

Лекция 14. Самодокументирующиеся программы

Принцип обработки данных – организация независимых файлов в одном файле, в котором каждая запись содержит все данные файлов, относящиеся к данному ключу.

Тем не менее, практика документирования программ противоречит собственным теориям. Обычно пытаемся поддерживать программу в виде, пригодном для ввода в машину, а независимый комплект документации, состоящей из текста и блок-схем, - в виде, пригодном для чтения человеком.

Результаты этого подтверждают мысль о неразумности поддержки независимых файлов. Программная документация получается удивительно плохой, а ее сопровождение - и того хуже. Вносимые в программу изменения не получают быстрого, точного и обязательного отражения в документе.

Предполагается, что правильным решением должно быть слияние файлов: включение документации в исходный текст программы. Это одновременно и сильный побудительный мотив к должному сопровождению, и гарантия того, что документация всегда будет под рукой у пользователя. Такие программы называют самодокументирующимися.

Очевидно, при этом неудобно, хотя и возможно, включать блок-схемы, если в этом есть необходимость. Но, приняв во внимание анахронизм блок-схем и использование преимущественно языков высокого уровня, становится возможным объединить программу с документацией.

Использование исходного кода программы в качестве носителя документации влечет некоторые ограничения. С другой стороны, непосредственный доступ читателя документации к каждой строке программы открывает возможность для новых технологий. Пришло время разработать радикально новые подходы и методы составления программной документации.

В качестве важнейшей цели должны попытаться предельно уменьшить груз документации - груз, с которым ни мы, ни наши предшественники толком не справились.

PL/I — разработанный в 1964 году язык программирования, созданный для научных, инженерных и бизнес-ориентированных вычислений.

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.

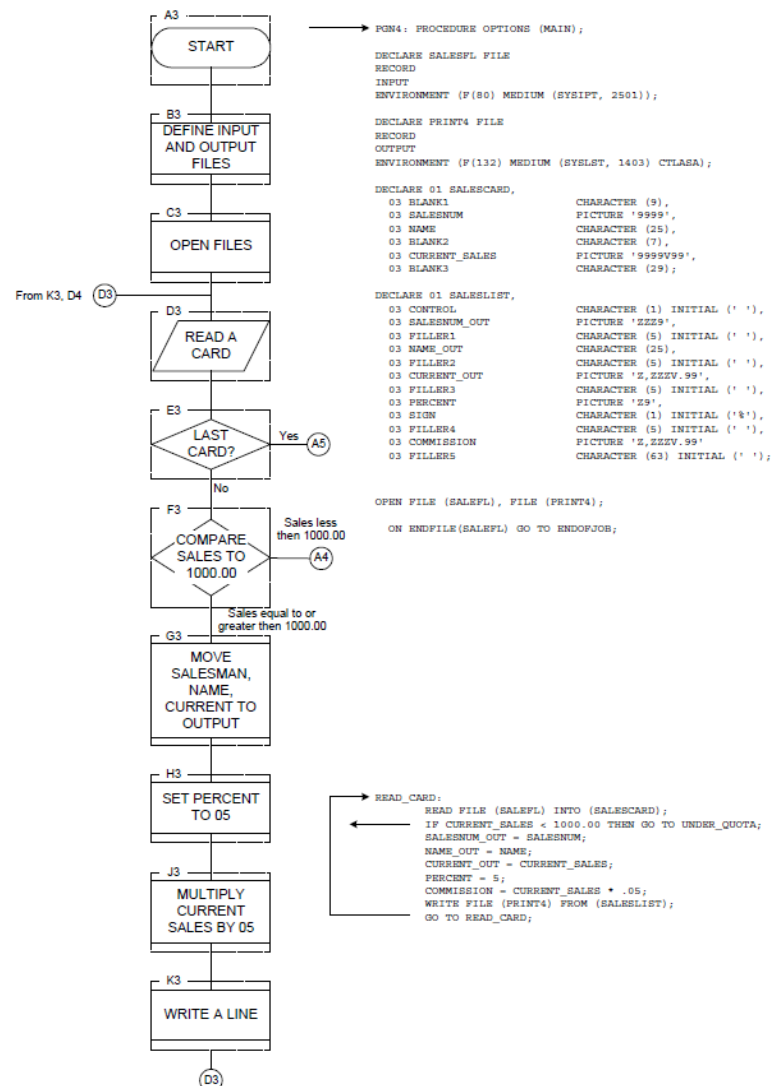


Рис. 1. Сравнение блок-схемы и соответствующей программы на PL/I
(фрагмент)

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.

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. На рисунке 2 показана самодокументирующаяся программа на языке PL/I.4 Числа в кружочках не являются ее частью, а служат метадокументацией для ссылок при обсуждении.

1. Используйте для каждого запуска свое имя задания и ведите журнал, в котором учитывайте предмет проверки, время и полученные результаты. 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.
2. Используйте мнемонические названия программы, включающие идентификатор версии - в предположении, что будет несколько версий. Здесь индекс - две младшие цифры года.

3. Включите текстовое описание в качестве комментариев к PROCEDURE.
4. 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.
5. Покажите связь с алгоритмом, описанным в книге: а) изменения; б) особенности использования; в) представление данных.
6. 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.
7. Поставьте метку в начале инициализации.
8. Поставьте метки перед группами операторов, соответствующие операторам алгоритма, описанного в книге.
9. Используйте отступы для показа структуры и группирования.
10. 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.
11. Вставьте строчные комментарии для пояснения всего, что неочевидно. При использовании изложенных выше приемов они окажутся короче и малочисленней, чем обычно.
12. 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. Каковы недостатки такого подхода к документированию? Они существуют, и в прежние времена были существенными, но сейчас становятся мнимыми.

```

(1) //QLT4 JOB ...
(2) QLTSTR7: 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, ALWAYS 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;

(8) K0: K = -1;
K1: I = 0;
K3: J = 2*I+1;
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 QLTSTR7;

```

Рис. 2. Самодокументирующаяся программа

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.

Однако одновременно 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.

Аналогичное возражение вызывает аргумент, что самодокументирующиеся программы требуют больше ввода с клавиатуры. В печатном документе требуется, по меньшей мере, одно нажатие на клавишу для каждого символа на каждый черновой экземпляр. В самодокументирующейся программе суммарное количество символов меньше, и на один символ приходится меньше нажатий на клавиши, так как черновики не перепечатываются.

А что же блок-схемы и структурные графы? Если используется только структурный граф самого высокого уровня, он вполне может содержаться в отдельном документе, поскольку редко подвергается изменениям. Но, конечно, его можно включить в исходный текст программы в качестве комментария, что будет благоразумно.