Программная логика верхнего уровня

(предлагаемая формулировка)

1. Общее описание

Для чего нужна формализация понятия «программная логика верхнего уровня»

При описании алгоритмов достаточно сложных режимов работы технологического оборудования программисты неизбежно начинают использовать процедуры описания программной логики высокого уровня, из которых и выполняется вызов низкоуровневых процедур, непосредственно управляющих технологическим оборудованием.

То, что программная логика верхнего уровня в программировании присутствует, является общепризнанным фактом. И алгоритмы работы программ (в текстовом или графическом формате) как раз программную логику верхнего уровня и описывают.

Но уже ставшая хронической проблема выделения четких алгоритмов работы из исходного кода программ наталкивает на мысль о том, что фраза «программная логика верхнего уровня» обозначает сейчас в программировании скорее общепринятое понятие с расплывчатыми границами, а не четкую логическую формулировку.

Строгая формализация понятия «программная логика верхнего уровня» должна заложить предпосылки для формализации и автоматизации процесса выделения алгоритмов работы программ из их исходного кода, что, в свою очередь, должно сделать исходный код ПО более читаемым для программиста и таким образом способствовать снижению издержек на разработку и сопровождение ПО.

Разделение программной логики на логику верхнего и нижнего уровня

Я не могу претендовать на особенную новизну моих утверждений, т.к. все они, хотя и в несколько фрагментированном виде, уже содержатся в ставших классикой учебниках по программированию. Но в виде четко изложенной технологии программирования я пока приводимых мной ниже тезисов не встречал (хотя столько сейчас везде книг и других материалов по

программированию, что стоит кому-нибудь сказать, что он придумал что-то новое, а ему и говорят: а эти мысли уже высказал такой-то там-то и там-то).

Правило разделения программной логики на логику верхнего и нижнего уровня я решил сформулировать путем выделения того, что относится к логике нижнего уровня:

1) действия по управлению каким-либо оборудованием;

2) действия, связанные с приемом и передачей информации: обмен данными по сети, операции по работе с файлами и базами данных и т.п.;

3) проведение каких-либо вычислений;

4) обработка данных конкретных типов: массивов, структур, объектов и т.п;

5) действия, связанные с запуском и контролем хода выполнения параллельных процессов.

Возможно, я забыл перечислить что-то еще, но понятно, что к логике нижнего уровня я отнес конкретные действия по приему, передаче и обработке какой-либо информации.

Тогда к логике верхнего уровня будет относиться описание последовательности выполнения низкоуровневых операций.

Что нужно для описания логики верхнего уровня на императивных языках программирования высокого уровня

(эти пункты я привожу без доказательства, исходя из моего практического программистского опыта и знаний, полученных при изучении программирования, т.к. 1) у меня не хватает математического образования для строгого математического анализа таких вопросов, и 2) я не претендую на всеохватность, а хочу обобщить ситуацию хотя бы для наиболее типичных случаев, возникающих при программировании и алгоритмизации уже написанных программ)

1) Минимально необходимый набор маршрутных операторов (см. теорему Бома-Якопини о структурном программировании):

- сложное условие;

- цикл;

- присвоение;

- вызов процедуры (функции).

2) Для действий над операндами в сложных условиях логики верхнего уровня используются следующие логические операторы: И/ИЛИ/НЕ

3) Для задания типов процедур и функций, используемых при описании логики верхнего уровня, используются следующие процедурные типы:

- ПРОЦЕДУРА (ПАРАМЕТР:ЦЕЛЫЙ); для посылки команды логике нижнего уровня;

- ФУНКЦИЯ (ПАРАМЕТР:ЦЕЛЫЙ):ЦЕЛЫЙ; для посылки команды логике нижнего уровня и получения от логики нижнего уровня информации о результатах выполнения низкоуровневых операций;

4) При описании логики верхнего уровня используются переменные целого типа в следующих случаях:

- как переменные цикла;

- для хранения текущего состояния процедуры логики верхнего уровня при управлении последовательностью выполнения низкоуровневых операций (например, при автоматном программировании);

- для хранения с целью последующего анализа результата запроса к логике нижнего уровня; результат запроса может быть:

- логическим значением: ДА (1) /НЕТ (0);

- одним из возможного набора определенных значений для выбора дальнейшего варианта действий;

- целым числом: верхней границей цикла или следующим значением переменной цикла.

2. Подробное описание

Нумерация процедур логики высокого уровня

Для программирования логики верхнего уровня (ЛВУ) я решил ввести нумерацию процедур логики верхнего уровня. Номер процедуры ЛВУ должен обязательно присутствовать в ее имени, например:

П2__процедура_ЛВУ

Номера процедур ЛВУ, в свою очередь, будут определять номера команд и запросов, передаваемых процедурой ЛВУ логике нижнего уровня.

Я посчитал, что для одной процедуры ЛВУ будет достаточно 100 команд и запросов. Поэтому номера команд и запросов процедуры ЛВУ должны задаваться так, чтобы выполнялось равенство:

N процедуры = N кз div 100,

где N кз – номер команды/запроса;

div – оператор целочисленного деления.

Тогда для логики нижнего уровня обработчик команды может иметь следующий вид:

```
procedure  _komanda(k:integer);
begin

    if      ((k div 100) = 1) then   komanda__1(k)
    else if ((k div 100) = 2) then   komanda__2(k)

end;
```

где процедуры komanda__1(k) и komanda__2(k) – это обработчики команд для процедур ЛВУ с номерами 1 и 2.

Пример процедуры ЛВУ

Приводимая ниже в качестве примера процедура ЛВУ написана на русскоязычной версии языка Компонентный Паскаль. При этом для наглядности я переопределил следующие русскоязычные ключевые слова:

- ЦИКЛ_ПОКА;
- КОНЕЦ_ЦИКЛА_ПОКА;
- КОНЕЦ_ПРОЦЕДУРЫ.

```
КОНСТАНТЫ
    ОТВЕТ_НЕТ                          = 0;
    ОТВЕТ_ДА                           = 1;

    К2__НАЧАЛО_ЧТЕНИЯ_СХЕМЫ            = 200;
    З2__КОНЕЦ_ЧТЕНИЯ_СХЕМЫ             = 201;
    К2__ЧТЕНИЕ_СТРОКИ                  = 202;
```

```
    З2__ЗАГОЛОВОК_СЧИТАН                = 203;
    З2__НЕТ_ВЫХОДА_ИЗ_ЦИКЛА             = 204;
    К2__ЧИТАТЬ_ЗАГОЛОВОК                = 205;
    З2__НАЧАЛО_ЭЛЕМЕНТА                 = 206;
    К2__ПРОДОЛЖИТЬ_ЧТЕНИЕ_ЭЛЕМЕНТА      = 207;
    К2__НАЧАТЬ_ЧТЕНИЕ_ЕЛЕМЕНТА          = 208;
    К2__ОКОНЧАНИЕ_ЧТЕНИЯ_СХЕМЫ          = 209;
    К2__ПРОВЕРИТЬ_НАЧАЛО_ЭЛЕМЕНТА       = 210;

ТИП

ПРОЦЕДУРА_ПАР_ЦЕЛЫЙ*    = ПРОЦЕДУРА(К:ЦЕЛЫЙ);
ФУНКЦИЯ_П_ЦЕЛ_Р_ЦЕЛ*    = ПРОЦЕДУРА(К:ЦЕЛЫЙ): ЦЕЛЫЙ;

ПЕРЕМЕННЫЕ
команда*:             ПРОЦЕДУРА_ПАР_ЦЕЛЫЙ;
запрос*:              ФУНКЦИЯ_П_ЦЕЛ_Р_ЦЕЛ;


ПРОЦЕДУРА  П2__СЧИТАТЬ_СХЕМУ;
НАЧАЛО
    команда(К2__НАЧАЛО_ЧТЕНИЯ_СХЕМЫ);
    ЦИКЛ_ПОКА ((запрос(З2__КОНЕЦ_ЧТЕНИЯ_СХЕМЫ) = ОТВЕТ_НЕТ) &
               (запрос(З2__НЕТ_ВЫХОДА_ИЗ_ЦИКЛА) = ОТВЕТ_ДА))
    ДЕЛАТЬ
      команда(К2__ЧТЕНИЕ_СТРОКИ);
      команда(К2__ПРОВЕРИТЬ_НАЧАЛО_ЭЛЕМЕНТА);
      ЕСЛИ (запрос(З2__ЗАГОЛОВОК_СЧИТАН) = ОТВЕТ_ДА) ТО
        ЕСЛИ (запрос(З2__НАЧАЛО_ЭЛЕМЕНТА) = ОТВЕТ_ДА) ТО
          команда(К2__НАЧАТЬ_ЧТЕНИЕ_ЕЛЕМЕНТА);
        ИНАЧЕ
          команда(К2__ПРОДОЛЖИТЬ_ЧТЕНИЕ_ЭЛЕМЕНТА);
        КОНЕЦ;
      ИНАЧЕ
        команда(К2__ЧИТАТЬ_ЗАГОЛОВОК);
      КОНЕЦ;
    КОНЕЦ_ЦИКЛА_ПОКА;
    команда(К2__ОКОНЧАНИЕ_ЧТЕНИЯ_СХЕМЫ);
    ВЫХОД;
КОНЕЦ_ПРОЦЕДУРЫ  П2__СЧИТАТЬ_СХЕМУ;
```

Для приведенной выше процедуры ЛВУ процедуры «команда» и «запрос» на самом деле являются процедурными переменными, которым были присвоены адреса соответствующих процедур из DLL логики нижнего уровня, исходный код которой был написан в среде программирования Delphi 4 на языке Object Pascal.

Выделение логики нижнего уровня в DLL – это только один из возможных вариантов, обусловленный для приведенного примера тем, что среда программирования Delphi 4 еще не позволяла программистам вводить в текст программы русскоязычные идентификаторы.

Современные среды программирования предоставляют программистам такую возможность, поэтому как логика верхнего уровня, так и логика нижнего уровня могут находиться в одной программе , и при этом их исходный код может быть записан с использованием русскоязычных идентификаторов (правда при этом ключевые слова языка программирования остаются англоязычными).

Правило четырех окон

Для одновременного редактирования как процедур ЛВУ, так и логики нижнего уровня, нужно в текстовом редакторе среды программирования одновременно держать открытыми как минимум 4 окна:

1) окно с именами и значениями констант и запросов для процедуры ЛВУ;
2) окно с исходным кодом процедуры ЛВУ;
3) окно с исходным кодом обработчика для команды (или запроса) процедуры ЛВУ;
4) окно с исходным кодом процедуры логики нижнего уровня, вызываемой из обработчика для команды (или запроса) процедуры ЛВУ.

Подробнее об обработчиках для команды и запроса процедуры ЛВУ

Ниже приведены определение структуры данных и исходный код для обработчиков команд и запросов процедуры П2__СЧИТАТЬ_СХЕМУ:

```
type  schit_shemu = record    // .29
         ti:integer;          // текущий индекс на схеме
         tek_str:integer;     // текущая строка записи
         te:integer;          // текущий элемент
         tw:integer;          // текушая вертикаль
         fl_wyh:integer;      // флаг выхода из цикла
         fl_cht_zgl:integer;  // флаг чтения заголовка
         nach_el:integer;
         stro, stro2:s_dl_stro;
         dls:integer;         // длина считанной строки
         tek_alt_A:s_dl_stro; // текущее альтернативное начало действия
end;


procedure komanda__2(k:integer);
begin
    if (k = K2__NACHALO_CHTENIQ_SHEMY) then begin
```

```
        nachalo_chteniq_shemy(form_dalvjaz.scs);
    end else if (k = K2__CHTENIE_STROKI) then begin
        chtenie_stroki(form_dalvjaz.scs);
    end else if (k = K2__CHITATX_ZAGOLOWOK) then begin
        chitatx_zagolowok(form_dalvjaz.scs);
    end else if (k = K2__PRODOLVITX_CHTENIE_ELEMENTA) then begin
        prodolvitx_chtenie_elementa(form_dalvjaz.scs);
    end else if (k = K2__NACHATX_CHTENIE_ELEMENTA) then begin
        nachatx_chtenie_elementa(form_dalvjaz.scs);
    end else if (k = K2__OKONCHANIE_CHTENIQ_SHEMY) then begin
        okonchanie_chteniq_shemy(form_dalvjaz.scs);
    end else if (k = K2__PROWERITX_NACHALO_ELEMENTA) then begin
        proweritx_nachalo_elementa(form_dalvjaz.scs);
    end;

end;


function  zapros__2(n:integer):integer;
var ot:integer;
begin
    if (n = Z2__KONEC_CHTENIQ_SHEMY) then begin
        if (form_dalvjaz.scs.tek_str = fajly.kol_str_zap) then ot := YES
                                                          else ot := NO;
    end else if (n = Z2__ZAGOLOWOK_SCHITAN) then begin
        ot := form_dalvjaz.scs.fl_cht_zgl;

    end else if (n = Z2__NET_WYHODA_IZ_CIKLA) then begin
        if (form_dalvjaz.scs.fl_wyh = NO) then  ot := YES
                                          else  ot := NO;
    end else if (n = Z2__NACHALO_ELEMENTA) then begin
        ot := form_dalvjaz.scs.nach_el;
    end;

    zapros__2 := ot;
end;
```

Из вышеприведенного исходного кода видно, что все процедуры логики нижнего уровня, вызываемые из обработчиков, разделяют одну и туже структуру данных form_dalvjaz.scs, описание которой приведено выше. Каждой процедуре ЛВУ на нижнем уровне логики должна соответствовать своя структура данных, объединяющая процедуры нижнего уровня, выполняющие команды и запросы этой процедуры ЛВУ.

Предполагается также, что процедуры ЛВУ не являются рекурсивными (иначе на нижнем уровне пришлось бы организовывать полноценный стек для процедур ЛВУ).

Есть и некоторые другие ограничения – любой достаточно внимательный программист, подробно изучив предлагаемую логику работы программы, сможет их перечислить.

Все это может показаться достаточно избыточным и громоздким. Спрашивается – а для чего все это нужно ?

Для чего все это нужно ?

1) Выделение ЛВУ, независимой от изменений в исходном коде низкоуровневых процедур, фактически означает проведение алгоритмизации программы – ее высокоуровневые алгоритмы становятся ясными и наглядными.

2) ЛВУ для основных императивных языков программирования высокого уровня становится унифицированной и может быть автоматически переведена с одного языка программирования на другой.

3) Визуализация алгоритмов ЛВУ становится делом техники и может выполняться автоматически.

4) исходный код ПО становится более читаемым для программиста, что способствует снижению издержек на разработку и сопровождение ПО.

Может, кому-нибудь предлагаемый материал покажется интересным.

Дмитрий_ВБ