ISaGRAF - это очень просто!

(часть IV, Язык программирования ST)

Гулько С.В., ХОЛИТ Дэйта Системс, г.Киев

В предыдущих выпусках мы познакомились с двумя языками программирования - **SFC** и **FBD**. В среде ISaGRAF они выступают в качестве основного инструментария, который используется для создания каркаса проекта. Графическое представление значительно облегчает построение приложения, так как дает возможность взглянуть на задачу как бы сверху, выделить основные логические узлы и правильно задать информационные потоки между компонентами. Т.е. это даже больше напоминает проектирование, нежели программирование! Однако не следует забывать и об обратной стороне высокого уровня - трудные задачи решаются достаточно просто, а вот простые, своего рода полировка, уже намного сложнее.

Почему так? Почему нельзя воспользоваться одним языком для создания всего приложения? Почему же, можно. Многие используют для этого только **FBD**, иногда можно встретить полностью законченные решения на **ST** или **LD**. Конечно, это дело вкуса, опыта и самого проекта, однако если делать "по науке", то следует использовать тот же **FBD**, где в качестве основных компонентов будут выступать Вами же созданные блоки. Такой подход чем-то аналогичен любому ремеслу, например, скульптор может изначально использовать для первичной обработки ту же болгарку, а уже потом в ход пойдут разнообразные молотки и зубила. В программировании ситуация очень похожая. Идеальных, пригодных для решения всех задач, языков нет (личное мнение автора), поэтому очень важным оказывается умение комбинировать их. ISaGRAF в этом отношении дает ряд преимуществ, так как он уже включает в себя 6 (шесть!) разноплановых языков, каждый из которых закрывает ту или иную нишу. Плюс эти языки можно комбинировать - можно написать функцию на одном и вызывать ее, используя конвенцию другого. Например, переходы и события **SFC** можно описать на **ST**, функции **ST** усовершенствовать на **IL** или предоставить пользователю основной интерфейс, выполненный на **LD**. Как Вы уже догадались, речь в этом номере пойдет именно об этих "промежуточных" языках.

Сразу хочу извиниться за то, что использовал слово промежуточный. Все три языка являются полноценными и независимыми, достаточно много людей создают приложения исключительно на них, не применяя **SFC** и **FBD**. Такой подход оправдан, когда создается небольшой проект, не содержащий большого количества хитросплетений алгоритмов. А вот крупное приложение часто лучше разбить на несколько частей, которые будут реализовываться на разных языках. В случае правильного выбора языков такой подход даст неоспоримые преимущества в вопросах последующего сопровождения, так как весь алгоритм будет "на виду" и лишь некоторые его составные части будут сокрыты.

Итак, приступим....

Язык ST

ST (Structured Text, Структурированный Текст) - это структурированный язык высокого уровня, разработанный для процессов автоматизации. Этот язык в основном используется для создания сложных процедур, которые не могут быть легко выражены при помощи графических языков. **ST** является языком, предназначенным для описания действий внутри шагов, выражения условий языка **SFC** или действий и тестов языка **FC**. Язык был сознательно упрощен, но таковы были требования решаемых задач. В **ST** Вы не встретите ни работы с памятью, ни прямого доступа к портам, да они и не нужны. Если программа будет работать под управлением окружения ISaGRAF, то все эти вопросы берет на себя целевая система.

Синтаксис языка напоминает Pascal, в нем нет каких-либо подводных камней и особенностей использования, так что освоить его будет достаточно просто. **ST** программа - это список **ST** предложений. Каждое предложение заканчивается точкой с запятой (";"). Имена, используемые в исходном коде (идентификаторы переменных, константы, ключевые слова и т.п.), разделяются неактивными разделителями (пробелами, концами строк или табуляторами) или активными разделителями, которые имеют строго определенное назначение (например, разделитель ">" обозначает сравнение "больше чем"). В текст

```
if Pump  and Level<100.0 then
   V1:=V1+1.0;
end_if;
if Podacha_s and Level<100.0 then
   V2 := V2 + 0.5;
end_if;
Level := V1 + V2;
if V2<>0.0 then
   Concentartion := (V2/(V1+V2))*100.0;
else
   Concentration := 0.0;
end_if;
if (Osn_zadvijka or Avar_zadvijka)  and  Level>=2.0 then
   Kv:= V1/Level;
   Kc:= V2/Level;
   Level := Level - 2.0;
   V1:=V1 - 2.0*Kv;
   V2:=V2 - 2.0*Kc;
end_if;
```

могут быть произвольно введены комментарии. Комментарий должен начинаться символами "(*" и заканчиваться "*)".

Основные предложения языка **ST**:

- предложение присваивания (**Variable := expression**;);
- вызов функционального блока;
- предложения выбора (**IF, THEN, ELSE, CASE**...);
- итерационные предложения (**FOR, WHILE, REPEAT**...);
- управляющие предложения (**RETURN, EXIT**...);
- специальные предложения для связи с такими языками как **SFC**.

Операции с данными

Все языки стандарта IEC 61131 обладают жесткой типизацией. Это значит, что если переменная объявлена как **Integer**, то Вы не сможете выполнить операцию по ее сложению с, предположим, строковой переменной. Статическая типизация заставляет вырабатывать более жесткий и организованный подход к программированию, и еще на стадии компиляции будут выявлены ошибки присвоения или обработки разнотиповых и несовместимых данных.

Рассмотрим такой пример:

(bool1 AND bool2) - данное выражение является корректным, так как обрабатываются две переменных одного типа

(sin(12.2) – 4.3) - тоже корректно, так как оба значения имеют тип **REAL**

(bool1 + sin(12.2)) - приведет к ошибке во время компиляции, так как содержит операции над разными типами.

Скобки, которые использовались в предыдущем примере, могут также служить для задания приоритетов выполнения операций или же для разбития сложного выражения на более простые составные части.

Например:

```
3*4+5 = 17;
3*(4+5) = 21;
```

Как видите, введение скобок значительно изменило результат. В ISaGRAF используется стандартная схема приоритетности операций - операции над битами и логические, умножение, деление и затем уже сложение и вычитание (схема сокращена для упрощения изложения).

В **ST** используется следующая операция присвоения:

```
переменная := значение;
```

"значение" может представлять собой как функцию, так и переменную или выражение. Важно лишь соблюдение типов данных и направления.

Это значит, что Вы не сможете осуществить запись во входную переменную или же чтение из выходной переменной.

Примеры записи этой операции:

```
var1:= 123;
var1_real:= var1/2.2;
var2_real:= sin(var1_real);
```

Управление ходом программы if-then-else-end_if

Одной из наиболее часто используемых операций является сравнение. Синтаксис стандартной конструкции **if-else** в **ST** следующий:

```
if логическое_выражение then
(*команды*)
elsif логическое_выражение then
(*команды*)
else (*команды*)
end_if;
```

Для данной конструкции обязательным является исключительно наличие **if-end_if**, дополнительные же условия могут и отсутствовать. Команд **elsif** может быть больше, чем одна, что позволит превратить конструкцию **if-then-else** в булевый аналог операции **case-of**. Не стоит пренебрегать использованием **elsif**, это достаточно мощный инструмент, правда, к его использованию нужно привыкнуть. В отличие от **elsif**, число которых может быть более одного, и которые позволяют задавать условие, в конструкции **if** может существовать только один **else**. Этот блок не позволяет назначать какие-либо условия и его выполнение будет осуществлено только в том случае, если ни одно из условий оператора сравнения не оказалось истинным. Это своего рода запасной вариант, когда ничто другое не помогает и не срабатывает.

Рассмотрим небольшой пример использования:

```
if manual and not (alarm) then
        level := manual_level;
        bx126 := bi12 OR bi45;
elsif  over_mode then
        level := max_level;
elsif over_mode and not (alarm) then
        level := max_level/2;
else  level := (lv16 * 100) / scale;
end_if;
```

Как видите, для того, чтобы использовать **if**, нужно задать логическое условие. Им может быть как переменная типа boolean, так и выражение (2 > 1) или результат, возвращаемый функцией

```
if is_odd(value) then (*Число нечетное, выполняем вычисления*)
else  (*Четное число*)
end_if;
```

Оператор case-of-end_case

Оператор **case** предназначен для создания листов сравнения на основе целочисленных констант:

```
case целочисленное_выражение of
    значение:
        действие;
        действие;
    значение: действие;
    значение,значение: действие;
else
    действие;
end_case;
```

По большому счету, **case** представляет собой частный случай **if**. Приблизительно аналогичную функциональность можно получить с использованием **if-elsif**, однако код выйдет более громоздким. Предположим, стоит задача написания функции декодирования ошибок. На вход функции будет подаваться код ошибки, а выходным значением будет текстовая строка с описанием, доступным для человеческого понимания. Давайте сделаем два небольших примера, один на **if**, второй на **case** и сравним результат.

Для начала определим в Словаре три целочисленных константы и присвоим им следующие значения:

```
CONNECTION_ERROR = 1
ENGINE_ERROR     = 2
GENERAL_ERROR    = 3
```

Не вдаваясь в подробности создания функций, перейдем сразу к написанию кода для **if**:

```
if error_code = CONNECTION_ERROR
then
    description:= 'Ошибка подключения';
elsif error_code = ENGINE_ERROR
then
    description:='Проблемы в модуле двигателя';
elsif error_code = GENERAL_ERROR
then
    description:='Общая ошибка';
else
    (*Если мы попали сюда - произошло что-то из ряда вон выходящее, так как код ошибки не объявлен. Пора серьезно поговорить с разработчиками*)
    description:='Неизвестная ошибка';
end_if;
```

Теперь перепишем приведенный выше кусок кода с использованием **case**:

```
case error_code of
      CONNECTION_ERROR:
          description:='Ошибка подключения';
      ENGINE_ERROR:
          description:='Проблемы в модуле двигателя';
      GENERAL_ERROR:
          description:='Общая ошибка';
   else
      description:='Неизвестная ошибка';
end_case;
```

Как видите, во втором примере код получился несколько компактнее и, на мой субъективный взгляд, более удобочитаемый по сравнению с **if** аналогом.

Оператор return

Кроме операторов ветвления, на ход выполнения программы непосредственно влияет оператор **return**. В его функции входит прекращение работы текущей функции или подпрограммы с последующей передачей управления "наверх". В отличие от других классических языков программирования, в ISaGRAF оператор **return** не используется для возврата того или иного значения.

Предположим, есть функция, которая выполняется только в том случае, если значение переменной ***current_value*** больше 12.5. В итоге получится следующий код:

```
if current_value <= 12.5 then
    (*Выходим, так как значение меньше допустимого*)
    return;
end_if;
    (*Если мы здесь, то можно быть уверенным, что значение current_value больше чем 12.5*)
```

Операторы организации циклов

Как-то на одном из сайтов IT-новостей мне довелось прочесть забавное обсуждение, какой же из языков программирования лучше. Кто-то в шутку высказал предположение, что **html**, что вызвало две ударных волны противоположной направленности. Часть людей, опять же в шутку, стала отстаивать данное утверждение. Вторая половина, не обладающая столь развитым чувством юмора, развязала "святую войну". Какие только доводы не приводились в пользу того, что **html** не является полноценным языком программирования - начиная от вполне разумных, и заканчиваю прямым переходом на личность и списком вычеркнутых модератором нецензурных выражений. В этом хаосе родилась замечательная мысль - в **html** нет циклов, следовательно, это не язык программирования в прямом смысле этого слова.

Что ж, языки стандарта IEC 61131 можно считать нормальными с этой точки зрения. Каждый из них позволяет организовывать циклическое выполнение участков кода. Да, в зависимости от того, какой язык используется, меняется и сам метод, однако не меняется суть.

В **ST** существует 3 оператора, позволяющие создать циклическое выполнение кода. Все они обладают важной особенностью, о которой необходимо помнить - вследствие синхронной природы целевой системы ISaGRAF, переменные, имеющие направление ***Input*** или ***Output***, не обновляются ядром до полного выхода из цикла. Об этом следует помнить, и тогда Вам удастся избежать достаточно большого числа недоразумений.

Цикл for

Это простой и наиболее часто встречающийся в классическом программировании вариант. Цикл основан на использовании целочисленного счетчика с последующим сравнением его значения с граничным.

Синтаксис команды следующий:

```
for index:=min to max by step do
    (*код*)
end_for;
```

index представляет собой счетчик, значение которого будет увеличено (или уменьшено) на ***step*** при каждой последующей итерации. Если Вы не указали значение ***step***, то шаг цикла в таком случае будет равен 1.

min является начальным значением счетчика и может быть как константой, так и переменной, вычисляемой или возвращаемой из функции.

max - граница, при достижении которой переменной ***index*** происходит выход из цикла.

Рассмотрим небольшой пример:

```
for index:=1 to 10 by 2 do
      result := result*index;
end_for;
```

Цикл while

Цикл **while** работает с булевыми условиями:

```
while логическое_условие do
      (*код*)
end_while;
```

Логическим условием может быть как переменная, так и выражение или результат вычисления в функциональном блоке. Цикл будет выполняться до тех пор, пока логическое условие будет истинно. Особенность **while** является то, что тест на истинность выполняется перед тем, как будут обрабатываться первые команды из секции кода.

```
while(DataIsReady()) do
      line := ReadLine();
      processLine(line);
end_while;
```

Цикл repeat-until

Данный цикл по синтаксису очень похож на **while** за одним небольшим исключением - сначала выполняется код, а в конце будет произведен тест логического условия. Это значит, что блок кода выполнится хоты бы один раз вне зависимости от того, истинно условие или нет.

```
repeat
   (*код*)
until логическое условие
end_repeat;
```

Команда выхода из цикла exit

Иногда возникает ситуация, когда необходимо принудительно завершить выполнение цикла. Например, стоит задача поиска в массиве чисел или символов. По логике приложения хватает первого совпадения. Как бы мы поступили, не будь доступна команда **exit**? Вероятно, сохранили бы найденное значение в промежуточной переменной и подняли бы условный флаг, что значение найдено, после чего продолжили бы сканирование всех, уже ненужных элементов массива. Это не столь критично, когда мы располагаем массой процессорного времени, однако в контроллерах это отнюдь не так, и чем быстрее завершится операция, тем лучше.

Проиллюстрируем все вышесказанное небольшим примером:

```
length := mlen(message);
found := NO;
for index := 1 to length by 1 do
   code := ascii (message, index);
   if (code = searched_char) then
      found := YES;
      exit;
   end_if;
end_for;
```

Редактор ST

Может показаться странным, почему я решил коснуться вопроса редактора в конце статьи, а не в ее начале. Отвечу - посмотрите на синтаксис самого языка. Все описание поместилось на несколько журнальных страницах, что говорит о легкости и доступности **ST**. Язык включает в себя небольшой набор команд, которых вполне достаточно для того, чтобы разработать сложное приложение. Не забывайте, что все функциональные блоки можно превратить в вызов функций, которые затем можно вызвать уже из **ST**.

В заключении скажу пару слов о редакторе. На его панель выведены кнопки управления, в соответствие которым поставлены команды языка, т.е. программирование получается полу-визуальным, если можно так сказать. При нажатии на кнопки, в поле редактирования будут возникать синтаксические конструкции, которые затем нужно будет заполнять кодом. Никто не запрещает набирать текст вручную (что, зачастую, быстрее), но никто и не заставляет использовать исключительно кнопки. Есть свобода выбора, а это главное.

КОНТАКТЫ:
тел: (044) 492-31-08, 492-31-09
e-mail: s.gulko@isagraf.com.ua