

Система программирования Модула-2 представляет собой систему из четырех относительно независимых компонент:

- 1) компилятора с языка Модула-2;
- 2) универсального редактора текстов;
- 3) средств отладки программ;
- 4) набора библиотек стандартных процедур.

#### Запуск компилятора

Запуск компилятора осуществляется с помощью утилиты m2, либо с помощью турбо-компилятора e2 (описание см. в томе "Утилиты ОС Excelsior").

Модула-2 компилятору передается текст программы, оформленный в соответствии с описанием языка. В системе принято стандартное расширение для имени файла, содержащего определяющий модуль - .d ; реализующий или программный модуль - .m . Если расширение отсутствует, компилятор добавляет расширение .m : program.m.

В результате компиляции определяющего модуля получается символьный файл (симфайл); реализующего модуля - симфайл и кодовый файл (кодофайл); программного модуля - симфайл и кодофайл одновременно.

Имена симфайлов и кодофайлов получаются из имени модуля (не из имени файла) добавлением расширителей '.sym' и '.cod' соответственно.

Сначала компилируют определяющий модуль, если он есть, затем - реализующий. Если программа состоит из нескольких модулей, при компиляции необходимо следить за порядком импорта, иначе при запуске задачи возможен конфликт версий.

#### Сообщения компилятора

Компилятор выдает на русском языке сообщение о характере ошибки, приводится номер строки и сама строка, в которой допущена ошибка, а также помечается предположительно место в строке, где надо искать ошибку (в случае турбо-компилятора происходит последовательное позиционирование курсора на место ошибки), например:

Ожидался символ ';' ;

104: PROCEDURE Next(i,j: INTEGER\$, A: ARRAY OF CHAR);  
Число ошибок: 1

Если в процессе компиляции не обнаружено ошибок, на экран выдается сообщение:

Modula 2.50 [P2.5] (c) KRONOS. v1.9 /12-Jan-88/ <h> "help.m"  
строк 242 время 06ср + 12io код: 325w "help.cod"

В результате компиляции на текущей директории появятся файлы help.sym и help.cod.

## Перечень сообщений Модуля-2 компилятора

- 1) Не удалось открыть симфайл для внешнего модуля
- 2) Слишком много идентификаторов!
- 3) Переполнена хеш таблица
- 4) Переполнение стека выражений!
- 5) Неожиданный конец исходного текста!
- 6) Переполнение строкового пула!
- 7) Переполнена куча (heap) таблицы символов!
- 8) Слишком много типов!
- 9) Должно быть имя блока
- 10) Неправильный (неконстантный) или вырожденный отрезок
- 11) Метки CASE должны быть из [0..32767]; размах < 256
- 12) Отсутствует 'h' после шестнадцатеричного
- 13) Незакрытая или слишком длинная строка!
- 14) Непонятный знак игнорируется
- 15) Невидимый объект
- 16) Повторно объявлен
- 17) Слишком глубокая вложенность
- 18) Неправильное начало программы
- 19) Неудачное обращение к файловой системе
- 20) Должен быть тип
- 21) Слишком длинный список
- 22) Типы несовместимы
- 23) Должен быть скалярный тип
- 24) Ошибка в конструкторе типа
- 25) Должен быть простой (1 слово) тип
- 26) Неправильное константное выражение
- 27) Неправильное выражение
- 28) Должна быть переменная
- 29) Должен быть массив
- 30) Не обладает адресом
- 31) Не обладает значением
- 32) Неправильный вид
- 33) Должна быть запись
- 34) Слишком сложное условное выражение
- 35) Это константное выражение?
- 36) Ожидался оператор
- 37) Ошибка в описаниях
- 38) Недопустимо в определяющем модуле
- 39) Где MODULE ??!
- 40) Слишком много EXIT'ов!
- 41) Шаг не из [-128..127]
- 42) EXIT вне LOOP'a
- 43) Это не модуль
- 44) Неправильный вызов
- 45) Выход за границы диапазона
- 46) Такая метка уже была
- 47) Функция вместо процедуры или vice versa
- 48) Экспорт невозможен. Объект уже объявлен
- 49) Код записывается байтами! [0..0FFh]
- 50) Неправильное число параметров

- 51) Это не процедура
- 52) Должен быть тип указателя
- 53) Должен быть тип множества
- 54) Извините, но здесь нельзя писать RETURN!
- 55) Переполнение (исчерпание) в константном выражении
- 56) Это не симфайл
- 57) Устаревшая версия симфайла
- 58) Переполнение таблиц при записи симфайла
- 59) Нереализованная процедура
- 60) Скрытый тип должен быть однословным
- 61) Слишком большой размер типа (больше > 256KW)
- 62) Попытка подсунуть чужой симфайл (не на Модуле-2)
- 63) Повторный FORWARD
- 64) Слишком большая процедура
- 65) Слишком большой модуль
- 66) Незакрытый комментарий начавшийся в строке ;
- 67) Неправильный синтаксис строки
- 68) Неправильное употребление IMPORT FROM ...
- 69) Конфликт версий (по времени компиляции)
- 70) SEQ параметр не последний
- 71) Пока допустимо только SEQ x) простой тип
- 72) В CASE нужна хоть одна альтернатива;
- 73) Слишком мало памяти для работы компилятора;
- 74) Такое использование ARRAY OF не реализованно;
- 75) Противоречивая спецификация параметра;
- 76) Присваивание VAL переменной (Только Для Чтения);
- 77) Извините, VAL переменные только в DEFINITION MODULE;
- 78) Нет памяти для компилятора;
- 79) Компилятор прерван;
- 80) Неизвестная ошибка.

#### Условия функционирования системы программирования

СП Модуля-2 работает в среде операционной системы Excelsior. Для функционирования ОС Excelsior рекомендуется следующая структура системного носителя.

Системный носитель должен содержать:

- директорию, содержащую коды драйверов внешних устройств и командные файлы конфигурации системы - etc;
- директорию, содержащую коды библиотек стандартных процедур и утилит - bin;
- директорию, содержащую симфайлы библиотек стандартных процедур - sym;
- директорию, содержащую тексты библиотек стандартных процедур - lib.

Компилятор и редактор загружаются в оперативную память. На директории bin, sym и текущую директорию должен быть установлен путь (см. том "Кронос для начинающих").

Кроме указанных файлов, для успешной работы редактора и монитора-подсказки help на директории bin должны присутствовать следующие специальные файлы:

- .ex.\$\$\$;
- .ex.help;

- .ex.keys.
- help.help
- help.\$\$\$.

Убедиться в их наличии можно с помощью утилиты ls.

## О СТАНДАРТЕ ЯЗЫКА МОДУЛА-2

Язык Модула-2 реализован компилятором полностью в соответствии с сообщением о языке и изменениями, внесенными Н.Виртом.

Кроме того, реализованы некоторые расширения, выработанные и рекомендованные группой по стандартизации языка Модула-2, заседание которой состоялось в Новосибирске в 1987 г.

В разделе 3 приводятся ограничения языка текущей реализацией Модула-2 компилятора.

## Расширения языка Модула-2

### 1. Расширено понятие Обозначения.

```
$ Обозначение = КвалИдент
{ "." Идентификатор | "[" СписВыражений "]" | "^"
  | "(" СписВыражений ")"
}
```

Примеры:

```
TYPE ptr = POINTER TO RECORD f1,f2: INTEGER END;
```

```
PROCEDURE func(): ptr;
.....
```

```
VAR i: INTEGER;
```

```
CHAR(i):='a';
func()^.f1:=1;
```

### 2. Разрешены присваивания формальных массивов целиком.

Компилятор вставляет динамические проверки того, что длина массива в левой части равна длине массива в правой части оператора присваивания. Для литерных массивов размеры сравниваются не на равенство, а на больше или равно.

### 3. Добавлены стандартные процедуры:

#### 1) ASSERT(BOOLEAN [,INTEGER]);

Если b=TRUE, то вызов эквивалентен пустому оператору, иначе исполнение программы завершается. Второй параметр может отсутствовать, а если он есть, то это некоторая дополнительная информация о причине завершения. Например:

```
ASSERT(adr=NIL,НетПамяти);
```

где НетПамяти - это константа 0С.

#### 2) BYTES( Обозначение | Тип )

Аналогична процедуре SIZE, но выдает размер объекта или типа в байтах. Процедура SIZE выдает размер в единицах адресации (для процессоров Кронос в словах).

3) BITS( Обозначение | Тип )

Аналогична процедуре SIZE, но выдает размер объекта или типа в битах. Для отрезков и перечислимых типов выдает минимальное число битов необходимое для представления значений.

4) LOW( ARRAY OF T ): T;

Аналогична процедуре HIGH. Выдает значение нижней границы массива.

5) SMALL( CHAR ): CHAR;

Аналогична процедуре CAP. Выдает маленькую букву по большой.

4. У процедуры HALT разрешен необязательный параметр типа INTEGER, значение которого системно-зависимо. Вызов процедуры HALT с параметром означает аварийное завершение задачи и может приводить к действиям, облегчающим отладку. Вызов процедуры HALT без параметра означает нормальное завершение задачи, неотличимое от выполнения возврата из нулевой процедуры головного модуля.

5. Введена конструкция, позволяющая импортировать из какого-либо модуля все экспортируемые этим модулем объекты.

```
$ Импорт = [ FROM Идентификатор ] IMPORT СписИдент ";"  
          | IMPORT FROM СписИдент ";"
```

Пример:

```
IMPORT FROM Terminal;  
(* доступны все объекты модуля Terminal *)  
.....  
WriteString('hello');
```

6. Введены процедуры с переменным числом параметров. Такие процедуры позволяют существенно облегчить использование библиотек ввода/вывода и уменьшить число процедур в них.

```
$ ФормальныеПараметры =  
  "(" [ФПСекция {";" ФПСекция} [Последовательность]] ")"  
  "[" ":" КвалИдент ].
```

```
$ Последовательность = SEQ идент ":" Тип.
```

Последним параметром у процедуры может быть параметр - последовательность. При вызове процедуры вместо такого формального параметра может быть последовательность фактических параметров (в том числе и пустая). Внутри процедуры такой параметр аналогичен параметру типа ARRAY OF T.

Пример:

```
PROCEDURE Min(SEQ x: INTEGER): INTEGER;  
  VAR min,i: INTEGER;  
  BEGIN ASSERT(HIGH(x)>=0);  
    min:=x[0];  
    FOR i:=1 TO HIGH(x) DO
```

```

        IF x[i]<min THEN min:=x[i] END
    END;
    RETURN min
END Min;

```

.....

```
i:=Min(1,2,3,4);
```

Параметр-последовательность SEQ x: T может быть передан целиком процедуре с формальным параметром SEQ y: T1, если тип T совместим с типом T1.

```

PROCEDURE p(SEQ x: INTEGER);
.....
    i:=Min(x);
.....

```

Замечание: если тип последовательности WORD, то фактическими параметрами могут быть не только однословные объекты, но и структуры. В этом случае передается только их адрес и копирования не происходит.

7. Введен механизм, позволяющий передавать структурные параметры по значению, но без копирования их (так называемая передача параметров по доступу). Это средство позволяет получить более эффективные процедуры и уменьшить требования к памяти. Компилятор контролирует, что значение этих параметров не меняется в получающей их процедуре.

\$ ФПСекция = [ VAR | VAL ] СписИдент ":" ФормТип.

8. В DEFINITION MODULE добавлены VAL-переменные.

9. Введена неявная конкатенация строк.

```

$ Строка      = Подстрока { Подстрока | Восьмеричное"с" }
$ Подстрока   = "'" { литера } "'" | '"' { литера } '"'.

```

Пример:

```
CONST тревога = ' ' 7с 'Спасайся кто может' 7с 12с 15с;
```

10. Введены комментарии как в языке Ада: --. Вся строка за '--' считается комментарием.

Пример.

```
CONST high=1024;  -- верхняя граница массива.
```

#### Ограничения, накладываемые реализацией

Текущая реализация компилятора допускает не более:

- 16 выходов EXIT из LOOP-цикла;
- 256 символов в строчной константе;
- 64 имен в списке имен;

- 7 параметров у процедуры (ARRAY OF и SEQ передаются двумя параметрами).
- 256 байт размах значений меток в операторе CASE. Метки только положительные.

.PAGE

## ПРИМЕР ЗАПУСКА ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

### 1. Описание действия программы

При входе в монитор программы оператор набирает произвольную строку текста, после чего происходит запуск одного за другим столько процессов, сколько символов в строке. Каждый процесс выдает на экран в случайной позиции отведенной программой колонки соответствующий ему символ, высвечивая его со случайной интенсивностью. После нажатия оператором любой клавиши начинается завершение процессов. Выход из монитора - передача пустой строки.

### 2. Текст программы.

Идентификаторы программы являются одновременно комментариями к ней.

```
MODULE sampBEST; (* Leo 02-Mar-88. (c) KRONOS *)
```

```
IMPORT Scheduler, SYSTEM, Heap, Terminal, Resource, Random;
```

```
FROM Terminal  IMPORT  SetCrs, Home, Clear, ClearLine
                  , print, Pressed, Read
                  , UnderLined, Cursor;
```

```
FROM Edit      IMPORT  ReadString;
```

```
FROM Strings   IMPORT  Str1;
```

```
MODULE ЭКРАН;
```

```
FROM Scheduler IMPORT  Gate, InitGate, EnterGate, ExitGate;
```

```
EXPORT ЗахватитьЭкран, ОсвободитьЭкран;
```

```
VAR Lock: Gate; (* "Замок" для использования ЭКРАНА *)
```

```
PROCEDURE ЗахватитьЭкран;
```

```
BEGIN
```

```
    EnterGate(Lock);
```

```
END ЗахватитьЭкран;
```

```
PROCEDURE ОсвободитьЭкран;
```

```
BEGIN
```

```
    ExitGate(Lock);
```

```
END ОсвободитьЭкран;
```

```
BEGIN
```

```
    InitGate(Lock);
```

```

END ЭКРАН;

MODULE Процессы;

IMPORT ЗахватитьЭкран, ОсвободитьЭкран;

FROM Heap      IMPORT  ALLOCATE, DEALLOCATE;
FROM Terminal  IMPORT  SetCrs, Write, Home, Clear, print
                        , LowIntensity, HighIntensity, Cursor
                        , UnderLined;
FROM Scheduler IMPORT  Signal, InitSignal, Send, Wait
                        , ProcessId, MakeProcess, Start
                        , SetHaltSignal;
FROM Random    IMPORT  RanGe;
FROM Resource  IMPORT  Final;

EXPORT Запустить, Остановить, ОстановитьВСЕ;

PROCEDURE интенсивность(i: INTEGER);
  VAR x: INTEGER;
BEGIN
  CASE ABS(i) OF
    |1: x:=HighIntensity(0);
        x:= LowIntensity(1);
    |2: x:= LowIntensity(0);
        x:=HighIntensity(0);
    |3: x:= LowIntensity(0);
        x:=HighIntensity(1)
  END;
END интенсивность;

CONST ПУСТО=ProcessId(NIL);
      размер=256;

TYPE ПамятьПроцесса=ARRAY [0..размер-1] OF INTEGER;

VAR BCE: ARRAY [0..79] OF
  RECORD
    процесс: ProcessId;
    буква   : CHAR;
    конец   : Signal;
    стой    : BOOLEAN;
    память  : POINTER TO ПамятьПроцесса;
  END;

  всего: INTEGER;

  ВзялСвоеТамГдеУвиделСвое: Signal;

  Колонка: INTEGER;

PROCEDURE Табло;
BEGIN
  ЗахватитьЭкран;
  SetCrs(0,25);
  интенсивность(3);

```



```

    print("РАБОТАЕТ ПРОЦЕССОВ #$$$2d#", всего);
    ОсвободитьЭкран;
END Табло;

PROCEDURE Печатальщик;
    VAR МояБуква: CHAR;
        МояКолонка: INTEGER;
            строка: INTEGER;
                счетчик: INTEGER;
                    инт: INTEGER;
BEGIN
    МояКолонка:=Колонка; Send(ВзялСвоеТамГдеУвиделСвое);
    МояБуква:=BCE[МояКолонка].буква;
    счетчик :=0;
    LOOP
        IF BCE[МояКолонка].стой THEN EXIT END;
        ЗахватитьЭкран;
        интенсивность(счетчик MOD 3 + 1);
        строка :=RanGe(2,16);
        -- случайное число в диапазоне [2..16]
        SetCrs(строка,МояКолонка);
        Write(МояБуква);
        ОсвободитьЭкран;
        INC(счетчик);
    END;
    интенсивность(2);
    ЗахватитьЭкран;
    FOR строка:=2 TO 18 DO
        SetCrs(строка,МояКолонка); Write(" ");
    END;
    ОсвободитьЭкран;
    HALT;
END Печатальщик;

PROCEDURE Запустить(колонка: INTEGER; чтопечатать: CHAR);
BEGIN
    IF (колонка<0) OR (колонка>HIGH(BCE)) THEN RETURN END;
    WITH BCE[колонка] DO
        IF процесс#ПУСТО THEN RETURN END;
        ALLOCATE(память,размер);
        IF память=NIL THEN RETURN END;
        процесс:=MakeProcess(Печатальщик,память,размер);
        SetHaltSignal(процесс,конец);
        буква:=чтопечатать;
        стой:=FALSE;
        Колонка:=колонка;
        Start(процесс);
        Wait(ВзялСвоеТамГдеУвиделСвое);
    END;
    INC(всего); Табло;
END Запустить;

PROCEDURE Остановить(колонка: INTEGER);
BEGIN

```

```

IF (колонка<0) OR (колонка>HIGH(BCE)) THEN RETURN END;
WITH BCE[колонка] DO
  IF процесс=ПУСТО THEN RETURN END;
  стой:=TRUE;
  Wait(конец);
  DEALLOCATE(память,размер);
  процесс:=ПУСТО;
END;
DEC(всего); Табло;
END Остановить;

PROCEDURE ОстановитьBCE;
  VAR i: INTEGER;
BEGIN
  FOR i:=LOW(BCE) TO HIGH(BCE) DO Остановить(i) END;
  интенсивность(2);
  IF Cursor(1)#0 THEN END;
  IF UnderLined(0)#0 THEN END;
END ОстановитьBCE;

VAR i: INTEGER;

BEGIN
  Home; Clear;
  всего:=0;
  InitSignal(ВзялСвоеТамГдеУвиделСвое);
  FOR i:=LOW(BCE) TO HIGH(BCE) DO
    WITH BCE[i] DO
      процесс:=ПУСТО; InitSignal(конец);
    END;
  END;
  IF Cursor(0)#0 THEN END;
  Final(ОстановитьBCE);
END Процессы;

PROCEDURE Показать(s: ARRAY OF CHAR);
  VAR i: INTEGER; char: CHAR;
BEGIN
  i:=0;
  WHILE (i<=HIGH(s)) & (s[i]#0c) & (Pressed()=0) DO
    Запустить(i,s[i]); INC(i)
  END;
  char:=Read();
  WHILE (i>=0) DO DEC(i); Остановить(i) END;
END Показать;

VAR s: ARRAY [0..80] OF CHAR;

BEGIN
  Str1(s,'          Kronos is the best computer in the world.');
```

```

  LOOP
    Home; Clear;
    SetCrs(23,14);
    print("каждый столбик печатается отдельным процессом");
```

```
SetCrs(19,20);
print("НАЖМИТЕ ЛЮБУЮ КНОПКУ ДЛЯ ОСТАНОВКИ");
SetCrs(21,00); ClearLine; print("%s\r",s);
Показать(s);
SetCrs(19,00); ClearLine; SetCrs(19,25);
print("НАБЕРИТЕ СВОЮ СТРОКУ");
SetCrs(21,00); ClearLine;
IF UnderLined(1)#0 THEN END;
IF Cursor(1)#0 THEN END;
print("%s\r",s);
ReadString("",s);
IF Cursor(0)#0 THEN END;
IF UnderLined(0)#0 THEN END;
IF s[0]=0c THEN HALT END;
END;
END sampBEST.
```