САМОДОКУМЕНТИРУЮЩИЕ ПРОГРАММЫ

КАКАЯ ДОКУМЕНТАЦИЯ ТРЕБУЕТСЯ?

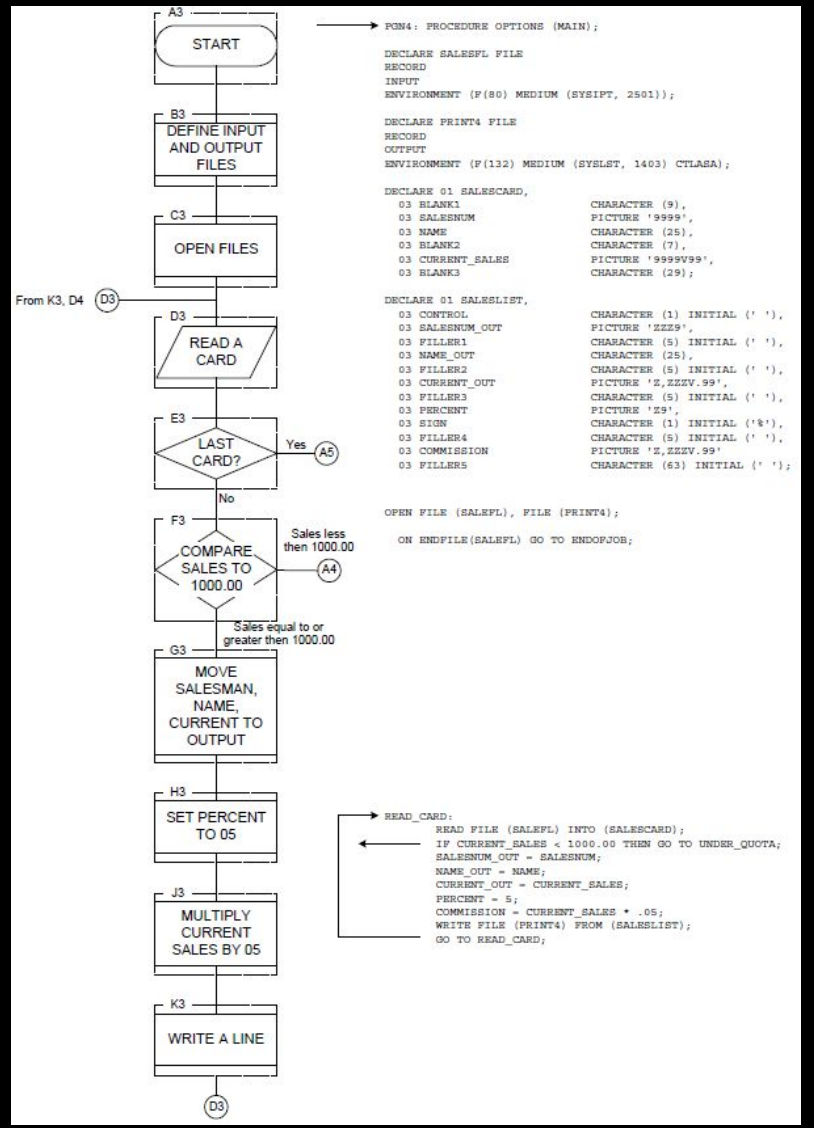
- **Чтобы использовать программу.** Каждому пользователю требуется словесное описание программы
- *Назначение.* Что является главной функцией программы и причиной ее написания?
- *Среда.* На каких машинах, аппаратных конфигурациях и конфигурациях операционной системы будет она работать?
- *Область определения и область значений.* Каковы допустимые значения входных данных? Какие правильные значения выходных результатов могут появиться?
- *Реализованные функции и использованные алгоритмы.* Что конкретно может делать программа?
- *Форматы ввода-вывода*, точные и полные.
- *Инструкция по работе*, в том числе описание вывода на консоль и устройство вывода при нормальном и аварийном завершении.
- *Опции.* Какой выбор предоставляется пользователю в отношении функций? Каким образом нужно его задавать?
- *Время работы.* Сколько времени занимает решение задачи заданного размера на заданной конфигурации?
- *Точность и проверка.* Какова ожидаемая точность результатов? Какие имеются средства проверки точности?

ОПРЕДЕЛЕНИЕ

- Самодокументирующие программы – это один из основных принципов обработки данных учит, что безрассудно стараться поддерживать синхронность независимых файлов. Значительно лучше собрать их в один файл, в котором каждая запись содержит все данные их обоих файлов, относящиеся к данному ключу.

ПОДХОД

- Первое предложение состоит в том, чтобы разделы программы, обязанные присутствовать в ней согласно требованиям языка программирования, содержали как можно больше документации. Соответственно, метки, операторы объявления и символические имена включают в задачу передать читателю как можно больше смысла.



ПОДХОД

- Второе предложение - в максимальной мере использовать пространство и формат, чтобы улучшить читаемость и показать отношения подчиненности и вложенности.
- Третье предложение - включить в программу необходимую текстовую документацию в виде параграфов комментариев. В большинстве программ достаточно иметь построчные комментарии. В программах, отвечающих жестким стандартам организаций на "хорошее документирование", их часто слишком много. Однако даже в этих программах обычно недостаточно параграфов комментариев, которые действительно способствуют понятности и обозримости целого.

РАЗРАБОТКА ПРОГРАММЫ

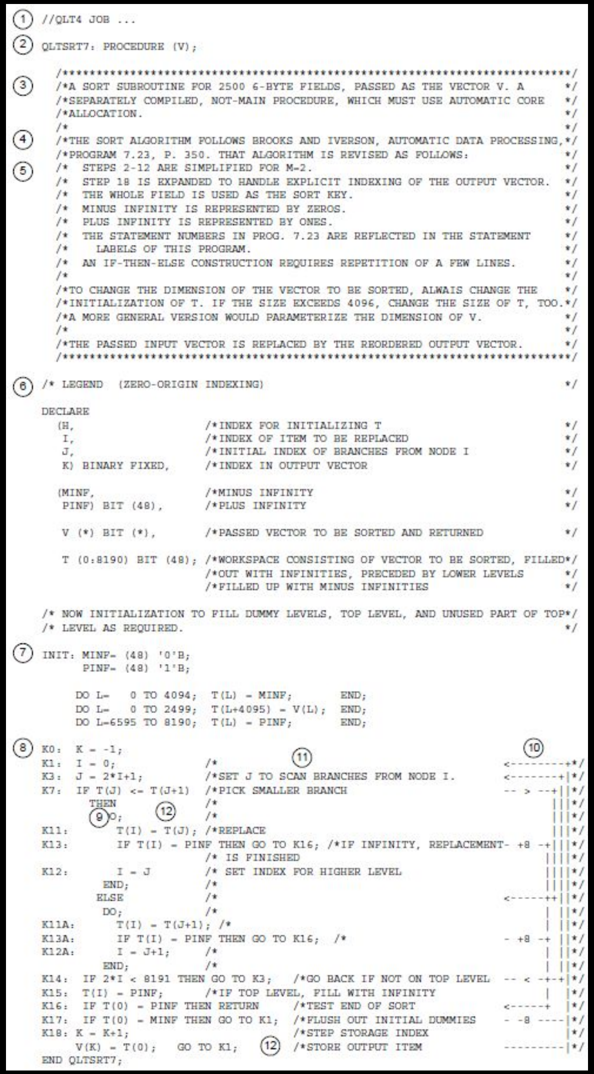
- Поскольку документация встраивается в используемые программой структуру, имена и форматы, значительную часть этой работы необходимо проделать, когда программу только начинают писать. Но именно тогда и нужно писать документацию. Поскольку подход на основе самодокументирования сокращает дополнительную работу, меньше препятствий к его осуществлению.

НЕКОТОРЫЕ ПРИЕМЫ

- Используйте для каждого запуска свое имя задания и ведите журнал, в котором учитывайте предмет проверки, время и полученные результаты. Если имя состоит из мнемоники (здесь QLT) и числового суффикса (здесь 4), то суффикс можно использовать в качестве номера запуска, связывающего запись в журнале и листинг. При этом для разных прогонов требуются свои карты задания, но их можно делать колодами с дублированием постоянных данных.
- Используйте мнемонические названия программы, включающие идентификатор версии - в предположении, что будет несколько версий. Здесь индекс - две младшие цифры года.
- Включите текстовое описание в качестве комментариев к PROCEDURE.
- Для документирования алгоритмов ссылайтесь, где можно, на литературу. Это экономит место, адресует к более полному освещению, чем можно дать в программе, и дает возможность знающему читателю пропустить ссылку, оставляя уверенность, что он вас поймет.
- Покажите связь с алгоритмом, описанным в книге: а) изменения; б) особенности использования; в) представление данных.
- Объявите все переменные. Используйте мнемонику. Используйте комментарии для превращения оператора DECLARE в полноценную легенду. Обратите внимание, что он уже содержит имена и описания структур, нужно лишь дополнить его описаниями назначения. Сделав это здесь, вы избежите отдельного повторения имен и структурных описаний.
- Поставьте метку в начале инициализации.
- Поставьте метки перед группами операторов, соответствующие операторам алгоритма, описанного в книге.
- Используйте отступы для показа структуры и группирования.
- Вручную поставьте стрелки, показывающие логический порядок операторов. Они очень полезны при отладке и внесении изменений. Их можно поместить на правом поле места для комментариев и сделать частью вводимого в машину текста.
- Вставьте строчные комментарии для пояснения всего, что неочевидно. При использовании изложенных выше приемов они окажутся короче и малочисленней, чем обычно.
- Помещайте несколько операторов на одной строке или один оператор на нескольких строках в соответствии с логической группировкой, а также, чтобы показать связь с описанием алгоритма

ВОЗРАЖЕНИЯ

- Возражения. Каковы недостатки такого подхода к документированию? Они существуют, и в прежние времена были существенными, но сейчас становятся мнимыми.
- Самым серьезным возражением является увеличение размера исходного текста, который нужно хранить. Поскольку практика все более тяготеет к хранению исходного кода в активных устройствах, это вызывает растущее беспокойство. Лично я пишу более краткие комментарии в программах на APL, которые хранятся на диске, чем в программах на PL/I, которые хранятся на перфокартах.
- Однако одновременно мы движемся к хранению в активных устройствах текстовых документов, доступ к которым и изменение осуществляется с помощью компьютеризированных текстовых редакторов. Как указывалось выше, слияние текста и программы сокращает общее количество хранимых символов.



```
(1) //QLT4 JOB ...

(2) QLTSRT7: PROCEDURE (V);

    /**************************************************************************/
(3) /*A SORT SUBROUTINE FOR 2500 6-BYTE FIELDS, PASSED AS THE VECTOR V. A      */
    /*SEPARATELY COMPILED, NOT-MAIN PROCEDURE, WHICH MUST USE AUTOMATIC CORE   */
    /*ALLOCATION.                                                              */
    /*                                                                         */
(4) /*THE SORT ALGORITHM FOLLOWS BROOKS AND IVERSON, AUTOMATIC DATA PROCESSING,*/
    /*PROGRAM 7.23, P. 350. THAT ALGORITHM IS REVISED AS FOLLOWS:              */
(5) /*  STEPS 2-12 ARE SIMPLIFIED FOR M=2.                                     */
    /*  STEP 18 IS EXPANDED TO HANDLE EXPLICIT INDEXING OF THE OUTPUT VECTOR.  */
    /*  THE WHOLE FIELD IS USED AS THE SORT KEY.                               */
    /*  MINUS INFINITY IS REPRESENTED BY ZEROS.                                */
    /*  PLUS INFINITY IS REPRESENTED BY ONES.                                  */
    /*  THE STATEMENT NUMBERS IN PROG. 7.23 ARE REFLECTED IN THE STATEMENT     */
    /*    LABELS OF THIS PROGRAM.                                              */
    /*  AN IF-THEN-ELSE CONSTRUCTION REQUIRES REPETITION OF A FEW LINES.       */
    /*                                                                         */
    /*TO CHANGE THE DIMENSION OF THE VECTOR TO BE SORTED, ALWAIS CHANGE THE    */
    /*INITIALIZATION OF T. IF THE SIZE EXCEEDS 4096, CHANGE THE SIZE OF T, TOO.*/
    /*A MORE GENERAL VERSION WOULD PARAMETERIZE THE DIMENSION OF V.            */
    /*                                                                         */
    /*THE PASSED INPUT VECTOR IS REPLACED BY THE REORDERED OUTPUT VECTOR.      */
    /**************************************************************************/

(6) /* LEGEND  (ZERO-ORIGIN INDEXING)                                          */

  DECLARE
    (H,                   /*INDEX FOR INITIALIZING T                           */
     I,                   /*INDEX OF ITEM TO BE REPLACED                       */
     J,                   /*INITIAL INDEX OF BRANCHES FROM NODE I              */
     K) BINARY FIXED,     /*INDEX IN OUTPUT VECTOR                             */

    (MINF,                /*MINUS INFINITY                                     */
     PINF) BIT (48),      /*PLUS INFINITY                                      */

     V (*) BIT (*),       /*PASSED VECTOR TO BE SORTED AND RETURNED            */

     T (0:8190) BIT (48); /*WORKSPACE CONSISTING OF VECTOR TO BE SORTED, FILLED*/
                          /*OUT WITH INFINITIES, PRECEDED BY LOWER LEVELS      */
                          /*FILLED UP WITH MINUS INFINITIES                    */

  /* NOW INITIALIZATION TO FILL DUMMY LEVELS, TOP LEVEL, AND UNUSED PART OF TOP*/
  /* LEVEL AS REQUIRED.                                                        */

(7) INIT: MINF= (48) '0'B;
          PINF= (48) '1'B;

       DO L=   0 TO 4094;  T(L) = MINF;       END;
       DO L=   0 TO 2499;  T(L+4095) = V(L);  END;
       DO L=6595 TO 8190;  T(L) = PINF;       END;

                                                                      (10)
(8) K0:  K = -1;                              (11)
    K1:  I = 0;              /*                                     <---------+*/
    K3:  J = 2*I+1;          /*SET J TO SCAN BRANCHES FROM NODE I.  <--------+|*/
    K7:  IF T(J) <= T(J+1)   /*PICK SMALLER BRANCH                  -- > --+||*/
           THEN              /*                                            |||*/
          (9)O;      (12)    /*                                            |||*/
    K11:       T(I) = T(J);  /*REPLACE                                     |||*/
    K13:       IF T(I) = PINF THEN GO TO K16; /*IF INFINITY, REPLACEMENT- +8 -+|||*/
                             /* IS FINISHED                               ||||*/
    K12:       I = J         /* SET INDEX FOR HIGHER LEVEL                ||||*/
             END;            /*                                           ||||*/
            ELSE             /*                                     <-----++||*/
             DO;             /*                                           | ||*/
    K11A:      T(I) = T(J+1); /*                                          | ||*/
    K13A:      IF T(I) = PINF THEN GO TO K16;  /*                   - +8 -+ ||*/
    K12A:      I = J+1;      /*                                           | ||*/
             END;            /*                                           | ||*/
    K14:  IF 2*I < 8191 THEN GO TO K3;   /*GO BACK IF NOT ON TOP LEVEL -- < -+-+|*/
    K15:  T(I) = PINF;       /*IF TOP LEVEL, FILL WITH INFINITY           |  |*/
    K16:  IF T(0) = PINF THEN RETURN     /*TEST END OF SORT         <-----+  |*/
    K17:  IF T(0) = MINF THEN GO TO K1;  /*FLUSH OUT INITIAL DUMMIES - -8 ----|*/
    K18: K = K+1;                        /*STEP STORAGE INDEX                 |*/
         V(K) = T(0);   GO TO K1;   (12) /*STORE OUTPUT ITEM        ---------|*/
    END QLTSRT7;
```

ЗАКЛЮЧЕНИЕ

- Подход на основе самодокументирования стимулирован применением языков высокого уровня и обретает наибольшую мощь и наивысшее оправдание в языках высокого уровня, используемых в режиме он-лайн, будь то в пакетном режиме или интерактивно. Как я доказывал, такие языки, и системы очень сильно облегчают жизнь программистов. Поскольку машины сделаны для людей, а не люди для машин, их использование оправдано как с экономической точки зрения, так и чисто по-человечески.

СПАСИБО ЗА ВНИМАНИЕ