенному уменьшению вычислений. После того как все многоугольники Тиссена вычислены, его веса вычисляются по количеству ячеек (по счетной таблице) для каждой станции относительно общего числа ячеек. Конечно, поскольку веса Тиссена в сумме дают единицу, то требуются границы для всех многоугольников, кроме одного.

Указанный алгоритм описан достаточно подробно в статье Кроли и Хартман (1985) для использования на любом компьютере. Хотя эффективность вычислений зависит от конкретной конфигурации сети станции, пример вычислений для бассейна реки Урал, показанный на рис. 2 потребовал только 32 секунды для вычисления весов на 12 Мгц компьютере PC-AT (80286 с копроцессором 80287). Это как минимум в десять раз быстрее, по сравнению с методами, которые используют определение ближайшей станции для каждой ячейки цифровой карты. На рис. 2 только 8.2% из 170128 ячеек на карте были исследованы для определения 4.1% ячеек, которые образуют границу многоугольника. По мере увеличения числа станций в приложениях достигаем уровня, когда число ячеек, используемое для определения границ многоугольника, совпадает с числом ячеек на карте. В этом случае систематическое исследование всех ячеек займет такое же время, как и установление границ всех многоугольников. Кроли и Хартман (1985) нашли, что в среднем разрешение, равное примерно 80 ячейкам на станцию, потребует примерно такого же времени, как и исследование

всех точек карты. В случае бассейна реки Урал, это означает, что алгоритм дает преимущество даже в случае, если сеть станций содержит около 2100 станций.

6. Построение базы данных

Мы применяем алгоритмы тиссенового усреднения для последующего использования в наборах данных. Прежде всего мы разделили данные на части по одной для каждого года всей записи. Затем обработали данные за каждый год. Это позволило нам использовать их в иттеративном режиме. Поскольку переработка данных за период с 1956 по 1976 год для бассейна реки Урал занимает порядка 2-3 дней, это позволяет нам возобновить вычисления, если они были прерваны, без потери предыдущих вычислений.

С целью рационализации вычисления тиссеновых площадей и исключения повторных вычислений пакет обработки данных задуман так, чтобы при повторяющаейся конфигурации станции можно было воспользоваться результатами предыдущих вычислений. Пакет программ был создан для достижения трех целей. Во-первых, он должен работать на персональных компьютерах. Во-вторых, должен работать в автоматическом режиме и исправлять ошибки. Когда встречается ошибка, она должна быть распознана и сделано предложение о ее исправлении, когда это возможно. Все подпрограммы объединены друг с другом на диске посредством головной программы. Эта программа управляет всеми данными и информацией о выполнении каждой подпрограммы. Таким образом, ни один из модулей не может быть запущен до тех пор, пока предыдущие модули не были завершены благополучно. В третьих, пакет должен быть простым в обращении, так чтобы не приходилось изменять программы между применениями. Пакет обработки данных состоит из двух основных частей: 1) ввода точечных измерений и их добавления к главной базе данных точечных измерений, и 2) изменения базы данных после площадного усреднения.

4.1 Управление базой данных точечных измерений

Для каждой станции должен создаваться файл, содержащий новые и измененнные данные каждый раз, когда изменяется база данных. Первая запись в файле, содержащем точечные измерения, имеет название станции и ее положение, определяемое широтой и долготой. Кроме того здесь, например, используются среднесуточные величины осадков и среднесуточные минимальные и максимальные температуры и даты. Записи не обязательно должны быть сделаны в хронологическом порядке или непрерывными. Эти файлы станций подвергаются обработке по программе нахождения ошибок для проверки размерности, интервала изменения и непротиворечивости. Все ошибки детально анализируются, так что они могут быть легко исправлены в программе проверки ошибок при повторном запуске. Остальные программы заканчиваются сообщениями об ошибках, если программа отыскания ошибок не проверила все файлы станций и если имеются ошибки.

База данных точечных измерений состоит из дважды объединенных листов, содержащихся в файлах прямого доступа. Лист состоит из блоков по одному для каждого дня. Первая запись (заголовок) в блоке содержит дату, флаг начала, указатель (соединяющий) на следующий логический заголовок (в хронологическом порядке) блока, и указатель на первую (логическую) запись станции в блоке. Оставшиеся записи в блоке представляют все точечные измерения, проведенные в этот день. Каждая запись в блоке содержит идентификатор станции (объединенную двоичную запись целых чисел, обозначающих широту и долготу), данные и переход к следующей логической записи. Таким образом, когда оказывается, что идентификатор перехода записи совпадает с идентификатором для всего блока, то определяется конец блока (конец данных для рассматриваемых суток). Флаг обработки указывает на то, проведена ли для данного блока обработка данных или нет и добавлен ли он к пространственно усредненной базе данных. Он принимает значение "новый", когда данные в блоке добавлены или изменены, и значение "старый", когда блок был использован для вычисления пространственных средних для следующей базы данных. Дан-

ные записи в блоке записаны в порядке, определяемом их широтой и долготой. Это упрощает поиск записи при исправлении и позволяет идентифицировать каждую комбинацию станций единственным способом. Порядок не является существенным для целей идентификации сети до тех пор, пока порядок в целом сохраняется. Максимальная длина файла в объединенной базе с записями длиной по 4 байта равняется 4294967296 записям, что достаточно для любых практических целей (в случае 100 работающих станций ежедневно или 101 записи в блоке, это позволяет хранить информацию за 116425 лет).

Новые данные легко добавляются к физическому концу базы данных точечных измерений. Они находятся при чтении отдельных файлов станций и соответствующие связи изменяются при включении новых данных в логическом порядке. Если получаемая база данных не содержит данных для какой-либо станции за какой-то день, то связи изменяются так, чтобы обойти соответствующую запись и таким образом исключить запись из логической последовательности. Поскольку логически связанные записи могут быть физически не последовательными, операции на диске могут значительно замедлять преобразование данных в базу данных точечных измерений. Реорганизация базы с тем, чтобы физическая последовательность записей в файле стала той же что и логическая, приводит к последовательному доступу логически связанных записей в процессе работы. В то же время перезапись восстанавливает мертвые области, оставшиеся от отсутствующих записей. Последовательный доступ дискового контролера исключает лишние "дергания" головки на диске в случае непоследовательной упаковки записей. Альтернативой листингу связей на диске является листинг связей в памяти, если конечно имеется достаточная память. Для малых компьютеров это невозможно. Использование РАМ дисков устраняет проблемы, связанные с износом головок и задержками ввода и вывода информации.

4.2 Описание базы данных после площадного усреднения

Отдельный файл прямого доступа записывается для усредненных по

площади данных в каждом из интересующих нас бассейнов. Каждый блок суточных записей в базе точечных данных соответствует суточной записи в файле для бассейна. Эффективность в редактировании площадных баз данных обеспечивается следующими условиями: 1) обрабатываются только блоки (данные) с новыми или измененными данными, 2) тиссеновы веса для повторяющихся конфигураций станций отыскиваются, а не вычисляются и 3) тиссеновы веса для новых сетей вычисляются эффективно. База данных точечных измерений сканируется путем прочтения каждого блока. Если блок имеет флаг "старый", он пропускается и читается следующий блок (по оглавлению), исключая лишний счет и ввод данных. Если блок имеет флаг "новый", то записи в блоке вводятся и обрабатываются.

В каждом типе записи (осадки, минимальные или максимальные температуры воздуха) работающие станции используются для построения идентификатора сети путем объединения идентификаторов станций в порядке, в котором они фигурируют в листинге. Поскольку порядок записей самосогласован, любая группа станций будет иметь тот же порядок, в котором гарантирована единственность идентификации каждой сети по мере ее вычисления и записи, так что она в дальнейшем может быть вызвана по имени вместо повторного вычисления. По мере вычисления и использования всех тиссеновых весов для различных сетей, они записываются в отдельной базе индексов в виде таблицы. В свою очередь индексы в соответствии со специальными кодами вычисляются по сети идентификаторов. Это обеспечивает быстрый поиск тиссеновых весов для каждой сети, избегая просмотр длинных записей в линейном поиске нужной сети. По мере использования базы данных, число различных сетей возрастает, площадное усреднение становится более эффективным, поскольку многие сети повторяются и расчитываются лишь новые конфигурации. Конечно это зависит от того, как сильно сеть станций меняется от одного вычисления к другому. По мере достижения максимально возможного числа сетей (что определяется возможной величиной диска), старые сети стираются и заменяются новыми.

В силу требований к памяти, где хранятся таблицы весов, цифровые карты и промежуточные переменные тиссеновых многоугольников, не всегда удается хранить цифровую карту в памяти машины во время вычислений. Она может быть записана на диске и прочитываться ряд за рядом когда это требуется во время вычислении. Поэтому обрабатывается лишь несколько станций и углы многоугольников хранятся в памяти до чтения карты и определения пересечений бассейна с многоугольником. Вычисления могут быть уменьшены при учете порядка обработки станций. Поскольку для идентификации сети порядок станций в суточном блоке записей не имеет значения (по мере того как он является самосогласованным), станции сначала вызываются по широте и затем по долготе. Так что физически близко расположенные станции записываются максимально близко. Часто многоугольники этих станций расположены вместе в районе цифровой карты. Вся карта не должна вводиться для определения пересечении бассейна с многоугольниками, поэтому возникает дальнейшее сокращение вычислений, что уменьшает число обращений типа ввод-вывод. Альтернативно карта может быть определена в терминах достаточно больших размеров ячеек (2км на 2км для Урала и 1км на 1 км для бассейна реки Эмба). Так что карта является достаточно малой, чтобы помещаться в памяти. Эта процедура является быстрой и она используется.

В процессе изменения пространственно усредненной базы данных каждый блок в базе данных точечных измерений отмечается как "старый". Следовательно он не обрабатывается в последующих вычисленях до тех пор, пока не добавляются новые или другие данные.

5. Програмное обеспечение

В качестве примера приведем описание использования изложенных выше методов в приложении к бассейнам рек Урала и Эмбы. Процедура использует программы, записанные в Приложении для персональных компьютеров. Сначала 22-летние суточные данные за период с 1956

по 1976 с 58 мететорологических станций в бассейне Урала и с 31 станции в бассейне реки Эмба записаны в 22 файлах по одному для каждого года записи. Каждый файл имеет шифр 19??.ZIP. Например данные за 1955 год с 58 +31 станций записаны в файле 1955.ZIP. Эти файлы располагаются в главной директории компьютера на диске D. Затем, были созданы файлы, содержащие все идентификаторы станций и положения для этих двух бассейнов под шифрами STNLIST.URA и STNLST.EMB. Были также созданы исходные версии всей базы данных площаных усреднений с шифрами URA01.PRV и EMB01.PRV и малые информационные файлы BSNNM (по одному из бассейна реки Урал и из бассейна Эмбы) AREA.URA, AREA.EMB, LENGTH.URA, LENGTH.EMB. Эти маленькие файлы описаны в комментариях к программам в Приложениях. Процедура в Приложении A OVERNITE.BAT использовалась для работы с программой REDUCE.BAT в приложении B сначала для Урала и затем в Приложении к Эмбе. Все програмное обеспечение находится на диске D в директории PROGRAMS.

REDUCE.BAT в Приложении B используется для построения средних по площади метеорологических данных по группам отдельных станций за каждый год исторической записи. Эта процедура используется для работы с программами, которые используют методику, описанную ранее в разделах 3 и 4, повторяя работу программ последовательно за каждый год. Она начинается с построения файла "DATALIST", который описывает шифры файлов всех наборов метеорологических данных. Затем для каждого года исторической записи эта процедура выбирает суточные данные для всех станций, объявляя этот год в файле и используя программу PREPYEAR.FOR, в приложении C. Затем эта программа проверяет все файлы метеорологических данных с целью проверки очевидных ошибок и используя программу ERRCHTCK.N14, в Приложении D. Затем по программе PROVSNAL.N14 преобразуются все метеорологические данные в базе данных точечных измерений, см. Приложение E. Далее преобразуется база данных точечных измерений, так что последовательный доступ также обеспечивается в логической последовательности по программе REPACK.N14, Приложение F. Окончательно, процедура REDUCE.BAT редактирует базу

пространственно осредненных данных по базе данных точечных измерений с использованием тиссеновых весов, вычисляемых по мере необходимости и используя либо программу DISAVURA.N14 (для бассейна Урала) либо DISAVEMB.N14 (для бассейна Эмбы). Обе эти программы были созданы ранее путем использования главного листинга DISAVMET.SCR, в Приложении G, в процессе работы программы PRDSAVMT.N14, в Приложении H.

6. Заключение

В таблице 1 для бассейна реки Урал и Таблице 2 для бассейна реки Эмбы иллюстрируется наличие доступной информации с 58 или 31 метеорологических станций, которая включает среднесуточные максимальные и минимальные температуры и величины осадков с 1955 по 1976 гг. Длины записи в Таблицах 1 и 2 вычислены для каждого года записи путем деления числа дней фактически доступных данных на каждой станции на число дней в году (чтобы получить величину станций-лет для каждой станции) и суммируя их по всем 58 или 31 станциям, соответственно. Легко увидеть, что 1955-59 гг. представляют период более низких плотностей записи для обоих бассейнов и после 1959 оба бассейна имеют более или менее стабильную сеть станций (возможное исключение составляет 1966 год). Периоды 1960-65 в бассейне реки Урал и 1971-1976 в бассейне реки Эмба также представляют собой периоды функционирования более плотной сети станций. В период с 1966 по 1970 год оба бассейна имели более низкую плотность сети станций по сравнению с годами до и после этого периода.

7. Литература

Коттафава и Ле Моли, 1969

Cottafava, G., and Le Moli, G., 1969. Automatic contour map. Communications of the ACM, Vol. 12, pp. 386-391.

Кроли и Ферронский, 1990

Т.Е. Кроли II, С.В. Ферронский. Влияние изменения климата на гидрологию Каспийского моря. Вып. 1. Построение цифровых карт водосборных бассейнов. М, 1990, с. 1-43.

Кроли и Хартман, 1985

Croley, T. E., II, and Hartmann, H. C., 1985. Resolving Thiessen polygons. Journal of Hydrology Vol. 76, pp. 363-379.

Дэйхоф, 1963

Dayhoff, M. O., 1963. A contour-map program for X-ray crystallography. Communications of the ACM, Vol. 6, pp. 620-622.

Дискин, 1969

Diskin, M. H., 1969. Thiessen coefficients by a Monte Carlo procedure. Journal of Hydrology, Vol. 8, pp. 323-335.

Дискин, 1970

Diskin, M. H., 1970. On the computer evaluation of Thiessen weights. Journal of Hydrology, Vol. 11, pp 69-78.

Форрест, 1974

Forrest, A. R., 1974. Computational geometry - achievements and problems. In: R. E. Barnhill and R. F. Reisenfeld (Editors),

Грин и Сибсон, 1978

Green, P. J., and Sibson, R., 1978. Computing Dirichlet tessellations in the plane. Computer Journal, Vol. 21, No. 2, pp. 167.

Григ, 1972

Grigg, A. O., 1972. A program for calculating Thiessen average rainfall. Transportation and Road Research laboratory, Crowthorne, Berkshire, TRRL Report LR 470, 20 pp.

Хип и Пинк, 1969

Heap, B. R., and Pink, M. G., 1969. Three contouring algorithms. National Physics Laboratory, Tedington, DNAM Report 81.

МакЛэин, 1974

McLain, D. H., 1974. Drawing contours from arbitrary data

points. Computer Journal, Vol. 17, pp. 318-324.

Шамос, 1978

Shamos, M. I., 1978. Computational geometry. Ph. D. Thesis, Department of Computer Science, Yale University, New Haven, Connecticut.

Ших, 1973

Shih, S. F., 1973. Modified Monte Carlo application to ground-water movement - the simultaneity procedures. Water Resources Research, Vol. 9, pp. 1029-1038.

Ших и Хамрик, 1975

Shih, S. F., and Hamrick, R. L., 1975. A modified Monte Carlo technique to compute Thiessen coefficients. Journal of Hydrology, Vol. 27, pp. 339-356.

Таблица 1. Количество станций-лет в записях метеорологических параметров для бассейна реки Урал

Год	Станций-лет (минимальная температура)	Станций-лет (максимальная температура)	Станций-лет (осадки)
1955	18.7	18.6	18.8
1956	20.2	20.2	20.2
1957	21.0	20.9	21.0
1958	21.8	21.8	21.8
1959	26.6	26.8	27.4
1960	33.3	32.4	34.4
1961	31.6	29.7	31.7
1962	33.1	32.2	34.4
1963	34.2	33.6	34.3
1964	32.6	32.4	32.5
1965	33.8	34.0	34.0
1966	26.7	26.7	26.8
1967	28.7	28.5	28.7
1968	29.5	29.4	29.6
1969	29.4	29.4	29.2
1970	29.3	29.5	29.5
1971	30.8	30.8	30.8
1972	30.8	30.8	30.8
1973	30.6	30.6	30.6
1974	29.8	29.8	29.8
1975	30.3	30.4	30.3
1976	30.6	30.6	30.5

Таблица 2. Количество станций-лет в записях метеорологических параметров для бассейна реки Эмба

Год	Станций-лет (минимальная температура)	Станций-лет (максимальная температура)	Станций-лет (осадки)
1955	11.9	11.8	12.0
1956	12.9	12.9	12.9
1957	13.6	13.4	13.6
1958	14.1	14.1	14.1
1959	14.9	14.9	15.4
1960	15.7	15.6	16.7
1961	15.6	14.0	15.6
1962	15.6	15.9	16.7
1963	16.5	15.9	16.5
1964	15.1	15.0	15.1
1965	16.4	16.6	16.6
1966	13.8	13.8	13.8
1967	14.8	14.8	14.8
1968	15.6	15.6	15.8
1969	15.5	15.5	15.4
1970	15.7	15.8	15.8
1971	16.9	16.9	16.9
1972	16.9	16.9	16.9
1973	17.0	17.0	17.0
1974	16.9	16.9	16.9
1975	16.4	16.4	16.4
1976	16.6	16.6	16.6

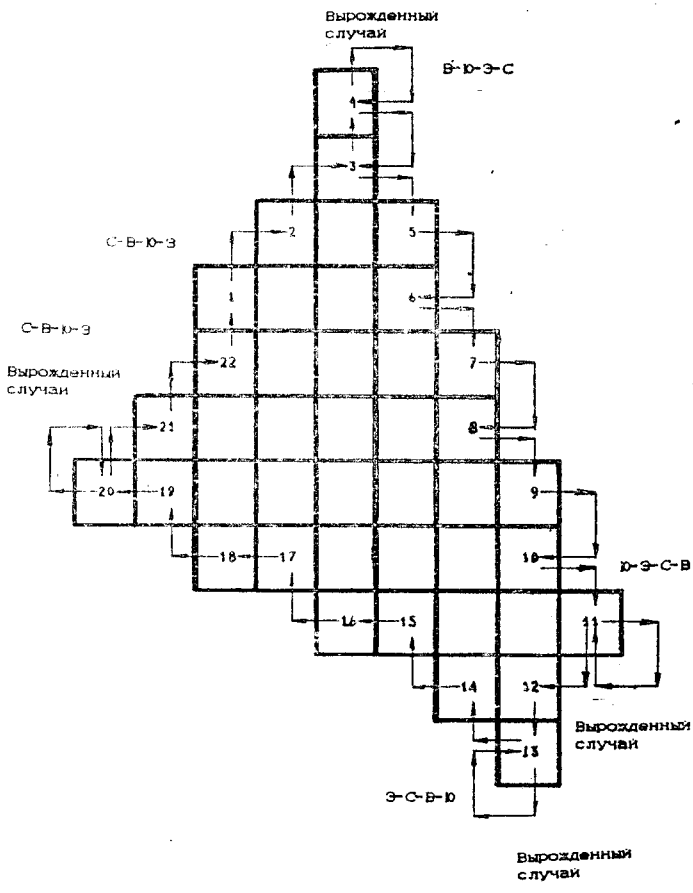


Рис. 1. Иллюстрация работы алгоритма определения границы тиссеновых многоугольников

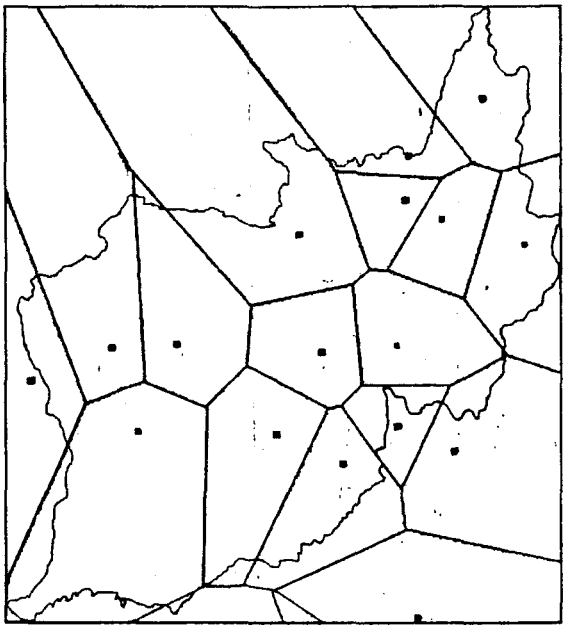


Рис. 2. Цифровая карта бассейна реки Урал и пример тиссеновой сети

Приложение А

```
@ECHO OFF
REM OVERNITE.BAT by TECII, 21 June 1990, TO run REDUCE.BAT for
REM both the Ural and the Embe river basins.
REM
SET D=D
SET P=D
SET N=URAL
CALL %P%:ROGRAMSEDUCE
IF EXIST ERROR.INT GOTO ERROR
SET N=EMBE
CALL %P%:ROGRAMSEDUCE
IF EXIST ERROR.INT GOTO ERROR
GOTO END
:ERROR
ECHO Error reported, OVERNITE job aborted!
:END
```

Приложение Б

```
@ECHO OFF
REM REDUCE.BAT by TECII, 21 June 1990, TO construct areally-ave-
REM raged daily data meteorological DATA from groups of individual
REM station files, one group for each year. Runs programs
REM ERRCHECK, PROVSNAL, REPACK, and DISAV%%% in stages so that IF
REM interrupted, NOT all is lost. Restart requires eliminating
REM 19??.ZIP files already processed.
REM
%D%:
CD N%
IF EXIST ERROR.INT DEL ERROR.INT
DIR 19??.ZIP | SORT > DATALIST
:LOOP
 %P%:ROGRAMSREPYEAR
 DIR MR*.DAT | SORT > PROVSNAL.LIS
 %P%:ROGRAMSRRCHECK
 IF ERRORLEVEL 1 GOTO ERROR
 %P%:ROGRAMSROVSNAL
 IF ERRORLEVEL 1 GOTO ERROR
 %P%:ROGRAMSEPACK
 IF ERRORLEVEL 1 GOTO ERROR
 %P%:ROGRAMSISAV%N%
 IF ERRORLEVEL 1 GOTO ERROR
 DEL *.DAT
 IF EXIST DATALIST GOTO LOOP
GOTO END
:ERROR
ECHO Error reported, DATA reduction aborted'
COPY BSNNM. ERROR.INT
:END
```

Приложение 1

```
      PROGRAM PrepareYearsData
      IMPLICIT NONE
      INTEGER I
      CHARACTER*12 L
      LOGICAL NothingLeft

      OPEN(UNIT = 1, FILE = 'DATALIST', STATUS = 'OLD')
      OPEN(UNIT = 2, FILE = 'DATALIST.NEW', STATUS = 'NEW')
      DO I = 1, *
       READ(1, 2) L
       WRITE(2,2) L
    2  FORMAT(A12)
    1 CONTINUE
      READ(1, 2) L
      L(9:9) = ' '
      CALL SYSTEM('PKUNZIP ' // L(1:4))
      CALL SYSTEM('RENAME *.' // L(3:4) // ' *.DAT')
      NothingLeft = .TRUE.
    3 CONTINUE
       READ(1, 2, END = 4) L
       WRITE(2,2) L
       NothingLeft = .FALSE.
       GOTO 3
    4 CONTINUE
      CLOSE (1)
      CLOSE (2)
      CALL SYSTEM('DEL DATALIST')
      CALL SYSTEM('REN DATALIST.NEW DATALIST')
      IF (NothingLeft) CALL SYSTEM('DEL DATALIST')
      END
```

Приложение D

```
      PROGRAM ERRorCHECK
c Updated by TEC II 28sep88, 26oct88 (for PC)
c This program is for use in procedure FORECAST.BAT. It checks all
c provisional met. files for obvious errors.  Messages are printed
c as errors are found. All data is tested and, at the end, if any
c errors have been found, the procedure terminates with  an  error
c condition.
c This program reads the file: BSNNM. and updates it; all forecast
c programs do the same with the exception of optional modules.
c This file is not to be disturbed by the user as it exists only for
c the purpose of each program keeping track of what other programs
c have been run as well as to reference specific basin information.
c It is a formatted direct access file with 13-byte records (the
c last two bytes of each record are a carriage return & line feed)
c and its contents are as follows:
c
c       1st record:   (FORMAT: A11)
c
c         Name  -  basin name, left adjusted, in all upper case.
c
c       2nd record:   (FORMAT: 2I3, A4, 1X)
c
c # # switch
c | | |     |
c | | |____|____lake precipitation computation switch:
c | |            "BASN" => estimate over-the-lake precipitation
c | |                      with land depths
c | |            "LAKE" => estimate over-the-lake precipitation di-
c | |                  rectly from nearby gages (subbasin "0" file)
c | |_____program designator number; each program has its own and
c |            places it in BSNNM. so programs can check for proper
c |            sequence
c |_________number of subbasins in basin
c
```

```
c The remaining records are filled in during a forecast session and
c are rewritten by the programs only.  They have no meaning after
c the session is over. Each record is a date (day, month, year) in
c the format 2I3, I5.

c 3rd through 9th records:   (FORMAT) 2I3, I5)
c
c 3rd record:  base date (see PROVSNAL.NI4 and REPACK.NI4)
c 4th record:  provisional date and data
c 5th record:  forecast and date
c 6th record:  plot start date
c 7th record:  date of earliest new data since WATOUT.NI4 last run
c 8th record:  date of earliest new data since LUMPSTOR.NI4 last run
c 9th record:  date of earliest new data since CMOFPOSR.NI4 last run
c
c This program reads the list of files to check as PROVSNAL.LIS;
c this contains the full file specification of each file to be
c opened and read by this program.  Each record of PROVSNAL.LIS
c should contain one file specification (filename) left-adjusted
c in the record, not to exceed 80 bytes.
c
c
c THIS PROGRAM MUST BE COMPILED TO DEFAULT TO 2-BYTE INTEGERS AND
c CONSTANTS!  SELECT THE "/T" SWITCH ON THE LAHEY FORTRAN COMPILER!
c
      IMPLICIT REAL (A - H, J - Z)
      IMPLICIT INTEGER (I)

      REAL Lat, Long
      INTEGER Tmax, Tmin, Precip
      CHARACTER*80 M$, Name$*80, File_Id*13, Sta_ID_No*15, Lake*10
      CHARACTER*9 Over_The_Lake*4, CrLf*2, Bn$*3
      COMMON / LakNam / Bn$

      CrLf = CHAR(13) // CHAR(10)
```

```
      OPEN(UNIT = 1, FILE = 'BOMAN', [illegible] = [illegible]
     + ACCESS = 'DIRECT' RECL = 14 [illegible] [illegible] )
      READ(1, FMT = 1000, [illegible] = 1) [illegible]
 1000 FORMAT(A10)
      Bo$ = Lake(1:3)
      READ(1, FMT = 2000, [illegible] = 2) [illegible] [illegible]
 2000 FORMAT([illegible]) [illegible]
      READ(1, FMT = 3000, [illegible] = 3) [illegible]
 3000 FORMAT([illegible])
      CLOSE (1)
      CALL Boma [illegible] MASTER [illegible]
      CALL Date[illegible] [illegible]
      iNoFiles   = 0
      iErr_Total = 0
      OPEN(UNIT = 1, FILE = 'PROV[illegible]', [illegible] = 'OLD', [illegible]
  222 READ(1, 1002, END = 999) Name$
 1002 FORMAT(A80)
      I = iStrLen(Name$)
      IF (I .EQ. 0 .OR. Name$(1:1) .NE. 'M') GOTO 222   ! skip
      Name$(9:9) = '.'
      File_ID = Name$(1:12)
      IF (File_ID .EQ. 'M0000000.PRV') GOTO 222       ! skip
      iErr = 0
      OPEN(UNIT = 2, FILE = Name$, STATUS = 'OLD')
      iNoFiles = iNoFiles + 1
      READ(2, 1002) M$
      I = iStrLen(M$)
      CALL Header_Format_Check(I, M$, File_ID, iErr)
      READ(M$(1:28), 1004, ERR = 20) Sta_ID_No, Lat, Long
 1004 FORMAT(1X, A7, 2F10.0)
   20 CALL Station_Check(Sta_ID_No, File_ID, iErr)
      CALL Lat_Long_Check(Lake, Lat, Long, File_ID, iErr)
      CALL Lat_Long_Compare(Lat, Long, File_ID, iNoFiles, iErr)
      CALL LatLong_Chk(Lat, Long, File_ID, Sta_ID_No, iErr)
```

```
      iLine = 1
111   iLine = iLine + 1
      READ(2, 1002, END = 99) M$
      I = LEN(M$)
      CALL Car. Format Check(I, M$, iLine, File_ID, iErr, iData_Err)
      READ(M$(1:23), 1006, ERR = 30) ID, iM, iY, Tmax, Tmin, Precip
1006  FORMAT(2(I3, I5, F14))
c
c If there are no problems with the format of the data, check the
c date.
c
 30   IF (iData_Err .EQ. 0) CALL Date_Check(ID, iM, iY, iYc, iLast_Date,
     + iLine, File_ID, iErr)
      IF (File_ID(1:2) .EQ. 'MS') THEN
      CALL US Data Check(Tmax, Tmin, Precip, iLine, File_ID, iErr)
      ELSE
      CALL Can.Data Check(Tmax, Tmin, Precip, iLine, File_ID, iErr)
      ENDIF
      GOTO 111
 99   CLOSE (2)
      iErr_Total = iErr_Total + iErr
      PRINT 1008, File_ID, iErr
1008  FORMAT(1X, 'File ', A13, ' has', I6, ' errors.', /)
      GOTO 222
999   CLOSE (1)
      IF (iErr_Total .EQ. 0) GOTO 333
998   OPEN(UNIT = 1, FILE = 'BSNNM.', STATUS = 'OLD', ACCESS =
     + 'DIRECT', RECL = 13, FORM = 'FORMATTED')
      WRITE(1, FMT = 2000, REC = 2) iNSB, 0, Over_The_Lake, CrLf
      CLOSE (1)
      CALL EXIT(1)

334   WRITE(6, 1501)
1501  FORMAT(1X, 'PROVSNAL.LIS must be present for processing by',
     + ' ERRCHECK.')
```

```
      GOTO 998
333 OPEN(UNIT = 1, FILE = 'BSNNM.', STATUS = 'OLD', ACCESS
     + = 'DIRECT', RECL = 13, FORM = 'FORMATTED')
c
c     "1" = designator for this program.
c
      WRITE(1, FMT = 2000, REC = 2) iNSB, 1, Over_The_Lake, CrLf
      CLOSE (1)
      END

      SUBROUTINE header_format_check(i, m$, file_id, ierr)
c
c     This routine checks that the provisional met. file header has
c     the proper line length and that there are entries for station
c     identification number, latitude, and longitude.
c
      IMPLICIT REAL (a-h,j-z)
      IMPLICIT INTEGER (i)
      CHARACTER*(*) m$, file_id*13

      IF(i .GE. 28) GOTO 10    !header must be at least 28 spaces to
      PRINT 1001, file_id      !accomodate 1X,A7,2F10.0 format.
1001 FORMAT(1x,'File',a13,'has a header of insufficient length.')
      ierr = ierr + 1
 10 IF(m$(1:1) .EQ. ' ') GOTO 11
      PRINT 1002, file_id
1002 FORMAT(1x,'File',a13,'has an incorrectly formatted header.')
      ierr = ierr + 1
 11 IF(m$(2:8) .NE. '        ') GOTO 12
      PRINT 1003, file_id
1003 FORMAT(1x,'File',a13,'has no station identification number.')
      ierr = ierr + 1
 12 IF(m$(9:18) .NE. '           ') GOTO 13
      PRINT 1004, file_id
1004 FORMAT(1x, 'File ', a13, ' has no latitude.')
```

```
      ierr = ierr + 1
   13 IF(m$(19:28) .NE. '            ') GOTO 14
      PRINT 1005, file_id
 1005 FORMAT(1x, 'File ', a13, ' has no longitude.')
      ierr = ierr + 1
   14 ibad = 0
      DO 20 ik=2,8
      ii = ICHAR(m$(ik:ik))
      IF(ii .GE. 48 .AND. ii .LE. 57) GOTO 20
      IF(ii .GE. 65 .AND. ii .LE. 90) GOTO 20
      ibad = ibad + 1
   20 CONTINUE
      IF(ibad .EQ. 0) GOTO 15
      PRINT 1006, file_id
 1006 FORMAT(1x, 'File ', a13, ' has an invalid station identification',
     + ' number.')
      ierr = ierr + 1
   15 ibad = 0
      DO 21 ik=9,18
      ii = ICHAR(m$(ik:ik))
      IF(ii .GE. 48 .AND. ii .LE. 57) GOTO 21
      IF(ii .EQ. 32 .OR.  ii .EQ. 43) GOTO 21
      IF(ii .EQ. 45 .OR.  ii .EQ. 46) GOTO 21
      ibad = ibad + 1
   21 CONTINUE
      IF(ibad .EQ. 0) GOTO 16
      PRINT 1007, file_id
 1007 FORMAT(1x, 'File ', a13, ' has an invalid latitude.')
      ierr = ierr + 1
   16 ibad = 0
      DO 22 ik=19,28
      ii = ICHAR(m$(ik:ik))
      IF(ii .GE. 48 .AND. ii .LE. 57) GOTO 22
      IF(ii .EQ. 32 .OR.  ii .EQ. 43) GOTO 22
      IF(ii .EQ. 45 .OR.  ii .EQ. 46) GOTO 22
```

```
      ibad = ibad + 1
   22 CONTINUE
      IF(ibad .EQ. 0) GOTO 17
      PRINT 1008, file_id
 1008 FORMAT(1x, 'File ', a13, ' has an invalid longitude.')
      ierr = ierr + 1
   17 RETURN
      END

      SUBROUTINE station_check(sta_id_no, file_id, ierr)
c
c This routine checks that the met. station file name is
c consistent with the station id contained in the file.
c Each station must also be comprised of exactly seven
c alphanumeric symbols, no spaces.
c
      IMPLICIT REAL (a-h,j-z)
      IMPLICIT INTEGER (i)
      CHARACTER*15 sta_id_no, file_id*13

      ibad = 0
      DO 10 ik=1,7
      ii = ICHAR(sta_id_no(ik:ik))
      IF(ii .GE. 48 .AND. ii .LE. 57) GOTO 10
      IF(ii .GE. 65 .AND. ii .LE. 90) GOTO 10
      ibad = ibad + 1
   10 CONTINUE
      IF(ibad .EQ. 0) GOTO 20
      PRINT 1000, file_id
 1000 FORMAT(1x, 'File', a13, ' has an improperly formatted header or',
     + ' an improper station name.')
      ierr = ierr + 1
   20 IF(file_id(1:2)   .NE. 'M0') GOTO 30
      IF(sta_id_no(1:1) .EQ.  '0') RETURN
      PRINT 1001, file_id
```

```
1001 FORMAT(1x, 'File ', a13, ' has a station id inconsistent with',
   + ' being a U.S. station.')
     ierr = ierr + 1
     RETURN
  30 IF (File_ID(1:2) .EQ. 'MR') GOTO 40
     IF(sta_id_no(1:1) .EQ. '6' .OR. sta_id_no(1:1) .EQ. '7') RETURN
     PRINT 1002, file_id
1002 FORMAT(1x, 'File ', a13, ' has a station id inconsistent with',
   + ' being a Canadian station.')
     ierr = ierr + 1
     RETURN
  40 IF (Sta_ID_No(1:1) .EQ. 'R') RETURN
     PRINT 1003, File_ID
1003 FORMAT(1X, 'File ', A13, ' has a station id inconsistent with',
   & ' being a Russian station.')
     ierr = ierr + 1
     RETURN
     END

     SUBROUTINE lat_long_check(lake, lat, long, file_id, ierr)
c
c This routine checks that the latitude and longitude of a station
c is within reasonable limits for the specific lake application.
c
     IMPLICIT REAL (a-h,j-z)
     IMPLICIT INTEGER (i)
     CHARACTER*13 file_id, lake*10

     PARAMETER (ino_lakes = 9)

     DIMENSION lat_north(ino_lakes), lat_south(ino_lakes)
     DIMENSION long_east(ino_lakes), long_west(ino_lakes)
c
c SUP   MIC   HUR   STC   ERI   ONT   CHA
c URA   EMB
```

```
C
      DATA lat_north/ 51.0, 47.0, 48.0, 44.0, 44.4, 45.5, 45.2,
     &                55.5, 51.0/
      DATA lat_south/ 45.7, 41.2, 42.0, 41.9, 40.2, 41.7, 43.0,
     &                45.0, 45.0/
      DATA long_east/-79.8,-83.7,-78.4,-80.4,-78.0,-74.5,-72.0,
     &                61.0, 60.0/
      DATA long_west/-93.5,-90.0,-85.5,-83.8,-85.5,-80.3,-74.5,
     &                48.0, 51.0/

      i = 0
      IF(lake .EQ. 'SUPERIOR  ') i = 1
      IF(lake .EQ. 'MICHIGAN  ') i = 2
      IF(lake .EQ. 'HURON     ') i = 3
      IF(lake .EQ. 'STCLAIR   ') i = 4
      IF(lake .EQ. 'ERIE      ') i = 5
      IF(lake .EQ. 'ONTARIO   ') i = 6
      IF(lake .EQ. 'CHAMPLAIN ') i = 7
      IF(lake .EQ. 'URAL      ') i = 8
      IF(lake .EQ. 'EMBE      ') i = 9
      IF(i .NE. 0) GOTO 10
      PRINT 1000
 1000 FORMAT(1x, 'Bad lake name in procedure-created file.')
      ierr = ierr + 1
      RETURN
   10 IF(i .LE. ino_lakes) GOTO 20
      PRINT 1001
 1001 FORMAT(1x, 'Under-dimensioned array in subroutine lat_long_check',
     + ' in program ERRCHECK.FOR......too many lake basins.')
      ierr = ierr + 1
      RETURN
   20 IF(lat  .LE. lat_north(i) .AND. lat  .GE. lat_south(i))  GOTO 30
      PRINT 1002, file_id
 1002 FORMAT(1x, 'File ', a13, ' has a latitude out of range or an',
     + ' improperly formatted header.')
```

```
      ierr = ierr + 1
   30 IF(long .LE. long_east(i) .AND. long .GE. long_west(i)) RETURN
      PRINT 1003, file_id
 1003 FORMAT(1x, 'File ', a13, ' has a longitude out of range or an',
     + ' improperly formatted header.')
      ierr = ierr + 1
      RETURN
      END

      SUBROUTINE lat_long_compare(lat, long, file_id, inofiles, ierr)

c This routine checks that no stations have identical latitudes
c or longitudes.  If any do, the user must choose only a single
c station with those coordinates.
c
      IMPLICIT INTEGER (a-p,r-z)
      IMPLICIT REAL (q)
      REAL lat, long
      CHARACTER*13, file_id, file_array

      PARAMETER (max_no_files = 250)

      DIMENSION file_array(max_no_files), qlat_array(max_no_files),
     +          qlong_array(max_no_files)

      qlat_array(inofiles)  = lat
      qlong_array(inofiles) = long
      file_array(inofiles)  = file_id
      inum = inofiles - 1
      DO 10 i = 1,inum
      IF(lat .EQ. qlat_array(i) .AND. long .EQ. qlong_array(i)) THEN
      PRINT 1000, file_id, file_array(i)
      ierr = ierr + 1
      ENDIF
   10 CONTINUE
```

```
 1000 FORMAT(1x, 'File ', a13, ' has the same latitude and longitude',
     +            ' as file ', a13, '.')
      RETURN
      END

      SUBROUTINE data_format_check(i, m$, iline, file_id, ierr,
     + idate_err)
c
c This routine checks that a line of data is the proper length
c and that there are entries for each date and each type of data.
c
      IMPLICIT REAL (a-h,j-z)
      IMPLICIT INTEGER (i)
      CHARACTER*(*) m$, file_id*13

      idate_err=0
      IF(i .GE. 23) GOTO 10          !data lines are at least 23 spaces for
      PRINT 1001, file_id, iline     !the 2I3, I5, 3I4 format plus comments.
 1001 FORMAT(1x, 'File ', a13, ' has line number', i6, ' of improper',
     + ' length.')
      idate_err=idate_err+1
      ierr = ierr + 1
   10 IF(m$(1:1) .EQ. ' ' .AND. m$(4:4) .EQ. ' ' .AND.
     +   m$(7:7) .EQ. ' ') GOTO 11
      PRINT 1002, file_id, iline
 1002 FORMAT(1x, 'File ', a13, ' has line number', i6,
     + ' improperly formatted.')
      idate_err=idate_err+1
      ierr = ierr + 1
   11 IF(m$(2:3) .NE. '  ') GOTO 12
      PRINT 1003, file_id, iline
 1003 FORMAT(1x, 'File ', a13, ' has no day for line number', i6, '.')
      idate_err=idate_err+1
      ierr = ierr + 1
   12 IF(m$(5:6) .NE. '  ') GOTO 13
```

```
      PRINT 1004, file_id, iline
 1004 FORMAT(1x, 'File ', a13, ' has no month for line number', i6, '.')
      idate_err=idate_err+1
      ierr = ierr + 1
   13 IF(m$(8:11)  .NE. '    ') GOTO 14
      PRINT 1005, file_id, iline
 1005 FORMAT(1x, 'File ', a13, ' has no year for line number', i6, '.')
      idate_err=idate_err+1
      ierr = ierr + 1
   14 IF(m$(12:15) .NE. '    ') GOTO 15
      PRINT 1006, file_id, iline
 1006 FORMAT(1x, 'File ', a13, ' has no maximum temperature for line',
     + ' number', i6, '.')
      ierr = ierr + 1
   15 IF(m$(16:19) .NE. '    ') GOTO 16
      PRINT 1007, file_id, iline
 1007 FORMAT(1x, 'File ', a13, ' has no minimum temperature for line',
     + ' number', i6, '.')
      ierr = ierr + 1
   16 IF(m$(20:23) .NE. '    ') GOTO 21
      PRINT 1008, file_id, iline
 1008 FORMAT(1x, 'File ', a13, ' has no precipitation for line number',
     + i6, '.')
      ierr = ierr + 1
   21 ibad = 0
      DO 22 ik=1,23
      ii = ICHAR(m$(ik:ik))
      IF(ii .GE. 48 .AND. ii .LE. 57) GOTO 22
      IF(ii .EQ. 32 .OR.  ii .EQ. 43 .OR. ii .EQ. 45) GOTO 22
      ibad = ibad + 1
   22 CONTINUE
      IF(ibad .EQ. 0) RETURN
      PRINT 1013, file_id, iline
 1013 FORMAT(1x, 'File ', a13, ' has invalid values in line number',
     + i6, '.')
```

```
      idate_err=idate_err+1
      ierr = ierr + 1
      RETURN
      END

      SUBROUTINE date_check(id, im, iy, iyc, ilast_date, iline,
     +                      file_id, ierr)
c
c This routine checks that the date given is a valid date and that
c it is within a reasonable time period (after the last year of the
c climatological data and before some future date).
c
      IMPLICIT REAL (a-h,j-z)
      IMPLICIT INTEGER (i)
      CHARACTER*13 file_id
      DIMENSION indim(12)
      DATA indim/31,28,31,30,31,30,31,31,30,31,30,31/

      CALL DateSequence(id, im, iy, idate)
      IF(idate .GE. ilast_date) GOTO 10
      PRINT 1000, file_id, iline
1000  FORMAT(1x, 'File ', a13, ' has a date that is not after the end',
     + ' of the climatological data for line', i6, '.')
      ierr = ierr + 1
 10   indim(2) = 28
      IF(iy .LT. iyc .OR. iy .GT. 1989) GOTO 20
      IF(im .LT. 1 .OR. im .GT. 12) GOTO 20
      IF(iy/4*4 .EQ. iy) indim(2) = 29
      IF(id .LT. 1 .OR. id .GT. indim(im)) GOTO 20
      CALL SequenceDate(ixd, ixm, ixy, idate)
      IF(ixd .NE. id .OR. ixm .NE. im .OR. ixy .NE. iy) GOTO 20
      RETURN
 20   PRINT 1001, file_id, iline
1001  FORMAT(1x, 'File ', a13, ' has an invalid date for line number',
     + i6, '.')
```

```
      ierr = ierr + 1
      RETURN
      END

      SUBROUTINE US data_check(tmax, tmin, precip, iline, file_id, ierr)

c This routine checks that maximum and minimum air temperatures
c and precipitation are approximately reasonable for U.S. stations.
c Temperatures are assumed to be degrees Fahrenheit and precipitation
c is assumed to be hundredths of inches.
c
c
      IMPLICIT INCAN (A-H,O-Z)
      IMPLICIT INTEGER (I)
      INTEGER tmax, tmin, precip
      CHARACTER*13 file_id

      IF(tmax .LT. -900) GOTO 10
      IF(tmax .LE.  110) GOTO 11
      PRINT 1001, file_id, iline
      ierr = ierr + 1
   11 IF(tmax .GE.  -60) GOTO 10
      PRINT 1002, file_id, iline
      ierr = ierr + 1
   10 IF(tmin .LT. -900) GOTO 20
      IF(tmin .LE.  110) GOTO 21
      PRINT 1003, file_id, iline
      ierr = ierr + 1
   21 IF(tmin .GE.  -60) GOTO 20
      PRINT 1004, file_id, iline
      ierr = ierr + 1
   20 IF(tmax .LT. -900 .OR. tmin .LT. -900) GOTO 30
      IF(tmin .LE. tmax) GOTO 30
      PRINT 1005,*file_id, iline
      ierr = ierr + 1
```

```
 30 IF(precip .GE. 0) GOTO 31
    IF(precip .LT. -900) RETURN
    PRINT 1006, file_id, iline
    ierr = ierr + 1
    RETURN
 31 IF(precip .LE. 1000) RETURN
    PRINT 1007, file_id, iline
    ierr = ierr + 1
    RETURN
1001 FORMAT(1x, 'File ', a13, ' has maximum temperature > 110 F',
   + ' degrees for line number', i6, '.')
1002 FORMAT(1x, 'File ', a13, ' has maximum temperature < -60 F',
   + ' degrees for line number', i6, '.')
1003 FORMAT(1x, 'File ', a13, ' has minimum temperature > 110 F',
   + ' degrees for line number', i6, '.')
1004 FORMAT(1x, 'File ', a13, ' has minimum temperature < -60 F',
   + ' degrees for line number', i6, '.')
1005 FORMAT(1x, 'File ', a13, ' has minimum temperature exceed',
   + ' maximum temperature for line number', i6, '.')
1006 FORMAT(1x, 'File ', a13, ' has negative precipitation that is',
   + ' not missing data for line number', i6, '.')
1007 FORMAT(1x, 'File ', a13, ' has precipitation > 10 inches for',
   + ' line number', i6, '.')
    END

      SUBROUTINE Can_data_check(tmax, tmin, precip, iline, file_id,ierr)
c
c This routine checks that maximum and minimum air temperatures
c and precipitation are approximately reasonable for Canadian
c stations.  Temperatures are assumed to be tenths of degrees
c Celcius and precipitation is assumed to be tenths of millimeters.
c
c NOTE: This routine is also used for the Russian data set since that
c data set uses the same (metric) units.
c
```

```
      IMPLICIT REAL (a-h,j-z)
      IMPLICIT INTEGER (i)
      INTEGER tmax, tmin, precip
      CHARACTER*13 file_id

      IF(tmax .LT.  -900) GOTO 10
      IF(tmax .LE.   500) GOTO 11
      PRINT 1001, file_id, iline
      ierr = ierr + 1
   11 IF(tmax .GE.  -530) GOTO 10
      PRINT 1002, file_id, iline
      ierr = ierr + 1
   10 IF(tmin .LT.  -900) GOTO 20
      IF(tmin .LE.   430) GOTO 21
      PRINT 1003, file_id, iline
      ierr = ierr + 1
   21 IF(tmin .GE.  -530) GOTO 20
      PRINT 1004, file_id, iline
      ierr = ierr + 1
   20 IF(tmax .LT. -900 .OR. tmin .LT. -900) GOTO 30
      IF(tmin .LE.  tmax) GOTO 30
      PRINT 1005, file_id, iline
      ierr = ierr + 1
   30 IF(precip .GE.    0) GOTO 31
      IF(precip .LT. -900) RETURN
      PRINT 1006, file_id, iline
      ierr = ierr + 1
      RETURN
   31 IF(precip .LE. 2540) RETURN
      PRINT 1007, file_id, iline
      ierr = ierr + 1
      RETURN
  001 FORMAT(1x, 'File ', a13, ' has maximum temperature >  43 C',
     + ' degrees for line number', i6, '.')
  002 FORMAT(1x, 'File ', a13, ' has maximum temperature < -53 C',
```

```
   + ' degrees for line number', i6, '.')
1003 FORMAT(1x, 'File ', a13, ' has minimum temperature >  43 C',
   + ' degrees for line number', i6, '.')
1004 FORMAT(1x, 'File ', a13, ' has minimum temperature < -53 C',
   + ' degrees for line number', i6, '.')
1005 FORMAT(1x, 'File ', a13, ' has minimum temperature exceed',
   + ' maximum temperature for line number', i6, '.')
1006 FORMAT(1x, 'File ', a13, ' has negative precipitation that is',
   + ' not missing data for line number', i6, '.')
1007 FORMAT(1x, 'File ', a13, ' has precipitation > 25.4 centimeters',
   + ' for line number', i6, '.')
     END

     INTEGER FUNCTION istrlen(symbol)
     IMPLICIT INTEGER (a - z)
     CHARACTER*(*) symbol
     i = LEN(symbol)
   1 IF(i .EQ. 0) GOTO 2
     IF(symbol(i:i) .NE. ' ') GOTO 2
     i = i - 1
     GOTO 1
   2 istrlen = i
     RETURN
     END

     SUBROUTINE DateSequence(Day, Month, Year, SequenceNumber)
c
c This routine computes the number of the date, with days numbered
c sequentially from -32768, corresponding to 12 June 1898, through
c 32767, corresponding to 15 November 2077.  All variables, cons-
c tants, and arithmetic are 16-bit integer only.
c
     IMPLICIT NONE
     INTEGER*2 Day, Month, Year, SequenceNumber
     IF (Year .LT. 1900 .OR. (Year .EQ. 1900 .AND. Month .LT. 3)) THEN
```

```
      CALL DS(Day, Month, Year, SequenceNumber, 1896)
      SequenceNumber = SequenceNumber - 835
         SequenceNumber = SequenceNumber - 32767
         RETURN
      ENDIF
      IF (Year .LT. 1988 .OR. (Year .EQ. 1988 .AND. Month .LT. 3)) THEN
      CALL DS(Day, Month, Year, SequenceNumber, 1900)
      SequenceNumber = SequenceNumber - 32142
      RETURN
      ENDIF
      CALL DS(Day, Month, Year, SequenceNumber, 1988)
      RETURN
      END

      SUBROUTINE DS(Day, Month, Year, SequenceNumber,BaseYear)
c
c This routine computes the number of the date, with days numbered
c sequentially from 1 (corresponding to the base date).  The base
c date must be 1 March 1900 or 1 March on any LEAP YEAR not
c greater than 2010.  Dates must be between the base date and the
c base date plus 32766.  [I. e., the logic is correct only for
c dates post-1900 and pre-2100 since 1900 and 2100 are not leap
c years but 2000 is.  Thus, tests for non-leap years on century
c boundaries except those divisible by 400 which are leap years,
c are avoided.]  This routine is set up to use only 2-byte integer
c variables and to avoid intermediate use of reals in the
c evaluation of expressions.  See Dr. Dobb's Journal 80:66-70.
c
      IMPLICIT NONE
      INTEGER*2 Day, Month, Year, SequenceNumber, BaseYear
      SequenceNumber = Year - BaseYear - 1 + (Month + 9) / 12
      SequenceNumber = 365 * SequenceNumber + SequenceNumber / 4
     +                    + (153 * MOD(Month + 9, 12) + 2) / 5 + Day
      RETURN
      END
```

```
      SUBROUTINE SequenceDate(Day, Month, Year, SequenceNumber)
c
c This routine computes the date of the number, with days numbered
c sequentially from -32768, corresponding to 12 June 1898, through
c 32767, corresponding to 15 November 2077. All variables, constants,
c and arithmetic are 16-bit integer only.
c
      IMPLICIT NONE
      INTEGER*2 Day, Month, Year, SequenceNumber, Intermediate
      IF (SequenceNumber .LT. -32141) THEN
         Intermediate = SequenceNumber + 32767
         Intermediate = Intermediate + 835
         CALL SD(Day, Month, Year, Intermediate, 1896)
           RETURN
       ENDIF
       IF (SequenceNumber .LE. 0) THEN
          Intermediate = SequenceNumber + 32142
          CALL SD(Day, Month, Year, Intermediate, 1900)
          RETURN
       ENDIF
       CALL SD(Day, Month, Year, SequenceNumber, 1988)
       RETURN
       END

       SUBROUTINE SD(Day, Month, Year, SequenceNumber,BaseYear)
c
c The routine computes the date of the number, with days numbered
c sequentially from 1 (corresponding to the base date). The base
c date must be 1 March 1900 or 1 March on any LEAP YEAR not
c greater than 2010. Dates must be between the base date and
c the base date plus 32766. [I. e., the logic is correct only for
c dates post-1900 and pre-2100 since 1900 and 2100 are not leap
c years but 2000 is. Thus, tests for non-leap years on century
c boundaries except those divisible by 400 which are leap years,
c are avoided.] This routine is set up to use only 2-byte integer
```

```
c variables and to avoid intermediate use of reals in the  evalua-
c tion of expressions. See Dr. Dobb's Journal 80:66-70.
c
      IMPLICIT NONE
      INTEGER*2 Day, Month, Year, SequenceNumber, BaseYear
      Year = (SequenceNumber - 1 - SequenceNumber / 1461) / 365
      Day = SequenceNumber - 365 * Year - Year / 4
      Month = MOD ((5 * Day - 3) / 153 + 2, 12) + 1
      Year = Year + BaseYear + 1 - (Month + 9) / 12
      Day = Day - (153 * MOD (Month + 9, 12) + 2) / 5
      RETURN
      END

      SUBROUTINE latlong_chk(lat,long,file_id,sta_id_no,ierr)
C*****************************************************************
C THIS SUBROUTINE WILL CHECK THE LATITUDE AND LONGITUDE OF A STATION
C WITH A MASTER LIST. IF THE LOCATION OF THE STATION IS DIFFERENT,
C THEN A WARNING WILL BE PRINTED OUT AND THE FILE WILL BE CORRECTED.
C
C WRITTEN BY Clifford McCardell on November 14, 1988
c for the forecast package and the evaporation forecast package
C
c*****************************************************************
      IMPLICIT none
      INTEGER  MaxStations
      PARAMETER (MaxStations = 250)
      COMMON /blockname/nostations,stations,latlon
      REAL lat,long
      REAL latlon(MaxStations,2)
      INTEGER ierr,INDEX,nostations,oldindex,temp, Flag, Maximum
      CHARACTER*13 file_id,line*80,sta_id_no*15
      CHARACTER*7  stations(MaxStations)

c
c     ilat=jnint(lat*1000.0)    !make sure it returns
```

```
c     ilong=jnint(long*1000.0) !an integer*4 value
c     print*,ilat,ilong
c
c     do binary search of array for station number
c     index=nint(nostations/2.0)
      INDEX = 1
      oldindex=nostations
      Flag = 0
      Maximum = LOG(NoStations * 1.) / LOG(2) + 2
10    CONTINUE
      IF (stations(INDEX).eq.sta_id_no(1:7)) GOTO 30
      Flag = Flag + 1
      IF (Flag .GE. Maximum) THEN
      PRINT*,'Station '//sta_id_no(1:7)//' is not in MASTER LIST'
      ierr=ierr+1
      RETURN
      ENDIF
      IF (stations(INDEX).gt.sta_id_no(1:7)) THEN
      temp=INDEX
      INDEX=INDEX-NINT(ABS(INDEX-oldindex)*1./2.0 + 0.1)
      oldindex=temp
      ELSE
      temp=INDEX
      INDEX=INDEX+NINT(ABS(INDEX-oldindex)*1./2.0 + 0.1)
      oldindex=temp
      ENDIF
      IF (index.le.0) INDEX=1
      IF (index.gt.nostations) INDEX=nostations
      GOTO 10
30    CONTINUE
      IF ((latlon(INDEX,1).eq.lat).AND.
     +          (latlon(INDEX,2).eq.long)) RETURN !FILE IS OK
      PRINT*,'File '//file_id//' has error in station location, it',
     + ' does not match the MASTER LIST.'
      PRINT*,'***'//file_id//'has been changed to match the MASTER',
```

```
     * ' LIST,***'
      OPEN(UNIT=3,FILE='$$DUMMY$',STATUS='new',CARRIAGECONTROL='list',
     *          ERR=49)
c     read(2,*,err=49)
      WRITE(3,1000) sta_id_no,latlon(INDEX,1),latlon(INDEX,2)
40    READ(2,1001,END=50) line
      WRITE(3,1001) line
      GOTO 40
1000 FORMAT(ix,a7, 2f10.3)
1001 FORMAT(a80)
50    CLOSE (2, STATUS = 'DELETE')
      CLOSE(3)
      CALL SYSTEM('RENAME $$DUMMY$ ' // File_ID)
      OPEN(UNIT=2,FILE=file_id,STATUS='old',CARRIAGECONTROL='list')
      READ(2,*)
      RETURN
49    PRINT*,'error in recreating file '//file_id
      PRINT*,'***no file created***'
      ierr=ierr+1
      CLOSE(3,STATUS='delete',ERR=99)
99    RETURN
      END

      SUBROUTINE SORT
c
c This routine reads in stations from a file, stores them in an
c array, and then sorts the array.
c The 2-D array stores Stfname, Latiude, and Longitude
c
c Written by Rick Markham, 14 November 1988
c
      CHARACTER*3 Bn$
      INTEGER  MaxStations
      PARAMETER (MaxStations = 250)
      COMMON/Blockname/Count,Array,Array2
```

```
      COMMON / LakNam / Bn$
      INTEGER J,I,L,JJ,NN,Count
      CHARACTER Stfname*7
      CHARACTER*7 Array(MaxStations),Tmost
      REAL Lat,Long,Array2(MaxStations,2),Tmplat,Tmplong

      Count=0
c
c     Open file that contains the list of stations to be sorted
c
      OPEN (UNIT=1,FILE='STNLIST.'//bn$,STATUS='old')
      READ(1, 500)
10    READ (1,500,END=999) Stfname,Lat,Long
      Count = Count + 1
      Array(Count) = Stfname
      Array2(Count,1) = Lat
      Array2(Count,2) = Long
      GOTO 10
999   CONTINUE
      CLOSE (1)
c
c     Sort the array by station number
c     Selection Sort
c
      NN = Count - 1
      DO 100 J=1,NN
c
c     Find the location of the smallest
c
      L=J
      JJ = J + 1
      DO 200 I = JJ,Count
      IF (Array(L) .LT. Array(I)) GOTO 200
      L = I
200   CONTINUE
```

```
c
c     Interchange Array(L) with A(J)
c
      Tmpst=Array(L)
      Tmplat= Array2(L,1)
      Tmplong= Array2(L,2)
      Array(L) = Array(J)
      Array2(L,1) = Array2(J,1)
      Array2(L,2) = Array2(J,2)
      Array(J) = Tmpst
      Array2(J,1) = Tmplat
      Array2(J,2) = Tmplong
100   CONTINUE
c
c     Format statements
c
500   FORMAT(A7,f10.0,f10.0)
      END
```

Приложение Е

```
      PROGRAM PROVISIONAL_update
c
c Updated by TEC II, 28Sep88, 26Oct88 (for PC)
c
c This routine is the interface between the user and the system for
c entering new provisional meteorological data into the linked-list
c file: PROVSNAL.xxx.  xxx is read from file DSNNM, and is defined
c as follows:
c
c       xxx = SUP  (Lake Superior Basin)
c             MIC  (Lake Michigan Basin)
c             HUR  (Lake Huron Basin)
c             STC  (Lake St. Clair Basin)
c             ERI  (Lake Erie Basin)
c             ONT  (Lake Ontario Basin)
c             CHA  (Lake Champlain Basin)
c             URA  (Ural River Basin in USSR)
c             EMB  (Embe River Basin in USSR)
c
c This program is to be used in a "batch" mode; that is, the user
c mustprepare input files for station information to be added and
c then execute this program.
c
c NOTE:  THE DATA BASE (PROVSNAL.xxx) SHOULD BE CREATED, MODIFIED,
c OR UPDATED ONLY WITH THIS PROGRAM!!!!!!!
c
c The input file: PROVSNAL.LIS must have each record containing
c the name of a file to be incorporated into PROVSNAL.xxx.  Each
c file named must follow the following name convention:
c
```

```
c 0000000.PRV
c !      !
c !______!_________seven-character station identification string
c                   which must agree with the seven characters of the
c                   corresponding station identification string
c                   (STATION_ID) in the first record of the file,
c
c Each station file must have the following structure (followed
c exactly):
c             First record:
c
c             STATION_ID ## ## (format: 1x, a7, 2f10.3)
c             !         ! !! !!
c             !         ! !! !!__longitude in decimal degrees
c             !         ! !!____latitude in decimal degrees
c             !_________!_____station identification string with
c                               left-most character zero for U. S.
c                               stations and non-zero for Canadian
c                               stations
c             Remaining records:
c
c         dd mm yyyy Tmax Tmin Precip (format: 2i3, i5, 3i4)
c         !! !! !  ! !  ! !  ! !   !
c         !! !! !  ! !  ! !  ! !____!__depth of daily precipitation,
c         !! !! !  ! !  ! !  !         in inch/100 for U. S.,
c         !! !! !  ! !  ! !  !         in mm/10 for Canadian.
c         !! !! !  ! !  ! !__!_______minimum daily air temperature,
c         !! !! !  ! !  !             in degrees F for U. S.,
c         !! !! !  ! !  !             in degrees C/10 for Canadian.
c         !! !! !  ! !__!__________maximum daily air temperature,
c         !! !! !  !                in degrees F for U. S.,
c         !! !! !  !                in degrees C/10 for Canadian.
c         !! !! !__!_____________four-digit year of data's date
c         !! !!_*_______________two-digit month of data's date
c         !!_________________two-digit day of data's date
```

```
c This program creates or updates the file: PROVSNAL.xxx which is
c composed of "blocks", one for each day of record, arranged in a
c linked sequential order with every day represented by one block.
c The file is a direct access file with a fixed record length of
c 14 bytes; therefore, it must not be edited normally since an
c editor will create sequential access files which are unusuable
c in DISAVMET. All records are unformatted.  Each block has the
c following structure:
c
c        First block record:
c
c  date blink * link (unformatted: 2, 4, 4, 4)
c  !  ! !   ! ! !  !
c  !  ! !   ! ! !__!__pointer to next logical record
c  !  ! !   ! !_____used only in first physical record
c  !  ! !   !       as pointer to last physical record: if negative,
c  !  ! !   !       indicates block has been processed by DISAVGMET
c  !  ! !   !       ("old"); if positive, indicates block must yet be
c  !  ! !   !       processed by DISAVMET ("new")
c  !  ! !___!_____pointer to first record of next logical data block
c  !__!_________date sequence number
c
c        Other block records:
c
c ## ## Tmin Tmax Precip link (unformatted: 2, 2, 2, 2, 2, 4)
c !! !! !  ! !  ! !    ! !  !
c !! !! !  ! !  ! !    ! !__!__pointer to next logical record
c !! !! !  ! !  ! !____!_____depth of daily precipitation in
c !! !! !  ! !  !             hundredths of millimeters
c !! !! !  ! !__!_____________maximum daily air temperature in
c !! !! !  !                hundredths of degrees Centigrade
c !! !! !__!_____________minimum daily air temperature in
c !! !!                   hundredths of degrees Centigrade
c !! !!_________________longitude in hundredths of decimal degrees
c !!___________________latitude in hundredths of decimal degrees
```

```
c THIS PROGRAM MUST BE COMPILED TO DEFAULT TO 2-BYTE INTEGERS AND
c INTEGER CONSTANTS!  SELECT THE "/T" SWITCH ON THE LAHEY FORTRAN
c COMPILER!
      IMPLICIT NONE
      INTEGER*2 Max_No_Days
c
c     Can handle up to 30 years = 10958 days.
c
      PARAMETER(Max_No_Days = 10958)

      INTEGER*2 Nb, Base_Day, Base_Month, Base_Year, Base, Lat, Long,
     & Start, iNDEX, Date, Last, Latitude, Longitude, Day, Month, Year,
     & Tmax, Tmin, Precip, Number, MinT, MaxT, Prec, Kk, M1s, M2s, M3s,
     & M4s, M5s, iStrLen, Number_Of_Stations
      INTEGER*4 Rno, Rpo, Rec_No_Of_Date(Max_No_Days), bLink, Rc, Link,
     & nbLink, Rn, Zro4, zOn4, Dum1, M, N, I, K
      REAL*4 qLatitude, qLongitude
      LOGICAL*1 Crash
      CHARACTER*10 Name, Name$*80, Station_ID*8, Bn$*3
      CHARACTER*4 Over_The_Lake, St_Id, Station, CrLf*2
      COMMON Lat, Long

      CrLf = CHAR(13) // CHAR(10)
c
c     Four-byte integer constants.
c
      Zro4 =       0
      zOn4 =       1
c
c WARNING! The following date is the BASE DATE and corresponds  to
c the first day of the provisional data  which  MUST  BE  the  day
c after the last day of the climatologic  data  records  available
c for the application. The  user  will  not  be  allowed  by  this
c program to add or change any provisional data prior to this base
c date nor is it assumed that he/she will have any  reason  to  do
```

```
c so.  DO NOT CHANGE THIS BASE DATE as it  is  used  in  companion
c programs and must agree.  When changed, the previous version  of
c file: PROVSNAL.xxx must be amended to correspond to the new base
c date by deleting all earlier data and repacking  the  file.  The
c user should only perform this operation with program: REPACK  by
c changing the base date and  running  that  program;  once  done,
c earlier provisional  data  is  lost  and  unrecoverable.    This
c operation need only be performed  infrequently  as  climatologic
c data becomes available and the forecast data bases  are  rebuilt
c by GLERL (say, once every two years or so)  or  when  the  file:
c PROVSNAL.xxx becomes too  large  to  manage  efficiently  as  a
c linked list.
c
c     Open BSNNM. and get basin name and base date.
c
      OPEN(UNIT = 4, FILE = 'BSNNM.', STATUS = 'OLD',
     &     ACCESS = 'DIRECT', RECL = 13, FORM = 'FORMATTED')
      READ(4, FMT = 1100, REC = 1) Name
 1100 FORMAT(A10, 1X)
      READ(4, FMT = 1101, REC = 2) Nb, I, Over_The_Lake
 1101 FORMAT(2I3, A4, 1X)
      READ(4, FMT = 1102, REC = 3) Base_Day, Base_Month, Base_Year
 1102 FORMAT(2I3, I5)
      IF (I .NE. 1) THEN
       WRITE(6, 1009)
 1009  FORMAT(1X, 'Invalid sequencing of forecast programs.',
     &  /, 1X, 'PROVSNAL may only be run after ERRCHECK.')
       CLOSE (4)
       CALL EXIT(1)
      ENDIF
      CALL DateSequence(Base_Day, Base_Month, Base_Year, Base)
      Bn$ = Name(1:3)
c
c     Open data base.
c
```

```
      OPEN(UNIT = 1, FILE = 'PROVSNAL.' // Bn$, STATUS = 'OLD',
     & ACCESS = 'DIRECT', RECL = 14, ERR = 885, FORM ='UNFORMATTED')
      GOTO 886
  885 OPEN(UNIT = 1, FILE = 'PROVSNAL.' // Bn$, STATUS = 'NEW',
     & ACCESS = 'DIRECT', RECL = 14, FORM = 'UNFORMATTED')
      WRITE(1, REC = zOn4) Base, Zro4, zOn4, Zro4
  886 READ(1, REC = zOn4, ERR = 999) Start, bLink, Rc, Link
      Rc = IABS(Rc)
      IF (Start .NE. Base) THEN
       WRITE(6, 1004) Bn$
 1004  FORMAT(1X, 'PROVSNAL.', A5, ' has been altered by a'
     &  ' program other than PROVSNAL.', /,
     &  1X, 'If PERACKed, have data for PROVSNAL must'
     &    ' agree with PERACK.', /,
     &  1X, 'If not, then go to last backups and begin'
     &  ' again.')
       CLOSE (4)
       CALL EXIT(1)
      ENDIF
c
c     Starting logical record is always first physical one.
c
      bLink = zOn4
c
c     Start should always equal Base.
c
      INDEX = Start - Base
c
c     Note position of block starts.
c
      WRITE(6, 30)
   30 FORMAT(1H+, 'Reading daily data blocks links...            ')
c     'DoWhile' Construct...
    2 IF (bLink .NE. Zro4) THEN
       READ(1, REC = bLink, ERR = 999) Date, nbLink, Dum1, Link
```

```
      INDEX = INDEX + 1
      IF (INDEX .GT. Max_No_Days) THEN
       WRITE(6, 1007) Bn$
 1007  FORMAT(1X, 'Insufficient number of days allowed.', /,
     &   1X, 'Updating of PROVSNAL.', A3, ' aborted.')
       CLOSE (4)
       CALL EXIT(1)
      ENDIF
      Rec_No_Of_Date(INDEX) = bLink
      bLink = nbLink
      GOTO 2
     ENDIF
     Last = INDEX + Base - 1
     Crash = .FALSE.
c
c    Input list of filenames for user-supplied input files.
c
     OPEN(UNIT = 2, FILE = 'PROVSNAL.LIS', STATUS = 'OLD', ERR = 5)
     Number_Of_Stations = 0
  87 CLOSE (3)
  75 READ(2, 1000, END = 5) Name$
1000 FORMAT(A80)
     IF (iStrLen(Name$) .EQ. 0 .OR. Name$(1:1) .EQ. ' ') GOTO 75
     Name$ = Name$(1:8) // '.' // Name$(10:12)
     OPEN(UNIT = 3, FILE = Name$, STATUS = 'OLD')
     READ(3, 1002, END = 87) Station_ID, qLatitude, qLongitude
1002 FORMAT(A8, 2F10.0)
     Number_Of_Stations = Number_Of_Stations + 1
     WRITE(6, 31) Number_Of_Stations, Station_ID(2:8)
  31 FORMAT(1H+, 'Processing station', I4, ' (', A7,
    & ') ...                    ')
     Latitude = INT(ANINT(qLatitude * 100.))
     Longitude = INT(ANINT(qLongitude * 100.))
     Lat = Latitude
     Long = Longitude
```

```
      CALL ID_String(Station)
c
c     Main loop for interaction with user.
c
c     Get block date from user input.
c
  100 READ(3, 1062, END = 87) Day, Month, Year, Tmax, Tmin, Precip
 1062 FORMAT(2I3, I5, 3I4)
c
c     Convert to units used by remainder of forecast package.
c
      IF (Station_ID(2:2) .NE. '0') THEN
       IF (Tmax .LT. -900) THEN
     Tmax = -9999                                  ! Missing data.
       ELSE
     Tmax = Tmax * 10                              ! Degrees C x 100.
       ENDIF
       IF (Tmin .LT. -900) THEN
     Tmin = -9999                                  ! Missing data.
       ELSE
     Tmin = Tmin * 10                              ! Degrees C x 100.
       ENDIF
       IF (Precip .LT. 0) THEN
     Precip = -9999                                ! Missing data.
       ELSE
     Precip = Precip * 10                          ! Millimeters x 100.
       ENDIF
      ELSE
       IF (Tmax .LT. -900) THEN
        Tmax = -9999                                ! Missing data.
       ELSE
        Tmax = INT(ANINT(((Tmax-32.)*5./9.)*100.)) ! Degrees Cx100.
       ENDIF
       IF (Tmin .LT. -900) THEN
        Tmin = -9999                                ! Missing data.
```

```
        ELSE
         Tmin=INT(ANINT(((Tmin-32.)*5./9.)*100.)) ! Degrees Cx100.
        ENDIF
        IF (Precip .LT. 0) THEN
         Precip = -9999                              ! Missing data.
        ELSE
         Precip=INT(ANINT((Precip/100.*25.4)*100.)) ! Millimetersx100.
        ENDIF
       ENDIF
c
c      Find corresponding block in file.
c
       CALL DateSequence(Day, Month, Year, Number)
       IF (Number .LT. Base) THEN
        Crash = .TRUE.
        WRITE(6, 1003) Bn$
 1003   FORMAT(1X, 'Attempt to add data previous to base year.', /,
     &  1X, 'Updating of PROVSNAL.', A3, ' terminated.')
        GOTO 5
       ENDIF
       IF (Number .GE. Start .AND. Number .LE. Last) GOTO 6
c
c Fill in file with missing data (blocks with no data records) for
c all intervening dates up to and including the date given by user.
c
       IF (Number - Base + 1 .GT. Max_No_Days) THEN
        Crash = .TRUE.
        WRITE(6, 1008) Bn$
 1008   FORMAT(1X, 'Insufficient number of days allowed.', /,
     &  1X, 'Updating of PROVSNAL.', A3, ' terminated.')
        GOTO 5
       ENDIF
c
c      Change last block header pointers.
c
```

```
      Rno = Rec_No_Of_Date(Last - Base + 1)
      READ(1, REC = Rno, ERR = 999) Date, bLink, Dum1, Rpo
c
c     Header points to addition.
c
      IF (Rpo .EQ. Zro4) THEN
c
c      No data, skip search of records.
c
       WRITE(1, REC = Rno) Date, Rc + zOn4, Dum1, Rc + zOn4
      ELSE
c
c      Data exists, preserve pointer.
c
       WRITE(1, REC = Rno) Date, Rc + zOn4, Dum1, Rpo
c
c      Find end of data block.
c
       Rno = Rpo
   56  READ(1,REC=Rno,ERR= 999) Lat, Long, MinT, MaxT, Prec, Rpo
       IF (Rpo .EQ. Zro4) GOTO 55
       Rno = Rpo
       GOTO 56
c
c      Last record points to addition
c
   55  WRITE(1, REC = Rno) Lat, Long, MinT, MaxT, Prec, Rc + zOn4
      ENDIF
c
c     Not executed if number = last + 1.
c
      DO 8 Kk = Last + 1, Number - 1
       Rc = Rc + zOn4
c
c      Add block header records only.
```

```
c
        WRITE(1, REC = Rc) Kk, Rc + zOn4, zOn4, Rc + zOn4
        Rec_No_Of_Date(Kk - Base + 1) = Rc
    8 CONTINUE
      Rc = Rc + zOn4
c
c     Pointers to end of file.
c
      WRITE(1, REC = Rc) Number, Zro4, zOn4, Zro4
      Last = Number
      Rec_No_Of_Date(Last - Base + 1) = Rc
c
c     Have now to add new data record to block.
c
c     First, search block.
c
    6 Rn = Rec_No_Of_Date(Number - Base + 1)
c
c     K => to old first data record.
c
      READ(1, REC = Rn, ERR = 999) Date, bLink, Dum1, K
c
c     Strip "old" flag.
c
      Dum1 = IABS(Dum1)
      M = K
      N = Rn
   16 I = M
c
c If at end of block, must add record; otherwise replace old with
c new.
c
      IF (I .EQ. bLink) GOTO 10
      READ(1, REC = I, ERR = 999) Lat, Long, MinT, MaxT, Prec, M
      CALL ID_String(St_ID)
```

```
c
c     If equal, must replace old with new.
c
      IF (St_ID .EQ. Station) GOTO 11
c
c     If greater than, must add new in alphanumeric sequence.
c
      IF (St_ID .GT. Station) GOTO 10
      N = I
      GOTO 16
c
c     Add to existing data.
c
c     Don't add data if all missing.
c
   10 IF (      Tmax .LT. -9000
     & .AND.    Tmin .LT. -9000
     & .AND. Precip .LT. -9000) GOTO 100
      Rc = Rc + zOn$
c
c     Flag block as new.
c
      WRITE(1, REC = Rn) Date, bLink, Dum1, K
c
c     Throw away pointer to this record.
c
      READ(1, REC = N, ERR = 999) M1s, M2s, M3s, M4s, M5s
c
c     Replace to point to addition.
c
      WRITE(1, REC = N) M1s, M2s, M3s, M4s, M5s, Rc
c
c     New record points to old next one.
c
      WRITE(1, REC = Rc) Latitude, Longitude, Tmin, Tmax, Precip, I
```

```
      GOTO 100
c
c     Change existing data.
c
   11 IF (      Tmax .LT. -9000
     & .AND.    Tmin .LT. -9000
     & .AND. Precip .LT. -9000) THEN
c
c       Flag block as new.
c
        WRITE(1, REC = Rn) Date, bLink, Dum1, K
c
c       Throw away pointer to this record.
c
        READ(1, REC = N, ERR = 999) M1s, M2s, M3s, M4s, M5s
c
c       Replace to point around this one.
c
        WRITE(1, REC = N) M1s, M2s, M3s, M4s, M5s, M
      ELSE
c
c      Flag block as "new".
c
       WRITE(1, REC = Rn) Date, bLink, Dum1, K
c
c      Replace with new record.
c
       WRITE(1, REC = I) Latitude, Longitude, Tmin, Tmax, Precip, M
      ENDIF
      GOTO 100
c
c     Read error in PROVSNAL.xxx.
c
 999  WRITE(6, 2005) Bn$
2005  FORMAT(1X,'Problem with "PROVSNAL.',A3,'";updating aborted.')
```

```
c
c     Invalidate any program use.
c
      WRITE(4, FMT = 1103, REC = 2) Nb, -1, Over_The_Lake, CrLf
 1103 FORMAT(2I3, A4, 1X, A2)
      CLOSE (4)
      CALL EXIT(1)
c
c     End of update session.
c
    5 WRITE(6, 32)
   32 FORMAT(1H+, 'Closing and checking provisional data base... ')
      READ(1, REC = zOn4, ERR = 999) Date, bLink, Dum1, Link
      IF (Dum1 .LT. Zro4) Rc = -Rc
      WRITE(1, REC = zOn4) Date, bLink, Rc, Link
      I = Zro4
      M = Zro4
      Kk = Zro4
      IF (Link .NE. Zro4 .AND. Link .NE. bLink) THEN
       Dum1 = Link
   23  READ(1, REC = Dum1, ERR = 999)Latitude,Longitude,Tmin,Tmax,
     &  Precip, Dum1
       IF (Tmin .GT. -9000) I = I + zOn4
       IF (Tmax .GT. -9000) M = M + zOn4
       IF (Precip .GT. -9000) Kk = Kk + zOn4
       IF (I .GT. Zro4 .AND. M .GT. Zro4 .AND. Kk .GT. Zro4) GOTO 22
       IF (Dum1 .NE. bLink) GOTO 23
      ENDIF
   22 CLOSE (1)
      CLOSE (5)
      CLOSE (2)
c
c     Date = base actually already checked.
c
      IF (Date .NE. Base .OR. Link .EQ. bLink) THEN
```

```
      WRITE(6, 1200) Bn$
 1200 FORMAT(1X, 'PROVSNAL.', A3, ' does not have valid data'
     & ' on base date.')
      ELSE
       IF (I .EQ. Zro4 .OR. M .EQ. Zro4 .OR. Kk .EQ. Zro4)
     & WRITE(6, 1200) Bn$
      ENDIF
      IF (Crash) THEN
       CLOSE (4)
       CALL EXIT(1)
      ENDIF
c
c     2 = this program's designator.
c
      WRITE(4, FMT = 1103, REC = 2) Nb, 2, Over_The_Lake, CrLf
      CLOSE (4)
c
c     Successful data base building; repack data base.
c
      WRITE(6, 33)
   33 FORMAT(1H+, '                                                ',
     & /, 1H+)
      END

      SUBROUTINE ID_String(St_ID)
c
c This routine returns the concatenated binary image of the two
c 2-byte integers, contained in the unlabeled common block, as a
c 4-byte string. An effort is made to enable subsequent ordering
c by latitude first and then longitude. Two-byte integers are
c stored with least significant byte first; these are then
c switched in order here for the i. d. string.
c
      IMPLICIT NONE
      CHARACTER*4 St_ID, StaID
```

```
      COMMON StaID
      St_ID = StaID(2:2) // StaID(1:1) // StaID(4:4) // StaID(3:3)
      RETURN
      END

      INTEGER*2 FUNCTION iStrLen(Symbol)
      IMPLICIT NONE
      INTEGER*2 I
      CHARACTER*(*) Symbol
      I = LEN(Symbol)
c     'DoWhile' Construct...
    1 IF (I .NE. 0 .AND. Symbol(I:I) .EQ. ' ') THEN
       I = I - 1
       GOTO 1
      ENDIF
      iStrLen = I
      RETURN
      END

      SUBROUTINE DateSequence(Day, Month, Year, SequenceNumber)
c
c This routine computes the number of the date, with days numbered
c sequentially from -32768, corresponding to 12 June 1898, through
c 32767, corresponding to 15 November 2077. All variables,
c constants, and arithmetic are 16-bit integer only.
c
    IMPLICIT NONE
    INTEGER*2 Day, Month, Year, SequenceNumber
    IF (Year.LT.1900.OR.(Year .EQ. 1900 .AND. Month .LT. 3)) THEN
       CALL DS(Day, Month, Year, SequenceNumber, 1896)
       SequenceNumber = SequenceNumber - 835
       SequenceNumber = SequenceNumber - 32767
       RETURN
    ENDIF
    IF (Year.LT.1988.OR.(Year .EQ. 1988 .AND. Month .LT. 3)) THEN
```

```
      CALL DS(Day, Month, Year, SequenceNumber, 1900)
      SequenceNumber = SequenceNumber - 32142
      RETURN
     ENDIF
     CALL DS(Day, Month, Year, SequenceNumber, 1988)
     RETURN
     END

     SUBROUTINE DS(Day, Month, Year, SequenceNumber,BaseYear)
c
c This routine computes the number of the date, with days numbered
c sequentially from 1 (corresponding to the base date).  The  base
c date must be 1 March 1900 or 1 March on  any  LEAP  YEAR  not
c greater than 2010.  Dates must be between the base date and the
c base date plus 32766.  [I. e., the logic is  correct  only  for
c dates post-1900 and pre-2100 since 1900 and 2100 are  not  leap
c years but 2000 is.  Thus, tests for non-leap years  on  century
c boundaries except those divisible by 400 which are leap  years,
c are avoided.]  This routine is set up to use only 2-byte integer
c variables and  to  avoid  intermediate  use  of  reals  in  the
c evaluation of expressions.  See Dr. Dobb's Journal 80:66-70.
c
    IMPLICIT NONE
    INTEGER*2 Day, Month, Year, SequenceNumber, BaseYear
    SequenceNumber = Year - BaseYear - 1 + (Month + 9) / 12
    SequenceNumber = 365 * SequenceNumber + SequenceNumber / 4
   &                 + (153 * MOD(Month + 9, 12) + 2) / 5 + Day
    RETURN
    END
```

Приложение F

```
      PROGRAM Repack
c     Updated by TEC II 9apr90, 28sep88, 26OctBB (for PC).
c
c  This routine reads the master provisional data base,
c PROVSNAL.xxx and rewrites it in its logical order, sequentially.
c See comments in PROVSNAL.N14 for file structure. This repacking
c eliminates any "dead" space and allows subsequent disk I/O on
c the file to proceed sequentially, thus eliminating "head
c thrashing" on the disk drive consequent with a linked list
c spread unsequentially. This program is not necessary for
c correct update and forecast operations as both PROVISNAL and
c DISAVGMET read linked lists correctly; however, used regularly,
c it will enhance subsequent disk access operations. The
c disadvantages are an increase in the total amount of processing
c during any one update/forecast session that also repacks and
c decreased data integrity reliability since the entire master
c provisional data base, PROVSNAL.xxx, is read and rewritten each
c time.
c NOTE: This program will truncate all data earlier than the "base
c date" contained in the file: BSNNM. Care should be exercised
c that this file is never modified by the user except as a conse-
c quence of running programs that are part of the forecast
c apackage.
c THIS PROGRAM MUST BE COMPILED TO DEFAULT TO 2-BYTE INTEGERS AND
c INTEGER CONSTANTS! SELECT THE "/T" SWITCH ON THE LAHEY FORTRAN
c COMPILER!
      IMPLICIT NONE
      INTEGER*2 MaxAllowable
      PARAMETER (MaxAllowable = 300)   ! 300 stations per day maximum.
      INTEGER*2 Ms(5, MaxAllowable), Day, Month, Year, Base, Date
      INTEGER*4 Zero, zOne, bLink, Rn, Link, M2, I, Max_No, J, obLink,
     & NoOfRec, K
      CHARACTER*3 Bn$
```

```
c   Four-byte integer values
    Zero = 0
    zOne = 1
c
c.  Get basin name and base date.
c
    OPEN(UNIT = 1, FILE = 'BSNNM.', STATUS = 'OLD',
   & ACCESS = 'DIRECT', RECL = 13, FORM = 'FORMATTED')
    READ(1, FMT = 1001, REC = 1) Bn$
1001 FORMAT(A3)
    READ(1, FMT = 1002, REC = 3) Day, Month, Year
1002 FORMAT(2I3, I5)
    CLOSE (1)
    CALL DateSequence(Day, Month, Year, Base)
c
c Open PROVSNAL.xxx to be repacked and the PROVSNAL.xxx to
c receive the repacked  copy.
c
    CALL SYSTEM('RENAME PROVSNAL.' // Bn$ // ' PRVSNLOL.*')
    OPEN(UNIT = 1, FILE = 'PRVSNLOL.' // Bn$, STATUS = 'OLD',
   & ACCESS = 'DIRECT', RECL = 14, FORM = 'UNFORMATTED')
    OPEN(UNIT = 2, FILE = 'PROVSNAL.' // Bn$, STATUS = 'NEW',
   & ACCESS = 'DIRECT', RECL = 14, FORM = 'UNFORMATTED')

    bLink = zOne      ! Start with 1st logical rec. (also 1st
                      ! physical rec.).
    Rn = zOne         ! Start write with first physical record.
    Max_No = Zero     ! Initialize maximum record count.
    K = Zero          ! Initialize record count
    READ(1, REC = zOne) Date, obLink, NoOfRec, Link
    NoOfRec = IABS(NoOfRec) + zOne  ! so "100%" is never printed.
  2 obLink = bLink
    READ(1, REC = bLink) Date, bLink, M2, Link
    K = K + zOne
    IF (K / 100 * 100 .EQ. K) WRITE(6, 1) K * 100 / NoOfRec
```

```
  1 FORMAT(1H+, 'Repacking', I3, '%')
    IF (Date .LT. Base) GOTO 2            ! Skip data prior to Base.
    bLink = obLink
    K = K - zOne
  4 READ(1, REC = bLink) Date, bLink, M2, Link  ! Read data block
    K = K + zOne
    IF (K / 100 * 100 .EQ. K) WRITE(6, 1) K * 100 / NoOfRec
    I = Zero
  3 IF (bLink .NE. Link) THEN
    I = I + zOne
    IF (I .GT. MaxAllowable) THEN
    CLOSE(1)
    CLOSE(2)
    CALL SYSTEM('IF EXIST PROVSNAL.'//Bn$//' DEL PROVSNAL.'//Bn$)
    CALL SYSTEM('RENAME PRVSNLOL.' // Bn$ // ' PROVSNAL.*')
    WRITE(6, 34)
34   FORMAT('+Number of stations exceeds dimensions in REPACK.    ')
    CALL EXIT(1)
    ENDIF
    READ(1,REC=Link)Ms(1,I),Ms(2,I),Ms(3,I),Ms(4,I),Ms(5,I),Link
    K = K + zOne
    IF (K / 100 * 100 .EQ. K) WRITE(6, 1) K * 100 / NoOfRec
    GOTO 3
    ENDIF
c
c Write the data block as a linked list but arranged sequentially.
c
    IF (Max_No .LT. I) Max_No = I
    WRITE(2, REC = Rn) Date, Rn + I + zOne, M2, Rn + zOne
c
c   This loop is skipped if I = 0.
c
    DO 6 J = zOne, I
    Rn = Rn + zOne
    WRITE(2, REC = Rn) Ms(1,J), Ms(2,J), Ms(3,J), Ms(4,J), Ms(5,J),
```

```
     &  Rn + zOne
   6 CONTINUE
     Rn = Rn + zOne
     IF (Link .GT. Zero) GOTO 4
c
c    Rewrite last data block header (and last record if there is any)
c    so pointers are set to EOF.
c
c    Number of last record written.
c
     Rn = Rn - zOne
     IF (I .EQ. Zero) THEN
     WRITE(2, REC = Rn) Date, Zero, M2, Zero
     ELSE
     WRITE(2, REC = Rn - I) Date, Zero, M2, Rn - I + zOne
     WRITE(2, REC = Rn) Ms(1,I), Ms(2,I), Ms(3,I), Ms(4,I), Ms(5,I),
    &  Zero
     ENDIF
c
c Rewrite first physical record to update the last (physical)
c record number.
c
     READ(2, REC = zOne) Date, bLink, M2, Link
     IF (M2 .LT. Zero) Rn = -Rn
     WRITE(2, REC = zOne) Date, bLink, Rn, Link
c
c Put maximum number of stations beyond linked list, in last
c record plus 1.
     WRITE(2, REC = IABS(Rn) + zOne) Max_No
     CLOSE (1)
     CLOSE (2)
     WRITE(6, 33)
  33 FORMAT(1H+, '                                                    ')
     CALL SYSTEM('DEL PRVSNLOL.' // Bn$)
     END
```

```
      SUBROUTINE DateSequence(Day, Month, Year, SequenceNumber)
c
c This routine computes the number of the date, with days numbered
c sequentially from -32768, corresponding to 12 June 1898, through
c 32767, corresponding to 15 November 2077. All variables, constants,
c  and arithmetic are 16-bit integer  only.
c
      IMPLICIT NONE
      INTEGER*2 Day, Month, Year, SequenceNumber
      IF (Year.LT.1900.OR. (Year .EQ. 1900 .AND. Month .LT. 3)) THEN
      CALL DS(Day, Month, Year, SequenceNumber, 1896)
      SequenceNumber = SequenceNumber - 835
      SequenceNumber = SequenceNumber - 32767
      RETURN
      ENDIF
      IF (Year.LT.1988 .OR. (Year .EQ. 1988 .AND. Month .LT. 3)) THEN
      CALL DS(Day, Month, Year, SequenceNumber, 1900)
      SequenceNumber = SequenceNumber - 32142
      RETURN
      ENDIF
      CALL DS(Day, Month, Year, SequenceNumber, 1988)
      RETURN
      END

      SUBROUTINE DS(Day, Month, Year, SequenceNumber,BaseYear)
c
c This routine computes the number of the date, with days numbered
c sequentially from 1 (corresponding to the base date).  The base
c date must be 1 March 1900 or 1 March on any LEAP YEAR not
c greater than 2010.  Dates must be between the base date and the
c base date plus 32766.  [I. e., the logic is correct only for
c dates post-1900 and pre-2100 since 1900 and 2100 are not leap
c years but 2000 is.  Thus, tests for non-leap years on century
c boundaries except those divisible by 400 which are leap years,
c are avoided.]  This routine is set up to use only 2-byte integer
```

```
c variables and to avoid intermediate use of reals in the
c evaluation of expressions.  See Dr. Dobb's Journal 80:66-70.
c
      IMPLICIT NONE
      INTEGER*2 Day, Month, Year, SequenceNumber, BaseYear
      SequenceNumber = Year - BaseYear - 1 + (Month + 9) / 12
      SequenceNumber = 365 * SequenceNumber + SequenceNumber / 4
     &                     + (153 * MOD(Month + 9, 12) + 2) / 5 + Day
      RETURN
      END
```

Приложение 6

```
      PROGRAM DIStributedAVerageMETeorology
c
c Updated by TEC II 28sep88, 26Oct88 (for PC), 17jan90 (for variable
c length), 28feb90, 2mar90, 21jun90, 09aug90.
c Updated by TSH 08jun90.
c
c Handling of 1-station networks modified by TEC on 17jan90 as per
c changes made by TSH on 14aug89 to EDSAVMET.SRC.
c Informational message, on computation of Thiessen weights, added
c for use of same weights as last time by TEC II 28feb90.
c Handling of Russian rivers (Ural and Emba) added by TSH on 08jun90.
c Corrected for non-unity cell size by TEC II 21jun90.
c Corrected for no lake (subbasin 0) present in map by TEC II
c 21jun90. Informational message, on computation of Thiessen
c weights, corrected & history log of Thiessen weights computation
c added by TEC II 09aug90.
c
c This routine updates the provisional subbasin data files from the
c provisional data file: PROVSNAL.xxx. It computes Thiessen-weighted
c areal averages for minimum and maximum air temperatures and
c precipitation if they are flagged as new data in the provisional
c data file and removes the flags. Otherwise old data values,
c computed previously, are left unchanged in the provisional
c subbasin data files, named as follows:
c
c          xxx%%.PRV
c          ! !!!
c          ! !!!______two-digit subbasin number
c          !_!______three-character Great Lakes Basin identifier:
c                        SUP  (Superior)
c                        MIC  (Michigan)
c                        HUR  (Huron)
c                        STC  (St. Clair)
```

```
c                         ERI  (Erie)
c                         ONT  (Ontario)
c                         CHA  (Champlain)
c                         URA  (Ural River)
c                         EMB  (Embe River)
c
c Each of the provisional subbasin files is an unformatted,
c direct-access file with a record length of 6 bytes. The first
c record contains the basin/subbasin numerical identifier (e. g.
c 119 corresponds to basin 1, Superior, subbasin 19), the number
c of days in the file, and the starting date sequence number, in
c that order as three 2-byte integers. All other records contain
c Tmin, Tmax, and Precip in that order as three 2-byte integer
c numbers; each record corresponds to a day's data and they are
c present in chronological order.
c The provisional data file, PROVSNAL.xxx, is a direct-access
c linked list that must have blocks of data, one for each day, in
c chronological (logical) order. Every day between the beginning
c and end dates must be represented even if there are no data for
c that day. Each block must have individual station data arranged
c in an order that is unique for that group of stations; i. e.,
c every time the same set of stations reoccurs, they must be
c listed in exactly the same order. This is taken care of by
c PROVSNAL. The first record in a block contains the following
c information: Date, bLink, Rc, and Link (respectively: 2, 4, 4,
c and 4 bytes, unformatted). Date = sequence number of the date for
c this block, bLink = record number of header for next day's data,
c Rc = number of the last physical record, appearin in the first
c physical record only, otherwise a non-zero number (if negative,
c indicates "old" data previously acted upon by this program and
c therefore already present in the provisional subbasin data
c files; if positive, indicates "new" additional or changed data
c to be acted upon now in the update process),and Link = record
c number of first station's data for this block. The following
c records (one for each station) in the block contain the latitude
```

```
c (2 bytes), longitude (2 bytes), daily minimum and maximum air
c temperatures (2 bytes apiece), daily precipitation (2 bytes),
c and a 4-byte pointer to the next logical record, for a total of
c 14 bytes in an unformatted record.  Units are, respectively:
c decimal degrees times 100 north of equator, decimal degrees
c times 100 east of prime meridian (negative if west), degrees
c Celsius times 100, degrees Celsius times 100, and millimeters
c times 100 (depth).  Missing data in any of the last 3 data
c fields must be denoted by a value of -9999 for each of the
c missing values respectively.  If all data is missing for a
c station, then no records need be present in the data block.
c
c This program also notes the earliest "new" provisional data
c processed and appends the corresponding date to file:  BSNNM.
c for use insubsequent processing by other programs.
c
c This program is prepared for compilation by program PRDSAVMT.NI4
c and must be compiled before use each time.
c
c
c THIS PROGRAM MUST BE COMPILED TO DEFAULT TO 2-BYTE INTEGERS AND
c INTEGER CONSTANTS! SELECT THE "/T" SWITCH ON THE LAHEY FORTRAN
c COMPILER!
      IMPLICIT NONE
      INTEGER*2 Map_Width, Map_Height, Max_No_Sta, No_Subbasins,
     & Max_No_Networks, Hash_Size, Buffer_Length, Record_Length, NwStLg,
     & CellSize
      CHARACTER*3 Basin_Name
c
c 'Map_Width' is the number of the right column of the map (0 is the
c number of the left column of the map).  'Map_Height' is the number
c ofthe top row of the map (0 is the number of the bottom row of the
c map).
      PARAMETER (Map_Width = ************** filled in by PRDSAVMT *****
      PARAMETER (Map_Height = ************** filled in by PRDSAVMT ****
```

```
      PARAMETER (Max_No_Sta = **************** filled in by PRDSAVMT **
      PARAMETER (No_Subbasins = ************** filled in by PRDSAVMT **
      PARAMETER (Basin_Name = **************** filled in by PRDSAVMT **
      PARAMETER (Buffer_Length = ************* filled in by PRDSAVMT **
      PARAMETER (CellSize = ****************** filled in by PRDSAVMT **

      PARAMETER (Max_No_Networks = 1000)
      PARAMETER (Hash_Size = 1500)
      PARAMETER (Record_Length = Map_Width + 1)
      PARAMETER (NwStLg = Max_No_Sta*4)

      INTEGER*2 Q, Qmet(1:Max_No_Sta), Qqmet(1:Max_No_Sta, 1:3),
     & Qwmet(0:No_Subbasins, 1:3), Hash_Table(1:Hash_Size),
     & Chain_Table(1:Max_No_Networks), Last_Use(1:Max_No_Networks),
     & Error, Basin_Designator, iBuffer(Buffer_Length), Latitude,
     & Longitude, I, Total_No_Networks, Number, C2,
     & Ibn, No_Rec, K, Rce, No_Days,
     & Number_Of_Records, J, Sta_No, No_Sta, No_Networks,
     & Network_In_Memory, Date, Data_No, First_Day, First_Month,
     & First_Year, Tmin, Tmax, Precip, Subbasin, Symbol_Size,
     & Network_No, Earliest_Last_Use, Network_To_Delete, Last_Day,
     & Last_Month, Last_Year, Starts, Day, Month, Year,
     & SrtN(1:Max_No_Sta), N, Ntwrk_Sym_Sz(1:Max_No_Networks)
      INTEGER*4 Zero, zOne, B2, R2, L2, Rcp, Length
      REAL*4 Qx(1:Max_No_Sta), Qy(1:Max_No_Sta), Qqx(1:Max_No_Sta), Qq,
     & Qqy(1:Max_No_Sta), qCount(0:No_Subbasins), SrtX(1:Max_No_Sta),
     & SrtY(1:Max_No_Sta),
     & qThiessen_Weights(1:Max_No_Sta, 0:No_Subbasins),
     & qThsnWts(1:Max_No_Sta, 0:No_Subbasins + 2)
      CHARACTER CrLf*2, Bn$*3
      CHARACTER*1 zMap(0:Map_Width, 0:Map_Height)
      CHARACTER*(Record_Length) zMap_Record(0:Map_Height)
      CHARACTER*4 X$(1:Max_No_Sta), Y$(1:Max_No_Sta), Over_The_Lake
      CHARACTER*6 Buffer(Buffer_Length)
      CHARACTER*(NwStLg) Network_ID, Network_Symbol_To_Delete
```

```
      LOGICAL*1 Saved, First_New_Data, Found, Always

      COMMON /HshTbls/ Hash_Table, Chain_Table, Ntwrk_Sym_Sz
      COMMON /Weights/ qThsnWts, qCount
      COMMON /Digital/ zMap
      COMMON /Sta_Loc/ Qx, Qy, SrtX, SrtY, SrtN, N
      COMMON Latitude, Longitude
      EQUIVALENCE (zMap, zMap_Record, Buffer),
     & (qThsnWts, qThiessen_Weights, Buffer)

      Cr$ = CHAR(13)//CHAR(10)
      Bn$ = Basin_Name
      Zero = 0
      zOne = 1

      IF (Bn$ .EQ. 'SUP') Basin_Designator = 1
      IF (Bn$ .EQ. 'MIC') Basin_Designator = 2
      IF (Bn$ .EQ. 'HUR') Basin_Designator = 3
      IF (Bn$ .EQ. 'STC') Basin_Designator = 4
      IF (Bn$ .EQ. 'ERI') Basin_Designator = 5
      IF (Bn$ .EQ. 'ONT') Basin_Designator = 6
      IF (Bn$ .EQ. 'CHA') Basin_Designator = 7
      IF (Bn$ .EQ. 'URA') Basin_Designator = 8
      IF (Bn$ .EQ. 'EMB') Basin_Designator = 9

c Get starting date from PROVSNAL.xxx and check number of stations.
c
      OPEN(UNIT = 1, FILE = 'PROVSNAL.'//Bn$, STATUS = 'OLD', ACCESS
     & = 'DIRECT', RECL = 14, ERR = 900, FORM = 'UNFORMATTED')
c
c Inspect database to see if all are already processed data; exit
c normallyif no new data processing is required.
c
      Rcp = zOne
      Found = .FALSE.
```

```
c 'DoWhile' Construct ...
40 IF (Rcp .NE. Zero) THEN
   READ(1, REC = Rcp, ERR = 900) Number, B2, R2, L2
   Found = Found .OR. R2 .GT. Zero
   Rcp = B2
   GOTO 40
   ENDIF
   IF (.NOT. Found) THEN
   CLOSE(1)
   WRITE(6, 41)
41  FORMAT(1X, 'No new data present; Thiessen averages skipped...')
   CALL EXIT(0)
   ENDIF
   READ(1, REC = zOne, ERR = 900) Number, B2, R2, L2
   READ(1, REC = ABS(R2) + zOne, ERR = 900) R2
   No_Sta = ABS(R2)
   IF (No_Sta .GT. Max_No_Sta) THEN
   WRITE(6, 18) No_Sta
18  FORMAT(1X, 'Too many stations are present (', I3, ') for'
  &  ' processing.',
  &  /,1X,'Change "Maximum_Number_Of_Stations" in program: PRDSAVMT',
  &  /,1X,'and recompile.  Use it to build new source file for this',
  &  /,1X,'basin: DISAVMET.NI4 and recompile.  The Thiessen weights',
  &  /,1X,'data base will be rebuilt.')
   CLOSE (1)
   CALL EXIT(1)
   ENDIF
   Rcp = zOne

c Input subbasin areas and subbasin cell counts in map.
c
   OPEN(UNIT = 2, FILE = 'AREA.'//Bn$, STATUS = 'OLD')
   READ(2, 501) (qCount(i), i = 1, No_Subbasins)
501 FORMAT(E13.6E2)
   READ(2, 501) qCount(0)
```

```
      DO 8765 I = 0, No_Subbasins
      qCount(I) = qCount(I) / 1000000. / CellSize ** 2
 8765 CONTINUE
      CLOSE (2)
c
c  Initialize the number of networks computed thus far in this
c  routine. Set "Saved" TRUE to avoid saving uninitialized weights.
c
      Total_No_Networks = 0
      Saved = .TRUE.
c
c  Input the code map for the subbasins within the basin.
c
       OPEN(UNIT = 2, FILE = Bn$//'BYTCD.MAP', STATUS = 'OLD', RECL =
     & Record_Length, ACCESS = 'DIRECT', FORM = 'UNFORMATTED')
       DO 20 J = 0, Map_Height
       READ(2) zMap_Record(J)
   20  CONTINUE
       CLOSE (2)
c
c Get number of records and starting date from subbasin 01
c provisionaldata file.
c
       OPEN(UNIT = 7, FILE = Bn$//'01.PRV', STATUS = 'OLD', ACCESS
     & = 'DIRECT', RECL = 6, FORM = 'UNFORMATTED')
       READ(7, REC = 1) Ibn, No_Rec, K
       CLOSE (7)
       Always = No_Rec .EQ. 0 .OR. K .NE. Number
       IF (Always) THEN
       K = Number
       No_Rec = 0
       ENDIF
       No_Rec = No_Rec + 1
       Rcs = 2 + Number - K
       First_New_Data = .FALSE.
```

```
      No_Days = Number - K
      Number_Of_Records = W
c
c       Open the intermediate provisional subbasin data file.
c
      OPEN(UNIT = 5, STATUS = 'SCRATCH', ACCESS = 'DIRECT', RECL = 6,
     & FORM = 'UNFORMATTED')
c
c Open the intermediate Thiessen weight table file.
c
      OPEN(UNIT = 3, FILE = 'THSN_DAT.'//Bn$, STATUS = 'OLD', ACCESS
     & = 'DIRECT', RECL = (No_Subbasins + 1) * Max_No_Sta * 4, FORM
     & = 'UNFORMATTED', IOSTAT = Error)
      Found = Error .EQ. 0
c
c Old weights available, check for hash tables...
c
      IF (Found) THEN
      OPEN(UNIT = 7, FILE = 'THSN_HSH.' // Bn$, STATUS = 'OLD', ACCESS
     &  = 'DIRECT', RECL = 2, FORM = 'UNFORMATTED', IOSTAT = Error)
      Found = Error .EQ. 0
      ENDIF
c
c Old weights and hash tables exist, check dimensions...
c
      IF (Found) THEN
      READ(7) I
      READ(7) J
      READ(7) K
      READ(7) Sta_No
      READ(7) No_Sta
      READ(7) Ibn
      READ(7) No_Networks
      IF (I .NE. NwStLg .OR. J .NE. Hash_Size .OR.
     &  K .NE. Max_No_Networks .OR. Sta_No .NE. No_Subbasins .OR.
```

```
     &  No_Sta .NE. Max_No_Sta .OR. Ibn .NE. Basin_Designator) THEN
      CLOSE (7, STATUS = 'DELETE')
      Found = .FALSE.
      ENDIF
      ENDIF
C
C Old weights, hash tables, and dimensions O.K, check symbol table...
C
      IF (Found) THEN
      OPEN(UNIT = 2, FILE = 'THSN_SYM.'//Bn$, STATUS = 'OLD', ACCESS
     &  = 'DIRECT', RECL = NwStLg, FORM = 'UNFORMATTED', IOSTAT = Error)
      Found = Error .EQ. 0
      ENDIF
C
C Either initialize or read hash tables...
C
      IF (.NOT. Found) THEN
      CLOSE (7, STATUS = 'DELETE', IOSTAT = Error)
      CLOSE (3, STATUS = 'DELETE', IOSTAT = Error)
      OPEN(UNIT = 3, FILE = 'THSN_DAT.'//Bn$, STATUS = 'NEW', ACCESS
     &  = 'DIRECT', RECL = (No_Subbasins+1)*Max_No_Sta*4, FORM
     &  = 'UNFORMATTED')
      CLOSE(2, STATUS = 'DELETE', IOSTAT = Error)
      OPEN(UNIT = 2, FILE = 'THSN_SYM.'//Bn$, STATUS = 'NEW', ACCESS
     &  = 'DIRECT', RECL = NwStLg, FORM = 'UNFORMATTED')
      No_Networks = 0
      DO 3 I = 1, Hash_Size
      Hash_Table(I) = 0
 3    CONTINUE
      ELSE
      DO 502 I = 1, Hash_Size
      READ(7) Hash_Table(I)
502   CONTINUE
      DO 503 I = 1, Max_No_Networks
      READ(7) Chain_Table(I)
```

```
503  CONTINUE
     DO 504 I = 1, Max_No_Networks
     READ(7) Last_Use(I)
504  CONTINUE
     DO 505 I = 1, Max_No_Networks
     READ(7) Ntwrk_Sym_Sz(I)
505  CONTINUE
     CLOSE (7)
     Network_In_Memory = 0
     ENDIF
c
c    Open file for saving Thiessen-weight-computation history log.
c
     OPEN(UNIT = 4, FILE = 'HSTRYLOG.' // Bn$, STATUS = 'OLD',
   & ACCESS = 'DIRECT', RECL = 8, FORM = 'UNFORMATTED', ERR = 420)
420  CLOSE(4, STATUS = 'DELETE', ERR = 421)
421  OPEN(UNIT = 4, FILE = 'HSTRYLOG.' // Bn$, STATUS = 'NEW',
   & ACCESS = 'DIRECT', RECL = 8, FORM = 'UNFORMATTED', ERR = 420)
c
c    Read a block of data from PROVSNAL.xxx.
c
c    Rcp = 0 => at end-of-file
c
c    'DoWhile' Construct...
120  IF (Rcp .NE. Zero) THEN
     READ(1, REC = Rcp, ERR = 900) Date, B2, R2, L2
     No_Days = No_Days + 1
     IF (R2 .LT. Zero .AND. .NOT. Always) THEN
c
c    Flag = (old); skip block.
c
     Rcp = B2
c
c Preserve old values in provisional subbasin data files by skipping
c record.  (They must be there from a 'previous update').
```

```
c
    IF (Rcs .GT. No_Rec) GOTO 890
    Rcs = Rcs + 1
    CALL SequenceDate(Day, Month, Year, Date)
    WRITE(6, 10) Day, Month, Year
10   FORMAT(1H+, 'Processing:', 2I3, I5,
   &   '.                                        ')
    ELSE
c
c Flag = "new", must calculate weighted data.
c
    Rcp = L2
    Data_No = 0
    IF (.NOT. First_New_Data) THEN
    CALL SequenceDate(First_Day, First_Month, First_Year, Date)
    First_New_Data = .TRUE.
    ENDIF
    J = 1
c
c   At end of block if Rcp = B2.
c
c 'DoWhile' Construct...
B12   IF (Rcp .NE. B2) THEN
    READ(1, REC = Rcp, ERR = 900) Latitude, Longitude, Tmin, Tmax,
  &    Precip, Rcp
c
c   Concatenated binary Latitude & Longitude.
c
    CALL ID_String(X$(J))
    Qqx(J) = Latitude / 100.
    Qqy(J) = Longitude / 100.
    Qqmet(J, 1) = Tmin
    Qqmet(J, 2) = Tmax
    Qqmet(J, 3) = Precip
    J = J + 1
```

```
      GOTO 812
      ENDIF
      J = J - 1
c
c Convert station locations to map coordinates; not
c executed if J = 0.
c
      DO 130 Sta_No = 1, J
      CALL Compute_Map_Coordinates(Qqx(Sta_No), Qqy(Sta_No))
130   CONTINUE
c
c     Data_No = 1 => minimum temperature
c               2 => maximum temperature
c                  3 => precipitation
c
c        'DoWhile' Construct...
300   IF (Data_No .LT. 3) THEN
      Data_No = Data_No + 1
      CALL SequenceDate(Day, Month, Year, Date)
      WRITE(6, 30) Day, Month, Year, Data_No
30    FORMAT(1H+, 'Processing:', 2I3, I5,
     &  '; Data No. ', I1, '                                    )
      I = 1
      K = 0
c         'DoWhile' Construct...
112    IF (K .NE. J) THEN
        K = K + 1
       Q = Qqmet(K, Data_No)
       IF (Q .GT. -9000) THEN
       Y$(I) = X$(K)
       Qx(I) = Qqx(K)
       Qy(I) = Qqy(K)
       Qmet(I) = Q
       I = I + 1
       ENDIF
```

```
      GOTO 112
      ENDIF
      No_Sta = I - 1
c
c If all is missing for this data item, skip weighting and fill
c in as missing.
c
      IF (No_Sta .EQ. 0) THEN
      DO 311 Subbasin = 0, No_Subbasins
      Qwmet(Subbasin, Data_No) = -9999
311    CONTINUE
c
c If this is a 1-station network then do not compute new
c Thiessen weights since it is unnecessary to do so.  Simply
c use the value at the station as the weighted value.
c
    ELSE IF (No_Sta .EQ. 1) THEN
c
c Write to the history log file, an unformatted, direct access
c file read and written to sequentially.  Each record has the
c following structure:
c
c ddccnnii          (unformatted, stored as 4 2-byte integers)
c     !!!!!!!!
c     !!!!!!!!__________________network index (network number)
c     !!!!!!__________________Total_No_Networks
c     !!!!__________________Hundreds place is code:
c     !!                      0: network added
c     !!                      1: network deleted
c     !!                      2: 1-station "network" added & dropped
c     !!                      3: network recalled
c     !!                      4: network repeated
c     !!                      Tens place is Data_No
c     !!__________________Date
c
```

```
      WRITE(4) Date, Data_No + 200, Total_No_Networks, 0
      DO 312 Subbasin = 0, No_Subbasins
      Qwmet(Subbasin, Data_No) = NINT(Qmet(1))
312     CONTINUE
      ELSE
c
c Construct network identifier.
c
      Network_ID = Y$(1)
      DO 132 Sta_No = 2, No_Sta
      Network_ID = Network_ID(1:(Sta_No - 1)*4) // Y$(Sta_No)
132     CONTINUE
      Symbol_Size = No_Sta * 4
c
c Compare network identifier with networks already computed.
c
      CALL Find_Symbol(Network_ID, Symbol_Size, Found, Network_No)
      IF (.NOT. Found) THEN
      IF (No_Networks .EQ. Max_No_Networks) THEN
c
c Tables full, find least used...
c
      Earliest_Last_Use = 32767
      DO 4 Network_No = 1, Max_No_Networks
      IF (Earliest_Last_Use .GT. Last_Use(Network_No)) THEN
      Earliest_Last_Use = Last_Use(Network_No)
      Network_To_Delete = Network_No
      ENDIF
4       CONTINUE
      READ(2, REC = Network_To_Delete) Network_Symbol_To_Delete
      I = Ntwrk_Sym_Sz(Network_To_Delete)
      WRITE(4) Date, Data_No + 100, Total_No_Networks,
     &       Network_To_Delete
      CALL Delete_Symbol(Network_Symbol_To_Delete, I, Network_No)
      ELSE
```

```
      Network_No = No_Networks + 1
      No_Networks = Network_No
      ENDIF
      CALL Store_Symbol(Network_ID, Symbol_Size, Network_No)
c
c Save current weights before calculating new ones, if necessary.
c
      IF (.NOT. Saved) THEN
c
c Skip save if current weights are ones to be deleted; will only
c happen (current is least-used) if Max_No_Networks = 1.
c
      IF (Network_In_Memory .NE. Network_No) THEN
      WRITE(6, 11) Day, Month, Year, Data_No
11        FORMAT(1H+, 'Processing:', 2I3, I5,
  &         '; Data No. ', I1, '; saving old weights.          ')
           CALL Invalidate_Old_Weights(Bn$, Error)
           WRITE(3, REC = Network_In_Memory) qThiessen_Weights
          ENDIF
         ENDIF
      WRITE(6, 13) Day, Month, Year, Data_No, Total_No_Networks + 1
13      FORMAT(1H+, 'Processing:', 2I3, I5,
   &       '; Data No. ', I1, '; finding new weights ', I6, '.')
      WRITE(4) Date, Data_No, Total_No_Networks + 1, Network_No
      CALL Thiessen_Weights(No_Sta)
      Saved = .FALSE.
      Total_No_Networks = Total_No_Networks + 1
      Network_In_Memory = Network_No
      ENDIF
      IF (Network_No .NE. Network_In_Memory) THEN
      IF (.NOT. Saved) THEN
      WRITE(6, 11) Day, Month, Year, Data_No
      CALL Invalidate_Old_Weights(Bn$, Error)
      WRITE(3, REC = Network_In_Memory) qThiessen_Weights
      ENDIF
```

```
      WRITE(6, 12) Day, Month, Year, Data_No
12      FORMAT(1H+, 'Processing: ', 2I3, I5,
   &       '; Data No. ', I1, '; reading old weights.          ')
      WRITE(4) Date, Data_No + 300, Total_No_Networks, Network_No
      READ(3, REC = Network_No) qThiessen_Weights
      Network_In_Memory = Network_No
      Saved = .TRUE.
      ELSE
      IF (Found) THEN
      WRITE(6, 16) Day, Month, Year, Data_No
16       FORMAT(1H+, 'Processing: ', 2I3, I5,
   &       '; Data No. ', I1, '; using previous weights.       ')
      WRITE(4) Date, Data_No + 400, Total_No_Networks, Network_No
      ENDIF
      ENDIF
      Last_Use(Network_No) = Date
      DO 200 Subbasin = 0, No_Subbasins
      Qq = 0.
      DO 201 Sta_No = 1, No_Sta
      Qq = Qq + Qmet(Sta_No) * qThiessen_Weights(Sta_No, Subbasin)
201      CONTINUE
      Qwmet(Subbasin, Data_No) = NINT(Qq)
200     CONTINUE
      ENDIF
      GOTO 300
      ENDIF
c
c Write updated values to provisional subbasin data intermediate file.
c
      Number_Of_Records = Number_Of_Records + 1
      WRITE(5) Rcs
      DO 321 Subbasin = 0, No_Subbasins
      WRITE(5) (Qwmet(Subbasin, Data_No), Data_No = 1, 3)
321   CONTINUE
      IF (Rcs .GT. No_Rec) No_Rec = Rcs
```

```
      Rcs = Rcs + 1
      ENDIF
       GOTO 120
      ENDIF
c
c     End of PROVSNAL.xxx.
c
      CALL SequenceDate(Last_Day, Last_Month, Last_Year, Date)
c
c     Save Thiessen weights and network hash tables.
c
      WRITE(6, 14)
   14 FORMAT(1H+,
     &'Saving Thiessen weights and network hash tables.          ')
      IF (.NOT. Saved) THEN
       CALL Invalidate_Old_Weights(Bn$, Error)
       WRITE(3, REC = Network_In_Memory) qThiessen_Weights
      ENDIF
      CLOSE (3)
      CALL Invalidate_Old_Weights(Bn$, Error)
      OPEN(UNIT = 7, FILE = 'THSN_HSH.' // Bn$, STATUS = 'NEW',
     & ACCESS = 'DIRECT', RECL = 2, FORM = 'UNFORMATTED')
      WRITE(7) NwStLg
      WRITE(7) Hash_Size
      WRITE(7) Max_No_Networks
      WRITE(7) No_Subbasins
      WRITE(7) Max_No_Sta
      WRITE(7) Basin_Designator
      WRITE(7) No_Networks
      DO 506 I = 1, Hash_Size
       WRITE(7) Hash_Table(I)
  506 CONTINUE
      DO 507 I = 1, Max_No_Networks
       WRITE(7)* Chain_Table(I)
  507 CONTINUE
```

```
      DO 508 I = 1, Max_No_Networks
       WRITE(7) Last_Use(I)
  508 CONTINUE
      DO 509 I = 1, Max_No_Networks
       WRITE(7) Ntwrk_Sym_Sz(I)
  509 CONTINUE
      CLOSE (7)
      CLOSE (2)
      CLOSE (4)
c
c Read temporary intermediate storage file for  subbasin  weighted
c data and update individual subbasin files.  No_Days = number  of
c records supposed to be in the subbasin  provisional  data  files
c minus header; if there are more after the last day  added,  they
c are ignored (should never happen with normal use of the forecast
c package).
c
    DO 5 Subbasin = 0, No_Subbasins
    WRITE(6, 15) Bn$ // CHAR(Subbasin / 10 + 48) //
   & CHAR(Subbasin - (Subbasin / 10) * 10 + 48)
15  FORMAT(1H+, 'Updating subbasin provisional data file: ', A5,
   & '.PRV              ')
    OPEN(UNIT = 7, FILE = Bn$ // CHAR(Subbasin / 10 + 48) //
   & CHAR(Subbasin - (Subbasin / 10) * 10 + 48) // '.PRV', STATUS =
   & 'OLD', ACCESS = 'DIRECT', RECL = 6, FORM = 'UNFORMATTED')
    READ(7, REC = 1) Ibn, No_Rec, K
    I = Number_Of_Records
    J = I
    IF (J .GT. Buffer_Length) J = Buffer_Length
    K = Subbasin + 2
    B2 = 1
c 'DoWhile' Construct...
 7  IF (J .GT. 0) THEN
    I = I - J
    DO 8 C2 = 1, J
```

```
      READ(5, REC = B2) iBuffer(C2)
      READ(5, REC = K) Buffer(C2)
      B2 = B2 + No_Subbasins + 2
      K = K + No_Subbasins + 2
8     CONTINUE
      DO 9 C2 = 1, J
      WRITE(7, REC = iBuffer(C2)) Buffer(C2)
9     CONTINUE
      J = I
      IF (J .GT. Buffer_Length) J = Buffer_Length
      GOTO 7
      ENDIF
      WRITE(7, REC = 1) Ibn, No_Days, Number
      CLOSE (7)
5     CONTINUE
      CLOSE (5, STATUS = 'DELETE')
c
c Set flags to "old" for processed data.
c
      Rcp = zOne
c     'DoWhile' Construct ...
    2 IF (Rcp .NE. Zero) THEN
      READ(1, REC = Rcp, ERR = 900) Date, B2, R2, L2
      IF (R2 .GT. Zero) WRITE(1, REC = Rcp, ERR = 900) Date, B2, -R2,L2
      Rcp = B2
      GOTO 2
      ENDIF
      CLOSE (1)
c
c       Get "Length" of forecast.
c
      OPEN(UNIT = 1, FILE = 'LENGTH.' // Bn$, STATUS = 'OLD',
     & ACCESS = 'DIRECT', RECL = 52, FORM = 'FORMATTED')
      READ(1, FMT = 1000, REC = 1) Length
 1000 FORMAT(45X, I5)
```

```
      CLOSE (1)
c
c Store earliest changed date in file BSNNM. so that rest of
c forecast package knows when to begin updating of other files.
c
      OPEN(UNIT = 1, FILE = 'BSNNM.', STATUS = 'OLD',
     & ACCESS = 'DIRECT', RECL = 13, FORM = 'FORMATTED')
      READ(1, FMT = 1005, REC = 2) I, J, Over_The_Lake
c
c  3 = designator for this program.
c
      WRITE(1, FMT = 1005, REC = 2) I, 3, Over_The_Lake, CrLf
      CALL DateSequence(Last_Day, Last_Month, Last_Year, J)
      J = J + 1          ! J = Sequence number of 1st day of forecast.
      IF (.NOT. First_New_Data)
     & CALL SequenceDate(First_Day, First_Month, First_Year, J)
c
c   Provisional data end date
c
      WRITE(1, FMT = 1002, REC = 4) Last_Day,Last_Month,Last_Year,CrLf
1002 FORMAT(2I3, I5, A2)
c
c Determine ending date for forecast, "Length" FULL months after end
c of data.
c
      CALL SequenceDate(Last_Day, Last_Month, Last_Year, J)
      IF (Last_Day .EQ. 1) THEN
      Last_Month = Last_Month + Length
      ELSE
      Last_Month = Last_Month + Length + 1
      ENDIF
      Last_Year = Last_Year + (Last_Month - 1) / 12
      Last_Month = MOD(Last_Month - 1, 12) + 1
      Last_Day = 1
      CALL DateSequence(Last_Day, Last_Month, Last_Year, I)
```

```
      I = I - 1
      CALL SequenceDate(Last_Day, Last_Month, Last_Year, I)
      WRITE(1, FMT = 1002, REC = 5) Last_Day, Last_Month,Last_Year,CrLf
c
c Determine starting date for model statistics output (1st of month
c of forecast beginning minus 18 months and 1 day) and save in BSNNM.
c
      CALL SequenceDate(Last_Day, Last_Month, Last_Year, J)
      Last_Month = Last_Month + 18
      Last_Year = Last_Year - 3 + (Last_Month - 1) / 12
      Last_Month = MOD(Last_Month - 1, 12) + 1
      Last_Day = 1
      CALL DateSequence(Last_Day, Last_Month, Last_Year, J)
      CALL SequenceDate(Last_Day, Last_Month, Last_Year, J)
      WRITE(1,FMT=1002,REC = 6) Last_Day, Last_Month,Last_Year,CrLf
c
c Write earliest date sequence number of  added/changed  data  for
c other files, comparing it to the date already  there  for  them,
c and writing the earliest back.  This number  is  used  by  other
c programs that update those files as the beginning of the update;
c at that time, the other programs correct this number to  reflect
c the update.
c
      CALL DateSequence(First_Day, First_Month, First_Year, Starts)
c
c Date of earliest new data put by DISAVMET since WATOUT.NI4 was
c last run  to update  xxx%%.SUM.
c
      READ(1, FMT = 1002, REC = 7) Last_Day, Last_Month, Last_Year
      CALL DateSequence(Last_Day, Last_Month, Last_Year, Date)
      IF (Date .GT. Starts) Date = Starts
      CALL SequenceDate(Last_Day, Last_Month, Last_Year, Date)
      WRITE(1,FMT=1002,REC = 7) Last_Day, Last_Month, Last_Year,CrLf
c
c Date of earliest new data put by DISAVMET since LUMPSTOR.NI4 was
```

```
c last run to update xxxALL.SUM.
c
      READ(1, FMT = 1002, REC = 8) Last_Day, Last_Month, Last_Year
      CALL DateSequence(Last_Day, Last_Month, Last_Year, Date)
      IF (Date .GT. Starts) Date = Starts
      CALL SequenceDate(Last_Day, Last_Month, Last_Year, Date)
      WRITE(1,FMT=1002, REC = 8) Last_Day, Last_Month, Last_Year,CrLf
c
c Date of earliest new data put by DISAVMET since CMOFPOSR.NI4 was
c last run to update xxxALL.MET.
c
      READ(1, FMT = 1002, REC = 9) Last_Day, Last_Month, Last_Year
      CALL DateSequence(Last_Day, Last_Month, Last_year, Date)
      IF (Date .GT. Starts) Date = Starts
      CALL SequenceDate(Last_Day, Last_Month, Last_Year, Date)
      WRITE(1, FMT = 1002, REC = 9) Last_Day, Last_Month, Last_Year,CrLf
      CLOSE (1)

      WRITE(6, 2005) Total_No_Networks
 2005 FORMAT(1H+, 'Successful completion of DISAVMET:', I6,
     &        ' New Thiessen networks computed! ')
      CALL EXIT(0)
c
c       Error handling...
c
c    Problems with PROVSNAL.xxx or with xxxXX.PRV.
c
 890 WRITE(6, 2001) Bn$
 2001 FORMAT(1X, 'End-of-file read before expected on old provisional',
     & /, 1X, 'subbasin data file: ', a3, 'XX.PRV',
     & /, 1X, 'Updating aborted!')
      GOTO 891

 900 WRITE(6, 2000) Bn$
 2000 FORMAT(1X, 'Data file, PROVSNAL.', A3,
```

```
     & ' is not present or is in error.', /, 1X, 'Updating aborted!')
 891 OPEN(UNIT = 2, FILE = 'BSNNM.', STATUS = 'OLD',
     &      ACCESS = 'DIRECT', RECL = 13, FORM = 'FORMATTED')
      READ(2, FMT = 1005, REC = 2) I, J, Over_The_Lake
1001 FORMAT(A10)
1005 FORMAT(2I3, A4, 1X, A2)
c
c     Invalidate use of any programs.
c
     WRITE(2, FMT = 1005, REC = 2) I, -1, Over_The_Lake, CrLf
     CLOSE (2)
     CALL EXIT(1)
     END

     SUBROUTINE Invalidate_Old_Weights(Bn$, Error)
     IMPLICIT NONE
     INTEGER*2 Error
     CHARACTER*3 Bn$
     OPEN(UNIT=7,FILE = 'THSN_HSH.' // Bn$, STATUS = 'OLD', ACCESS
    & = 'DIRECT', RECL = 2, FORM = 'UNFORMATTED', IOSTAT = Error)
     CLOSE (7, STATUS = 'DELETE', IOSTAT = Error)
     RETURN
     END

     INTEGER*2 FUNCTION Hash(Symbol, K, Hash_Size)
c
c This function returns the hash code for a particular symbol given
c the symbol length (K) and hash size.
c
     IMPLICIT NONE
     INTEGER*2 Max_No_Sta, NwStLg
     PARAMETER (Max_No_Sta =********* filled in by PRDSAVMT ****
     PARAMETER (NwStLg = Max_No_Sta * 4)

     CHARACTER*(NwStLg) Symbol
```

```
      INTEGER*2 K, Hash_Size

      INTEGER*2 J, I, Bit_No, Byte_No, M

      J = 0
      DO 1 I = 0, 14
      Bit_No = I + (K * 9 / 15)
      Byte_No = Bit_No / 8 + 1
      Bit_No = MOD(Bit_No, 8)
      M = IAND(ICHAR(Symbol(Byte_No:Byte_No)), ISHFT(128, -Bit_No))
      IF (M .NE. 0) J = IOR(J, ISHFT(1, I))
    1 CONTINUE
      Hash = MOD(J, Hash_Size) + 1
      RETURN
      END

      SUBROUTINE Find_Symbol(S, Ss, Found, Indecs)
c
c This routine searches for a given symbol in the hashed symbol
c table and, if it is found, returns Found = .TRUE. and its index
c in the symbol table. S is the symbol to store. Hash_table is
c the table to hash to; its contents point to positions in the
c Symbol_Table (on disk) and in the Chain_table. If a symbol
c hashes to the same location in the Hash_Table, then it will
c exist in the symbol table along the links established by the
c Chain_Table.
c
      IMPLICIT NONE
      INTEGER*2 Max_No_Sta, NwStLg, Max_No_Networks, Hash_Size
      PARAMETER (Max_No_Sta = *********** filled in by PROSAVMT ****
      PARAMETER (NwStLg = Max_No_Sta * 4)
      PARAMETER (Max_No_Networks = 1000)
      PARAMETER (Hash_Size = 1500)

      CHARACTER*(NwStLg) S
```

```
      INTEGER*2 Ss, Index
      LOGICAL*1 Found

      INTEGER*2 Hash_Table(0:Hash_Size-1), Table(1:Max_No_Networks),
     X Hash, Network_Number, No_Networks
      LOGICAL*1 Found(Max_Lgth_Sta)
      COMMON /Hash/Hash_Table, Table, Hash_Size

      Index = Hash_Table(Hash(Symbol, Symbol_size))
      IF (Index .EQ. 0) THEN
      Found = .FALSE.
      ELSE
      Start = .TRUE.
        DOWHILE (Index .
     X  .NE. 0 .AND. Table(Index) .NE. 0) THEN
      Index = Chain_Pointer(Index)
      READ(2, REC = Index) Sta
      GOTO 1
      ENDIF
      Found = S .EQ. Sta
      ENDIF
      RETURN
      END

      SUBROUTINE Store_Symbol(Symbol, Symbol_size, Index)
c
c This routine stores a symbol in the given hashed  symbol  table.
c See comments in the procedure: Find_Symbol.  Index is the place
c in the tables to store the symbol.  THIS ROUTINE DOES NOT  CHECK
c TO SEE IF SYMBOL IS ALREADY IN THE  TABLES  NOR  IF  THE  TABLES
c OVERFLOW!
c
      IMPLICIT NONE
      INTEGER*2 Max_No_Sta, NwStLg, Max_No_Networks, Hash_Size
      PARAMETER (Max_No_Sta = *********** filled in by PRDSAVMT ****
```

```
      PARAMETER (NwStLg = Max_No_Sta * 4)
      PARAMETER (Max_No_Networks = 1000)
      PARAMETER (Hash_Size = 1500)

      CHARACTER*(NwStLg) Symbol
      INTEGER*2 Symbol_Size, Indecs

      INTEGER*2 I, Hash, Hash_Table(1:Hash_Size),
     & Chain_Table(1:Max_No_Networks), Ntwrk_Sym_Sz(1:Max_No_Networks)
      COMMON /HshTbls/ Hash_Table, Chain_Table, Ntwrk_Sym_Sz

      WRITE(2, REC = Indecs) Symbol              ! Store in symbol table
      Ntwrk_Sym_Sz(Indecs) = Symbol_Size
      I = Hash(Symbol, Symbol_size, Hash_size)
      IF (Hash_Table(I) .EQ. 0) THEN
      Chain_Table(Indecs) = 0                    ! Start new chain
      ELSE
      Chain_Table(Indecs) = Hash_Table(I)        ! Insert in chain
      ENDIF
      Hash_Table(I) = Indecs                     ! Store in hash table
      RETURN
      END

      SUBROUTINE Delete_Symbol(S, Ss, Indecs)
c
c This routine finds and deletes a given symbol  from  the  hashed
c symbol table and returns the index in the symbol  table  of  the
c new vacancy.    See  comments  in  the  procedure:  Find_Symbol.
c ASSUMES THAT SYMBOL EXISTS IN THE HASHED  SYMBOL  TABLE;  IT  IS
c UNCHECKED HERE!!!
c
      IMPLICIT NONE
      INTEGER*2 Max_No_Sta, NwStLg, Max_No_Networks, Hash_Size
      PARAMETER (Max_No_Sta = ********* filled in by PRDSAVMT ******
      PARAMETER (NwStLg = Max_No_Sta * 4)
```

```
      PARAMETER (Max_No_Networks = 1000)
      PARAMETER (Hash_Size = 1500)

      CHARACTER*(NwStLg) S
      INTEGER*2 Ss, Indecs

      INTEGER*2 Hash_Table(1:Hash_Size),Chain_Table(1:Max_No_Networks),
     & Index_Save, Hash, Ntwrk_Sym_Sz(1:Max_No_Networks)
      CHARACTER*(NwStLg) Ste
      COMMON /HshTbls/ Hash_Table, Chain_Table, Ntwrk_Sym_Sz

      Index_Save = Hash(S, Ss, Hash_Size)
      Indecs = Hash_Table(Index_Save)
      READ(2, REC = Indecs) Ste
      IF (S .EQ. Ste) THEN
      Hash_Table(Index_Save) = Chain_Table(Indecs)
      ELSE
c 'DoWhile' Construct...
 1    IF (S .NE. Ste) THEN
      Index_Save = Indecs
      Indecs = Chain_Table(Indecs)
      READ(2, REC = Indecs) Ste
      GOTO 1
      ENDIF
      Chain_Table(Index_Save) = Chain_Table(Indecs)
      ENDIF
      RETURN
      END

      SUBROUTINE Thiessen_Weights(No_Stations)
c
c This routine calculates Thiessen weights for all meteorological
c stations, for each subbasin, for all subbasins.
c
c Inputs are: basin map, zMap(0:Map_Width,0:Map_Height)
```

```
c                 where  zMap(I, J)  is array of codes,
c                                 one code for each subbasin
c                 I  is the x-coordinate of an area on the map
c                 J  is the y-coordinate of an area on the map
c            number of meteorological stations in network, No_Sta
c             station coordinates, Qx(1:No_Sta), Qy(1:No_Sta)
c                where  Qx(Sta_No)  is the x-coordinate of station
c                Qy(Sta_No)  is the y-coordinate of station
c                   'Sta_No'  is station index between 1 and No_sta
c Outputs are:
c weight table, qThiessen_Weights(0:No_Subbasins, 1:No_Sta)
c          where 'qthiessen_weights(subbasin, Sta_No)
c          is the Thiessen weight for indicated
c          subbasin and station
c
c
c Thiessen weights are  determined by  computing  polygon maps
c (one at a time and disposing of each before  getting the next)
c and  counting the codes    of each subbasin in the  polygon
c row 'windows  (between left and right columns in each row).
c
      IMPLICIT NONE
      INTEGER*2 Map_Width, Map_Height, Max_No_Sta, No_Subbasins,
     & Dummy_Code

      PARAMETER (Map_Width = ******** filled in by PRDSAVMT *******
      PARAMETER (Map_Height = ******** filled in by PRDSAVMT ******
      PARAMETER (Max_No_Sta = ******** filled in by PRDSAVMT ******
      PARAMETER (No_Subbasins = ****** filled in by PRDSAVMT ******
      PARAMETER (Dummy_Code = No_Subbasins + 1)

      CHARACTER*1 zMap(0:Map_Width, 0:Map_Height)
      REAL*4 qThiessen_Weights(1:Max_No_Sta, 0:No_Subbasins + 2),
     & qCount(0:No_Subbasins), Q
      INTEGER*4 Qtw(1:Max_No_Sta, 0:No_Subbasins + 2), Qk, Qkk
```

```
      INTEGER*2 Lrow(0:Map_Height), Rrow(0:Map_Height), Sta_No, No_Sta,
     & Y1, Yu, No_Stations, Subbasin, No_Sta_Minus_One, L, R, I, J
C
C NOTE: The count array, "Qtw" above, and the  variables,  Qk  and
C Qkk below, must be typed  as  INTEGER*4  so  that  the  counting
C additions in the double DO-LOOP 2 below  and  the  summation  in
C DO-LOOP 205 are very fast and "Qtw" is dimensionally  compatable
C with "qThiessen Weights".  If INTEGER*4 is unavailable, then use
C REAL*4.
C
      COMMON /Weights/ qThiessen_Weights, qTmm
      COMMON /Digital/ zMap
      COMMON /Polygon/ Sta_No, No_Sta, Y1, Yu, Lrow, Rrow
      EQUIVALENCE (qThiessen_Weights, Qtw)

      No_Sta = No_Stations
      IF (No_Sta .EQ. 1) THEN
      DO 1 Subbasin = 0, No_Subbasins
      qThiessen_Weights(1, Subbasin) = 1.0
 1    CONTINUE
      ELSE
      No_Sta_Minus_One = No_Sta - 1
      DO 2 Sta_No = 1, No_Sta_Minus_One
      DO 3 Subbasin = 0, No_Subbasins + 2
      Qtw(Sta_No, Subbasin) = 0
 3     CONTINUE
      CALL Thiessen_Polygon
      DO 4 J = Y1, Yu
      L = Lrow(J)
      R = Rrow(J)
      DO 5 I = L, R
      Subbasin = ICHAR(zMap(I, J))
      Qtw(Sta_No, Subbasin) = Qtw(Sta_No, Subbasin) + 1
5      CONTINUE
4     CONTINUE
```

```
 2  CONTINUE
    DO 6 Sta_No = 1, No_Sta_Minus_One
    Qtw(Sta_No, 0) = Qtw(Sta_No, 0) + Qtw(Sta_No, Dummy_Code)
 6  CONTINUE
    DO 7 Subbasin = 0, No_Subbasins
    Qk = 0
    Q = qCount(Subbasin)
    IF (Q .EQ. 0) THEN
    Q = 1.0
    Qk = 1.0
    ENDIF
    DO 8 Sta_No = 1, No_Sta_Minus_One
    Qkk = Qtw(Sta_No, Subbasin)
    Qk = Qk + Qkk
    qThiessen_Weights(Sta_No, Subbasin) = Qkk / Q
8   CONTINUE
    qThiessen_Weights(No_Sta, Subbasin) = 1. - Qk / Q
 7  CONTINUE
    ENDIF
    RETURN
    END

      SUBROUTINE Thiessen_Polygon
c
c     This routine generates a map of the Thiessen polygon for the
c     given station.  Only the polygon edges are identified.
c
c Inputs are:  station index, Sta_No
c              total number of stations, No_Sta
c         map coordinates of each station (expressed in map units),
c              Qx(1:No_Sta) and Qy(1:No_Sta)
c Outputs are:
c polygon map expressed as left and right abcissa for each
c ordinate, Lrow(0:Map_Height) and Rrow(0:Map_Height)
c map upper most and lower most ordinate values, Yu and Yl
```

```
c
c
c Polygon edges are found by first locating an  initial  point  on
c the edge of the polygon, or in the polygon but on  the  edge  of
c the map, and then searching along the edge (of  the  polygon  or
c map, whichever dominates) to establish other edge points.    The
c search pattern is initially north, east, south, and west  (NESW)
c until the next point is found in both the map and polygon.   The
c search pattern is rotated (ESWN, SWNE, WNES, etc.)     whenever
c the search returns to the previous point found (no  next  point
c found) and the search  continues  until  the  edge  is  completely
c traversed.   A point is identified as being in the polygon  if
c the polygon station is the closest of all stations to it.
c
      IMPLICIT NONE
      INTEGER*2 Map_Width, Map_Height, Max_No_Sta

      PARAMETER (Map_Width = ******** filled in by PRDSAVMT *********
      PARAMETER (Map_Height = ***** filled in by PRDSAVMT *********
      PARAMETER (Max_No_Sta = ****** filled in by PRDSAVMT *********
      REAL*4 Qx(1:Max_No_Sta), Qy(1:Max_No_Sta), SrtX(1:Max_No_Sta),
     & SrtY(1:Max_No_Sta), V
      INTEGER*2 Search_Pattern_X(1:4, 0:5), Search_Pattern_Y(1:4, 0:5),
     & Lrow(0:Map_Height), Rrow(0:Map_Height), Sta_No, No_Sta, Yl, Yu,
     & No_Pattern_Rotations, N, J, I, StaNo, iStart, jStart, iLast,
     & jLast, iFirst, jFirst, Ii, Jj, Search, SrtN(1:Max_No_Sta)
      LOGICAL*1 First_Cell, Closest_Station
      COMMON /Sta_Loc/ Qx, Qy, SrtX, SrtY, SrtN, N
      COMMON /Polygon/ Sta_No, No_Sta, Yl, Yu, Lrow, Rrow
c
c Initialize search pattern and number of pattern rotations
c
      DATA Search_Pattern_X /  0,  1,  0, -1,
     &                         1,  0, -1,  0,
     &                         0, -1,  0,  1,
```

```
     &                          -1,  0,  1,  0,
     &                           0,  1,  0, -1,
     &                           1,  0, -1,  0/
      DATA Search_Pattern_Y /    1,  0, -1,  0,
     &                           0, -1,  0,  1,
     &                          -1,  0,  1,  0,
     &                           0,  1,  0, -1,
     &                           1,  0, -1,  0,
     &                           0, -1,  0,  1/
      No_Pattern_Rotations = 0
      N = No_Sta
c
c Initialize upper most and lower most ordinates to null map.
c
      Yl = Map_Height
      Yu = 0
c
c   Initialize left and right row limits to null map.
c
      DO 99 J = 0, Map_Height
      Lrow(j) = Map_Width
      Rrow(j) = 0
 99   CONTINUE
c
c Store station coordinates and numbers in order of closest
c to Sta_No.
c
      DO 1 I = 1, No_Sta
      SrtX(I)=(Qx(I)-Qx(Sta_No)) ** 2 + (Qy(I) - Qy(Sta_No)) ** 2
      SrtN(I) = I
    1 CONTINUE
      DO 2 I = 1, No_Sta - 1
      StaNo = SrtN(I)
      V = SrtX(I)
      DO 3 J = I + 1, No_Sta
```

```
      IF (SrtX(J) .LT. V) THEN
          SrtX(I) = SrtX(J)
          SrtX(J) = V
          V = SrtX(I)
          SrtN(I) = SrtN(J)
          SrtN(J) = StaNo
          StaNo = SrtN(I)
         ENDIF
    3  CONTINUE
    2 CONTINUE
      DO 4 I = 1, No_Sta
       SrtX(I) = Qx(SrtN(I))
       SrtY(I) = Qy(SrtN(I))
    4 CONTINUE
c
c Find map or off-map location of station and find an initial
c point on the map, on the polygon edge with nothing to the north.
c
c NOTE: Since station coordinates are already shifted to economize
c on distance calculations, we must "unshift" them to determine
c which cell that the station is in. Alternatively, we could use
c the shifted coordinates and routine Closest_Station to see which
c cell is closest to the station (which would be the cell that the
c station is in); but that is too much work.
c
      I = Qx(Sta_No) + 0.5
      J = Qy(Sta_No) + 0.5

      IF (I .GT. Map_Width) THEN
       IF (J .GT. Map_Height) THEN
c       Station is northeast of map...
        GOTO 981
       ELSE
        IF (J .GE. 0) THEN
c        Station is east of map...
```

```
          GOTO 971
         ELSE
c         Station is southeast of map...
          GOTO 961
         ENDIF
        ENDIF
       ELSE
        IF (I .GE. 0) THEN
         IF (J .GT. Map_Height) THEN
c         Station is north of map...
          GOTO 951
         ELSE
          IF (J .GE. 0) THEN
c          Station is in map...
           GOTO 941
          ELSE
c          Station is south of map...
           GOTO 931
          ENDIF
         ENDIF
        ELSE
         IF (J .GT. Map_Height) THEN
c         Station is northwest of map...
          GOTO 921
         ELSE
          IF (J .GE. 0) THEN
c          Station is west of map...
           GOTO 911
          ELSE
c          Station is southwest of map...
           GOTO 901
          ENDIF
         ENDIF
        ENDIF
       ENDIF
```

```
c
c Search from northwest corner east and continue if nothing found.
c
  981 I = 0
      J = Map_Height
c     'DoWhile' Construct...
  982 IF (I .LE. Map_Width) THEN
       IF (Closest_Station(I, J)) GOTO 999
       I = I + 1
       GOTO 982
      ENDIF
c
c Search from northeast corner south and exit if nothing found.
c
  971 I = Map_Width
      J = Map_Height
c     'DoWhile' Construct...
  983 IF (J .GE. 0) THEN
       IF (Closest_Station(I, J)) GOTO 999
       J = J - 1
       GOTO 983
      ENDIF
      RETURN
c
c Search from northeast corner south and continue if nothing found.
c
  961 I = Map_Width
      J = Map_Height
c     'DoWhile' Construct...
  962 IF (J .GE. 0) THEN
       IF (Closest_Station(I, J)) GOTO 999
       J = J - 1
       GOTO 962
      ENDIF
c
```

```
c     Search from southeast corner west and exit if nothing found.
c
      I = Map_Width
      J = 0
c     'DoWhile' Construct...
  963 IF (I .GE. 0) THEN
       IF (Closest_Station(I, J)) GOTO 999
       I = I - 1
       GOTO 963
      ENDIF
      RETURN
c
c Search from southwest corner east and exit if nothing found or
c continue if something found.
c
  931 I = 0
      J = 0
c     'DoWhile' Construct...
  932 IF (I .LE. Map_Width) THEN
       IF (Closest_Station(I, J)) GOTO 941
       I = I + 1
       GOTO 932
      ENDIF
      RETURN
c
c     Search north until out of polygon.
c
c     'DoWhile' Construct...
  941 IF (Closest_Station(I, J)) THEN
       J = J + 1
       IF (J .GT. Map_Height) GOTO 942
       GOTO 941
      ENDIF
c
c     Go back into polygon.
```

```
c
  942 J = J - 1
      GOTO 999
c
c Search from northwest corner south and continue if nothing found.
c
  901 I = 0
      J = Map_Height
c     ...
  902 IF (J .GE. 0) THEN
       IF (Closest_Station(I, J)) GOTO 999
       J = J - 1
       GOTO 902
      ENDIF
c
c     Search from southwest corner east and exit if nothing found.
c
      I = 0
      J = 0
c     'DoWhile' Construct...
  903 IF (I .LE. Map_Width) THEN
       IF (Closest_Station(I, J)) GOTO 999
       I = I + 1
       GOTO 903
      ENDIF
      RETURN
c
c Search from northwest corner east and continue if nothing found.
c
  921 I = 0
      J = Map_Height
c     'DoWhile' Construct...
  922 IF (I .LE. Map_Width) THEN
       IF (Closest_Station(I, J)) GOTO 999
       I = I + 1
```

```
      GOTO 922
      ENDIF
c
c Search from northwest corner south and exit if nothing found.
c
  911 I = 0
      J = Map_Height
      GOTO 983
c
c     Search from northwest corner east and exit if nothing found.
c
  951 I = 0
      J = Map_Height
      GOTO 903
c
c     Edge of polygon (or map) reached, store coordinates
c     and initialize lower-most and upper-most ordinates.
c
  999 iStart = I
      jStart = J
      Yl = J
      Yu = J
      Lrow(J) = I
      Rrow(J) = I
c
c     Initialize "last-point-found" and "first-point-found"
c     coordinates to impossible values and set First_Cell.
c
      iLast = Map_Width * 2
      jLast = Map_Height * 2
      iFirst = Map_Width * 2
      jFirst = Map_Height * 2
      First_Cell = .TRUE.
c
c     Search for edge in clockwise pattern.
```

```
c
c      'DoWhile' Construct...
    5 IF (No_Pattern_Rotations .LT. 6) THEN
        Ii = I
        Jj = J
        Search = 1
c       'DoWhile' Construct...
    6   IF (Search .LT. 4) THEN
         Ii = Ii + Search_Pattern_X(Search, No_Pattern_Rotations)
         Jj = Jj + Search_Pattern_Y(Search, No_Pattern_Rotations)
       IF (Ii.LE.Map_Width.AND. Ii .GE. 0 .AND. Jj .LE. Map_Height
     &   .AND. Jj .GE. 0) THEN
          IF (Closest_Station(II, JJ)) THEN
c
c Found next edge point; test to see if first and store if not.
c
            I = Ii
            J = Jj
            IF (iLast .EQ. iStart .AND. jLast .EQ. jStart .AND.
     &       I .EQ. iFirst .AND. J .EQ. jFirst) RETURN
            IF (First_Cell) THEN
             iFirst = I
             jFirst = J
             First_Cell = .FALSE.
            ENDIF
            IF (J .LT. Yl) Yl = J
            IF (J .GT. Yu) Yu = J
            IF (I .LT. Lrow(J)) Lrow(J) = I
            IF (I .GT. Rrow(J)) Rrow(J) = I
            iLast = I
            jLast = J
            Search = 0
           ENDIF
          ENDIF
          Search = Search + 1
```

```
      GOTO 6
     ENDIF
C
C Rotate search pattern; if search pattern  changed  more  than  5
C times, have degenerate case  of  one  point  is  only  thing  on
C polygon edge and on map.
C
      No_Pattern_Rotations = No_Pattern_Rotations + 1
      GOTO 5
     ENDIF
     RETURN
     END

     LOGICAL*1 FUNCTION Closest_Station(I, J)
C
C This routine returns .TRUE. if the closest station to point  (I,
C J) is Sta_No and .FALSE. if not; assumes the Sta_No is the first
C station represented in the arrays: SrtX, SrtY, and SrtN, defined
C in the calling routine.  If the arrays represent all stations in
C the network sorted in order of distance from Sta_No,  then  this
C routine will find .FALSE.'s faster than if  not  ordered.    The
C structure of the IF statement herein is  designed  to  guarantee
C this routine always gives unique assignment of  closest  station
C for any (and all) points no matter what the value of Sta_No;  i.
C e., if this routine returns .TRUE. for one station, then it will
C return .FALSE. for all other stations for the same point (I, J).
C This is a consideration only when a point is equidistant from two
C stations, a rare but possible problem.
C
     IMPLICIT NONE
     INTEGER*2 Max_No_Sta

    PARAMETER (Max_No_Sta = ******* filled in by PRDSAVMT *********

    REAL*4 X(1:Max_No_Sta), Y(1:Max_No_Sta), SrtX(1:Max_No_Sta),
```

```
     & SrtY(1:Max_No_Sta), W, V
       INTEGER*2 I, J, No_Sta, StaNo, SrtN(1:Max_No_Sta)
       COMMON /Sta_Loc/ X, Y, SrtX, SrtY, SrtN, No_Sta

       W = (SrtX(1) - I) ** 2 + (SrtY(1) - J) ** 2
       Closest_Station = .FALSE.
       DO 1 StaNo = 2, No_Sta
       V = (SrtX(StaNo) - I) ** 2 + (SrtY(StaNo) - J) ** 2
       IF (V .LE. W) THEN
       IF (V .EQ. W) THEN
       IF (SrtN(StaNo) .LT. SrtN(1)) RETURN
       ELSE
       RETURN
       ENDIF
       ENDIF
     1 CONTINUE
       Closest_Station = .TRUE.
       RETURN
       END

         SUBROUTINE Compute_Map_Coordinates(X, Y)
c
c This routine accepts the latitude and longitude of a point and
c returns the x and y map coordinates of the point representing
c the 'horizontal' and 'vertical' distances, respectively, from
c an 'origin' on the map of (0.5, 0.5).
c
c The map is a polyconic projection about a central meridian
c (88.0000 degrees west of the prime meridian for Superior).
c
c On entry,
c x is latitude in degrees and decimal degrees, positive from the
c equator north.
c y is longitude in degrees and decimal degrees, positive from the
c prime meridian east (negative west of the p. m.).
```

```
c
c On exit,
c x is distance in kilometers from point (0.5, 0.5) right.
c y is distance in kilometers from point (0.5, 0.5) up.
c
      IMPLICIT NONE
      REAL*4 R_M, X_Offset, Y_Offset, Radius

      PARAMETER (R_M = ********** filled in by PRDSAVMT *********
      PARAMETER (X_Offset = ***** filled in by PRDSAVMT *********
      PARAMETER (Y_Offset = ***** filled in by PRDSAVMT *********
      PARAMETER (Radius = *** filled in by PRDSAVMT *********

      REAL*4 X, Y, Lat, Lon, Tp, Snpl
c
c Get latitude in radians and longitude in radians from reference
c meridian.
c
      Lat = X / 57.29578
      Lon = (Y - R_M) / 57.29578
      Tp = TAN(Lat)
      Snpl = SIN(Lat) * Lon
      X = Radius * SIN(Snpl) / Tp
      Y = Radius * (Lat + (1. - COS(Snpl)) / Tp)
      X = X + X_Offset
      Y = Y + Y_Offset
      RETURN
      END

      SUBROUTINE ID_String(St_ID)
c
c This routine returns the concatenated binary image of the two
c 2-byte integers, contained in the unlabeled common block, as a
c 4-byte string. An effort is made to enable subsequent ordering
c by latitude first and then longitude.  Two-byte integers are
```

```
c stored with least significant byte first; these are then
c switched in order here for the i. d. string.
c
      IMPLICIT NONE
      CHARACTER*4 St_ID, StaID
      COMMON StaID
      St_Id = StaID(2:2) // StaID(1:1) // StaID(4:4) // StaID(3:3)
      RETURN
      END

      SUBROUTINE DateSequence(Day, Month, Year, SequenceNumber)
c
c This routine computes the number of the date, with days numbered
c sequentially from -32768, corresponding to 12 June 1898, through
c 32767, corresponding to 15 November 2077.  All variables,
c constants, and arithmetic are 16-bit integer only.
c
    IMPLICIT NONE
    INTEGER*2 Day, Month, Year, SequenceNumber
    IF (Year .LT. 1900 .OR. (Year .EQ. 1900 .AND. Month .LT. 3)) THEN
    CALL DS(Day, Month, Year, SequenceNumber, 1896)
      SequenceNumber = SequenceNumber - 835
      SequenceNumber = SequenceNumber - 32767
      RETURN
    ENDIF
    IF (Year .LT. 1988 .OR. (Year .EQ. 1988 .AND. Month .LT. 3)) THEN
    CALL DS(Day, Month, Year, SequenceNumber, 1900)
    SequenceNumber = SequenceNumber - 32142
    RETURN
      ENDIF
      CALL DS(Day, Month, Year, SequenceNumber, 1988)
      RETURN
      END

      SUBROUTINE DS(Day, Month, Year, SequenceNumber, BaseYear)
```

```
c
c This routine computes the number of the date, with days numbered
c sequentially from 1 (corresponding to the base date).  The  base
c date must be 1 March 1900 or 1 March on  any  LEAP  YEAR  not
c greater than 2010.  Dates must be between the base date and the
c base date plus 32766.  [I. e., the logic is  correct  only  for
c dates post-1900 and pre-2100 since 1900 and 2100 are  not  leap
c years but 2000 is.  Thus, tests for non-leap years  on  century
c boundaries except those divisible by 400 which are leap  years,
c are avoided.]  This routine is set up to use only 2-byte integer
c variables and  to  avoid  intermediate  use  of  reals  in  the
c evaluation of expressions.  See Dr. Dobb's Journal 80:66-70.
c
      IMPLICIT NONE
      INTEGER*2 Day, Month, Year, SequenceNumber, BaseYear
      SequenceNumber = Year - BaseYear - 1 + (Month + 9) / 12
      SequenceNumber = 365 * SequenceNumber + SequenceNumber / 4
     &                    + (153 * MOD(Month + 9, 12) + 2) / 5 + Day
      RETURN
      END

      SUBROUTINE SequenceDate(Day, Month, Year, SequenceNumber)
c
c This routine computes the date of the number, with days numbered
c sequentially from -32768, corresponding to 12 June 1898, through
c 32767, corresponding  to  15  November  2077.   All  variables,
c constants, and arithmetic are 16-bit integer only.
c
      IMPLICIT NONE
      INTEGER*2 Day, Month, Year, SequenceNumber, Intermediate
      IF (SequenceNumber .LT. -32141) THEN
         Intermediate = SequenceNumber + 32767
         Intermediate = Intermediate + 835
         CALL SD(Day, Month, Year, Intermediate, 1896)
         RETURN
```

```
      ENDIF
      IF (SequenceNumber .LE. 0) THEN
         Intermediate = SequenceNumber + 32142
         CALL SD(Day, Month, Year, Intermediate, 1900)
         RETURN
      ENDIF
      CALL SD(Day, Month, Year, SequenceNumber, 1900)
      RETURN
      END

      SUBROUTINE SD(Day, Month, Year, SequenceNumber, BaseYear)
c
c This routine computes the date of the number, with days numbered
c sequentially from 1 (corresponding to the base date).  The  base
c date must be 1 March 1900 or  1  March  on  any  LEAP  YEAR  not
c greater than 2010.  Dates must be between the base date and  the
c base date plus 32766.  [I. e., the logic  is  correct  only  for
c dates post-1900 and pre-2100 since 1900 and 2100  are  not  leap
c years but 2000 is.  Thus, tests for non-leap  years  on  century
c boundaries except those divisible by 400 which are leap  years,
c are avoided.]  This routine is set up to use only 2-byte integer
c variables and to avoid intermediate use of reals in the evaluation
c of expressions.  See Dr. Dobb's Journal 80:66-70.
c
      IMPLICIT NONE
      INTEGER*2 Day, Month, Year, SequenceNumber, BaseYear
      Year = (SequenceNumber - 1 - SequenceNumber / 1461) / 365
      Day = SequenceNumber - 365 * Year - Year / 4
      Month = MOD ((5 * Day - 3) / 153 + 2, 12) + 1
      Year  = Year + BaseYear + 1 - (Month + 9) / 12
      Day = Day - (153 * MOD (Month + 9, 12) + 2) / 5
      RETURN
      END
```

Приложение Н

```
      PROGRAM PRepareDISAVgMeT
c
c by TEC II, last updated 26Oct88 (for PC), 23dec88
c Russian map setups added by TSH 08jun90
c
c This routine reads the source code file: DISAVMET.SRC and
c modifies it, in preparation for its compilation, for use in a
c specific Great Lakes Basin or Russian river basin.
c This is done so that only a single master copy of source code,
c DISAVMET.SRC, must be maintained.  Information is passed to this
c routine through the file: BSNNM.
c
c NOTE: Subbasin numbers must be assigned in the map so that
c consecutive numbers represent adjacent subbasins (physically
c touching) on the map. Thus any consecutive range of numbers
c corresponds to a "closed set" (not to more than one
c mutually-exclusive subsets). E. g., subbasins 11 and 12 on the
c Superior map must have 2 cells defined between them in the St.
c Mary's river as code '23' (a dummy code on the Superior map for
c connecting subsets) or the subbasins should be renumbered so that
c 11 and 12 become either 1 and 22 or 22 and 1, respectively.
c Another example is the joining of islands with their associated
c subbasins with dummy codes. The only exception to this rule is
c for the area outside of the entire basin.  E. g., "subbasin"
c code 24 in the Superior map represents the area outside of the
c entire Lake Superior Basin.  Likewise, "subbasin" code 0
c represents the lake which is also outside of all of the actual
c subbasins.
c
c The Russian river basin maps contain only 1 subbasin (the entire
c basin) and therefore have only 2 possible codes: 1=in the basin,
c 3=out-of-basin.
c
c The Max_No_Sta$ variable governs the size of the Thiessen
```

```
c weights data base that will be maintained on the disk.  Changing
c this value will destroy the old Thiessen weights data base files
c  (THSN_DAT.xxx,  THSN_HSH.xxx,  and  THSN_SYM.xxx)  and  begin
c building the data base again. Thus, it is  desirable  to  change
c this variable infrequently which  can  be  achieved  by  setting
c Max_No_Sta$ large; on the other hand, setting it too large  will
c waste disc space.
c
      IMPLICIT NONE
      INTEGER*2 No_Sta, I, N_Map_Width, N_Map_Height, N_Max_No_Sta,
     & N_No_Subbasins, N_R_M, N_X_Offset, N_Y_Offset, N_Radius,
     & N_Switch, N_BufLen, N, J, iStrLen, No_Subbasins, Map_Height,
     & Map_Width, Count, K, N_Cell_Size
      INTEGER*4 Buffer Length, BL2
      CHARACTER*3 Map_Width$(9), Map_Height$(9), Max_No_Sta$(9), L,
     & Line*100, No_Subbasins$(9), Name(9)*10, X_Offset(9)*12,
     & Y_Offset(9)*12, R_M(9)*6, Radius(9)*6, Buffer_Length$*6,
     & Cell_Size(9)

c                    SUP   MIC   HUR   STC   ERI   ONT   CHA
c                    URA   EMB
      DATA Max_No_Sta$  /'150','200','200','100','200','250','100',
     &                   '100','100'/
      DATA No_Subbasins$/'022','027','029','007','021','015','007',
     &                   '001','001'/
      DATA Map_Width$   /'759','471','551','239','583','455','183',
     &                   '391','420'/
      DATA Map_Height$  /'515','607','575','183','423','391','311',
     &                   '433','340'/
      DATA Cell_Size    /'001','001','001','001','001','001','001',
     &                   '002','001'/
      DATA X_Offset     / '  406.039150',                        SUP
     &                    '  227.973530',                        MIC
     &                    '  279.796362',                        HUR
     &                    '   64.526407',                        STC
```

```
&                    '  298.661220',                          ERI
&                    '  262.877366',                          ONT
&                    '   89.026790',                          CHA
&                    '  204.165222',                          URA
&                    '  214.502169'/                          EMB
  DATA Y Offset     / '  5121.205120',                        SUP
&                    '  4586.657700',                         MIC
&                    '  4709.046822',                         HUR
&                    '-4646.650407',                          STC
&                    '-4473.260070',                          ERI
&                    '-4632.908225',                          ONT
&                    '-4784.124320',                          CHA
&                    '  2454.377914',                         URA
&                    '-4896.904350'/                          EMB
  DATA X N          / '-87.00',                               SUP
&                    '-86.87',                                MIC
&                    '-81.77',                                HUR
&                    '-82.71',                                STC
&                    '-81.73',                                ERI
&                    '-76.90',                                ONT
&                    '-73.34',                                CHA
&                    '  54.99',                               URA
&                    '  55.85'/                               EMB
c
c Map calibrations yielded rmse = 1.63861 km (HURON),  1.03296  km
c (STCLAIR), and 0.456165 km (ONTARIO).
c
  DATA Radius       / '6365.2',                               SUP
&                    '6363.7',                                MIC
&                    '6346.2',                                HUR
&                    '6328.6',                                STC
&                    '6338.3',                                ERI
&                    '6341.8',                                ONT
&                    '6339.6',                                CHA
&                    '3014.5',                                URA
```

```
  &                    '6034.0'/                                    EMB
    DATA Name         / 'SUPERIOR  ',
  &                    'MICHIGAN  ',
  &                    'HURON     ',
  &                    'STCLAIR   ',
  &                    'ERIE      ',
  &                    'ONTARIO   ',
  &                    'CHAMPLAIN ',
  &                    'URAL      ',
  &                    'EMBE      '/

cTSH       K = 0
cTSH       DO 1 I = 1, 6
c To do only USSR rivers use the 2 lines below. Otherwise
c use 2 lines above.
    K = 7                                                          TSH
    DO 1, I = 8, 9                                                 TSH
    K = K + 1
    Count = 0
    READ(No_Subbasins$(I), 4001) No_Subbasins
    READ(Map_Height$(I), 4001) Map_Height
    READ(Map_Width$(I), 4001) Map_Width
    READ(Max_No_Sta$(I), 4001) No_Sta
4001  FORMAT(I3)
       L = Name(I)(1:3)
       IF (K .GT. 1) WRITE(6, 4) L, Count
    4  FORMAT(1X, 'DISAV', A3, '.NI4', I5, ' lines')
c
c      Test if map available.
c
       IF (Radius(I) .NE.     0.0') GOTO 15
   12  WRITE(6, 3015) Name(I)
3015  FORMAT(1X,A10,'Basin map not available;no program generated!')
       CALL EXIT(1)
c
```

```
c       Prepare Buffer_Length string.
c
   15   Buffer_Length = Map_Height + 1
        Buffer_Length = Buffer_Length * (Map Width + 1) / 6
        BL2 = No_Subbasins + 3
        BL2 = BL2 * No_Sta * 4 / 2
        Buffer_Length = MIN(Buffer_Length, BL2)
        WRITE(Buffer_Length$, 4002) Buffer Length
 4002   FORMAT(I6)
c
c Update DISAVMET source file and place updated copy in
c current directory.
c
      OPEN(UNIT = 1, FILE = 'DISAVMET.SRC', STATUS = 'OLD')
      CALL SYSTEM(
     &  'IF EXIST DISAV' // L // '.NI4 DEL DISAV' // L // '.NI4')
      OPEN(UNIT = 2, FILE = 'DISAV' // L // '.NI4', STATUS = 'NEW',
     &  CARRIAGECONTROL = 'LIST')
      WRITE(2, 6)
    6 FORMAT('c',
     &     /, 'c This file is an intermediate file only and should',
     &     /, 'c be deleted immediately after first compilation.',
     &     /, 'c Changes to source code should only be made to file',
     &     /, 'c DISAVMET.SRC from which this file is derived by',
     &     /, 'c program PRDSAVMT.'
     &     /, 'c')
      N_Map_Width = 0
      N_Map_Height = 0
      N_Max_No_Sta = 0
      N_No_Subbasins = 0
      N_R_M = 0
      N_X_Offset = 0
      N_Y_Offset = 0
      N_Radius = 0
      N_Switch = 0
```

```
      N_BufLen = 0
      N_Cell_Size = 0
100   READ(1, 1000, END = 150) Line
1000  FORMAT(A100)
      N = iStrLen(Line)
      IF (Line(7:17) .EQ. 'PARAMETER (') GOTO 101
c
c       Skip null lines.
c
        IF (N .EQ. 0) GOTO 100
c
c       Skip comments.
c
        IF (Line(1:1) .EQ. 'C' .OR. Line(1:1) .EQ. 'c') GOTO 100
  140   WRITE(2, 1001) (Line(J:J), J = 1, N)
 1001   FORMAT(100A1)
        Count = Count + 1
        IF (Count / 100 * 100 .EQ. Count) WRITE(6, 3) L, Count
    3   FORMAT(1H+, 'DISAV', A3, '.NI4', I5, ' lines')
        GOTO 100

  101   IF (Line(18:26) .NE. 'Map_Width') GOTO 102
        Line = Line(1:26) // ' = ' // Map_Width$(I) // ')'
        N_Map_Width = N_Map_Width + 1
        N = 33
        GOTO 140
  102   IF (Line(18:27) .NE. 'Map_Height') GOTO 103
        Line = Line(1:27) // ' = ' // Map_Height$(I) // ')'
        N_Map_Height = N_Map_Height + 1
        N = 34
        GOTO 140
  103   IF (Line(18:27) .NE. 'Max_No_Sta') GOTO 104
        Line = Line(1:27) // ' = ' // Max_No_Sta$(I) // ')'
        N_Max_No_Sta = N_Max_No_Sta + 1
        N = 34
```

```
      GOTO 140
104   IF (Line(18:29) .NE. 'No_Subbasins') GOTO 601
      Line = Line(1:29) // ' = ' // No_Subbasins$(I) // ')'
      N_No_Subbasins = N_No_Subbasins + 1
      N = 36
      GOTO 140
601   IF (Line(18:20) .NE. 'R_M ') GOTO 602
      Line = Line(1:20) // ' = ' // R_M(I) // ')'
      N_R_M = N_R_M + 1
      N = 30
      GOTO 140
602   IF (Line(18:25) .NE. 'X_Offset') GOTO 603
      Line = Line(1:25) // ' = ' // X_Offset(I) // ')'
      N_X_Offset = N_X_Offset + 1
      N = 41
      GOTO 140
603   IF (Line(18:25) .NE. 'Y_Offset') GOTO 604
      Line = Line(1:25) //    = '// Y_Offset(I) // ')'
      N_Y_Offset = N_Y_Offset + 1
      N = 41
      GOTO 140
604   IF (Line(18:23) .NE. 'Radius') GOTO 605
      Line = Line(1:23) // ' = ' // Radius(I) // ')'
      N_Radius = N_Radius + 1
      N = 33
      GOTO 140
605   IF (Line(18:27) .NE. 'Basin_Name ') GOTO 606
      Line = Line(1:27) // ' = ''' // L // ''')'
      N_Switch = N_Switch + 1
      N = 36
      GOTO 140
606   IF (Line(18:30) .NE. 'Buffer_Length') GOTO 607
      Line = Line(1:30) // ' = ' // Buffer_Length# // ')'
      N_BufLen = N_BufLen + 1
      N = 40
```

```
      GOTO 140
  607 IF (Line(18:25) .NE. 'CellSize') GOTO 140
      Line = Line(1:25) // ' = ' // Cell_Size(I) // ')'
      N_Cell_Size = N_Cell_Size + 1
      N = 32
      GOTO 140

  105 CLOSE(2, STATUS = 'DELETE')
      WRITE(6, 3005)
 3005 FORMAT(1X, 'Master source code for DISAVMET.NI4 has been ',
     & 'unacceptably altered!', /, 1X, 'Forecast aborted!')
      CALL EXIT(1)

  888 WRITE(6, 1203) L
 1203 FORMAT(1X,'Problem with "PROVSNAL.',A3,'";updating aborted.')
      CALL EXIT(1)
C
C     Updating complete; check for proper number of changes.
C
  150 IF (N_Map_Width .NE. 3) GOTO 105
      IF (N_Map_Height .NE. 3) GOTO 105
      IF (N_Max_No_Sta .NE. 8) GOTO 105
      IF (N_No_Subbasins .NE. 2) GOTO 105
      IF (N_R_M .NE. 1) GOTO 105
      IF (N_X_Offset .NE. 1) GOTO 105
      IF (N_Y_Offset .NE. 1) GOTO 105
      IF (N_Radius .NE. 1) GOTO 105
      IF (N_Switch .NE. 1) GOTO 105
      IF (N_BufLen .NE. 1) GOTO 105
      CLOSE (1)
      CLOSE (2)
      WRITE(6, 3) L, Count
    1 CONTINUE
      END
```

```
      INTEGER*2 FUNCTION iStrLen(Symbol)
      IMPLICIT NONE
      INTEGER*2 I
      CHARACTER*(*) Symbol
      I = LEN(Symbol)
    1 IF (I .EQ. 0) GOTO 2
      IF (Symbol(I:I) .NE. ' ') GOTO 2
      I = I - 1
      GOTO 1
    2 iStrLen = I
      RETURN
      END
```

Т.Е. Кроли II, С.В. Ферронский. Влияние изменения климата на гидрологию Каспийского моря. Вып. 2. Пространственное усреднение точечных метеорологических измерений. М., 1990, с. 1÷130.

Подписано в печать 15/I-91 *г.*
Заказ № 12 *. Тираж* 70 *экз. Объем* 7,5 *п.л.* Цена 2 руб.

Отпечатано в Дипломатической академии МИД СССР

МЕЖДУВЕДОМСТВЕННЫЙ ГЕОФИЗИЧЕСКИЙ КОМИТЕТ
ПРИ ПРЕЗИДИУМЕ АН СССР

Т.Е. Кроли II, С.В. Ферронский

ВЛИЯНИЕ ИЗМЕНЕНИЯ КЛИМАТА НА ГИДРОЛОГИЮ КАСПИЙСКОГО МОРЯ

Вып. 1

Построение цифровых карт водосборных бассейнов

Москва 1990

УДК 551.521

Данная работа является первой из серии, посвященной результатам проекта "Влияние изменения климата на гидрологию Каспийского моря". Проектом намечается построение гидрологических моделей бассейна Каспийского моря и исследование влияния изменений климата по сценариям моделей общей циркуляции атмосферы на гидрологические процессы бассейна.

Описан способ построения цифровых карт водосборного бассейна для использования в программах обработки данных, разработанных в Лаборатории Великих озер (НОАА, Энн Арбор, Мичиган, США) Кроли и Хартман (1985). Цифровые карты бассейнов требуются для определения пространственно усредненных величин по точечным значениям метеорологических переменных. Методические аспекты и содержание программ, необходимых для построения цифровых карт по массивам граничных точек бассейна, описаны в тексте, а сами программы приведены в приложениях.

Ответственный редактор:
Проф. Г.Н. Голубев

© Междуведомственный геофизический комитет АН СССР, М., 1990 г.

SOVIET GEOPHYSICAL COMMITTEE OF THE USSR ACADEMY OF SCIENCES

T.E. Croley II, S.V. Ferronsky

CLIMATE CHANGE IMPACTS ON THE HYDROLOGY OF THE CASPIAN SEA

Issue 1

Building digital maps of the hydrological basins

Moscow 1990

This paper is the first in a series describing work on the project: "Climate change impacts on the hydrology of the Caspian Sea". The project anticipates the construction of hydrological models of the Caspian Sea and its contributing hydrological basins and the study of climate change impacts (taken from atmospheric global circulation model simulations) on hydrological processes.

Below, we describe construction of digital maps of hydrological basins for use with data reduction procedures developed at the Great Lakes Environmental Research Laboratory (Croley and Hartmann, 1986a). A digital map of a basin is required for determining spatially-averaged hydrometeorological variables from point values of these variables. The programs used for building digital maps from boundary points are listed in the Appendices and they are described in the text.

Editor:
Prof. B.N. Bolubev

1. Введение

Проектом предусматривается выполнение следующих основных этапов: 1) построение цифровых карт бассейна Каспийского моря, 2) пространственное усреднение гидрометеорологической информации суточного разрешения со всех станций бассейна, 3) построение гидрологической модели бассейна, 4) калибровка гидрологической модели, 5) построение модели, описывающей испарение с Каспийского моря, включающее те же четыре этапа работ, 6) прогноз гидрологических изменений по климатическим сценариям, 7) статистический анализ результатов.

Пространственное усреднение гидрометеорологической информации, полученной со всех станций бассейна, проводится методом Тиссена. Этот метод является более точным по сравнению с простым арифметическим усреднением, поскольку он учитывает неравномерную плотность расположения станций. Метод состоит в определении весовой доли каждой станции, пропорциональной площади ее влияния. Площадь влияния станции ограничена отрезками, равноудаленными от соседних станций (рис.1) и образующими многоугольник. Отношение площади многоугольника к площади всего бассейна определяет весовую долю басейна, которая покрывается данной станцией. Проводя аналогичные вычисления для всех станций, получим ряд весовых значений, отражающих относительное влияние каждой станции бассейна. Эти веса в сумме равны единице и используются как множители при вычислении пространственно усредненных значений гидрометеорологических величин внутри бассейна.

Пересечение многоугольника Тиссена с площадью бассейна может быть определено посредством тестирования всех точек на карте и выделения тех из них, которые являются общими и для многоугольника и для площади бассейна (Кроли и Хартман, 1985) . Обычно граница бассейна представляется на карте в виде ряда точек, соединенных отрезками прямой. Точность представления возрастает с количеством

используемых точек. Такая процедура при использовании этого типа карт связана с проверкой, лежит ли тестируемая точка внутри или вне бассейна. Дискин (1970), Григ (1972), Ших и Хамрик (1975) усовершенствовали существовавшие ранее методы выделения границы, предложив подсчитывать число пересечений границы с прямой, выходящей в произвольном направлении из тестируемой точки. Если число пересечений равняется нулю или четному числу, то рассматриваемая точка лежит вне бассейна. При определении весовых долей Тиссена эта проверка должна выполняться всякий раз, как только берется другая сеть станций . Если оцифровывается вся карта, так что она превращается в массив "точек" или "ячеек", то каждая ячейка подвергается проверке, лежит ли она внутри бассейна или нет, только один раз. При дальнейших вычислениях весов Тиссена достаточно прочитать эту информацию, не проводя проверок, связанных с различными сетями. Такая процедура приводит к сокращению времени счета в случае, если веса Тиссена вычисляются более чем для одной сети станций.

2. Цифровые карты

Цифровая карта представляет собой двумерный массив "ячеек", каждая из которых есть квадрат, лежащий в области бассейна и обычно имеющий размер 1x1 или 2x2 км. Индексы массива описывают координаты ячеек на карте. Каждый элемент массива представляет собой пару чисел. Одно число соответствует расстоянию от начала координат до выбранной ячейки, измеренному параллельно центральному меридиану (меридиану отсчета); другое соответствует расстоянию, измеренному перпендикулярно к меридиану отсчета. Каждой ячейке массива присваивается код области карты, которой она принадлежит. В простейшем случае кодом будут нуль или единица, что указывает на положение ячейки вне или внутри гидрологического бассейна. В общем случае код будет выражаться числом, присвоенным области, в которой расположена ячейка. На рис. 1, 2 показан пример цифровой карты для бассейна реки Урал.

Цифровая карта формируется посредством наложения декартовой системы координат на выбранную область карты, как это показано, например, для реки Урал на рис. 3. Она определяет, в какую область попадает каждая элементарная ячейка этой карты. При другом способе формирования карты можно использовать координаты точек на границе района (наиболее просто, осуществляется в процессе оцифровки). Несколько подобных карт отдельных областей могут затем совмещаться, образуя многорайонную карту.

3. Построение

Дадим описание, того как определяются координаты границы области для формирования по ним цифровой карты.

3.1. Преобразование карт

Для перевода точек карты, определяемых по их широте и долготе, в декартовы координаты воспользуемся поликонической проекцией. Эта проекция использовалась исключительно для крупномасштабного картирования в США до 1950 годов (Шнайдер, 1982). Поликоническая проекция части поверхности сферы состоит из ряда конических проекций, в которых каждая линия широты проецируется на конус, касательный к сфере вдоль линии широты. Затем поверхность каждого конуса разворачивается на плоскости. Вывод уравнений преобразования приводится на рис. 4. Эта проекция популярна благодаря простоте построения (Дитц и Адамс, 1945). Она широко использовалась Геологической службой США для топографической обработки результатов съемок. Из практики США, на широтном отрезке до 1036 км по обе стороны от меридиана отсчета ошибка в масштабе не превышает 1%. Поскольку в направлении меридиана отсчета и вдоль линий равной долготы точность расстояний высокая, то эта проекция в основном используется для областей карт, ориентированных с востока на запад. Координаты (x, y) на рис. 4 могут быть в дальнейшем смещены путем добавления к ним

дополнительных слагаемых, так что начало координат карты будет расположено в удобном месте. Для описания карты требуется знание четырех параметров: меридиана отсчета, радиуса Земли, и смещений по осям x и y.

Параметры поликонической проекции находятся по программе POLYCON.FOR, приведенной в приложении А. Она использует при вычислениях ряд пробных точек карты, заданных одновременно в виде пары географических координат (широта,долгота) и расстояния (x,y), которые измерены перпендикулярно и параллельно к меридиану отсчета соответственно. Пробные значения радиуса Земли и меридиана отсчета также должны быть заданы. Отметим, что меридианы на карте не параллельны. В то время как начало координат может быть выбрано произвольно, оси координат должны быть параллельны меридиану отсчета. Первую пробную точку карты устанавливаем точно, путем приближений, определяя смещения по осям x и по y. Таким образом, ошибки в измерении расстояния x и y от начала координат отражаются в смещениях по x и по y. Ошибки измерения минимизируются путем выбора первой пробной точки в таком месте, где широта и долгота могут быть определены максимально точно.

Программа использует оставшиеся пары широта-долгота и расстояния на карте для определения меридиана отсчета и величины радиуса Земли, которые минимизируют сумму квадратов расстояний между положениями точек, измеренных пользователем, и полученными при преобразованиях пар широта-долгота и их координат на карте. Точки должны быть выбраны аккуратно и определены с минимальной ошибкой измерения и расположены так, чтобы большая их часть была равномерно распределена по той части карты, где ошибка измерения должна быть минимальной. Аппроксимация карты приводит к нахождению наилучшего положения меридиана отсчета, лежащего наиболее близко к пробному меридиану (в случае, если не было допущено ошибок при измерениях или оценке, то наилучшее приближение совпадает с пробным). Если при аппроксимации была указана только одна точка,

то значения меридиана отсчета и радиуса Земли будут также совпадать с пробными.

Все расстояния получаем в единицах, указанных пользователем. Программа POLYCON.FOR в Приложении А определяет константу для перевода измерений на карте, проводимых в дюймах, и расстояний на карте, измеренных в километрах. Широты (градусы к северу от экватора) и долготы (градусы к востоку от меридиана отсчета) должны задаваться в десятых долях градуса, так как в программе они переводятся в радианы. Меридиан отсчета также задан в десятых долях градуса к востоку от исходного меридиана.

3.2. Нормализация параметров карты

После определения параметров проекции карты вычисляют параметры, необходимые для получения нормализованного массива. В программе MAKEMAP.FOR (Приложение В) использованы параметры поликонической карты, размеры карты и выбранный размер ячеек. С ее помощью параметры и расстояния пересчитываются в терминах единичной ячейки и вводится сдвиг карты на половину ячейки, так что вычисления расстояния от выбранной точки до центра ячейки могут быть произведены по левому нижнему углу ячейки. Исключение добавления половины ячейки к обеим координатам либо для точки, либо для вершины ячейки при каждом вычислении расстояния может привести к значительному сокращению времени счета. Эти координаты используются при вычислении в программах площадного усреднения методом Тиссена, при обработке данных в почти реальном масштабе времени, в программах прогнозов, в климатических программах (Кроли и Хартман, 1986б), а также в других моделях, где важно экономить время вычисления.

3.3. Граничные преобразования

После того как параметры проекции карты найдены, точки границы

области могут быть использованы для определения ячеек массива цифровой карты, расположеных внутри области, и ячеек вне этой области. Программа MAKEMAP.FOR в Приложении В позволяет считать файл точек границы и строить цифровую карту области. Точки должны располагаться в порядке обхода границы, а первая и последняя точки должны совпадать. Точки должны состоять из пар "широта-долгота", причем широта стоит вначале и каждая пара составляет одну запись.

Пользователь должен вставить параметры поликонической карты (полученные из программы POLYCON.FOR), размеры карты, и желаемый размер клетки (размер ячейки). Все размеры должны быть самосогласованы. Все широты и долготы должны быть выражены как градусы к востоку от первоначального меридиана (например, 88В--88З). Широты должны быть указаны в градусах к северу от экватора. Ширина и длина карты задаются расстояниями (в милях, километрах, и т.д.) от начала координат на карте к правой границе и к верхней границе, соответственно, для всей области, которая должна быть оцифрована. Размеры ячейки задаются шириной и длиной и определяют разрешение цифровой карты.

Поскольку положения точек на карте переводятся в координаты, точность задается равной 1/100000 стороны ячейки. Те ячейки, которые располагаются ближе чем на 1/1000000 стороны ячейки от границы, считаются расположенными на границе. Эта точность ограничивает размеры используемых карт до 2147·2147 ячеек для 4-битовых целых [zmap(0:2146, 0:2146) в программе MAKEMAP.FOR]. Размеры карты должны выбираться так, чтобы пары "широта/долгота" преобразовались в координаты площади размером -2147:+2146 на -2147:+2146, дабы избежать переполнения целых чисел.

Каждая ячейка на карте тестируется, для того чтобы определить, находится ли она на границе. Это делается путем пересчета числа пересечений границы с горизонтальным сегментом линии, направленной к востоку от центра ячейки. Если линия

пересекает границу нулевое или четное число раз, то точка находится вне границы. Этот тест не работает для точек расположенных непосредствено на границе (в пределах одной милионной от края ячейки). Так что небольшое частичное пересечение, считаемое как единица добавляется к счетчику пересечений как десять. Тест на число пересечений с границей будет рассматривать эти точки как расположенные на границе. Последнее пересечение для каждой ячейки отражает точно относительное положение каждой точки. Одно положение показывает, сколько пересечений возникло с границей. Если ячейка расположена на границе, то ей приписывается код 1. В противном случае код равен 0.

4. Пример

В заключение приведем пример построения цифровой карты для бассейна реки Урал, впадающей в Каспийское море. Измерения проводились, используя многолистные карты масштаба 1:500,000 составленные ГУГКом СССР и Министерством обороны США. Было получено 18 файлов, каждый из которых описывает отрезок границы на одной из карт. Для каждого отрезка измерено в среднем по 18 точек как в терминах широты и долготы так и в терминах расстояний: абсциссы X перпендикулярно к удобно выбранному меридиану отсчета и ординаты Y паралельно меридиану отсчета. Эти измерения приведены в Таблице 1 для первого отрезка. Первоначальные пробные значения радиуса земли были приняты равными 6028.441 км и 56.451° широты, соответственно. Программа POLYCON.FOR в Приложении A затем использовалась для получения истинных параметров поликонической проекции карты, приведенных в Таблице 2. Величина радиуса Земли может показаться заниженной, но она отражает наилучшее приближение в терминах невязок масштаба на исходной карте, так что это не совсем радиус Земли.

После установления наилучшего приближения для каждого отрезка карты и после использования всех отрезков для определения пар

координат широт и долгот для каждой части границы для совмещения всех кусков была применена обычная поликоническая проекция. Эта проекция определялась при радиусе Земли, равном среднему арифметическому всех радиусов Земли, вычисленному по всем восемнадцати кускам, входящим с весами, пропорциональными количеству используемых точек внутри каждого куска. Этот радиус равняется 6028.955 км. Меридиан отсчета взят посередине района (84.99°). Значения X и Y сдвигов карты выбраны таким образом, чтобы центр карты был распо-расположен в левом нижнем углу бассейна ($X_{сдвиг}$= 407.330432 км, $Y_{сдвиг}$= -4901.756093 км). Ширина карты и ее высота выбирались удобным образом для определения верхнего правого угла карты (ширина карты = 779.119319 км, длина карты = 863.196554 км). Величина ячеек выбрана удобной для представления карты в виде двумерного массива в памяти вычислительной машины класса IBM-PC (2км на 2км). После обработки всех 18 кусков границы полученный файл широт и долгот для всего бассейна реки Урал использован в программе MAKEMAP.FOR в Приложении B для построения цифровой карты, показанной на Рис. 1. Окончательный набор параметров приведен в Таблице 3. Ширина и высота карты переопределяются так, что граница становится толщиной в одну ячейку, параметры карты изменяются, как показано в Таблице 3, для ширины в одну ячейку. Параметры представляют сдвинутую на половину ячейки карту, так что ячейки в массиве координат карты определяются теперь не по углу, а по центру. И, наконец, программа CONVRTRW.FOR в Приложении C использована для преобразования цифровой карты в форму, полезную для других программ, использующих площадное усреднение. На рис.5 показан пример цифровой карты бассейна реки Урал, полученной авторами.

5. Заключение

Здесь изложена методика построения цифровых карт, которая используется авторами при осуществлении проекта. Выполненная часть работ по построению цифровых карт для бассейна Каспийского моря свидетельствует об удобстве ее использования для практики и надежности работы программ при всех ситуациях.

Таблица 1. Координаты точек, измеренные на одной из карт бассейна реки Урал

No	Широта (°)	Долгота (°)	X (дюйм)	Y (дюйм)	No	Широта (°)	Долгота (°)	X (дюйм)	Y (дюйм)
1	46.333	48.5	13.066	5.092	14	47.000	50.0	22.124	10.901
2	46.333	49.0	16.088	5.052	15	47.000	50.5	25.112	10.924
3	46.333	49.5	19.124	5.041	16	47.333	48.5	13.181	13.868
4	46.333	50.0	22.155	5.049	17	47.333	49.0	16.158	13.831
5	46.333	50.5	25.172	5.076	18	47.333	49.5	19.132	13.827
6	46.666	48.5	13.099	8.016	19	47.333	50.0	22.106	13.825
7	46.666	49.0	16.114	7.983	20	47.333	50.5	25.076	13.850
8	46.666	49.5	19.131	7.964	21	47.666	48.5	13.225	16.791
9	46.666	50.0	22.134	7.971	22	47.666	49.0	16.176	16.787
10	46.666	50.5	25.151	8.004	23	47.666	49.5	19.138	10.737
11	47.000	48.5	13.141	10.940	24	47.666	50.0	22.094	10.746
12	47.000	49.0	16.136	10.910	25	47.666	50.5	25.047	10.762
13	47.000	49.5	19.129	10.891					

Таблица 2. Параметры поликонической проекции для одной из карт бассейна реки Урал, полученные при помощи программы POLYCON.FOR

x сдвиг = 0.2362498710E+03

y сдвиг = -0.4817413329E+04

Меридиан отсчета = 49.590°

Радиус земли = 0.6032137921E+04

С.К.О. ошибки = 0.107119E+00 км

С.К.О. ошибки (на карте) = 0.892680E-02 в дюймах

Таблица 3. Параметры для построения цифровой карты бассейна реки Урал, полученные с помощью программы MAKEMAP.FOR

X сдвиг	=	204.165222	ячейкам
Y сдвиг	=	-2450.377930	ячейкам
Базисный меридиан	=	54.99°	
Радиус Земли	=	3014.477539	ячейкам
Ширина карты	=	391	ячейкам
Длина карты	=	433	ячейкам
Размер ячейки карты	=	1	ячейкам

6. Литература

Кроли и Хартман (1986а)

T.E. Croley, H.C. Hartmann. Mapping drainage areas on digital maps form boundary coordinates. GLERL open file report № 501, June 1986, 31 pp.

Кроли и Хартман, (1985)

T.E. Croley, H.C. Hartmann. Resolving Thiessen polygons. Journal of Hydrology, 1985, 76, p. 363-379.

Кроли и Хартман, (1986б)

T.E. Croley, H.C. Hartmann. Near real-time forecasting of large lake water supplies. National Technical Information Service, Springfield, Virginia 22161, NOAA Technical Memorandum GLERL-61, 1986, 80 pp.

Дитц и Адамс, (1945)

C.H. Deetz, O.S. Adams. Elements of Map Projection with applications to map and chart construction. Greenwood Press, New York, 1945, pp.60-69.

Дискин (1970)

M.H. Diskin. On the computer evaluation of Thiessen weights. Journal of Hydrology, 1970, 11, p. 69-78.

Григ (1972)

A.O. Brigg. A program for calculating Thiessen average rainfall. Transportation Road Research Laboratory, Crowthorne, Berkshire, TRRL Report LR 470, 1972, 20 pp.

Ших и Хамрик (1975)

S.F. Shih, R.L. Hamrick. A modified Monte Carlo technique to compute Thiessen coefficients. Journal of Hydrology, 1975, 27, p. 1029-1038.

Снайдер (1982)

J.P. Snyder. Map projection used by the U.S. Geological Survey. Geological Survey Bulletin 1532, United States Government Printing Office, Washington, D.C., pp. 123-134.

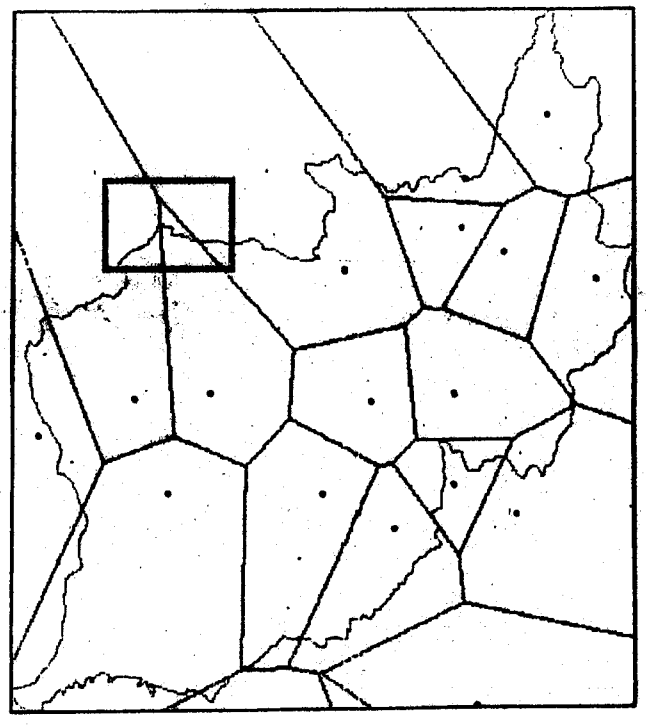


Рис. 1. Цифровая карта бассейна реки Урал с многоугольниками Тиссена

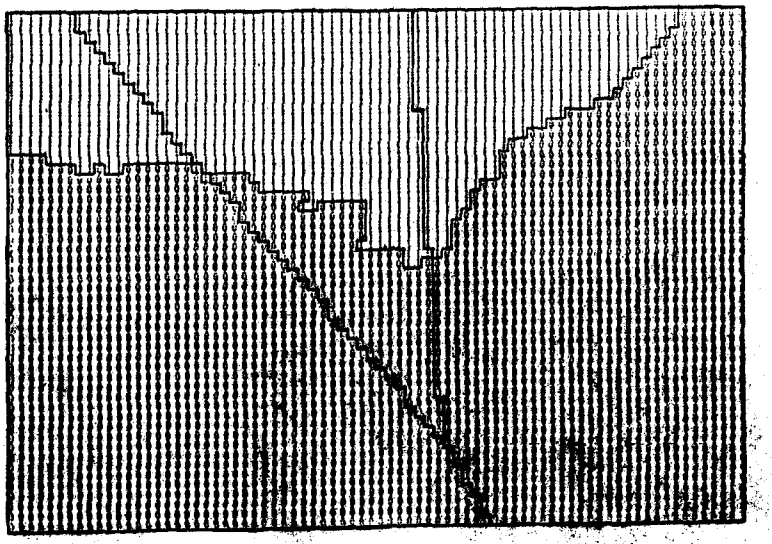


Рис. 2. Увеличенная часть цифровой карты бассейна реки Урал, показанная на рис. 1

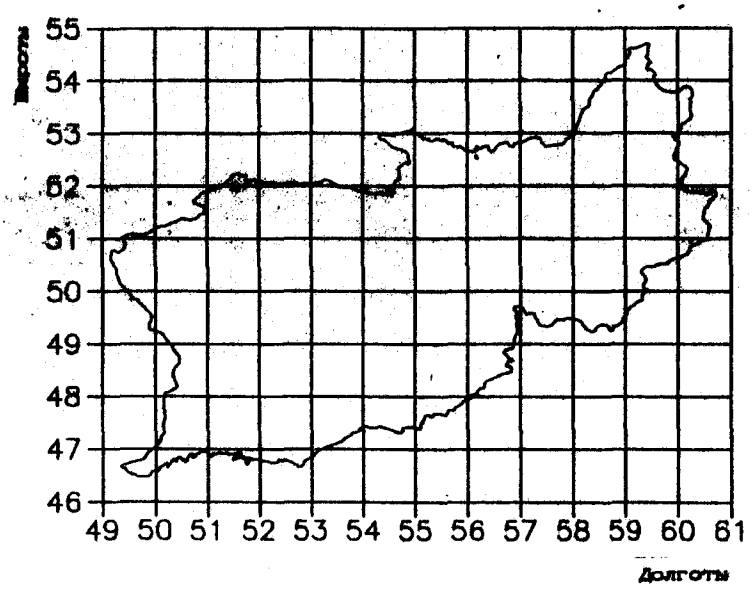
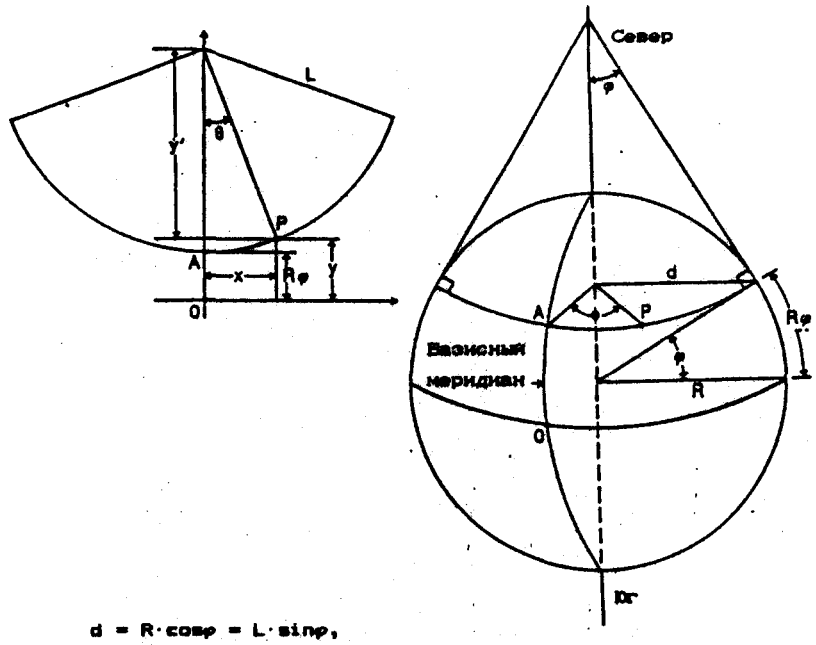


Рис. 3. Положение гидрологического бассейна реки Урал



$d = R \cdot \cos\varphi = L \cdot \sin\varphi,$

$\overline{AP} = \theta \cdot d,$

$\theta = \overline{AP} / L = \theta \cdot \sin\varphi,$

$x \approx L \cdot \sin\theta = R \cdot \sin(\theta \cdot \sin\varphi)/\tan\varphi,$

$y = R \cdot \varphi + L - y' = R \cdot (\varphi + (1 - \cos \cdot (\theta \sin\varphi)/\tan\varphi).$

Рис. 4. Уравнения поликонического преобразования карт.

Рис. 5. Пример цифровой карты бассейна реки Урал

Приложение А

```
      PROGRAM POLYCONIC
c
c Adapted 26 June 1990 for IBM-PC and compatibles for use in
c the Soviet Union on the Caspian Sea basins (from version
c written 28 March 1986 for the U.S. Great Lakes at the Great
c Lakes Environmental Research Laboratory) by:
c
c          Thomas E. Croley II and Sergei V. Ferronsky
c          Soviet Geophysical Committee
c          Academy of Sciences of the USSR
c          Molodezhnaya 3
c          Moscow, 117296
c          USSR
c
c This routine determines polyconic-projection map parameters that
c provide the "best" mapping of latitude/longitude coordinates to
c the user's map measurements for a series of user-supplied points.
c It also then converts a series of user measurements to
c latitude/longitude coordinates for subsequent use (input:
c ????????.DAT, output: ????????.LL). "Best" refers to minimum
c sum-of-squared distances between locations of points transformed
c (from latitude/longitude to map coordinates) and the user-
c supplied measurements. See comments in subroutine:
c 'POLYCONIC_MAP_APPROXIMATION' for more detail and directions for
c obtaining input values.
c
c The parameters can then be used with the subroutine:
c "COMPUTE_MAP_DISTANCES" in other applications to map lati-
c tude/longitude coordinate pairs to distances from the map
c origin.
c
c Inputs:   inputs to be provided by the user:
c
```

```
c   qRef_Meridian      initial trial value of the map's reference
c                      meridian, in decimal degrees east of the
c                      prime meridian (e.g., 88 W = -88 E)
c
c   qEarth_Radius      initial trial value of the radius of the
c                      earth, in any convenient units
c
c   Nax                maximum number of user_supplied points to
c                      be used in making the polyconic map
c                      approximation; Nax > 0
c
c   qInit_Lat          array of latitudes, in decimal degrees
c                      north of the equator, for the user-sup-
c                      plied points, 1, ..., no. of measurements
c
c   qInit_Long         array of longitudes, in decimal degrees east
c                      of the prime meridian, for the user_supplied
c                      points, 1, ..., no. of measurements
c
c   qXds               array of distances, in same units as
c                      qEarth_Radius (Conversion = 1.0 or to be
c                      converted to same units as qEarth_Radius),
c                      measured from the user's map origin
c                      perpendicular to a central meridian to each
c                      of the user-supplied points,
c                      1, ..., no. of measurements
c
c   qYds               array of distances, in same units as
c                      qEarth_Radius (Conversion = 1.0 or to be
c                      converted to same units as qEarth_Radius),
c                      measured from the user's map origin parallel
c                      to a central meridian to each of the user-
c                      supplied points,
c                      1, ..., no. of measurements
c
```

```
c   Conversion          conversion factor to convert from units
c                       associated with qXds and qYds to units
c                       associated with the mEarth_radius.
c
c.Outputs:
c   qX_Offset           map offset, perpendicular to reference
c                       meridian, used in subroutine:
c                       "COMPUTE_MAP_DISTANCES" to locate points on
c                       the map in same units as qEarth_Radius
c
c    qY_Offset          map offset, parallel to reference meridian,
c                       used in subroutine: "COMPUTE_MAP_DISTANCES"
c                       to locate points on the map in same units as
c                       qEarth_Radius
c
c   qR_M                reference meridian in decimal degrees, to
c                       the nearest hundredth of a degree
c
c   qRadius             radius of the earth in same units as
c                       qEarth_Radius
c
c   qRMSE               root-mean-square-error between approximation
c                       and user-supplied map measurements in same
c                       units as qEarth_Radius
c
c The map origin represents that point which will be the lower
c left corner of the (0,0) cell in the resulting digital map.
c
c
   IMPLICIT INTEGER*4 (A - P, R - Z)
   IMPLICIT DOUBLE PRECISION (Q)
   PARAMETER (Max = 100)
c
   PARAMETER (qEarth_Radius = 6028.441) ! initial value, kilometers
   PARAMETER (qRef_Meridian =   56.451) ! initial value
```

```
c
      DIMENSION qInit_Lat(Nax), qInit_Long(Nax), qXds(Nax), qYds(Nax)
      CHARACTER*8 FileName
      COMMON /MEASUR/ qInit_Lat, qInit_Long, qXds, qYds

      Conversion = 2.54/100./1000.*500000.  ! inches to kilometers
      DO 1 I = 1, Nax
       qInit_Lat(I) = 0.
       qInit_Long(I) = 0.
       qXds(I) = 0.
       qYds(I) = 0.
    1 CONTINUE
c
c     Input file name for data file; input data file.
c
      CALL Rite('Map file name (extension assumed to be ".DAT") >')
      READ(5, 6) FileName
      WRITE(6, 6)
      L = StringLength(FileName)
      DO 30 I = 1, L
      K = ICHAR(FileName(I:I))
      IF (K.GT.96.AND.K.LT.123) FileName(I:I) = CHAR(IAND(K, 95))
   30 CONTINUE
      OPEN(UNIT = 1, FILE = FileName(1:L) // '.DAT',STATUS = 'OLD')
      I = 1
    3 READ(1, 4, ERR = 11) qInit_Lat(I), qInit_Long(I)
      READ(1, 5, ERR = 11) qXds(I), qYds(I)
      qXds(I) = qXds(I) * Conversion
      qYds(I) = qYds(I) * Conversion
      I = I + 1
      GOTO 3
   11 Num_Measurements = I - 1
      IF(num_measurements.LT.1.OR.num_measurements.GT.Nax)STOP
      qR_M       = ANINT(qRef_Meridian*100.)/100.
      qRadius    = qEarth_Radius
```

```
c Determine map attributes, rounding the computed reference meri-
c dian to the nearest hundreth of a degree.
c
      CALL POLYCONIC_MAP_APPROXIMATION
     +  (num_measurements, qX_Offset, qY_Offset, qRadius, qR_M, qRMSE)
      qR_M = ANINT(qR_M*100.)/100.
      WRITE(6,6000) qX_Offset, qY_Offset, qR_M, qRadius, qRMSE,
     +  qRMSE / Conversion
6000 FORMAT(1H+, 'x offset  = ', e17.10e2, '                    ',
     +            '                ', /,
     +        1x, 'y offset  = ', e17.10e2, /,
     +        1x, 'Ref. Mer. = ', f9.2, ' Degrees', /,
     +        1x, 'Radius    = ', e17.10e2, /,
     +        1x, 'rmse is approximately ', e13.6e2, /,
     +        1x, 'rmse is approximately (meas. units) ', e13.6e2, //)
      I = 0
c
c Convert user measurements in second half of data file to lati-
c tude-longitude coordinates and write to output file.
c
      OPEN(UNIT = 2, FILE = FileName(1:L) // '.LL', STATUS = 'NEW')
      WRITE(2, 6001) qX_Offset, qY_Offset, qR_M, qRadius, qRMSE,
     + qRMSE / Conversion
6001 FORMAT('x offset  = ', e17.10e2, /,
     +        'y offset  = ', e17.10e2, /,
     +        'Ref. Mer. = ', f9.3, ' Degrees', /,
     +        'Radius    = ', e17.10e2, /,
     +        'rmse is approximately ', e13.6e2, /,
     +        'rmse is approximately (meas. units) ', e13.6e2, //)
   10 READ(1, 5, ERR = 9) qXm, qYm
      I = I + 1
      qX = qXm * Conversion
      qY = qYm * Conversion
      CALL ComputeMapCoordinates(qX, qY)
      WRITE(2, 7) qX, qY
```

```
      IF (I / 100 * 100 .EQ. I) WRITE(6, 8) I
      GOTO 10
    9 CLOSE (1)
      CLOSE (2)
      WRITE(6, 8) I
    2 FORMAT(I3)
    4 FORMAT(2F6.3)
    5 FORMAT(F7.3, 1X, F7.3)
    6 FORMAT(A8)
    7 FORMAT(2F10.4)
    8 FORMAT(1H+, I4, ' lines processed.')
      END

      SUBROUTINE POLYCONIC_MAP_APPROXIMATION
     +  (n, x_offset, y_offset, radius, r_m, rmse)
c
c This routine returns the values of all necessary variables to
c  approximate a polyconic map projection to the user's map:
c
c This routine accepts n (>0) map point coordinates given as both
c latitude & longitude pairs [latitude(i), longitude(i), i  =  1,
c ..., n] and  as  user-   measured  map  distances  [xgiven(i),
c ygiven(i), i = 1, ...,n] where x is measured perpendicular to a
c convenient reference meridian & y is measured parallel  to  the
c convenient reference meridian. Both x & y are distances from an
c a user-selected origin. Usually, x(1) = y(1) = 0, but  this  is
c not necessary here. As only point #1 is used  to  fix  the  map
c offsets, it is very important that this point be determined  as
c accurately as possible. Errors in the other points  are  spread
c evenly in determining the parameters but point  #1  errors  are
c reflected directly in the x_offset and  y_offset    parameters.
c Initial values of radius and r_m must also be given.
c
c The user should fix an origin (coordinates  of  the  lower  left
c corner of the map) and the ordinate  axis  generally  should  be
```

```
c taken parallel to a meridian (line of longitude) that is central
c to the mapped area. It is important to realize that this routine
c will "find" a best-fit reference  meridian that is close to  the
c chosen convenient meridian that the user selects (if there  were
c no measurement or mapping errors, the routine would find exactly
c what the user selects).  The user  should  input  the   chosen
c meridian as the initial trial for this program.  Note  that  the
c ordinate axis is not parallel to the meridian passing through the
c origin in general (assuming that the origin is to the lower left
c of the area of interest and the chosen meridian is central to the
c area of interest).
c
c x_offset: a lateral correction to fix point [x(1), y(1)]
c y_offset: a longitudinal correction to fix point [x(1), y(1)]
c r_m:      the reference meridian for the polyconic map approxima.
c radius:   the reference value of the earth's radius for the approx.
c
c All distances are in the units given by the user.  Latitudes  and
c longitudes must be given in decimal degrees; r_m is in  decimal
c degrees. Longitudes and r_m are in reference to degrees east  of
c the prime  meridian.
c
c a point exactly on the map; the remaining establish the values of
c "r_m" and "radius" that minimize the sum-of-squared-errors of the
c resulting  polyconic map approximation's values with respect  to
c the remaining points. Therefore, all  points  should  be  chosen
c carefully; they should  be determined with  minimum  measurement
c error and be placed so that the  majority are evenly spread over
c the portion of the map where the minimum  approximation error is
c desired. As the first point is fixed exactly on  the  map,  its
c location should be central to the area of greatest interest. Note
c that if only one point is passed to this routine (n =  1),  then
c the values of r_m and radius returned to the calling program are
c the same  as the initial values.
c
```

```
      IMPLICIT DOUBLE PRECISION (a - m, o - z)
      IMPLICIT INTEGER*4 (n)
      PARAMETER (Nax = 100)
      DIMENSION xgiven(Nax),ygiven(Nax),latitude(Nax),longitude(Nax)
      COMMON /MEASUR/ latitude, longitude, xgiven, ygiven
      COMMON /zzzzz2/ rx_offset, ry_offset, rradius, rr_m, s
      COMMON /zzzzz3/ nn, ncount
      ncount = 0
      rradius = radius
      rr_m = r_m
      nn = n
      CALL Determine_Offsets
      CALL Sum_of_Squared_Errors(s)
      IF(n .EQ. 1) GOTO 4
      ss = s
    3 CALL Determine_R_M
      CALL Determine_Radius
      IF(ABS(s - ss)/s .LT. .00000001) GOTO 4
      ss = s
c     if(ncount .gt. 20000) stop
      GOTO 3
    4 x_offset = rx_offset
      y_offset = ry_offset
      radius = rradius
      r_m = rr_m
      rmse = SQRT(s/n)
      RETURN
      END
c
      SUBROUTINE DETERMINE_R_M
      IMPLICIT DOUBLE PRECISION (a - m, o - z)
      IMPLICIT INTEGER*4 (n)
      PARAMETER (Nax = 100)
      DIMENSION latitude(Nax),longitude(Nax),xgiven(Nax),ygiven(Nax)
      COMMON /MEASUR/ latitude, longitude, xgiven, ygiven
```

```
      COMMON /zzzzz2/ x_offset, y_offset, radius, r_m, s
      COMMON /zzzzz3/ n, ncount
      step = 0.1
    2 ndirection = 1
      r_m = r_m + ndirection*step
      CALL Determine_Offsets
      CALL Sum_of_Squared_Errors(ss)
      IF(ss .GT. s) GOTO 3
      s = ss
      GOTO 4
    3 r_m = r_m - ndirection*step
      ndirection = -1
    4 ncount = ncount + 1
      WRITE(6, 1000) nCount, Radius, R_M, SQRT(S / N)
 1000 FORMAT(4h+No.,i6,',R:',e13.6e2,',M:',e13.6e2,',S:',e13.6e2)
c     if(ncount .gt. 20000) stop
      r_m = r_m + ndirection*step
      CALL Determine_Offsets
      CALL Sum_of_Squared_Errors(ss)
      IF(ss .GE. s) GOTO 6
      s = ss
      GOTO 4
    6 ncount = ncount + 1
      WRITE(6, 1000) ncount,radius,r_m,SQRT(s/N)
      r_m = r_m - ndirection*step
      Step = Step / 10.
      IF (Step .GT. 0.00005) GOTO 2
      RETURN
      END
c
      SUBROUTINE DETERMINE_RADIUS
      IMPLICIT DOUBLE PRECISION (a - m, o - z)
      IMPLICIT INTEGER*4 (n)
      PARAMETER (Nax = 100)
      DIMENSION latitude(Nax),longitude(Nax),xgiven(Nax),ygiven(Nax)
```

```
      COMMON /MEASUR/ latitude, longitude, xgiven, ygiven
      COMMON /zzzzz2/ x_offset, y_offset, radius, r_m, s
      COMMON /zzzzz3/ n, ncount
      step=0.1/6380.*Radius ! step is about 100 m, in user's units
    2 ndirection = 1
      radius = radius + ndirection*step
      CALL Determine_Offsets
      CALL Sum_of_Squared_Errors(ss)
      IF(ss .GT. s) GOTO 3
      s = ss
      GOTO 4
    3 radius = radius - ndirection*step
      ndirection = -1
    4 ncount = ncount + 1
      WRITE(6, 1000) nCount, Radius, R_M, SQRT(S / N)
 1000 FORMAT(4h+No.,i6,', R:',e13.6e2,', M:',e13.6e2,', S:',e13.6e2)
c     if(ncount .gt. 20000) stop
      radius = radius + ndirection*step
      CALL Determine_Offsets
      CALL Sum_of_Squared_Errors(ss)
      IF(ss .GE. s) GOTO 6
      s = ss
      GOTO 4
    6 ncount = ncount + 1
      WRITE(6, 1000) ncount,radius,r_m,SQRT(s/N)
      radius = radius - ndirection*step
      Step = Step / 10.
      IF (Step .GT. 0.00005 / 6380. * Radius) GOTO 2
      RETURN
      END
c
      SUBROUTINE DETERMINE_OFFSETS
      IMPLICIT DOUBLE PRECISION (a - h, o - z)
      IMPLICIT INTEGER*4 (n)
      PARAMETER (Nmx = 100)
```

```
      DIMENSION latitude(Nax),longitude(Nax),xgiven(Nax),ygiven(Nax)
      COMMON /MEASUR/ latitude, longitude, xgiven, ygiven
      COMMON /zzzzz2/ x_offset, y_offset, radius, r_m, sum_err
      x_offset = 0
      y_offset = 0
      x = latitude(1)
      y = longitude(1)
      CALL Compute_Map_Distances(x, y)
      x_offset = xgiven(1) - x
      y_offset = ygiven(1) - y
      RETURN
      END
c
      SUBROUTINE SUM_OF_SQUARED_ERRORS(s)
      IMPLICIT DOUBLE PRECISION (a - m, o - z)
      IMPLICIT INTEGER*4 (n)
      PARAMETER (Nax = 100)
      DIMENSION latitude(Nax), longitude(Nax), xgiven(Nax), ygiven(Nax)
      COMMON /MEASUR/ latitude, longitude, xgiven, ygiven
      COMMON /zzzzz2/ x_offset, y_offset, radius, r_m, sum_err
      COMMON /zzzzz3/ n, ncount
      s = 0.
      DO 2 ni = 1, n
      x = latitude(ni)
      y = longitude(ni)
      CALL Compute_Map_Distances(x, y)
    2 s = s + (x - xgiven(ni))**2 + (y - ygiven(ni))**2
      RETURN
      END
c
      SUBROUTINE COMPUTE_MAP_DISTANCES(x, y)
      IMPLICIT DOUBLE PRECISION (a - z)
      COMMON /zzzzz2/ x_offset, y_offset, radius, r_m, sum_err
      lat = x/57.29578
      lon = (y - r_m)/57.29578
```

```
      tp = TAN(lat)
      snpl = SIN(lat)*lon
      x = radius*SIN(snpl)/tp
      y = radius*(lat + (1. - COS(snpl))/tp)
      x = x + x_offset
      y = y + y_offset
      RETURN
      END
C
      SUBROUTINE ComputeMapCoordinates(X, Y)
      IMPLICIT DOUBLE PRECISION (A - Z)
      COMMON /Zzzzz2/ Xoffset, Yoffset, Radius, RefMer, SumErr
      L1 = 45.0
      L2 = RefMer
      dY = 1.0
      dX = 1.0
      Latitude = L1 + dY
      Longitude = L2 + dX
      CALL Compute_Map_Distances(L1, L2)
    1 oL1 = L1
      oL2 = L2
      L1 = Latitude
      L2 = Longitude
      CALL Compute_Map_Distances(L1, L2)
      dY = (Y - L2) / (L2 - oL2) * dY
      dX = (X - L1) / (L1 - oL1) * dX
      Latitude = Latitude + dY
      Longitude = Longitude + dX
      IF (ABS(dY) .GT. .00001 .OR. ABS(dX) .GT. .00001) GOTO 1
      X = Latitude
      Y = Longitude
      RETURN
      END
C
      SUBROUTINE Rite(String)
```

```
      INTEGER I, L, StringLength
      CHARACTER*(*) String
      L = StringLength(String)
      WRITE(6, 1) (String(I:I), I = 1, L)
    1 FORMAT(1H&, 79A1)
      RETURN
      END
C
      INTEGER FUNCTION StringLength(Symbol)
      IMPLICIT NONE
      INTEGER L
      CHARACTER*(*) Symbol
      L = LEN(Symbol)
C
C     Strip leading blanks...
C
    1 IF (L .GT. 0 .AND. Symbol(1:1) .EQ. ' ') THEN
      Symbol = Symbol(2:L)
      L = L - 1
      GOTO 1
      ENDIF
C
C     "Strip" trailing blanks...
C
    2 IF (L .GT. 0 .AND. Symbol(L:L) .EQ. ' ') THEN
      L = L - 1
      GOTO 2
      ENDIF
      StringLength = L
      RETURN
      END
C
```

Приложение Б

```
      PROGRAM MakeMap
c
c Adapted 26 June 1990 for IBM-PC and compatibles for use in the
c Soviet Union on the Caspian Sea basins (from version written 28
c March 1986 for the U.S. Great Lakes at the Great Lakes
c Environmental Research Laboratory) by:
c
c                 Thomas E. Croley II and Sergei V. Ferronsky
c                 Soviet Geophysical Committee
c                 Academy of Sciences of the USSR
c                 Molodezhnaya 3
c                 Moscow, 117296
c                 USSR
c
c This routine reads a file of boundary points and constructs a
c digital map of the area bounded by the boundary. The points must
c be in order going around the boundary and the first and last
c point must be the same.
c
c The points are assumed to be latitude/longitude pairs with
c latitude first and each pair constituting one record. These
c values are assumed to be in decimal degrees.
c
      IMPLICIT INTEGER*4 (a - p, r - y)
      IMPLICIT REAL*4 (q)
      IMPLICIT CHARACTER*1 (z)
      DOUBLE PRECISION Slope_DP, qX_Offset, qY_Offset, qRef_Meridian,
     +                 qEarth_Radius, qMap_Width, qMap_Height,
     +                 qCell_Size
      DOUBLE PRECISION qR_M, qRadius, qXoffset, qYoffset, qX, qY
      CHARACTER*13 CoorFile, MapFile
c
c This user must supply polyconic map parameters (obtained with
c program: POLYCON.FOR), map dimensions, and desired cell size
```

```
c ("pixel" size). All length dimensions must be in consistent
c units. All latitudes and longitudes are assumed to be decimal
c degrees; longitudes must be expressed as degrees east of the
c prime meridian (e.g., 88 W = -88 E).  'qMap_Width' and
c 'qMap_Height' represent the distance (mi, km, etc.) from the map
c origin to the right boundary and top boundary, respectively, of
c the entire area to be mapped. 'qCell_Size' represents the width
c and height of each individual grid cell in the digital map, and
c thus determines the resolution of the digital map.
c
c The user also must supply the number of latitude-longitude pairs
c used to describe the map boundary, represented by
c 'Max_Num_of_Points'. The file containing the lat.-long. pairs is
c named by 'CoorFile', and the file to contain the digital map is
c named by 'MapFile'. It is assumed here that the map projection,
c represented by the supplied parameters, will map the supplied
c "map boundary" within the non-negative quadrant; i.e., at a
c minimum the map projection boundaries lie on or outside of the
c "map boundary" of the mapped basin or lake.
c
c
c ***************************************************************
c ***              User-supplied information                  ***
c ***************************************************************
c                                                               ***
      PARAMETER ( qX_Offset          =   487.338432     )      ***
      PARAMETER ( qY_Offset          = -4981.754893     )      ***
      PARAMETER ( qRef_Meridian      =    54.99         )      ***
      PARAMETER ( qEarth_Radius      =  6828.955        )      ***
      PARAMETER ( qMap_Width         =   779.119319     )      ***
      PARAMETER ( qMap_Height        =   863.196554     )      ***
      PARAMETER ( qCell_Size         =     2.           )      ***
                                                               ***
      PARAMETER ( Max_Num_of_Points =  1595             )      ***
      PARAMETER ( CoorFile           = 'URAL.LL'        )      ***
```

```
      PARAMETER ( MapFile            = 'URABYTCD.RAW'  )          ***
c                                                                 ***
c ************** End of User-supplied information ****************
c
c
c ***************************************************************
c ***                      User-supplied information            ***
c ***************************************************************
c                                                                 ***
c     PARAMETER ( qX_Offset         =   214.082176     )         ***
c     PARAMETER ( qY_Offset         = -4897.304482     )         ***
c     PARAMETER ( qRef_Meridian     =    55.85         )         ***
c     PARAMETER ( qEarth_Radius     =  6033.984        )         ***
c     PARAMETER ( qMap_Width        =   419.111610     )         ***
c     PARAMETER ( qMap_Height       =   338.625887     )         ***
c     PARAMETER ( qCell_Size        =     1.           )         ***
c                                                                 ***
c     PARAMETER ( Max_Num_of_Points =   665            )         ***
c     PARAMETER ( CoorFile          = 'EMBE.LL'        )         ***
c     PARAMETER ( MapFile           = 'EMBBYTCD.RAW'   )         ***
c                                                                 ***
c *************** End of User-supplied information ***************
c
c
c Add half cell to width and height to allow possibility  of  map
c cell being on the boundary and resulting cell will be in the map;
c add one cell also for borders on other edge of map; add one cell
c also so that resulting truncation will include any  fraction  of
c cell in integer width or height; subtract one since Map_Width and
c Map_Height represent array dimensions that start with 0 (not 1).
c
      PARAMETER (Map_Width= (qMap_Width/qCell_Size + 1. + 1. +.5)-1)
      PARAMETER (Map_Height = (qMap_Height/qCell_Size+1.+ 1. +.5)-1)

      PARAMETER (Record_Length = Map_Width + 1)
```

```
      DIMENSION zmap(0:Map_Width, 0:Map_Height)
      CHARACTER*(Record_Length) zMap_Record(0:Map_Height)
      DIMENSION qLat(Max_Num_of_Points), qLong(Max_Num_of_Points)
      DIMENSION Xcoor(Max_Num_of_Points), Ycoor(Max_Num_of_Points)
      DIMENSION Xmax(Max_Num_of_Points), Ymax(Max_Num_of_Points)
      DIMENSION Xmin(Max_Num_of_Points), Ymin(Max_Num_of_Points)

      COMMON /Block1/ Xcoor, Xmax, Xmin
      COMMON /Block2/ Ycoor, Ymax, Ymin
      EQUIVALENCE (Xcoor, qLat), (Ycoor, qLong)
      EQUIVALENCE (zMap, zMap_Record)

      COMMON /MapDat/ qXoffset, qYoffset, qR_M, qRadius

      CALL UNDER0(.TRUE.) ! LAHEY FORTRAN extension to set
                          ! underflow to 0.
      qRadius = qEarth_Radius / qCell_Size
      qR_M = qRef_Meridian
c
c     Add one cell to offsets to give a one-cell border on
c     the left and bottom.
c
      qXoffset = qX_Offset / qCell_Size + 1.
      qYoffset = qY_Offset / qCell_Size + 1.
c
c Report "shifted" parameters so that any use of  these  reported
c parameters  will result  in  comparisons  of  converted  points
c (latitude/longitude) to  map cell midpoints.
c
      WRITE(6, 1001) qXoffset - 0.5, qYoffset - 0.5, qR_M, qRadius,
     +               Map_Width, Map_Height, 1
1001 FORMAT(
     +1x, 'Digital map parameters in normalized cell units...',//,
     +7x, 'X Offset            = ', e17.10e2, /,
     +7x, 'Y Offset            = ', e17.10e2, /,
```

```
   +7x, 'Reference Meridian = ', f9.2, ' Degrees', /,
   +7x, 'Earth''s Radius     = ', e17.10e2, /,
   +7x, 'Map Width          = ', I17 /,
   +7x, 'Map Height         = ', I17, /,
   +7x, 'Map Cell Size      = ', I17, //)
c
c Read in all latitude-longitude pairs.  The  user  may  need  to
c modify this section to fit the specific  file  available,  which
c contains the  lat.-long. pairs describing the area to be mapped.
c
    OPEN(UNIT = 1, FILE = CoorFile, STATUS = 'old')
    I = 1
111 CONTINUE
    READ(1, 1000, END = 222) qLat(I), qLong(I)
1000 FORMAT(2f10.4)
    IF(I .GT. Max_Num_of_Points) THEN
    WRITE(6, 2223)
2223  FORMAT(1x, 'Number of points exceeds allowable dimensions!',
   +     /, 1x, 'Map making aborted!')
    CLOSE(1)
    STOP
    ENDIF
    I = I + 1
    GOTO 111
222 CONTINUE
    Number_Of_Points = I - 1
    CLOSE(1)
c
c Convert point positions to map coordinates.
c
c NOTE:  A precision is hereby set to one millionth of a map
c        cell edge. This limits usable map dimensions to 2147 x
c        2147 cells for 4-byte integers [zmap(0:2146, 0:2146)].
c        The map must be chosen so  that all latitude/longitude
c        pairs convert to map coordinates within the area:
```

```
c         -2147:+2146 by -2147:+2146 to avoid integer overflow.
c         Cells in the map which are closer than one millionth of
c         a cell  edge to a boundary will thus be interpreted as
c         being on the boundary subsequently.
c
      DO 40 I = 1, Number_Of_Points
      qX = qLat(I)
      qY = qLong(I)
      CALL Compute_Map_Coordinates (qX, qY)
      Xcoor(I) = NINT(qX * 1000000.)
      Ycoor(I) = NINT(qY * 1000000.)
      WRITE(6, 4322) IFIX(I * 100. / Number_Of_Points)
4322  FORMAT(1h+, 'conv', I3, '%')
 40   CONTINUE
      IF(Xcoor(1) .NE. Xcoor(Number_Of_Points)
     + .OR. Ycoor(1) .NE. Ycoor(Number_Of_Points)) THEN
      WRITE(6, 2222)
2222  FORMAT(1x, 'Last point must be the same as the first point!',
     +       /, 1x, 'Map making aborted!')
      STOP
      ENDIF
      Number_Of_Points = Number_Of_Points - 1
c
c     Initialize map.
c
      DO 50 Y = 0, Map_Height
      DO 50 X = 0, Map_Width
      zMap(X, Y) = CHAR(0)
 50   CONTINUE
c
c     Store boundary line segment limits.
c
      DO 99 I = 1, Number_Of_Points
      Xmax(I) = MAX(Xcoor(I), Xcoor(I + 1))
      Xmin(I) = MIN(Xcoor(I), Xcoor(I + 1))
```

```
      Ymax(I) = MAX(Ycoor(I), Ycoor(I + 1))
      Ymin(I) = MIN(Ycoor(I), Ycoor(I + 1))
 99 CONTINUE
c
c Boundary Crossing Test...
c
c For each cell in the map, test if it is within the boundary  by
c counting the number of times a horizontal line segment extending
c east from the  cell's midpoint crosses the boundary. If the line
c crosses zero or an  even number of times, the point is not within
c the boundary.  This test  doesn't  ork  correctly  for  points
c "exactly" on the boundary (within one millionth of a cell  edge
c here) and so a small "partial crossing" count is added  to  the
c crossing count so that the test for number of boundary crossings
c will consider such points to be within the boundary. The  final
c relative position of each cell; the ones place shows on how many
c boundaries the cell exactly lies and the tens  place  shows  how
c many crossings of a boundary occurred. If a cell is  within  the
c boundary, it is given a code of 1; otherwise, it is given a code
c of zero.
c
      DO 20 iY = 0, Map_Height
      Y = 500000 + iY * 1000000
      DO 22 iX = 0, Map_Width
      X = 500000 + iX * 1000000
      bCount = 0
      DO 30 I = 1, Number_Of_Points
      IF(Y .LE. Ymax(I) .AND. Y .GE. Ymin(I)) THEN
      IF(Ymax(I) .EQ. Ymin(I)) THEN
      IF(X .GE. Xmin(I) .AND. X .LE. Xmax(I)) bCount = bCount + 1
      ELSE
      Slope_DP = Ycoor(I + 1) - Ycoor(I)
      Slope_DP = (Xcoor(I + 1) - Xcoor(I)) / Slope_DP
      xIntercept = (Y - Ycoor(I)) * Slope_DP * 1.0000000001
      xIntercept = Xcoor(I) + xIntercept
```

```
      IF(X .EQ. xIntercept) bCount = bCount + 1
      IF(Y .LT. Ymax(I)) THEN
      IF(X .LT. xIntercept) bCount = bCount + 10
      ENDIF
      ENDIF
      ENDIF
   30 CONTINUE
      IF(bCount / 20 * 20 .NE. bCount) zMap(iX, iY) = CHAR(1)
   22 CONTINUE
      WRITE(6, 4321) iY, Map_Height
 4321 FORMAT(1H+, I4, ' of ', I4)
   20 CONTINUE
c
c Output the finished map...
c
      OPEN(UNIT = 1, FILE = MapFile, STATUS = 'OLD', RECL =
     + Record_Length, ACCESS = 'DIRECT', FORM = 'UNFORMATTED',
     + IOSTAT = J)
      CLOSE(1, STATUS = 'DELETE', IOSTAT = J)
      OPEN(UNIT = 1, FILE = MapFile, STATUS = 'NEW', RECL =
     + Record_Length, ACCESS = 'DIRECT', FORM = 'UNFORMATTED')
      DO 21 J = 0, Map_Height
      WRITE(1) zMap_Record(J)
   21 CONTINUE
      CLOSE (1)
      END

      SUBROUTINE COMPUTE_MAP_coordinates(X, Y)
      IMPLICIT DOUBLE PRECISION (A - Z)
      COMMON /MapDat/ X_Offset, Y_Offset, R_M, Radius
      Lat = X / 57.29578
      Lon = (Y - R_M) / 57.29578
      Tp = TAN(Lat)
      Snpl = SIN(Lat) * Lon
      X = Radius * SIN(Snpl) / Tp
```

```
Y = Radius * (Lat + (1. - COS(Bnpl)) / Tp)
X = X + X_Offset
Y = Y + Y_Offset
RETURN
END
```

Приложение С

```
      Program CONVeRTRaW
c Program to convert raw subbasin codes to code map
c
c  Raw code map has either a "zero" or "one" in each cell. A zero
c  corresponds to"out of the basin" and a one corresponds to "in
c  the basin". The codeing scheme required by DISAVMET and other
c  programs is as follows:
c         code = 0 corresponds to "in the lake"
c                1                "in subbasin 1"
c                .
c                .
c                .
c                N                "in subbasin N"
c               N+1 dummy code for "in the lake" but joining pieces
c               N+2                "out of basin"
c
c  So, for only one subbasin (N=1), must map 1 to 1 and 0 to 3.
c
      IMPLICIT NONE
      INTEGER MapWIDTH, MapHeight, RecordLength
      CHARACTER*12 RawCodeMapName, NewCodeMapName
      PARAMETER (MapWidth = 391)
      PARAMETER (MapHeight = 433)
      PARAMETER (RawCodeMapName = 'URABYTCD.RAW')
      PARAMETER (NewCodeMapName = 'URABYTCD.MAP')

      PARAMETER (RecordLength = MapWidth + 1)
      Integer I,J
      CHARACTER*1 zMap(0:MapWidth, 0:MapHeight), Zero
      CHARACTER*(RecordLength) zMapRecord (0:MapHeight)
      Equivalence (zMap, zMapRecord)

      Zero = CHAR(0)
      OPEN(UNIT = 1, FILE = RawCodeMapName, STATUS = 'OLD', RECL=
```

```
 +  RecordLength, ACCESS = "DIRECT', FORM = 'UNFORMATTED')
    DO 21 J = 0, MapHeight
    READ(1) zMapRecord(J)
21  CONTINUE
    CLOSE (1)
    DO 1 I = 0, MapWidth
    DO 2 J 0, MapHeight
    IF (zMap(I,J).EQ.Zero) zMap(I,J) = CHAR(3)
 2  CONTINUE
 1  CONTINUE
    OPEN (UNIT = 1, FILE = NewCodeMapName, STATUS = 'NEW', RECL=
 +  RecordLength, ACCESS = 'DIRECT', FORM = 'UNFORMATTED')
    DO 22 J = 0, MapHeight
    WRITE(1) zMapRecord(J)
22  CONTINUE
    CLOSE(1)
    END
```

Т.Е. Кроли II, С.В. Ферронский. Влияние изменений климата на гидрологию Каспийского моря. Вып. 1. Построение цифровых карт водосборных бассейнов. М., 1990, с. 1-43.

Подписано в печать 15/I-91 г.

Заказ № 11 . *Тираж* 70 *экз. Объем* 2,6 *п.л.* Цена 20 коп

Отпечатано в Дипломатической академии МИД СССР