ТАРТУСКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

ТРУДЫ

ВЫЧИСЛИТЕЛЬНОГО ЦЕНТРА

57

ТАРТУ
1989

ТАРТУСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМЫ УПРАВЛЕНИЯ ОБРАБОТКОЙ ДАННЫХ

ТРУДЫ ВЫЧИСЛИТЕЛЬНОГО ЦЕНТРА

Выпуск 57

ТАРТУ 1989

Утверждено на заседании совета математического факультета ТГУ 23 декабря 1988 года

СИСТЕМЫ УПРАВЛЕНИЯ ОБРАБОТКОЙ ДАННЫХ.
Труды вычислительного центра.
Выпуск 57.
На русском языке.
Тартуский государственный университет.
ЭССР, 202400, г.Тарту, ул.Юликооли, 18.
Ответственный редактор Ю. Тапфер.
Подписано к печати 6.02.1989.
МВ 01329.
Формат 60x84/16.
Бумага ротаторная.
Машинопись. Ротапринт.
Условно-печатных листов 6,51.
Учетно-издательских листов 5,11. Печатных листов 7,0.
Тираж 300.
Заказ № 41.
Цена 1 руб.
Типография ТГУ, ЭССР, 202400, г.Тарту, ул.Тийги, 78.

СИСТЕМА ПРОГРАММИРОВАНИЯ НА БАЗЕ ЯЗЫКА Си ДЛЯ ПЭВМ ЕС

Т. Кельдер, М. Мериете

В настоящей статье описывается система программирования, предназначенная для разработки и выполнения на ПЭВМ ЕС программ, написанных на языке программирования Си. Входной язык системы соответствует оригинальному определению языка Си (см. [1]), с учетом уточнений и некоторых рекомендаций, собранных в проекте стандарта ANSI по языку Си (см. [2]). В составе программного обеспечения ПЭВМ ЕС разработанная система представляет собой версию 2 системы программирования СиМ86, первая версия которой в качестве входного языка использует подмножество языка Си.

Версия 2 системы программирования СиМ86 (в дальнейшем просто: система СиМ86) изложена в данной статье по следующей схеме. В первом разделе описывается назначение и состав системы СиМ86. Во втором разделе дается схема трансляции в целом, выделяются основные подзадачи компилятора и приводятся принятые в данной реализации соглашения. В третьем разделе описывается структура объектного модуля и связи между функциями. В следующих разделах коротко описываются способы пользования системой и её возможности, в том числе режимы работы компилятора и общий состав библиотеки системы СиМ86.

1. Назначение и состав системы СиМ86

Система СиМ86 разработана для ПЭВМ ЕС в операционной системе М86. В целом в состав системы входят:

1) компилятор с языка Си в машинный код ПЭВМ ЕС;

2) библиотека стандартных функций;

3) системные файлы-заголовки

и следующие утилиты операционной системы М86:

4) текстовой редактор;

5) сборщик объектных модулей СБОР86;

6) отладчик ОП86.

Система СиМ86 работает при стандартном программном и техническом обеспечении ПЭВМ ЕС 1841 в операционной системе М86. Создана также рабочая версия системы, предназначенная для операционной системы Альфадос. Система СиМ86 также работает при стандартном минимальном программном и техническом обеспечении ПЭВМ ЕС 1840.

Система выполняет следующие функции:

1) предоставляет пользователю все средства языка Си, ориентированного на задачи системного программирования;

2) обеспечивает применение в разрабатываемых программах стандартного программного обеспечения ПЭВМ ЕС в рамках операционной системы М86;

3) предоставляет широкий набор библиотечных функций в составе, становившимся стандартным в различных реализациях языка Си для ПЭВМ (см. [2]), с учётом особенностей операционной системы М86 и технических возможностей ПЭВМ ЕС типа 1841, а также типа 1840.

2. Схема трансляции и общее устройство компилятора

В компиляторе применяется однопроходная неоптимизирующая трансляция методом рекурсивного спуска. Данный метод выбран в целях повышения быстродействия и достижения сравнительно умеренного объёма компилятора для работы на относительно ограниченных по возможностям моделях ПЭВМ ЕС. С другой стороны, выбранный однопроходный метод влечёт за собой некоторые несущественные ограничения в исходном языке.

Транслируемый текст задаётся в одном файле, возможно, содержащим директивы препроцессора для включения других файлов в транслируемый текст. Результат трансляции - объектный модуль на машинном коде ПЭВМ ЕС - записывается в процессе трансляции в выходной файл.

В процессе компиляции транслируемая строка помещается в исходный буфер. При этом буфер заполняется по мере необходимости. Таким образом, входной текст сканируется в стиле "дай лексему".

В зависимости от языковой конструкции, начинающейся на обрабатываемой строке, вызывается соответствующая функция компилятора, т.е. каждому нетерминалу в грамматике исходного языка соответствует функция компилятора, и рекурсия в правиле вывода, определяющим данный нетерминал, приводит к рекурсии и в соответствующих функциях компилятора.

Обработка определений (объявлений) переменных и директив препроцессора приводит к дополнению соответствующих таблиц компилятора, т.е. таблицы имён и таблицы макроподстановок, а обработка инициаторов - к выводу кода инициатора.

При обработке определений функции сперва обрабатывается заголовок функции - проверяется корректность объявления аргументов, создаётся и заполняется таблица локальных имён транслируемой функции. После этого обрабатывается тело транслируемой функции. В зависимости от языковой конструкции вызывается соответствующая процедура компилятора.

В этих же процедурах вызываются подпрограммы генерации объектного кода соответствующих операций.

Для сосредоточения внимания на конкретных подзадачах трансляции представлялось целесообразным выделить подзадачи компилятора и средства их реализации в виде абстрактных типов данных.

Таким образом, для каждой подзадачи описываются соответствующие структуры данных и основные операции, допускаемые на этих структурах. Описываемые в таком виде средства реализации используются в компиляторе, следуя вышеприведённой схеме трансляции. Подробности их применения рассматриваются при описании соответствующих структур конкретной подзадачи. Выделяются следующие подзадачи:

1) общение с пользователем;
2) ввод и буферизация исходного текста;
3) макрообработка исходного текста;
4) работа с таблицами имён;
5) разбор описаний и определений;
6) разбор операторов управления;
7) разбор выражений;
8) генерация кода и построение объектного модуля;
9) диагностика ошибок и продолжение разбора.

Более подробно эти подзадачи и связанные с ними структуры данных описываются в статье [3].

Кроме общей схемы трансляции и выделения подзадач приведём принятые в данной реализации соглашения относительно входа компилятора, модели памяти самого компилятора и оттранслированной программы.

Входом компилятора является текстовой файл, составленный на языке СиМ86 и оформленный при помощи произвольного текстового редактора операционной системы М86 (редактор ПАСКАЛЬ редактор РТ). Подробное описание входного языка см. в [4].

Выходом компилятора является объектный модуль, оформленный согласно требованиям сборщика СБОР86 (система АСМ86) операционной системы М86. Так как формат объектного модуля соответствует формату фирмы Intel (Intel relocatable object module format), то это позволяет перевести оттранслированные модули из операционной системы М86 в операционную систему Альфадос.

Языком реализации компилятора выбран язык ассемблера АСМ86. Такой выбор служит цели достижения более высокой эффективности компилятора как в смысле быстродействия, так и в смысле пользования памятью.

По модели памяти сам компилятор состоит из четырёх сегментов:

а) кодовый сегмент CODE объёмом около 40К байтов;

б) сегмент статических данных и магазина DATA (объём статических данных около 10К байтов);

в) сегмент динамических данных EXTRA;

г) сегмент объектного модуля X1.

Объёмы последних двух сегментов определяются при сборке компилятора и могут быть зависимыми от конфигурации ПЭВМ.

Распределение памяти в сегментах EXTRA и X1 происходит под управлением подсистемы компилятора для динамического распределения памяти.

Общение с пользователем реализуется в компиляторе на простом языке команд, т.е. задание на компиляцию одного исходного файла представляет собой некоторую командную строку. Мнемоника команд и операндов задана в четвертом разделе данной статьи.

Язык общения - русский или английский, может быть конкретизирован пользователем. В зависимости от языка общения в компиляторе, естественно, меняется и язык диагностических сообщений. Более того, представление всех сообщений для компилятора в виде файлов сообщений с простой организацией позволяет пользователю путем добавления соответствующего файла сообщений ввести любой новый язык сообщений.

3. Структура объектного модуля и связь между функциями

Каждому исходному файлу соответствует один объектный модуль, выводимый компилятором в выходной файл. Имя этого файла совпадает с именем исходного файла, типом файла является OBJ. Компилятором поддерживается малая модель памяти, т.е. в объектном модуле существуют только два сегмента, максимальной длиной 64К байтов каждый: сегмент кода и сегмент данных. Все секции сегмента кода объединяются в группу под названием CGROUP, все секции сегмента данных объединяются в группу DGROUP.

При компиляции исходного файла в общем случае создаются следующие секции:

- для кодовой части всех функций транслируемого файла создаётся секция CODE;

- для внешних данных создаётся секция, название которой совпадает с названием файла;

- для статических строк каждой функции создаётся отдельная секция, название которой строится из названия функции с добавлением символа $ в конце имени.

Для каждой внешней переменной или функции создаётся элемент в таблице внешних имён. Для каждой переменной или функции, определённых как extern, создаётся свой элемент в таблице внешних ссылок. Одинаковые внешние имена из прописных и строчных букв различаются, т.е. преобразование имён не производится.

В пределах одной функции, исходя из методов доступа, данные разделяются на следующие типы памяти.

1) Регистровая память. Для работы доступны 6 регистров: ax, bx, cx, dx, si и di. Регистр bp применяется для адресации параметров, локальных переменных и рабочих полей. Регистр sp указывает на вершину магазина. При входе в функцию путем обращения к подпрограмме CCSAVE сохраняются все регистры кроме ax, а при выходе подпрограмма CCRESTOR восстанавливает все регистры кроме ax.

2) Память, доступная прямой адресацией. К данной категории памяти принадлежат все статические данные (внешние или внутренние), а также данные из других объектных модулей (объявленные через extern).

3) Параметры функций. Данный тип памяти адресуется регистром bp с положительными смещениями. Параметры размещаются в магазине (куда ссылается bp) в обратном порядке, т.е. наименьшие смещения имеют последние параметры. При обращении к функции, тип (значение) которой отличается от целого или ссылки, формируется дополнительный параметр - ссылка на поле для значения функции. Восстанавливать состояние магазина должна вызываемая функция. Это выполняется подпрограммой CCRESTOR, которой в качестве аргумента в регистре bx передаётся общая длина параметров в байтах.

4) Локальные переменные функции. Данный тип памяти адресуется регистром bp с отрицательными смещениями. Переменные, определённые раньше, имеют меньшие по абсолютной величине смещения.

5) Локальная рабочая память функции. Используется для сохранения промежуточных результатов вычислений с длинными и плавающими операндами, а также для значений функций нецелого типа. Эта область расположена непосредственно над областью локальных переменных и эти две области составляют адресуемую локальную память функции.

Общая длина локальной памяти для всех блоков функции вычисляется во время компиляции и память выделяется непрерывной областью полностью при входе в функцию путем установки регистра sp перед этой областью памяти.

6) Магазинная память. Эта область памяти находится над локальной рабочей памятью и непосредственно не адресуема. Используется для передачи параметров в вызываемую функцию и для сохранения промежуточных значений вычислений целой ариф-

метики при нехватке регистров. Переполнение магазина во время выполнения программы не проверяется.

Значения функций возвращаются либо в регистре ax (для функции целого типа или типа указатель), либо в специальном поле, обеспечиваемом вызывающей программой. В данном случае в регистре ax возвращается адрес этого поля.

4. Пользование системой

Для работы с компилятором требуется ПЭВМ типа 1841 или 1840 с памятью от 256К байтов и, как минимум, с одним дисководом. При вызове компилятора в команде задаётся имя исходного текста (первым аргументом) и желаемые режимы компиляции. Отсутствие же аргументов в команде означает включение режима трансляции нескольких исходных текстов. При этом для каждого транслируемого текста в диалоге запрашиваются имя текста и режимы компиляции. Например, командой

```
1>СИМ86
```

запускается компилятор для трансляции нескольких текстов. Последует подсказка

```
Введите командную строку:
```

по которой пользователь вводит имя транслируемого файла и режимы компиляции. Введение пустого ответа приводит к завершению работы.

Имя исходного текста указывается в качестве первого аргумента в командной строке компилятора. Допускаются имена согласно наименованиям файлов в операционной системе М86. Типом заданного исходного файла по умолчанию принимается СИ, типом объектного файла - OBJ.

За именем исходного файла в команде могут следовать один или несколько режимов компиляции. Режимы разделяются пробелами. Каждый режим имеет два обозначения: русское и латинское. Режим обозначается прописной или строчной буквой. Перечисляя ниже режимы, укажем латинское обозначение в скобках.

И(S)диск - параметр задаёт диск для исходного файла. Диск задаётся его именем согласно М86. По умолчанию исходный файл читается с текущего диска.

В(I)диск - параметр задаёт диск для включаемых файлов. По умолчанию принимается текущий диск.

О(O)диск - параметр задаёт устройство для выходного файла. По умолчанию выбирается текущий диск. Указанием символа "-" или "z" вместо имени диска транслятору сообщается, что выходной файл не создаётся.

П(P)устройство - задаёт устройство для вывода протокола. В протоколе приводятся транслируемый текст и диагностика трансляции. Независимо от объявления данного режима, в процессе трансляции на экран выдаются сообщения об ошибках. В качестве устройства допускаются:

- символ "Э" или "X" - вывод протокола на экран;

- символ "П" или "Y" - вывод протокола на печать;

- символ "Д" или "D" - вывод на печать только сообщений об ошибках;

- символ "-" или "Z" - пропуск протокола (принимается по умолчанию).

Я(L)язык - определяет язык для сообщений компилятора. По умолчанию остаётся тот же самый язык, который был уже определён. Язык задается одним символом, который соответствует

последнему символу x в названии errorsx используемой таблицы сообщений компилятора. Имеются две таблицы:

- errorsa - сообщения на английском языке;

- errorsr - сообщения на русском языке.

Таким образом, параметр Ar (или Lr) определяет русский, а Aa (или La) английский язык.

P(R) - допускается трансляция исходных текстов с русской лексикой. При этом допускаются русские служебные слова (см. [1]) и буквы русского алфавита в составе имён. Для этого на диске включаемых текстов должен быть размещён файл-заголовок RUSS.H. По умолчанию этот режим не принимается.

M(M) - в протоколе компиляции приводится исходный текст после макрообработки и препроцесса. По умолчанию приводится текст в исходном виде.

5. Структура и состав библиотеки СиМ86

В данном разделе коротко описываются основные классы функций библиотеки системы СиМ86.

1) Неявно вызываемые функции, названия которых имеют префикс CC (латинские буквы). В состав таких функций входят:

- функции, порождающие среду выполнения программы, или же функции обработки командной строки;

- функции сохранения и восстановления регистров;

- функции пересылки одних областей памяти в другие;

- функции преобразования данных;

- функции длинной арифметики и сравнения;

- функции арифметики и сравнения с плавающей точкой;

- вспомогательные функции системы ввода-вывода.

2) Функции ввода-вывода для консоли пользователя. В состав таких функций входят:

- ввод символа с консоли а также вывод его на экран или на печать;

- ввод строки с консоли;

- вывод строки на экран или на печать;

- ввод-вывод информации с преобразованием из внутренней формы представления во внешнюю и наоборот.

3) Функция для последовательной обработки файлов. В состав таких функций входят:

- открытие и закрытие файлов для ввода или для вывода;

- ввод-вывод символа;

- ввод-вывод строки;

- ввод-вывод заданного количества байтов;

- функции для позиционирования файлов;

- ввод-вывод информации с преобразованием из внутренней формы представления в внешнюю и наоборот.

4) Функции прямой обработки файлов. В состав таких функций входят:

- открытие и закрытие файлов;

- ввод-вывод заданного количества блоков;

- функции для позиционирования файлов.

5) Функции для обработки символов. В состав таких функций входят:

- определение принадлежности символа в заданный класс символов;

- преобразование строчных букв в прописные или же прописных букв в строчные.

6) Функции для обработки строк и полей памяти. В состав таких функций входят:

- поиск в строке вхождений символов или цепочек;
- пересылка строк и полей памяти;
- сравнение строк и полей памяти;
- сцепление строк;
- заполнение полей памяти заданным символом;
- преобразование данных из внутреннего представления во внешнее и наоборот.

7) Математические функции, в том числе:

- степенные и экспоненциальные функции;
- логарифмические функции;
- тригонометрические функции;
- обратные тригонометрические функции;
- некоторые специальные функции.

8) Графические функции, позволяющие изображать на экране графическую информацию. В состав таких функций входят:

- выбор графического режима и цвета изображения;
- функции для вывода на экран простейших изображений (отрезок, прямоугольник, окружность и т.д.);
- чтение и запись на графический экран;
- определение и выбор окон;

9) Функции для динамического распределения памяти.

10) Функции, представляющие некоторые возможности операционной системы М86. В состав таких функций входят:

- служба времени и даты;
- выполнение командного файла;
- удаление и переименование файлов и т.д.

6. Заключение

Языком реализации системы СиМ86 был выбран язык ассемблера АСМ86. Выбор языка ассемблера обусловлен намерением повышения эффективности системы по сравнению с первой версией системы СиМ88. Возникающие при использовании языка низкого уровня трудности преодолевались соблюдением при проектировании и программировании следующих принципов:

- проектирование и реализация сверху вниз, поддерживаемые принятым методом трансляции и позволяющие начинать отладку уже в ранних стадиях реализации;

- четкое выделение подзадач компилятора, их реализация и применение в стиле упрощенных абстрактных типов данных;

- высокая модульность всей системы: компилятор состоит из большого числа (около 250) сравнительно небольших модулей с чётко определёнными связями между модулями.

Соблюдаемая техника реализации в целом значительно способствовала и отладке компилятора - заметно облегчалась локализация ошибок и сама схема трансляции в некотором смысле управляла порядком разработки модулей компилятора.

Рабочая версия системы СиМ86 под операционной системой Альфадос служит основой доработки системы программирования для среды Альфадос (либо MS-DOS), также работающей на минимальном техническом обеспечении ПЭВМ ЕС. Несмотря на существование мощных оптимизирующих компиляторов для ПЭВМ (требующих больше и от самой ПЭВМ) система СиМ86 представляется целесообразным инструментом именно в составе минимально оснащенных моделей ПЭВМ ЕС.

С другой стороны, СиМ86 является и естественным дополнением средств программирования в ЕС ЭВМ, так как для этого же входного языка и по той же схеме трансляции реализована система программирования для ОС ЕС и СВМ ЕС (см. [5]). Целесообразным кажется использование этих двух систем программирования совместно в комплексе ПЭВМ ЕС - ЭВМ ЕС.

Литература

1. Kernighan, B.W., Ritchie, D.M. The C programming Language. Prentice-Hall, 1978. Русский перевод: Б. Керниган, Д. Ритчи, А. Фьюер. Язык программирования Си. Задачник по Си. Финансы и статистика, 1985.
2. Harbison, S.P., Guy, L.S. Jr. C: A reference manual. Second edition, Prentice-Hall, 1987.
3. Кельдер, Т., Мериете, М. Основные структуры данных компилятора СиМ86. Наст. сб. стр. 18-35.
4. Программное обеспечение персональных электронных вычислительных машин единой системы. Система программирования СиМ86. Версия 2. Руководство пользователя. 1988.
5. Кельдер, Т. Об одной реализации языка Си. Труды ВЦ ТГУ, 1988, 55, 3-18.

ОСНОВНЫЕ СТРУКТУРЫ ДАННЫХ КОМПИЛЯТОРА СиМ86

Т. Кельдер, М. Мериете

В настоящей статье описываются основные структуры данных компилятора СиМ86 (см. [1]), а также все основные операции с ними. При этом коротко характеризуются и все основные подсистемы компилятора.

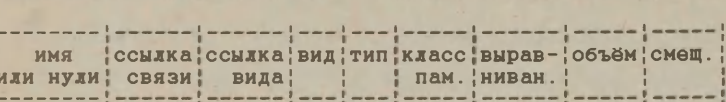
1. Таблица имён

Таблица имён служит сохранению всех описанных в транслируемом тексте имён (переменных, функций и т.п.), кроме меток операторов, и их атрибутов. Значения атрибутов конкретного имени определяют вид, класс памяти, распределение в памяти и тип соответствующего объекта. При помощи таблицы имён проверяются корректность объявлений и определений, корректность применения типов данных в операциях и другие контекстные условия. Накапливаемая в таблице информация используется также в процессе генерации объектного кода.

В целом таблица имён представляется как древовидная структура, при помощи которой указываются взаимосвязи элементов таблицы (например, связь имени записи с именем поля записи). По этой же структуре ведётся поиск, добавление и удаление элементов таблицы.

Объём таблицы определяется в процессе трансляции динамически - по мере необходимости выделяется память для нового элемента или же память элемента освобождается. Распределением памяти для таблицы имён занимается подсистема динамического распределения памяти.

Элемент таблицы имён, содержащий (возможно) имя объекта и атрибуты объекта, занимает 20 байтов следующей структуры:



Атрибуты своими значениями определяют "свойства" объекта следующим образом.

1) "Ссылка связи" определяет в случае объекта высшего уровня связь в таблице для поиска объекта, в случае же элементов составного объекта (записи, смеси или перечисления) связывает элементы одного и того же уровня.

2) "Ссылка вида" равна нулю или же ссылается на элемент таблицы, в котором уточняется вид данного объекта. Например, при переменной вида "запись" ссылаются на описание записи соответствующего типа.

3) Атрибут "вид" указывает производный тип (вид) объекта согласно следующим допускаемым значениям:

переменная	- 1;	указатель	- 2;
массив	- 3;	функция	- 4;
запись	- 5;	смесь	- 6;
перечислимый тип	- 7;	битовое поле	- 8.

Понятно, что некоторые виды (напр. битовое поле) могут появляться только в определённых языковых конструкциях.

4) Атрибут "тип" указывает простой тип объекта. При значении нуль данного атрибута обязательно существует ненулевая ссылка вида. Возможны следующие ненулевые значения:

символьный тип	- 1;	целый тип	- 2;
длинный тип	- 3;	целый без знака	- 4;
плавающий тип	- 5;	длинный плавающий	- 6;

5) Атрибут "класс памяти" указывает класс объекта:

внешний объявленный объект - 0 (класс extern);

внешний определённый объект - 1;

локальный автоматический объект - 2 (класс auto);

регистровый объект - 3 (класс register);

определение имени типа - 4 (класс typedef);

статический объект - 5 (класс static).

6) Атрибут "выравнивание" указывает возможное выравнивание объекта:

объект не выравнивается - 1;

объект выравнивается на слово - 2.

Примечание: Так как компилятор в данном варианте выравнивания объекта не производит, то данное поле в таблице имён можно считать зарезервированным для дальнейшего пользования.

7) Атрибут "объём" указывает требуемый объём памяти в байтах. Для некоторых объектов транслируемой программы этот же атрибут вместо объёма занимаемой памяти определяет следующую служебную информацию:

- в случае поля битов этот атрибут указывает константу сдвига для выделения значения поля битов;

- в случае функции здесь содержится ссылка на список параметров данной функции.

Значения простых типов данных занимают следующий объём памяти:

символьный (char)	- 1 байт;
целые (short, int, unsigned)	- 2 байта;
длинное целое (long)	- 4 байта;
указатель	- 2 байта;
плавающий (float)	- 4 байта;
длинный плавающий (double)	- 8 байтов.

8) Атрибут "смещение" определяет либо относительный адрес объекта в памяти либо какую-нибудь другую информацию, позволяющую оперировать данным объектом. Например, в случае внешних объявленных объектов здесь сохраняется индекс внешнего имени, в случае поля битов маска выделения значения объекта и т.п.

При трансляции элементы таблицы имён объединяются в две различные таблицы - в таблицу глобальных имён и в таблицу локальных имён. В таблице глобальных имён методом поиска служит расстановка с внешним сцеплением, в таблице локальных имён поиск линейный, обратный порядку создания таблицы.

Для констант перечислимых типов (enum) в таблицу создаётся дополнительный доступ.

Операции с таблицей имён выполняются в процессе трансляции практически на всех этапах трансляции кроме препроцесса. В зависимости от состояния трансляции вставляются новые элементы, сравниваются элементы или же удаляются.

Состояние и структура таблиц меняются динамически и в любой момент трансляции отражают все введённые уже в программе глобальные и локальные объекты.

Таблица глобальных имён заполняется при обработке определений и объявлений глобальных переменных, таблица локальных имён - при обработке параметров функции и других локальных объектов. Как правило, в операциях с таблицей имен определяются значения всех атрибутов за исключением (возможно) атрибутов распределения памяти. При появлении нескольких объектов с одним и тем же именем (например, при объявлении и последующем определении некоторого объекта), значения некоторых атрибутов могут переопределяться или добавляться. Операции включают в таблицу только корректные имена объектов.

2. Таблица макроопределений

В данном разделе описываются структуры данных, связанные с макроподстановкой лексем, т.е. таблица макроопределений и пул цепочек замены.

Элемент таблицы макроопределений создаётся по команде препроцессора #define. Каждый элемент занимает 20 байтов и размещается в динамической памяти компилятора. Таблица имеет структуру упорядоченного бинарного дерева.

Формат элемента таблицы макроопределений:

имя	ссылка влево	ссылка вправо	ссылка замены	ссылка парам.	длина замены	число парам.	резерв
0	8	10	12	14	16	17	18

1) Имя - идентификатор, заданный в команде препроцессора #define. По надобности имя сокращается до восьми символов или же добавляются в конце пробелы.

2) Ссылка влево, ссылка вправо - две ссылки связи, которые организуют таблицу как упорядоченное бинарное дерево.

3) Ссылка замены - указатель на соответствующую цепочку замены в пуле цепочек замены.

4) Ссылка параметров - указатель на дополнительную таблицу параметров при подстановке с параметрами (или 0, если параметров нет). Таблица параметров для каждой макроподстановки имеет аналогичную структуру упорядоченного бинарного дерева и те же поля, из которых заполнены только имя и две ссылки связи.

5) Длина замены - длина цепочки замены для данной макроподстановки.

6) Число параметров - число параметров (может быть нуль) для данной макроподстановки.

Пул цепочек замены, это непрерывная область (объёмом 2К байтов в динамической памяти, управляемая двумя ссылками:

- ссылка на первое свободное место;

- ссылка на конец области.

Операции с таблицей макроопределений выполняются при обработке препроцессорных команд #define, #ifdef, #ifndef и #undef, а также при макрообработке входной строки. Выделяются следующие операции:

а) добавление нового элемента в таблицу;

б) удаление элемента;

в) поиск элемента по имени;

г) макроподстановка лексем.

Отметим, что при удалении элемента соответствующие блоки таблицы освобождаются, но места в пуле цепочек замены не освобождаются. Переполнение данной области приводит к прекращению работы компилятора над данным файлом.

Рассмотрим несколько подробнее макроподстановку с параметрами. При появлении в обрабатываемой строке макровызова с параметрами таблица параметров заполняется так, что соответствие между формальными и действительными параметрами имеет тот же вид, что соответствие между макроопределениями и цепочками замены. После этого вызывается процесс макроподстановки рекурсивно, используя в качестве таблицы таблицу параметров для данной подстановки. Такой подход позволяет очень просто обрабатывать макроподстановки с несколькими уровнями (параметрами одной макроподстановки являются снова макроподстановки с параметрами и т.д.). Заметим, что рекурсия самих макроопределений (прямая или косвенная) не допускается, это приведёт к зацикливанию обработки и, как правило, к переполнению пуля цепочек замены.

3. Управляющий блок выражения

Управляющий блок выражения является основной структурой данных для обработки (под)выражения. Управляющие блоки выражений являются частью динамической памяти соответствующих подпрограмм компилятора для разбора подвыражений (т.е. операций определённого приоритета) и в соответствии с рекурсивными обращениями в компиляторе образуют дерево разбора выражения. В управляющий блок выражения объединены все атрибуты, которые характеризуют подвыражение и позволяют:

- проверять корректность применения операций;
- генерировать код для приведения типов операндов;
- генерировать код для операций;
- создавать управляющие блоки для результатов операций.

Длина каждого управляющего блока выражения - 20 байтов, а его формат:

тип адр.	флажки	объ. пам.	вид	тип	класс пам.	ссылка вида	ссылка знач.	ссылка табл.	значение
0	1	2	3	4	5	6	8	10	12

Указанные поля имеют следующие значения.

1) Тип адресаций - однобайтовое поле атрибута, определяющего способ доступа к данному операнду в объектной программе. Нулевое значение означает, что операнд - константа. Разряды 0 - 6 определяют следующие взаимоисключающие возможности адресации:

1h - регистровая переменная;

2h - рабочий регистр;

4h - локальная автоматическая переменная;

8h - магазин;

10h - внешняя (external) переменная;

20h - переменная с прямой адресацией (внешняя или локальная);

40h - локальное рабочее поле.

Последний, седьмой разряд - это флажок косвенности, который может появляться в комбинаций с любым другим разрядом. Флажок обозначает, что тип адресаций относится не к самому операнду, а к его адресу.

2) Флажки - в данном байте сохраняется ряд однобитных флажков, основные из них следующие:

1h - выражение представляет адрес значения; используется там, где предназначено автоматическое преобразование в адрес значения (напр., название массива, функции и т.п.);

2h - было явное взятие адреса (операция &);

4h - допускается взятие адреса;

10h - выражение представляет собой l-значение (допускается присваивание);

20h - имя переменной находится в локальной таблице имён;

40h - имя переменной находится в глобальной таблице имён; нулевое значение байта флажков допускается только при типе адресации 0 и означает константу 0.

3) Объём памяти - показывает в случае простых типов данных и строк требуемый объём памяти в байтах.

4) Атрибуты "вид", "тип" и "класс" памяти аналогичны соответствующим атрибутам элемента таблицы имен. Различие в том, что в данном случае они указывают вид, тип и класс памяти подвыражения, а не имени.

5) Ссылка вида - указатель на элемент таблицы имён (или 0), определяющий вид данного операнда в случае производных типов.

6) Ссылка значения - указатель на значение операнда, обыкновенно используется только в выражениях инициации для строчных констант. В данном поле запоминается длина операнда в случае записей и смесей.

7) Ссылка таблицы имён - указатель на элемент таблицы имён (или 0). Данный элемент определяет идентификатор, который являлся первичным выражением для данного операнда.

8) Значение - это восьмибайтное поле, содержимое которого зависит от типа адресации выражения. Если операнд константный, то здесь находится значение константы. В противном случае, со смещением 12 находится 1- или 2-байтовое значе-

ние, которое определяет операнд в объектной программе:

- в случае регистровых типов адресаций - номер регистра;

- в случае прямой адресации - смещение в сегменте;

- в случае локальной памяти - смещение в отношении регистра bp;

- в случае магазина - указатель логического стека;

- в случае внешней адресации - индекс внешнего имени.

Следующие 2 байта (со смещением 14) определяют сдвиг относительно первичного определения (может быть 0).

В случае регистровых типов адресации на поле со смещением 18 находится указатель на соответствующий элемент таблицы регистров (см. п. 4.).

Кроме дерева управляющих блоков выражений при компиляции выражения существенным является и глобальный атрибут "тип выражения". Возможны следующие значения (типы выражений):

константное выражение - 0;

выражение инициации - 1;

общее выражение - 2.

В константных выражениях (размеры массивов и т.п.) допускаются только константные операнды числовых типов. В выражениях инициации кроме констант допускаются и адресные операнды константного типа (т.е. адреса уже определённых объектов, в том числе строки-константы).

Блок управления выражением первоначально заполняется при обработке первичного выражения (идентификатор, числовая или символьная константа, строка). В ходе выполнения операций в выражении при помощи управляющих блоков для операндов генерируется соответствующий код и создается управляющий блок

для результата. При этом освобождаются ресурсы, которые были заняты под операндами.

Обнаружение синтаксической ошибки при компиляции выражения влечёт за собой (кроме сообщения об ошибке), как правило, создание управляющего блока результата типа "константа 0" и продолжение разбора с того же места.

4. Таблица распределения регистров

Таблица распределения регистров предназначена для управления регистрами при компиляции выражения, причем распределению подлежат 6 регистров: ax, bx, cx, dx, si и di. Регистр bp адресует динамическую память функции, регистр sp указывает на вершину стека. Сегментные регистры при выполнении компилируемой программы не меняются. Каждому из 6 регистров соответствует элемент таблицы распределения регистров. Таблица распределения регистров находится в памяти статических данных компилятора. Элемент таблицы распределения регистров имеет следующий формат:

номер регистра	флажки	ссылка на убв	ссылка вперёд	ссылка назад
0	1	2	4	6

1) номер регистра - задаётся в соответствии с машинным кодом ПЭВМ ЕС:

ax - 0; cx - 1; dx - 2; bx - 3; si - 6; di - 7.

2) В первом полубайте байта флажков 2 старших разряда определяют занятость регистра:

80h - регистр занят;

40h - регистр принадлежит регистровой переменной.

Второй полубайт определяет, является ли регистр базовым, и задаёт операнд r/m для постбайта кода:

для si - 4; для di - 5; для bx - 7.

3) Ссылка на управляющий блок выражения связывает элемент таблицы регистров с подвыражением, в котором занят регистр.

4) Две ссылки (вперёд и назад) связывают таблицу распределения регистров в двойной круговой список.

Кроме таблицы для распределения регистров пользуются еще глобальным указателем, указывающим на регистр, который распределяется следующим.

Операции с таблицей распределения регистров выполняются на этапе генерации объектного кода. В целом выделяются следующие примитивы.

а) Опустошение таблицы. Все регистры свободны, все указатели принимают исходные значения.

б) Зарезервирование регистра для регистровой переменной. Операция выполняется при определений регистровой переменной. Компилятор реализует первые два определения типа register для числовых переменных целого типа или для ссылок. Регистровыми переменными могут быть назначены регистры si и di. Связь с управляющим блоком выражения создаётся при обработке соответствующего первичного выражения.

в) Зарезервирование регистра. Если очередной регистр свободный, то он выделяется, иначе до этого выполняется операция разгрузки регистра в магазин.

г) Освобождение регистра. Освобождённый регистр ставится первым в списке регистров.

д) Замена регистра на регистр определённого типа. Регистры меняют свои места в списке, по необходимости генерируется соответствующий код. Существуют три типа замен:

- замена регистра на базовый регистр;
- замена регистра на ax;
- замена регистра на cx.

Замены необходимы для выполнения операций (в объектном коде) определённого типа (напр., базирование, умножение и т.п.).

е) Разгрузка регистра из магазина. Соответствующие замены производятся и в управляющем блоке выражения.

5. Таблица нерешённых переходов и меток

Так как компилятор однопроходный, нельзя непосредственно генерировать код для переходов вперёд. Для решения возникающейся проблемы введены две структуры данных:

- таблица логических меток;
- таблица нерешённых переходов.

Кроме того, с этими двумя таблицами тесно связана таблица меток goto, т.е. меток, определённых в исходной программе. Все эти таблицы находятся в динамической памяти компилятора, т.е. по надобности система распределения памяти вводит новый блок для таблицы или же освобождает блок. Элемент таблицы логических меток представляет собой двубайтовое поле. Элементы размещаются в блоках, размером по 20 байтов, а блоки связаны в список. На первое свободное место в таблице указывает глобальная ссылка.

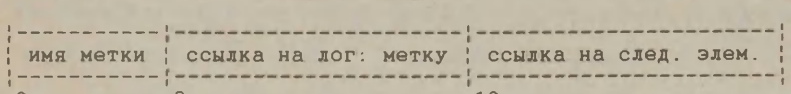
Элемент таблицы нерешённых переходов состоит из двух двубайтных полей. Они тоже размещаются в блоках размером по

20 байтов, связанных в список. Эти два поля содержат следующую информацию:

- относительный адрес операнда команды перехода в кодовой секции объектной программы;

- ссылка на соответствующую логическую метку.

Таблица меток goto состоит из элементов (по 12 байтов) для каждой появляющейся в исходной программе метки. Элемент этой таблицы имеет следующий формат:



1) Имя метки по необходимости сокращается до 8 байтов или добавляются пробелы.

2) Ссылка на логическую метку связывает элемент таблицы меток goto с соответствующей логической меткой.

3) Ссылка на следующий элемент связывает элементы таблицы в список.

Операции с данными таблицами выполняются на этапе компиляции структур управления исходной программы. Можно выделить следующие операции.

а) Зарезервирование логической метки. Операция выполняется при генерации перехода в объектной программе. Например, при компиляции предложения if...else резервируются метки для перехода на ветку else и для выхода из конструкции.

б) Присваивание значения логической метке.

в) Переход на логическую метку. Если метке уже присвоено значение, то генерируется полный код перехода; иначе создаётся новый элемент в таблице нерешённых переходов.

г) Добавление элемента в таблицу меток goto. Операция может выполняться в двух случаях:

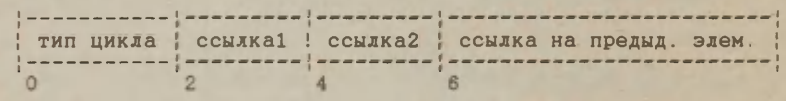
- при обнаружении новой метки в исходной программе;
- при обработке предложения goto.

В обоих случаях резервируется и соответствующая логическая метка. В первом случае ей присваивается значение.

д) Освобождение таблиц. Эта операция выполняется в конце компиляции тела функции. Переопределяются операнды переходов по таблице нерешённых переходов. При освобождении таблицы обнаруживается и отсутствие какой-либо метки.

6. Магазин операторов цикла

Магазин операторов цикла и переключателей используется для генерации кода операторов break или continue. Восьмибайтовый элемент этого магазина имеет формат:



1) Тип цикла определяет, с каким типом управляющей конструкции имеется дело (while, for, do или switch).

2) Ссылка1 является указателем на элемент таблицы логических меток для выхода из конструкции (по оператору break).

3) Ссылка2 является указателем на элемент таблицы логических меток для продолжения (по оператору continue). В случае переключателя здесь стоит ссылка на список вариантов. Вариант дается четырёхбайтовым полем, содержащим значение константного выражения (или специальное значение для default) и указатель на соответствующую логическую метку.

4) Ссылка на предыдущий элемент связывает элементы магазина. На вершину магазина указывает специальный глобальный указатель.

С магазином циклов допускаются следующие операции.

а) Создание нового элемента. Выполняется, если при разборе обнаружена соответствующая языковая конструкция.

б) Освобождение элемента. Выполняется при выходе из обработки соответствующей конструкции.

в) Переход по оператору break, continue или по соответствующему варианту.

7. Таблица служебных слов

Как известно, служебные слова в языке Си зарезервированы и не могут быть использованы в других целях. В компиляторе для разпознавания служебных слов имеется специальная функция расстановки, которая разделяет все служебные слова без коллизий. Каждому служебному слову присвоено два атрибута, распознаватель служебных слов выдаёт эти атрибуты для анализа.

Атрибутами служебных слов являются класс служебного слова и значение в классе. Выделяются следующие классы:

1 - класс памяти, возможные значения которого:

0 - extern; 1 - static; 2 - auto;

3 - register; 4 - typedef;

2 - префикс типа, возможные значения которого:

2 - short; 3 - long; 4 - unsigned;

3 - тип данных, возможные значения которого:

0 - char; 2 - int; 4 - float; 6 - double;

8 - struct; 10 - union; 12 - enum;

4 - оператор, возможные значения которого:

0 - continue; 2 - return; 4 - break;

6 - goto; 8 - if; 10 - while;

12 - for; 14 - do; 16 - switch;

18 - case; 20 - default;

5 - приставка оператора, возможно лишь одно значение:

0 - else;

6 - объём памяти, возможно лишь одно значение:

0 - sizeof.

Единственной операцией с таблицей служебных слов является распознавание служебного слова. Операция применяется на всех этапах синтаксического анализа.

8. Организация распределения памяти

Динамическая память компилятора выделяется из специального сегмента данных EXTRA. Явно в сегменте EXTRA существуют только восемь байтов, содержащие корень системы распределения памяти. После загрузки самого компилятора по длине сегмента EXTRA (на базовой странице памяти) инициируется система распределения памяти.

Рассматриваемая система является двухуровневой и работает по принципу сцепления свободных блоков.

1) Высший уровень - зарезервирование блока памяти произвольного размера.

2) Нижний уровень - зарезервирование блока памяти константной длины 20 байтов.

В основном работает нижний уровень. Нехватка памяти вызывает прекращение работы с данным исходным файлом.

9. Таблицы сообщений

Все сообщения компилятора объединены в специальные структуры данных - таблицы сообщений. Этих таблиц может быть несколько (на разных языках сообщений). Компилятор выбирает таблицу по специальному параметру в командной строке. Таблицы сообщений являются частью статических данных компилятора.

Формат таблицы сообщений:

```
<ссылка 0>
.........
<ссылка n>
<ссылка параметра> ... <ссылка параметра><сообщение 0>0
.....................................................
<ссылка параметра> ... <ссылка параметра><сообщение n>0
```

Первую часть таблицы сообщений составляют двухбайтовые относительные указатели на соответствующие сообщения. Перед произвольным сообщением могут стоять несколько (возможно, что ноль) указателей на переменные части данного сообщения (напр., имя файла, номер строки и т.п.). Операцией с таблицей сообщений является доступ к сообщению по индексу. Каждая часть компилятора, которая выдаёт то или иное сообщение, должна знать семантику данного сообщения, т.е. число и смысл параметров сообщения.

Литература

1. Кельдер, Т., Мериете, М. Система программирования на базе языка Си для ПЭВМ ЕС. Наст. сб. стр. 3-17.

РЕАЛИЗАЦИЯ ИНТЕРПРЕТАТОРА ЯЗЫКА СИ НА ИСКРА-226

К. Алев

Интерпретирующая система СиИ87-Искра предназначена для разработки и выполнения программ на языке программирования Си на вычислительной машине Искра-226. Система СиИ87-Искра состоит из двух частей - из транслятора с языка программирования СиИ87, преобразующего исходный текст во внутренний код и из интерпретатора внутреннего кода. Язык программирования СиИ87 является подмножеством языка Си, представленного в книге Кернигана и Ритчи [1].

Данная реализация языка Си является по своему характеру экспериментальной. Целью ставилось подробно разобраться в языке Си и в возможностях его реализации интерпретатором на Искра-226. Реализации интерпретатора языка Си способствовала интерпретирующая система Скоропись, которую характеризует лёгкое освоение, простота внесения изменений и расширений и возможность подключения новых систем. Вся система СиИ87-Искра программирована на языке Скоропись.

Реализация языка Си интерпретатором снижает быстродействие системы, но

1. значительно ускоряет разработку,
2. повышает гибкость и управляемость системы,
3. позволяет повысить эффективность системы в будущем.

Описанная реализация языка Си находится в стадии завершения отладки и опробования на программах. Данный интерпретатор имеет некоторые недостатки, касающиеся скорости интерпретации, но несмотря на это, система СиИ87-Искра является подходящим средством обучения языку Си.

В данной статье описывается состав системы СиИ87-Искра и особенности её реализации. Описываются группы кода интерпретатора и библиотека стандартных функций, а также приводятся ограничения языка СиИ87 по сравнению с полным языком Си, приведенным в [1].

1. Основные понятия и средства системы

Система СиИ87-Искра работает в среде системы Скоропись и использует её средства, в частности, управление загрузкой, редактора текстов, верхние уровни файловых систем, систему меню и др. Ниже описываются основные средства и понятия системы Скоропись, принимаемые при реализации системы СиИ87-Искра.

Системы Скоропись и СиИ87-Искра построены на основе таблиц. Таблица представляет собой поле динамической памяти со специальной организацией. Таблица состоит из элементов, имена которых уникальны. Каждый элемент таблицы может иметь несколько атрибутов и содержит текст (произвольной структуры). Структура таблицы позволяет организовать лексикографический просмотр содержащихся в ней элементов. Если элемент найден, то его атрибуты и текст доступны.

Над таблицами допускаются следующие операции:

1. Инициировать таблицы.

2. Вставить элемент с данным именем в таблицу.

3. Найти элемент с данным именем в таблице.

4. Удалить элемент с данным именем из таблицы.

Основные виды таблиц следующие:

1. Таблица имён при трансляции. Именем элемента таблицы является имя переменной, а атрибуты описывают тип, вид, адрес и др. характеристики переменной.

2. Системная таблица или система. Имя элемента таблицы совпадает с именем программы и текст элемента состоит из кодов, выполняемых интерпретатором.

3. Архив. Архив представляет форму хранения данных на внешних устройствах (НМД или НГМД). Именем элемента является имя программы, таблицы или подчинённого архива. Атрибуты элемента архива содержат информацию о дате последнего изменения, о локализации во внешней памяти и т.п., а текст элемента содержит любую специальную информацию.

4. Таблица макроопределений. Их используют в синтаксическом анализе исходного текста. Имя элемента таблицы совпадает с именем заменяемой лексемы и его текст содержит подстановки.

Таблица может быть объектом хранения и обмена, а также параметром, передаваемым программам.

Текст или программа является технологической единицей, т.е. единицей ввода, редактирования, хранения, трансляции, сборки и распечатки.

Общение с системой СиИ87-Искра происходит через систему меню, представляющую собой перечни предлагаемых режимов, причем каждому режиму соответствует отдельная строка. Для

выбора нужного режима следует ввести указатель меню (в виде инверсного свечения двух знакомест экрана) на нужную строку и нажать на клавишу run.

2. Основные ограничения языка СиИ87

Ввиду ограниченности объема данной статьи ниже рассматриваются только некоторые аспекты реализации языка СиИ87, отличающиеся от языка, заданного в описании [1]. Основные ограничения по сравнению с полным Си следующие:

1. В лексике отсутствуют прописные латинские и русские буквы, 8- и 16-константы, длинные и вещественные константы, из символьных констант отсутствуют \-константы и двоичное представление. Текстовая константа не содержит \-символа и не превышает длины одной строки исходного текста.

Имя (идентификатор) состоит из букв латинского алфавита и цифр, где первым символом имени является буква. Максимальная длина имени произвольная, но в данной реализации имена различаются по первым 8 символам.

В связи с отсутствием на клавиатуре Искра-226 символов "{" и "}", вместо них используются соответственно ключевые слова "begin" и "end".

Комментарии следует поместить на отдельные строки.

2. В языке СиИ87 допускаются два типа переменных: символьный (char) и целый (int). Из этих примитивных типов составляются следующие составные типы:

1) одномерный массив элементов примитивного типа;

2) указатель на объект примитивного типа;

3) функция, выдающая целое значение.

В языке СиИ87 отсутствуют следующие примитивные типы:

1) классы памяти static, internal, register и непосредственное определение классов памяти;

2) типы данных short, long, unsigned, float, double;

и следующие составные типы:

1) многомерные массивы и массивы указателей;

2) структуры (struct) и объединения (union);

3) типы указатель на массив и указатель на функцию.

3. Отсутствуют следующие выражения и операции:

1) явное преобразование типов (т.н. cast);

2) оператор sizeof;

3) условное выражение вида:

условие ? : выражение ; выражение ;

4) логические операции && и !! ;

5) выражения присваивания +=, -= и др.; существует только присваивание вида l-значение = выражение ;

6) операция "запятая".

4. Отсутствуют следующие определения объектов:

1) typedef;

2) инициализация;

3) возможность определить тип функции (все функции вырабатывают значение целого типа);

4) спецификация класса памяти extern.

5. Отсутствуют операторы do-while, switch-case, goto и метки операторов.

6. В препроцессе отсутствуют:

1) макроподстановка с параметрами;

2) включение текстового файла вида

   #include имя_файла

3) возможность т.н. условной трансляции - операторы #if, #ifdef, #else, #endif;

4) управление строками, т.е. #line.

3. Состав системы

Система СиИ87-Искра состоит из транслятора с языка СиИ87, названного smallc, и из интерпретатора внутреннего кода - cintpr. Все внутренние коды включаются в таблицу внутренних кодов, т.е. в систему пользователя или просто - в систему. Для выполнения программы на языке СиИ87 следует пройти следующие этапы (см. рис. 1):

1. Транслировать исходный текст во внутренний код и включить полученный внутренний код в систему.

2. Выполнить программу интерпретатором cintpr в системе. При интерпретации используется таблица стандартных функций fteek.

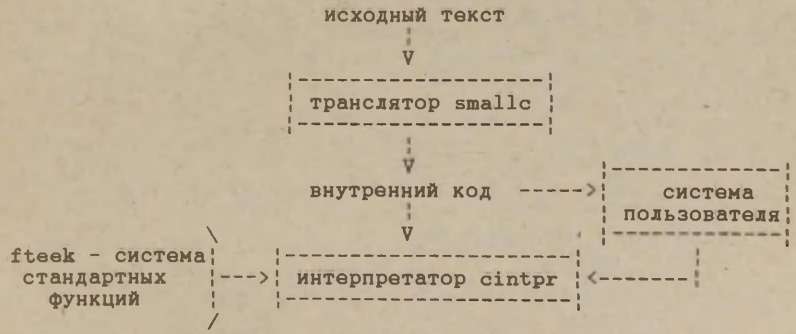


Рис. 1.

4. Работа с меню и с исходными текстами

Работа в разных режимах системы СиИ87-Искра происходит под управлением системы меню, которая содержит одно главное меню и несколько подменю. Главное меню даёт возможность работать в следующих режимах системы СиИ87-Искра:

1. Работа с исходными текстами.
2. Трансляция исходного текста.
3. Архивное обслуживание.
4. Вывод перечня глобальных переменных системы.
5. Выполнение программы в системе.

Каждому режиму главного меню соответствуют свои системные таблицы. Например, редактору текстов соответствует система scred, транслятору - системы transf и smallc, интерпретатору - cintpr и cintprt. В данной статье рассматриваются только режимы редактирования, трансляции и итерпретации. Режим "Архивное обслуживание" является подрежимом системы Скоропись и позволяет инициировать устройства и главный архив, создать подархивы и таблицы, копировать данные из одного архива в другой и т.п.

В режиме редактирования можно вводить текст программы на языке СиИ87 трёх видов:

1. Текст, содержащий только определения внешних (глобальных) переменных создаваемой системы. Например, пусть текст с именем global имеет следующий вид:

```
int input,array[10],*ptr;
char sym,*lptr;
```

2. Текст, содержащий только одно определение функции. Напри-

мер, пусть текст с именем func имеет следующий вид:

```
func(a)
    char a;
 begin if(a='x') return 1; else return 0; end
```

3. Текст, содержащий только макроопределения вида команды замены лексем:

```
#define имя подстановка
```

Например, следующий текст macro:

```
#define char 1
#define int  2
```

Такое распределение программ несколько ограничивает свободу программирования на языке Си, но способствует работе в интерактивном режиме.

5. Транслятор smallc

В режиме трансляции проводятся синтаксический и семантический анализы исходного текста и генерируется внутренний код. Для трансляции текста необходимо иметь на носителе систему пользователя для хранения внутренних кодов функций, таблицы внешних переменных системы и макроподстановок. Работать можно с созданной заранее системой, которую следует инициировать в режиме "Архивное обслуживание".

Транслятор имеет три режима: трансляция макроподстановок, трансляция внешних переменных системы и трансляция функций. Причиной трёхрежимной трансляции является интерпретатор, работающий с таблицами. Выбор нужного режима зависит от вида исходного текста. В ходе трансляции на экран и

на печать выдаётся протокол трансляции, состоящий из транслируемого текста и сообщений об обнаруженных ошибках. При удачном завершении трансляции можно записывать полученный внутренний код в систему пользователя. Режимы трансляции имеют разную схему и выдают разные по структуре внутренние коды.

При трансляции макрокоманд создаётся таблица макрокоманд, в которую включается каждая макрокоманда в виде отдельного элемента под именем макро и с текстом подстановки. При трансляции глобальных переменных все определения внешних объектов включаются под именами этих объектов в таблицу переменных. Каждый элемент данной таблицы имеет три атрибута, которые характеризуют тип (символьный или целый), вид (обычная переменная, массив или ссылка) и абсолютный адрес переменной. После трансляции исходного текста таблицы макроподстановок и глобальных переменных включаются в систему пользователя под именем mc и gl соответственно.

При трансляции функции предполагается, что исходный текст содержит только описание одной функции. После трансляции полученный внутренний код включается в систему пользователя под именем исходного текста.

При трансляции функций пользуются следующими таблицами:

1. Таблица макрокоманд для синтаксического анализа исходного текста (mc).
2. Таблица глобальных переменных (gl).
3. Таблица локальных переменных.

Транслятор функций работает методом рекурсивного спуска. При этом таблица глобальных переменных доступна на как-

дом этапе трансляции функции, но таблица локальных переменных - только в том блоке, где её определили. При входе в блок (т.е. при begin) создаётся новая таблица локальных переменных и копируются в неё все переменные из таблицы окружающего блока. При выходе из блока (т.е. при end) восстанавливается таблица окружающего блока. Все определения локальных переменных включаются во внутренний код, где кроме имени локальной переменной приводятся и его характеристики.

Структура внутреннего кода функции соответствует структуре функции, т.е. содержит блоки, описания локальных переменных, текстовые константы, операторы и выражения. Операторы во внутреннем коде представлены в польской записи. Трансляция выражения представляет собой рекурсивный процесс, где операции рассматриваются в убывающем порядке их приоритетов.

6. Интерпретатор cintpr

Для выполнения внутреннего кода некоторой функции необходимо иметь на носителе кроме системы интерпретатора систему стандартных функций fteek и систему пользователя, содержащую внутренние коды функций и таблицу внешних переменных системы.

Внутренний код интерпретируется слева направо. Схема интерпретации следующая:

1. Считается следующий байт во внутреннем коде.
2. По значению байта определяется выполняемая операция.
3. Пускается соответствующая программа интерпретатора или

выполняется некоторое другое действие.

Для интерпретации инициируется некоторая часть памяти. Оперативная память Си-машины состоит из трёх частей (см. рис. 2):

1. Память глобальных переменных системы. Объём памяти определяется таблицей глобальных переменных gl.

2. Область памяти, состоящий из рабочего стека и стека локальных переменных. Для этой области памяти выдаются 8200 байтов.

3. Стек текстовых констант объёмом 1000 байтов.

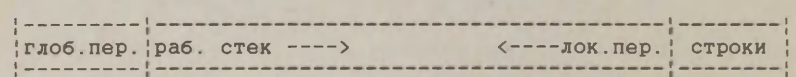


Рис. 2.

При интерпретации присутствуют таблица глобальных переменных и таблица локальных переменных. Интерпретатор реализован методом рекурсивного спуска: при следовании из блока в блок заменяются таблицы локальных переменных.

При коде вызова функции имя вызванной функции ищут в системе пользователя. Если он найден, то продолжается интерпретация внутреннего кода вызванной функции. В противном случае имя ищут в таблице стандартных функций fteek. Если функцию нашли, то она выполняется, в противном случае выдаётся сообщение об ошибке. При выходе из функции продолжается интерпретация кода, следующего за кодом вызова функции.

Выполнение операций реализуется соответственной программой кода. Коды Си-машины рассматриваются в следующем разделе.

Функцию можно выполнить и в режиме отладки (в системе

cintprt), где выполнение проводится точками останова и выводом изменений содержания памяти. Допускаются следующие команды отладки:

1. Определение или отмена точки останова в функции.
2. Отображение внутреннего кода функции с указанного адреса.
3. Вывод состава таблицы переменных.
4. Отображение содержания памяти Си-машины с указанного адреса.
5. Изменение содержания памяти по указанному адресу.

7. Коды Си-машины

Каждый код Си-машины имеет длину один байт. За кодом могут следовать несколько параметров. Параметрами могут быть имя переменной, некоторое 2-байтовое значение или текстовая константа. Коды Си-машины можно разделить в следующие группы:

1. Коды операторов (11 кодов).
2. Коды обмена информации (21 код).
3. Коды операций (20 кодов).

Коды операторов связаны с созданием таблиц локальных переменных (например, код начала блока, код вызова функции), восстановлением предыдущих таблиц локальных переменных (например, код прерывания цикла, код выхода из функции и др.) и сдвигом указателя внутреннего кода по метке.

Коды обмена информации осуществляют обмен информации между рабочим стеком и памятью. Длина операнда рабочего стека - два байта. Действия данных кодов можно распределить

следующим образом:

1. Читать информацию из памяти в рабочий стек.

2. Записать информацию из рабочего стека в память.

3. Читать абсолютный адрес переменной в рабочий стек.

4. Записать значение из рабочего стека в память по адресу, полученному из рабочего стека.

Коды операций предназначены для выполнения некоторых операций языка Си. Коды операций оперируют с операндами рабочего стека и результаты операций помещаются в рабочий стек.

8. Таблица стандартных функций

Таблица стандартных функций fteek содержит набор программ для ввода и вывода, для обработки символов, строк и полей памяти, для системного обслуживания. Все стандартные функций реализованы на языке Скоропись.

Для ввода с консоли и вывода на консоль никакие подготовительные действия не требуются. Пользователь может читать с консоли символ, целое значение или строку, вывести их на консоль или на печать. Для этого в библиотеке функций находятся такие типичные функции как getchar, getdec, gets, putchar, putdec, puts, printc, printw, prints.

В библиотеке стандартных функций предусмотрены функции для последовательной обработки файлов. Файл представляет собой поле динамической памяти, содержащее произвольный текст. Количество одновременно открытых файлов ограничено, т.е. одновременно может существовать один файл для чтения и

один файл для записи. Для открытия файла следует использовать функцию fopen, исходная информация которой задаёт название открытого файла и метод обработки, т.е. файл для чтения или записи. Если файл удалось открыть, то указателю файла присваивается начальный адрес этого файла. Длина файла для чтения определяется длиной исходного текста. Длина файла для записи не определена, т.е. в файл для записи можно писать информацию произвольной длины.

Литература

1. Kernighan, B.W., Ritchie, D.M., The C Programming Language. Prentice-Hall, 1978. Русский перевод: Б. Керниган, Д. Ритчи, А. Фьюер., Язык программирования Си. Задачник по Си. Финансы и статистика, 1985.
2. Hunter, B.C., Understanding C. Berkeley, 1984.

ФОРМАЛИЗАЦИЯ ПОНЯТИЯ КОММУНИКАТИВНОЙ СТРАТЕГИИ

М. Койт

Настоящая статья является продолжением работ, проводимых в рамках проекта ТАРЛУС (см., например, [4]), целью которых является моделирование человеческого общения.

Рассматриваем диалог между двумя коммуникантами - А и Б -, начинающийся с постановки одним из коммуникантов (А) цели $Ц^{А}$, и кончающийся достижением или снятием этой цели. Будем рассматривать такие диалоги, где цель А - заставить партнера Б делать некоторое действие Д. Цель считается достигнутой, когда Б выражает согласие делать Д.

Диалог представляет собой последовательность реплик коммуникантов А и Б:

$р_1^{А}$, $р_1^{Б}$, $р_2^{А}$, $р_2^{Б}$, ...

Каждая реплика состоит из одного или нескольких коммуникативных шагов (КШ). Примеры КШ: вопрос, ответ, команда, просьба (см. [3]). Формально КШ можно представлять в виде иерархических структур, т. н. фреймов. В репликах КШ выступают в языковой форме (в виде последовательностей слов).

В простейшем случае диалог состоит из двух реплик: А сообщает о своей цели (приказывает или просит Б делать Д), и Б согласится делать Д.

В более сложных случаях на пути к реализации цели $Ц^{А}$ перед А возникают препятствия: в ходе диалога оказывается,

например, что Б не хочет делать Д, или Б считает, что он не умеет делать Д, или Б боится последствий Д. Поэтому А должен применить определенный алгоритм воздействия на Б, чтобы преодолеть эти препятствия и добиться решения Б делать Д. Такой алгоритм достижения цели $Ц^{A}$ мы будем называть коммуникативной стратегией А.

Зафиксировав цель $Ц^{A}$, коммуникант А в ходе общения пытается направлять интеллектуальные и эмоциональные процессы Б (его желания, оценки, рассуждения) таким образом, чтобы заставить Б принять решение делать Д. При этом воздействие на партнера конкретно направлено на определенные аспекты его психики (желания, оценки, мнения, знания), от которых зависит принимаемое им решение. Такие аспекты психики коммуниканта называем его психологическими параметрами (коротко, П-параметрами; см. [3]). Решение делать Д зависит от конкретных значений П-параметров (от уровня желания, заинтересованности, страха и т.п.). Коммуникант А должен так построить свои реплики, чтобы их результатом было изменение значений П-параметров Б в нужном направлении: увеличение желания Б делать Д, уменьшение страха Б перед последствиями Д, дополнение его знаний и т. д.

В рамках применяемой коммуникативной стратегии А выбирает П-параметры Б, на которые оказывать влияние, и определяет в их значениях нужные изменения.

Для того, чтобы вести диалог с Б, коммуникант А должен иметь определенное представление о Б - модель партнера, включающую значения П-параметров партнера. В ходе диалога модель партнера может меняться: из ответных реплик партнера

А узнает, какое влияние оказывают его реплики на значения П-параметров Б и какие изменения нужно ввести в модель Б.

Кроме модели партнера коммуникант должен иметь знания об общих закономерностях общения, в том числе, знать, какие КШ можно применять для воздействия на определенные П-параметры партнера; какие значения П-параметров требуются для достижения $Ц^{A}$.

Имеется два вида П-параметров. На П-параметры первого вида можно воздействовать непосредственно, применяя определенные КШ. Напрмер, страх Б перед последствиями Д можно уменьшить, представляя информацию о том, что эти последствия не так вредны, как считает Б.

На П-параметры другого вида можно воздействовать только косвенно, через другие П-параметры. Например, желание Б делать Д можно увеличить, увеличив его страх перед последствиями не-Д или увеличив его оценку последствиям Д.

П-параметры формально можно представить в виде предикатов. В модель партнера войдут значения соответствующих предложений исчисления предикатов. Пусть значение предложения равно 0, +1 или -1, в зависимости от того, совпадает ли значение соответствующего П-параметра со значением, нужным для достижения цели $Ц^{A}$, либо оно больше или меньше нужного. Тем самым цель $Ц^{A}$ определяет разбиение рассматриваемого множества предложений трехзначного исчисления предикатов.

Диалог между А и Б происходит следующим образом.

Зафиксировав цель $Ц^{A}$, коммуникант А определяет П-параметры партнера, значения которых нужно проверить. Если все значения в модели партнера нули, то это означает, что все

П-параметры B уже имеют значения, требуемые для достижения $Ц^A$. Тем самым цель $Ц^A$ достигнута, и диалог не продолжается.

В противном случае A (случайно) выбирает П-параметр P или несколько П-параметров, значение которого отличается от нужного, и своей следующей репликой будет оказывать на него воздействие. Тем самым A ставит перед собой (под)цель превратить значение P в требуемое для достижения $Ц^A$. Если P принадлежит к таким П-параметрам, на которые непосредственно, с помощью КШ, можно оказывать влияние, то A выбирает один или несколько КШ и строит из них свою очередную реплику. Если на P непосредственно нельзя оказывать влияние, то A выбирает новый П-параметр среди тех, которые предназначены для косвенного воздействия на P, и дальше действует так же, как он действовал, выбрав P.

Интерпретируя ответную реплику партнера B, A вводит необходимые изменения в модель партнера и проверяет, достигнута ли цель изменения значения P. В утвердительном случае процесс повторяется с проверки модели партнера. В противном случае продолжается оказывание воздействия на P.

Рассмотрим формальную модель коммуникативной стратегии. Дополнительно к уже принятым обозначениям введем следующие.

Множество всевозможных П-параметров коммуниканта:

$$\bar{П} = \{П_1, \ldots, П_n\} = \bar{П}_1 \cup \bar{П}_2,$$

где $\bar{П}_1$ и $\bar{П}_2$ - множества П-параметров, на которые можно воздействовать, соответственно, непосредственно или косвенно.

Множество всевозможных коммуникативных шагов:

$$\bar{К} = \{К_1, \ldots, К_\ell\}.$$

Каждый коммуникант может применять КШ из множества $\bar{К}$.

Даны отображения:

$$\Phi_{11}: \bar{\Pi}_1 \to 2^{\bar{R}},$$

$$\Phi_{12}: \bar{\Pi}_1 \to 2^{\bar{R}},$$

$$\Phi_2 : \bar{\Pi}_2 \to 2^{\bar{\Pi}},$$

где $\Phi_{11}(\Pi_i)$ - множество КШ, применимых для увеличения значения П-параметра $\Pi_i \in \bar{\Pi}_1$;

$\Phi_{12}(\Pi_i)$ - множество КШ, применимых для уменьшения значения $\Pi_i \in \bar{\Pi}_1$;

$\Phi_2(\Pi_i)$ - множество других П-параметров, через изменение значений которых можно изменять значение $\Pi_i \in \bar{\Pi}_2$.

Модель партнера Б, имеющаяся у А (множество предложений трехзначного исчисления предикатов):

$$\bar{\Pi}^{Б}(Ц^{A}) = \{\Pi_1^{Б}(Ц^{A}), \ldots, \Pi_m^{Б}(Ц^{A})\},$$

где $\Pi_1^{Б}, \ldots, \Pi_m^{Б}$ - П-параметры.

Разбиение множества $\bar{\Pi}^{Б}(Ц^{A})$:

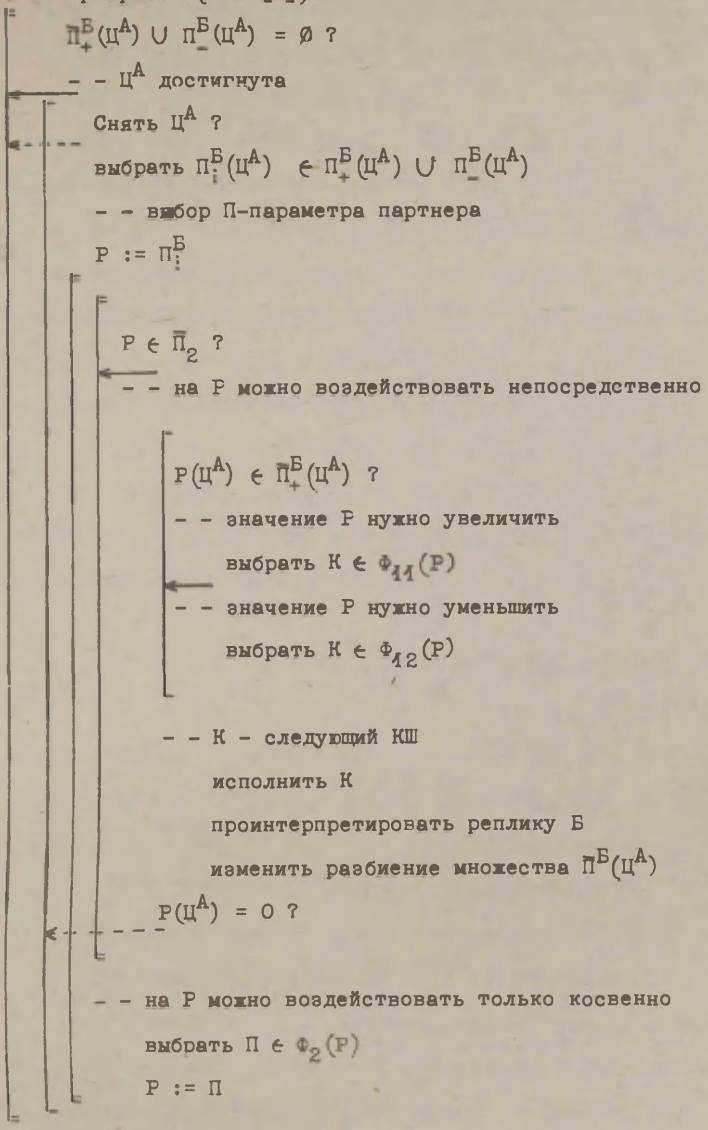
$$\bar{\Pi}^{Б}(Ц^{A}) = \bar{\Pi}_0^{Б}(Ц^{A}) \cup \Pi_+^{Б}(Ц^{A}) \cup \Pi_-^{Б}(Ц^{A}),$$

где $\bar{\Pi}_0^{Б}(Ц^{A})$ - множество предложений, значение которых 0 (т. е. значения соответствующих П-параметров Б совпадают с значениями, нужными для достижения $Ц^{A}$);

$\Pi_+^{Б}(Ц^{A})$ - множество предложений, значение которых -1 (значения соответствующих П-параметров Б нужно увеличить для достижения $Ц^{A}$);

$\Pi_-^{Б}(Ц^{A})$ - множество предложений, значение которых +1 (значения соответствующих П-параметров Б нужно уменьшить для достижения $Ц^{A}$).

Коммуникативную стратегию А представим в виде схематической программы (см. [2]):



Применение стратегии коммуникантом А можно представить как выбор и прохождение некоторого поддерева в дереве всевозможных подцелей цели $Ц^{A}$. Корень дерева соответствует исходной цели $Ц^{A}$. Каждая дуга соединяет вершины, соответствующие цели и ее подцели. Каждый выбор коммуникантом А П-параметра, на который он намерен воздействовать, означает постановку им очередной подцели (выбор следующей вершины дерева). Вершина считается пройденной, когда достигнута соответствующая цель. Прохождение поддерева начинается в некоторой висячей вершине и кончается в корне дерева.

Рассмотрим применение коммуникативной стратегии на примере следующего диалога (см [1]).

1^{o} А: Пойдем на лекцию о творчестве Сидорова.

2^{o} Б: Нет, вряд ли она будет интересной.

3^{o} А: Ну что ты, читает сам Иванов, будут слайды и магнитофонные записи.

4^{o} Б: А когда начало?

5^{o} А: Через час.

6^{o} Б: Хорошо, идем.

7^{o} А: (удовлетворен)

Проанализируем диалог по репликам.

1^{o}. Коммуникативной целью А является $Ц^{A}$ = "Б делает Д", где Д - "идти вместе с А на лекцию о творчестве Сидорова". А предполагает, что единственный П-параметр у Б, значение которого меньше требуемого - "знание цели $Ц^{A}$". Поэтому первая реплика А состоит из КШ "предложение".

2^{o}. Реплика Б содержит отказ и обоснование отказа. Интерпретируя ее, А узнает, что значения П-параметров Б "жела-

ние делать Д" и "оценка к последствиям Д" меньше нужного.

3°. А выбирает второй из этих П-параметров (ставит подцель "увеличить оценку Б к последствиям Д") и чтобы воздействовать на него, исполнит КШ "представление информации".

4°. Б не возражает против представленной информации. Из этого вытекает, что подцель А достигнута. Но выявляется еще один П-параметр Б, значение которого меньше требуемого - "знание о времени Д".

5°. Следующая подцель А - "увеличить знание Б". Для ее достижения А представляет информацию.

6°. Б не возражает против представленной информации и согласится делать Д - подцель, а также исходная цель А, достигнуты.

Дерево целей имеет корень и две висячих вершины, соответствующие исходной цели и двум ее подцелям, и две дуги.

Литература

1. Диненберг Ф.Г. Коммуникативная триада как базовая составляющая структуры диалога // Диалоговое взаимодействие и представление знаний. Новосибирск, 1985. С. 101 - 109.
2. Кихо Ю.К. Схематическое программирование // Тр. Вычисл. центра Тарт. ун-та. 1983. Вып. 50. С. 52 - 68.
3. Койт М.Э. Разработка формальной модели диалога // Учен. зап. Тарт. ун-та. 1987. Вып. 751. С. 60 - 70.
4. Койт М.Э., Ыйм Х.Я. Понятие коммуникативной стратегии в модели общения // Учен. зап. Тарт. ун-та. 1988. Вып. 793. С. 97 - 111.

ДИАЛОГОВЫЕ СРЕДСТВА В СУБД РАМА

А. Изотамм

В настоящей статье дается обзор подсистемы СУБД РАМА, представляющей возможность интерактивной работы с базой данных (фондом). В режиме диалога можно полностью провести все такие операции над записью, которые в пакетном режиме проводятся средствами языка DIL (см. [4], [5] и [6]). Кроме этого имеется также возможность выполнить некоторые функции языков манипулирования данными (см. [9], [10] и [11]) и вывода данных (см. [12] и [13])[1].

Диалог осуществляется средствами встроенного интерпретатора описаний файлов (см. [14]) и записей (см. [2]). У пользователя нет необходимости (а также и возможности) средствами какого-либо языка программирования управлять формой диалога или же формой представления информации на экране. Диалоговая подсистема реализована для ЭВМ типа ЕС-1060 на основе программных средств, разработанных сотрудниками ВЦ ТГУ Т. Кельдером и Т. Ээнма (см. [18]) для работы с локальными дисплеями типа ЕС-7920.

---

[1] Возможности печати таблиц рассматриваются в статье [15] настоящего сборника.

1. Работа на уровне фонда

Для инициализации диалога следует в заказ, написанный на языке TCL (см. [8]), включить предложение

```
//TERM DD DUMMY
```

а в последующем заказе пользователя должен быть EXEC-предложение фиктивной DIL-программы с непустым списком параметров. С точки зрения пользователя представляют интерес параметры LOGI, SPONT и DLG. Значение последнего задает логический адрес дисплея пользователя. Параметр LOGI дает возможность скопировать экран на АЦПУ в моменты прерываний, а при помощи параметра SPONT можно управлять контрольной печатью, которая сопровождает выполнение DIL-операторов. В дальнейшем будем опираться на следующий пример заказа:

```
// EXEC W5BAM,ID=W5,REGION=512K,DPRTY=10
//PUBLIC00 DD SYSOUT=X
//PUBLIC01 DD SYSOUT=X
//TERM DD DUMMY
//FILEDDNA DD DSN=PAM@@@@@.TEGUS,DISP=SHR
//FILEDDNB DD DSN=PAM@@@@@.TEGU,DISP=SHR
//GO.SYSIN DD DATA
*USER PUBLIC LANG=RUS
*FOND PAM
*STEP A
*EXEC DIL=SYSIN(CONTR) 'LOGI=N,SPONT=N,DLG=3C2'
*FILE TEGUS,REG=W
*FILE TEGU,REG=R
*SYSIN CONTR
```

По сравнению со статьей [8] имя системной процедуры теперь W5BAM, а в USER-предложении можно задавать язык диалога. Последнее означает, что все меню диалога, служебная и вспомогательная информация и сообщения выдаются либо на русском (RUS), английском (ENG) или эстонском (EST) языке. Фиктивной DIL-программой в приведенном примере служит пустая программа; параметры LOGI и SPONT выключены (для включения следовало бы в качестве значения задавать букву "Y").

Как известно, любая DIL-программа может в пакетном режиме работать только с записями, составленными по одному описанию (легенде) и принадлежащимися одному файлу. В режиме диалога таких ограничений нет, вернее - заказ на выполнение DIL-программы открывает все возможности интерактивной работы с фондом. Этим объясняется возможность задания двух файлов (TEGUS и TEGU) в приведенном примере.

Диалог начинается с показа (по USER-предложению) на экране имен пользователя фонда, списка DIL-параметров и информации о заказанных файлах. Для каждого файла выводится ведущая строка и по одной строке для каждой легенды записей данного файла. Следуя нашему примеру, где каждый файл состоит только из однотипных записей, эти строки выглядят так:

```
1. ЛЕГ=WPKS  P=W  ИМЯ=TEGUS  ДИСК=SYSTSO
   Р-ЛЕГ: WPKK    КОМПЛ=
2. ЛЕГ=PLK  P=R   ИМЯ=TEGU   ДИСК=SYSTSO
   Р-ЛЕГ: WPK     КОМПЛ=
```

Здесь именем описания первого файла TEGUS является WPKS, а физический файл находится на магнитном диске, носящем имя

SYSTSO. Структуру всех записей этого файла определяет легенда с именем WPKK. Пользователь может выбирать нужные легенды, записывая на соответствующее поле за словом КОМПЛ либо ничего, либо звездочку, либо одну или несколько имен, разделенных запятыми. Пустое значение означает, что данная легенда не используется, звездочка - что следует создавать комплект с именем легенды, а перечень имен - что данной легендой связываются комплекты с указанными именами (см. [9]).

После ввода заказа на комплекты система показывает на экране имена выбранных файлов, легенд и комплектов, и предлагает выбирать один из четырех вариантов работы:

а) работа с единственным комплектом;

б) перенос данных из одного комплекта в другой;

в) создание псевдомассива;

г) конец сеанса.

Эти возможности предлагаются снова после завершения любого из трех первых вариантов.

2. Работа с единственным комплектом

2.1. Макеты. Этот вариант дает возможность для работы с отдельными записями. На основе оттранслированной легенды (см. [3]) генерируются макеты экранов. Последние разделяются на два типа: макеты групп и макеты замков. Независимо от типа на каждом экране выделяются четыре поля - заголовок, тело, поле обмена служебной информацией и поле для внесистемных сообщений. В заголовке отражается текущее состояние комплекта: вариант работы, имя легенды, имя комплекта, ключи обрабатываемой записи, место и путь (см. [4], [9]). Служеб-

ная информация состоит из вопросов системы и ответов пользователя, а также из сообщений системы РАМА. Внесистемную информацию составляют сообщения оператора ЭВМ и других пользователей за другими дисплеями, не относящиеся к сеансу СУБД.

Как известно из [2], описание записи (легенда) имеет структуру корневого дерева, где непосредственные подчиненные вершины любой невисящей вершины составляют группы.. Для каждой группы легенды строится макет, состоящий из имени корня группы и строк для входящих в группу атомов. Для каждого атома отводится некоторое число строк на экране; одна группа может представляться на одном или нескольких экранах.

Рассмотрим, например, следующую легенду:

```
LEG WPK KEY=LAB TEXT
*1 LAB PICT=32   'ИМЯ ПОДРАЗДЕЛЕНИЯ'
*1 YHIK REP=1024 'ЕДИНИЦЫ ПУБЛИКАЦИЙ'
X( A SORT KEY=AD
X AA KEY=AUTOR,AD SORT
  *2 TIITEL REP=4 PICT=60  'ЗАГЛАВИЕ ЕДИНИЦЫ'
  *2 KAT SCOPE=[VM,YL,E,TRY,KK] 'КАТЕГОРИЯ: ЗАГРАНИЦА,
     СССР, ЭСТОНИЯ, УНИВЕРСИТЕТ, РУКОПИСЬ'
  *2 MARK PICT=16  'КЛЮЧЕВОЕ СЛОВО'
  *2 LIIK SCOPE=[MON,TART,TEES,POP,OP,MET] 'ТИП: МОНО-
     ГРАФИЯ, НАУЧН. СТАТЬЯ, ТЕЗИСЫ, ПОПУЛЯРНЫЙ, УЧЕБ-
     НОЕ ПОСОБИЕ, МЕТ-ПОСОБИЕ'
  *2 LKARV NAT PICT=3  'К-ВО СТРАНИЦ'
  *2 AD PICT=4 DEC  'ГОД ВЫПУСКА'
  *2 AUTOR REP=6 PICT=8  'АВТОР[Ы]'
END
```

На основе этой легенды строятся четыре макета групп. Первый макет для группы корня записи выглядит так:

```
                    WPK
LAB=
```

Второй макет строится для экземпляра повторяющейся группы YHIK, т.е. для одной единицы публикаций:

```
                  YHIK=1024,
KAT=
MARK=
LIIK=
LKARV=
AD=
```

Третий макет строится для множественного атома, т.е. для всех экземпляров повторяющейся группы TIITEL:

```
                  TIITEL=4,
1. TIITEL=
2. TIITEL=
3. TIITEL=
4. TIITEL=
```

Четвертый макет предназначен также для множественного атома, т.е. для всех авторов данной единицы:

```
                    AUTOR=6,
1. AUTOR=
2. AUTOR=
3. AUTOR=
4. AUTOR=
5. AUTOR=
6. AUTOR=
```

Три последние макета соответствуют повторяющимся группам; аттрибуты имен групп показывают максимально возможное количество экземпляров и их действительное количество (последнее в макете пусто и заполняется во время работы). Для каждого атома отводится поле его значения, снабженное символом-аттрибутом (см. [16], стр. 35), содержание которого зависит от описания атома и режима работы с записью. Например, если запись заказана только для чтения, то все поля значений атомов являются защищенными от изменений; в модифицируемой записи защищенными являются непустые поля констант, ключей основного доступа и псевдоатомов. Если атом имеет нетекстовый тип, то движение курсора на его поле вызывает автоматический перевод на верхний регистр клавиатуры; имена и значения атомов, являющихся ключами основного доступа, отображаются с повышенной яркостью.

В диалоговом режиме реализована и задача защиты данных, которая в ранних публикациях о СУБД РАМА осталась нерешен-

ной. Для этого была изменена семантика ключевого слова PACK (см. [2], стр. 30), которое связывается с корнем любой группы или с отдельными атомами и означает теперь защиту данных. Если хоть одна вершина легенды имеет свойство PACK, то все записи следует снабжать паролью. Если пользователь пароли не знает, то он не может модифицировать записи, а атомы со свойством PACK являются неотображаемыми на экране.

Выше было сказано, что макеты второго типа связываются с замками - они нужны для задания путей до необходимых мест. Ниже приведены все такие макеты для легенды нашего примера. Первый из них предназначен для ключа записи:

LAB=
УСЛОВИЕ=
ПРОШУ КЛЮЧИ

Второй макет предназначен для определения экземпляров повторяющейся группы YHIK:

INDEX=
УСЛОВИЕ=
ПРОШУ КЛЮЧИ

Так как YHIK является повторяющейся группой без явных ключей доступа, то в их роли выступает порядковый номер (индекс) экземпляра. Таким же образом выглядят макеты для ключей групп TIITEL и AUTOR - они также без явных ключей. Слово INDEX на первой строке соответствует неявному ключу группы YHIK, а на второй строке - такому же ключу собственной группы. Оба эти макеты одинаковы:

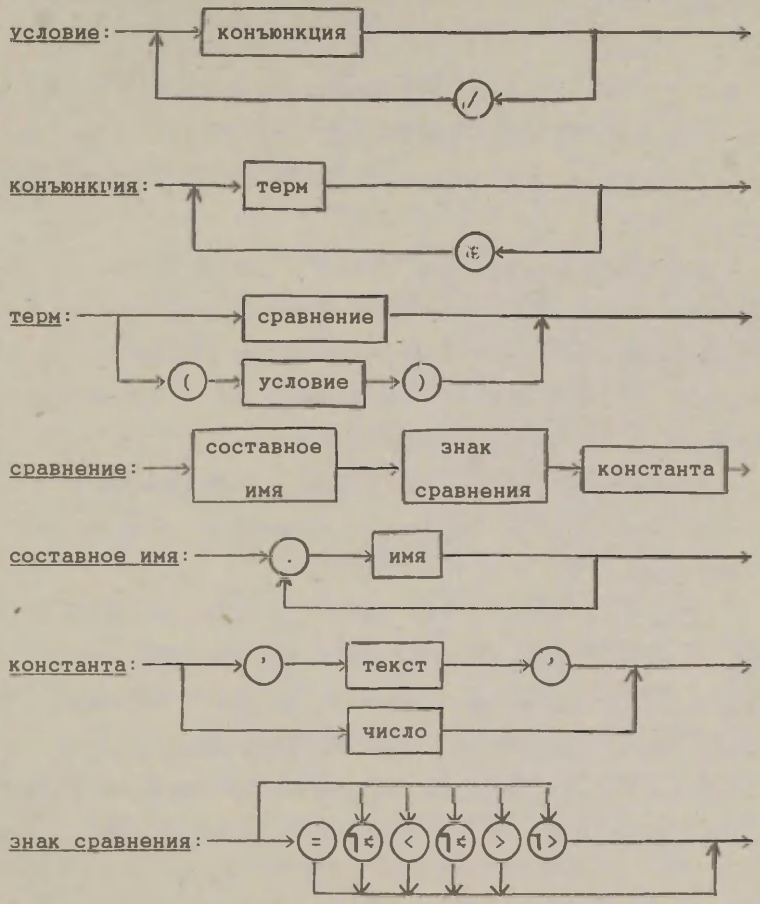
INDEX= INDEX=
УСЛОВИЕ=
ПРОШУ КЛЮЧИ

Последние два макета создаются для замков дополнительного доступа. Приведем из них полностью первый макет:

AD=
УСЛОВИЕ=
ПРОШУ КЛЮЧИ

Второй макет отличается от этого только перечнем спрашиваемых ключей - ими являются AUTOR и AD.

В качестве ключа следует задавать либо его конкретное значение, либо пустое значение или символ "*", означающий заказ на цикл. В последнем случае можно прибавить фильтр за словом УСЛОВИЕ: тогда система показывает (или печатает) только те группы (экземпляры повторяющихся групп), которые удовлетворяют заданному логическому условию. Условие следует записать, соблюдая следующие правила:



Например, если местом служит АА, и пользователя интересуют только монографии автора Тамм, то следует задавать

AUTOR=* AD=*
УСЛОВИЕ=.AUTOR='TAMM'&.LIIK='MON'
ПРОШУ КЛЮЧИ

Отметим, что вариант "AUTOR=TAMM,AD=*" является недопустимым, так как оба ключа принадлежат к одному и тому же замку, а частичный цикл по некоторым ключам не допускается (см. также [17]).

2.2. Служебные функции. Диалог в режиме работы с единственным комплектом начинается выводом на экран перечня функциональных клавишей и соответствующих им символов. Пользователю предусматривается возможность их переопределения. Предлагаемый набор является следующим:

1) ввести или показать следующий ("ввод" или буквы В/С);

2) игнорировать или окончить работу на данном уровне ("ПД1" или "К");

3) чистить экран ("ПФ10" или "Ч"); семантика этой функции такова: при вводе экземпляров повторяющейся группы на экране сохраняется изображение предыдущего экземпляра, и пользователю дается возможность заменить или исправить значения тех атомов, которые различаются по сравнению с предыдущим экземпляром; чистка экрана стирает все значения атомов с экрана;

4) печатать экран ("ПФ5" или "П") - на АЦПУ печатается копия экрана; этой функцией можно пользоваться тогда, когда в качестве значения параметра LOGI задан "N";

5) переход на DIL-режим ("ПФ12" или "#"); в этом режиме можно ввести любые DIL-операторы или порции (см. [4], [5] и [7]), объектом которых является обрабатываемая запись; DIL-команды следует задавать в одиночном порядке; признаком конца режима служит символ "конец пакета" - "###";

6) показать легенду ("ПФ1" или "Л");

7) показать возможности работы ("ПФ2" или "Т"); на экран выводится следующая таблица:

L [List]	- вывести на экран;
D [Delete]	- удалить из базы данных;
S [Stora]	- если есть, то заменить, иначе ввести;
R [Replace]	- если есть, то заменить, иначе ошибка;
A [Add]	- если нет, то ввести, иначе ошибка;
P [Print]	- контрольная печать на АЦПУ;
M [Modify]	- "завершение записи";
F [Form]	- печатать таблицу на АЦПУ.
B	- печатать список на АЦПУ

Вариант L дает возможность смотреть либо экземпляр группы (если заказан только один атом, то показывается вся группа), либо поддерево - он выводится на экран в прямом порядке прохождения (preorder-walk). При удалении (D) показывают пользователю удаляемое поддерево и предоставляется возможность отказаться от удаления. Вариант P печатает поддерево в скобочном представлении, F формирует таблицу (см. [15]), а B печатает простой список. Вариант M предназначен для создания

дополнительных доступов и вычисления значений псевдоатомов, не дожидаясь закрытия записи;

8) показать функции ("ПФ2" или "Н"); пользователю показывают рассматриваемый набор функций без возможности их переопределения (это можно сделать при помощи 11-той функции);

9) спросить режим работы ("ПД2" или "Р"); пользователю показывают таблицу возможностей работы (см. 7-ую функцию) с просьбой выбрать и ввести из них нужный вариант;

10) спросить язык сообщения ("ПФ4" или "Я") - эта функция дает возможность менять рабочий язык в ходе сеанса;

11) определить функции ("ПФ3" или "Ф");

12) начало дублирования печати ("ПФ8" или "+");

13) конец дублирования печати ("ПФ9" или "-"). В режиме дублирования печати на экран выводятся все печатаемые данные в тот момент, когда они наполняют буфер страницы (20 строк длиной по 128 символов). Так как длиной строки на экране является 80 символов, то пользователь может переключаться на просмотр левой или правой части страницы.

2.3. Работа с записью. В начале работы с записью система спрашивает у пользователя ключи записи. В ответ можно дать либо значение ключей, либо звездочку - в последнем случае организуется цикл по всем записям файла, описанным данной легендой. Пользователь может этот цикл фильтровать, задавая некоторое условие. Затем пользователю предлагают определить режим работы с записью при помощи экрана, приведенного на следующей странице. Конечно, система не допускает противоречия с заказанным режимом файла. Если в легенде предусмотрена защита данных паролью, то система предлагает ее ввести.

В таком случае, когда пароль не вводится или же она недействительна, возможно открытие записи только для чтения.

Режимы записи	
R [Read]	Только для чтения
D [Delete]	Удалить из базы данных
M [Modify]	Для чтения и записи
N [New]	Создать новую запись

Вариант=

В случае новой записи система, двигаясь в прямом порядке по дереву легенды, показывает макеты групп и по мере их заполнения создается физическая запись. Просмотр или модифицирование имеющейся записи происходит по умолчанию в режиме LIST. Отметим, что в режиме модифицирования записи пользователь в ходе просмотра может исправить значения незащищенных атомов (системой защищаются ключи основного доступа, а также константы и псевдоатомы). Для изменения режима следует пользоваться функцией "спросить режим работы".

Обработка записи происходит по следующей схеме:

1) у пользователя спрашивается место; в ответ следует ввести (составное) имя нужной вершины легенды, если запрос связан с дополнительным доступом, то следует задавать имя дополнительного доступа;

2) если на пути с корня легенды до места находятся замки, или местом служит корень повторяющейся группы или дополнительный доступ, то система показывает пользователю соответствующий макет замков; введенная пользователем информация

управляет дальнейшим поведением системы. Если запись модифицировалась во время сеанса, то в конце ее обработки пользователь должен решить, следует ли запись ввести в базу данных, или игнорировать сделанные корректуры. Об этом спрашивает система, задавая возможные ответы.

3. Перенос данных из одного комплекта в другой

Этот вариант дает возможность перекачивать данные из одной записи в другую. Записи не должны иметь обязательно совпадающиеся легенды, пользователь должен лишь заботиться об однородности структур данных. Например, атому нельзя присвоить корень повторяющейся группы, корню двухмерного массива - корень трехмерного массива и т.д. Корень повторяющейся группы с LIST-организацией может быть перенесен в качестве корня REP-группы, но обратное неосуществимо (нет информации о субординации экземпляров). Группы не должны быть обязательно одинаковой длины: если переносимая группа короче, то в составляемой группе "лишние" атомы останутся без значений, а в обратном случае "лишние" атомы не переносятся. Для переноса поддерево переводится в образ дерева текста (см. [3], [5]), а на новом месте оно переводится уже по новой легенде в физическое дерево. Таким образом, типы атомов могут не всегда совпадать; например, новый тип может в любом случае быть "текстом" необходимой длины.

Диалог о переносе начинается с вопроса, какой комплект выступает в роли "передатчика". Запись и место в записи определяются таким же образом, как при варианте с единственным комплектом. После создания требуемого "дерева текста" систе-

ма спрашивает имя комплекта "приемника", и после определения записи и места осуществляется перенос.

4. Создание псевдомассива

Понятие "псевдомассив" до сих пор не встречается в публикациях о СУБД РАМА. Тем не менее оно является естественным расширением возможностей системы: подразумевается такой массив, который (аналогично с псевдоатомом) заполняется значениями определенной функции. Если такой массив вычисляется по данным той же записи, где он сам находится, то его создание происходит в обычном порядке (см. [7]). Но в режиме диалога предоставляется новая возможность: псевдомассив сам находится в одной, а его аргументы - в другой записи. Приведем такой пример. Пусть создана следующая легенда:

```
LEG WPKK KEY=NIMI TEXT
*1 NIMI PICT=8  'ИМЯ ТАБЛИЦЫ'
*1 KAT SCOPE=[VM,YL,E,TRY,KK]
*1 LIIK SCOPE=[MON,TART,TEES,POP,OP,MET]
*1 TAB ARRAY[(KAT),(LIIK)] NAT
END
```

и пусть в конец приведенной выше легенды (WPK) добавлена еще вершина T:

```
    *2 AUTOR...
*1 T ARRAY[(KAT),(LIIK)] NAT PSEUDO=W1SAGED(YHIK,KAT)
END
```

Теперь можно в одной записи, созданной по легенде WPKK построить частотную таблицу по данным другой записи, создан-

ной по легенде WPK. Таблица имеет следующую форму (столбец и ряд SUM предназначены для сумм; см. также [15]):

		LIIK					
KAT	MON	TART	TEES	POP	OP	MET	SUM
VM							
YL							
E							
TRY							
KK							
SUM							

Система предлагает открыть "запись-приемник", где пользователь заказывает вершину TAB, а затем следует открыть "запись-передатчик", где (в режиме LIST) указывают на вершину T: притом можно задавать фильтр для выбора нужных экземпляров повторяющейся группы YHIK. Например, если необходимо получить частотную таблицу публикаций сотрудников Колло и Ээремаа, то экран замков должен выглядеть таким образом:

```
T01=*
T02=*
УСЛОВИЕ=.AUTOR=KOLLO/.AUTOR=AAREMAA
ПРОШУ КЛЮЧИ
```

Отметим, что имена ключей T01 и T02 являются системными: они сгенерированы, исходя из имени массива T и соответствуют первому и второму индексам без явных имен. Если же массив T описать в виде

```
...T ARRAY[KAT=(KAT),LIIK=(LIIK)] ...,
```

то именами ключей в макете будут KAT и LIIK.

Литература

1. Каазик Ю., Ээремаа К., Обзор системы РАМА для управления базой данных. Труды ВЦ ТГУ, 1986, 54, 3 - 15.
2. Изотамм А., Каазик Ю., Томбак М., Язык определения записи. Труды ВЦ ТГУ, 1978, 41, 7 - 64.
3. Изотамм А., Дерево описания записи в системе РАМА. Труды ВЦ ТГУ, 1980, 43, 3 - 35.
4. Каазик Ю., Ээльма П., Ввод и корректировка данных. Труды ВЦ ТГУ, 1978, 41, 75 - 96.
5. Изотамм А., Ввод данных в системе РАМА. Труды ВЦ ТГУ, 1986, 54, 16 - 32.
6. Каазик Ю., Толпин А., Реализация языка ввода данных. Труды ВЦ ТГУ, 1980, 45, 21 - 35.
7. Изотамм А., Завершение записи. Труды ВЦ ТГУ, 1988, 55, 19 - 31.
8. Ээремаа К., Язык управления заданиями системы РАМА. Труды ВЦ ТГУ, 1983, 50, 3 - 22.
9. Каазик Ю., Рауп А., Язык манипулирования данными. Труды ВЦ ТГУ, 1978, 41, 97 - 140.
10. Рауп А., Реализация языка манипулирования данными. Труды ВЦ ТГУ, 1980, 45, 36 - 65.
11. Тамме Т., О реализации языка манипулирования данными. Труды ВЦ ТГУ, 1985, 52, 15 - 26.
12. Пейал Я., Соо В., Томбак М., Язык вывода данных. Труды ВЦ ТГУ, 1982, 49, 42 - 61.
13. Пейал Я., Соо В., Реализация языка вывода данных. Труды ВЦ ТГУ, 1985, 52, 27 - 38.

14. Каазик Ю., Образование файлов. Труды ВЦ ТГУ, 1978, 41, 65 - 74

15. Изотамм А., Генератор таблиц. Наст. сборник.

16, Единая система электронных вычислительных машин, Операционная система, Общий телекоммуникационный метод доступа, Комплексы ЕС-7920, Руководство программиста, Е10.180 .003 Д1, 1980.

17. Изотамм А., Доступ к данным в системе РАМА. Труды ВЦ ТГУ, 1983, 50, 23 - 40.

18. Eenma T., DControl/DStarter-süsteem. 1987, рукопись.

ГЕНЕРАТОР ТАБЛИЦ

А. Изотамм

В статьях М. Томбака, Я. Пейала и В. Соо (см. [1] и [2]) приведено описание проблемно-ориентированного языка вывода данных системы РАМА (языка DOL), который дает возможность генерировать отчеты любой сложности. Ниже дается обзор т.н. генератора таблиц, реализующего некоторое подмножество языка DOL средствами встроенного интерпретатора.

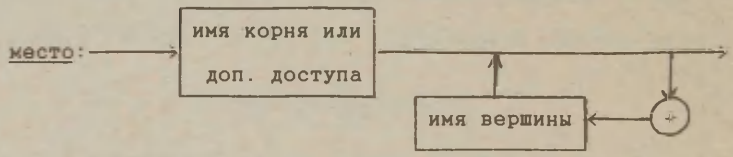
Специального языка для оформления и составления таблиц не понадобится - эти работы совершаются путем интерпретации легенды записи и физической записи в диалоговом режиме. Обзор диалоговых средств системы РАМА дается в статье [3] настоящего сборника. Предполагая, что читатель уже знакомился с этой статьей, в дальнейшем опускаются все ссылки на нее.

1. Постановка задачи

Объектом генератора таблиц (в дальнейшем ГТ) всегда является только одна физическая запись. В общем случае можно оформлять в виде отдельной таблицы любое поддерево (см. [5]) лишь с одним исключением: в этом поддереве пропускаются все массивы. Таким образом, каждый массив следует оформлять в виде самостоятельной таблицы.

Как известно из статей [6] и [3], в системе РАМА необходимое поддерево определяется тремя компонентами: место, путь и условие. Путь пропускается в том случае, когда определяемое поддерево идентифицируется однозначно местом, а условие пропускается тогда, когда все данные поддерева включаются в обрабатываемый набор. Рассмотрим эти компоненты.

1. Местом служит (составное) имя корня группы или дополнительного доступа. В последнем случае таблица оформляется по описанию той повторяющейся группы, которая содержит ключи дополнительного доступа, а ее экземпляры поместятся в таблице в том порядке, который определяется дополнительным доступом. Если из состава группы, определяемой местом, следует печатать только некоторые вершины (и также их поддеревья), то пользователь должен все такие вершины перечислять по следующему синтаксису:



2. Путь задается в виде последовательности значений ключей всех замков, как находящихся выше места, так и собственного замка, если местом служит корень повторяющейся группы или имя дополнительного доступа. Выбор данных в таблицу осуществляется или путем фиксирования значений ключей или заказом цикла над экземплярами повторяющейся группы.

3. Условие дает возможность для фильтрации данных, локализированных при помощи места и пути.

Таким образом, оформление таблицы зависит от места, а

содержание таблицы может быть ограничено путем и условием. Так как в самой таблице путь и условие в явном виде не отображаются, то перед каждой таблицей печатается специальная "экспресс-информация", пример[1] которой приведен на рис. 1.

```
                    EXPRESS-INFO
ЛЕГ=WPK  КОМПЛ=WPK  ЗАПИСЬ=ЛАБОРАТОРИЯ
МЕСТО=YHIK+TIITEL+AUTOR+AD
ПУТЬ: INDEX=*
УСЛОВИЕ=.AUTOR=ROOMELDI&(.LIIK=TART/.LIIK=TEES)&.AD=1985
```

Рис. 1.

В кратком обзоре общих положений оформления таблиц будем пользоваться терминологией, определенной в статье [1]. Таким образом, шапка - это древовидная структура с фиктивным корнем, элементами которой являются ортогоны - прямоугольники из шапки. Шапка столбцов является головкой таблицы, а шапка строк отображает распределение данных по некоторому дереву. Фиктивным корнем шапки столбцов является имя корня или дополнительного доступа, заданное в качестве места. Системой пользователю предоставляется возможность задавать текст заглавия таблицы, которое печатается перед шапкой столбцов. Если пользователь отказывается от этой возможности, а вершина места снабжена в легенде именем оформления (см. [5], стр. 17 - 18), то последнее будет заглавием, иначе таблица печатается без заглавия. Шапка столбцов строится в общем случае

---

[1] Легенда записи и общая среда приведены в статье [3].

(исключением является массив) в однозначном соответствии с легендой (с учетом дополнения определения места). Ортогоны той шапки заполняются именами оформления, а если они не заданы, то именами соответствующих вершин легенды. Ширина ортогона конечного уровня определяется как максимум из длины текста ортогона и длины соответствующего атома. Ширины ортогонов выших уровней определяются как суммы ширин ортогонов низших уровней. Ширина всей таблицы не ограничивается, но пользователь должен учитывать, что шириной печатной страницы является 128 символов. Если ширина таблицы превышает это число, то пользователю следует дополнить TCL-пакет заказа (см. [4]) необходимым количеством DD-карт (по одной на каждую "лишнюю" страницу).

Пользователю предоставляется возможность смотреть создаваемую таблицу на экране по мере заполнения отдельных страниц (смотреть можно как левые, так и правые части страниц).

Таблицы можно распределить на три основные типа: простые перечни, перечни с итоговыми строками и матрицы. Ниже рассматриваются возможности и поведение ГТ по каждому из этих типов в отдельности.

2. Перечень

Перечень - это такая таблица, где шапка строк отсутствует, а количество строк является произвольным. В системе ГТ перечень соответствует поддереву легенды, корню простой, альтернативной или повторяющейся группы (с исключением корня массива). Каждый экземпляр данной корневой группы отображается в таблице n строками ($n \geqslant 1$) в зависимости от состава

данной группы. Системой ведется счет экземпляров корневой группы. С этой целью в шапку добавлен самый левый ортогон "номер по порядку".

Приведем несколько примеров. Так как наши типографические возможности не позволяют в статье показать широкие таблицы, то ширины столбцов не всегда соответствуют описаниям данных. В качестве первого примера на рис. 2 приведена таблица, экспресс-информация которой представлена на рис. 1.

:N ПП:	TIITEL	: AD :	AUTOR :
: 1 :	РООМЕЛЬДИ Р. Э., О ПРОСТЫХ ПРАВОАЛЬ-	:1985:	ROOMELDI:
: :	ТЕРНАТИВНЫХ КОЛЬЦАХ. УЧ. ЗАПИСКИ ТГУ	: :	:
: :	1985, Н. 700, 60 - 70.	: :	:
: 2 :	РООМЕЛЬДИ Р. Э., О ДОКАЗАТЕЛЬСТВЕ	:1985:	ROOMELDI:
: :	ТОЖДЕСТВ В НЕАССОЦИАТИВНЫХ КОЛЬЦАХ.	: :	:
: :	XVIII ВСЕСОЮЗНАЯ АЛГЕБРАИЧЕСКАЯ КОН-	: :	:
: :	ФЕРЕНЦИЯ, КИШИНЕВ, 1985, С. 136.	: :	:

Рис. 2.

Легенда WPK, по которой построена эта таблица, приведена в статье [3] без комментариев по поводу вершины TIITEL. Значение этого атома разбита на четыре "подзначения" именно в интересах оформления таблицы. Например, если местом служит "A+TIITEL" и путем "*", то в результате получается отсортированный по годам выпуска список публикаций.

Следующие два примера на рисунках 3 и 4 основываются на легенде KOOLID. Так как используемый нами распознаватель СПТ WIRTH не допускает вне стринга букв русского алфавита, то

запасы значений определены как стринг (имена оформления задаются также стрингами в скобках). Хотя в дальнейшем в некоторых примерах экспресс-информация опущена, она в действительности всегда печатается.

```
LEG KOOLID KEY=KOOL TEXT  'ДАННЫЕ О ШКОЛЕ'
*1 KOOL PICT=50  'НАЗВАНИЕ ШКОЛЫ'
*1 KLASS REP KEY=NR SORT 'КЛАССЫ'
  *2 NR PICT=3  'НАПР. 1А, 10Б'
  *2 OP ('ПЕРЕЧЕНЬ УЧЕНИКОВ') REP=40,PICT=20,SORT
     KEY=NIMI,EESNIMI
    *3 EESNIMI  'ИМЯ УЧЕНИКА'
    *3 NIMI     'ФАМИЛИЯ УЧЕНИКА'
    *3 S SCOPE=[М,'Д']  'ПОЛ'
    *3 N INDEX  'ПОРЯДКОВЫЙ НОМЕР'
    *3 TABEL ('ТЕКУЩИЕ ОЦЕНКИ') ARRAY[SEM=4,(AINE)]
      *4 H REP NAT MAX=5 PICT=1.0  'ОЦЕНКИ'
      *4 KH PICT=1.2 PSEUDO=W1MEANE(H,H01) REAL
         'СРЕДНЯЯ ОЦЕНКА'
    *3 KMH PICT=1.2 PSEUDO=W1MEANE(TABEL,KH) REAL
       'ОБЩАЯ СРЕДНЯЯ ОЦЕНКА'
    *3 КОНТ INDEX BASEDOWN=KMH 'МЕСТО ПО ОБЩЕЙ СРЕДНЕЙ'
    *3 HUVI SCOPE=['НЕТ','ТЕХ','ИССК','СПОРТ','ПРОЧИЕ']
       'ИНТЕРЕСЫ'
  *2 DIS REP PICT=8
    *3 AINE SCOPE=['ЭСТ.ЯЗ',МАТЕМ.,'РУКОД.','ПЕНИЕ',
       'ФИЗ.ВП.','РУССК.'] 'ПРЕДМЕТЫ'
END
```

```
                    EXPRESS-INFO
ЛЕГ=KOOLID КОМПЛ=KOOLID  ЗАПИСЬ=СРЕДНЯЯ ШКОЛА N 1
МЕСТО=KLASS
ПУТЬ=1A
```

N ПП	NR	EESNIMI	NIMI	S	N	КМН	КОНТ	HUVI	DIS
1	1A	ПЕТЯ	АЛОВ	М	1	4.21	19	СПОРТ	ЭСТ.ЯЗ.
		КАТЯ	БЕЛАЯ	Д	2	5.00	1	ИССК	МАТЕМ.
		ВАНЯ	ИВАНОВ	М	3	3.44	37	НЕТ	ФИЗ.ВП
		.	.	.					ПЕНИЕ
									. . .
		ВАСЯ	ЯНОВ	М	40	4.06	27	ПРОЧИЕ	

Рис. 3.

Если задавать "МЕСТО=OP" и "ПУТЬ=1A,*", то получаем:

ПЕРЕЧЕНЬ УЧЕНИКОВ

N ПП	EESNIMI	NIMI	S	N	КМН	КОНТ	HUVI
1	ПЕТЯ	АЛОВ	М	1	4.21	19	СПОРТ
2	КАТЯ	БЕЛАЯ	Д	2	5.00	1	ИССК
.	.	.	.	.	.	.	.
40	ВАСЯ	ЯНОВ	М	40	4.06	27	ПРОЧИЕ

Рис. 4.

3. Перечень с итоговыми строками

К таблицам такого типа принадлежит множество экономических отчетов. Приведем на рисунке 5 сперва пример такой таблицы, затем текст соответствующей ей легенды, и рассмотрим, какие соглашения там соблюдаются.

ПРИХОД МАТЕРИАЛОВ

N ПП	МАТЕРИАЛ			
	ГРУППА	МАРКИ В ГРУППЕ		
		МАРКА	ПРИХОД	
			ВЕС	СТОИМОСТЬ
1	10	1000	1.10	2.20
			2.00	4.00
		ВСЕГО МАРКА	3.10	6.20
		2000	2.00	8.00
			3.00	12.00
		ВСЕГО МАРКА	5.00	20.00
	ВСЕГО ГРУППА		8.10	26.20
2	20	1000	1.00	1.00
			2.00	2.00
		ВСЕГО МАРКА	3.00	3.00
	ВСЕГО ГРУППА		3.00	3.00
ИТОГО НА СКЛАДУ			11.10	29.20

Рис. 5.

```
LEG MATE KEY=LADU REAL PICT=4.2
*1 LADU TEXT PICT=16  'СКЛАД'
*i T ('ПРИХОД МАТЕРИАЛОВ')
  *2 GRUPP ('МАТЕРИАЛ') REP SORT KEY=A
    *3 A ('ГРУППА')  DEC PICT=2
    *3 MARK ('МАРКИ В ГРУППЕ') REP SORT KEY=B
      *4 B ('МАРКА') DEC PICT=4
      *4 MAT ('ПРИХОД') REP
        *5 K ('ВЕС')
        *5 M ('СТОЙМОСТЬ')
      *4 SUMMA ('ВСЕГО МАРКА')
        *5 K PSEUDO=W1SUME(MAT,MAT.K)
        *5 M PSEUDO=W1SUME(MAT,MAT.M)
    *3 SUMMA ('ВСЕГО ГРУППА')
      *4 K PSEUDO=W1SUME(MARK,MARK.K)
      *4 M PSEUDO=W1SUME(MARK,MARK.M)
  *2 SUMMA ('ИТОГО НА СКЛАДУ')
    *3 K PSEUDO=W1SUME(GRUPP,GRUPP.K)
    *3 M PSEUDO=W1SUME(GRUPP,GRUPP.M)
END
```

Как видно, таблица на рис. 5 отличается от простого перечня перестановкой итоговых строк. Здесь мы имеем дело с таким случаем, где работой генератора таблиц следовало бы управлять средствами определенного языка. Однако, поскольку задача для ГТ была поставлена иначе, то пришлось найти возможности управления составлением таблицы путем добавления сигналов в легенду. Эти сигналы являются следующими:

а) именем корня поддерева сводки должно быть SUMMA;

б) структура поддерева сводки, а также имена всех его вершин должны совпадать со структурой и именами вершин того поддерева, где находятся суммируемые значения атомов (корнем этого поддерева является в нашем примере вершина МАТ);

в) все атомы в поддереве сводки должны быть псевдоатомами с заданными функциями вычисления их значений.

Если эти условия одновременно не удовлетворяются, то ГТ не считает поддерево связанным с итоговой строкой. Приведем на рис. 6 и 7 еще два примера, иллюстрирующие возможности разбиения вышеприведенной таблицы.

```
                EXPRESS-INFO
ЛЕГ=МАТЕ  КОМПЛ=МАТЕ  ЗАПИСЬ=СКЛАД
МЕСТО=GRUPP
ПУТЬ=20
```

МАРКИ В ГРУППЕ

N ПП	МАРКА	ПРИХОД	
		ВЕС	СТОИМОСТЬ
1	1000	1.00	1.00
		2.00	2.00
	ВСЕГО МАРКА	3.00	3.00
ВСЕГО ГРУППА		3.00	3.00

Рис. 6.

```
EXPRESS-INFO
ЛЕГ=МАТЕ  КОМПЛ=МАТЕ  ЗАПИСЬ=СКЛАД
МЕСТО=SUMMA
```

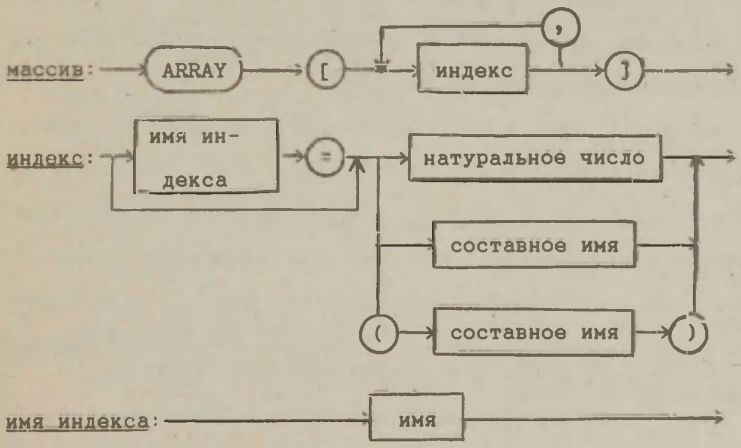
ИТОГО НА СКЛАДУ

N ПП	K	M
1	11.10	29.20

Рис. 7.

4. Матрица

Матрица является печатным изображением массива, т.е., повторяющейся группы со специальной организацией, который определяется по следующему синтаксису (см. также [5]):



Элементом массива может служить как атом, так и поддерево. Однако, ГТ не в силах составлять таблицу в таком случае, когда элементом является корень другого массива; если в поддереве, являющимся элементом, встречается опять массив, то он не изображается в матрице. Для каждого элемента массива выделяется одна клетка матрицы, т.е., поле, которое ограничивается горизонтальными и вертикальными линиями. Каждая матрица имеет шапку столбцов, которая строится по описаниям последнего индекса и элемента массива. Если этот индекс имеет n значений, то в шапку поместится слой из n ортогонов. Текстом такого ортогона служит значение индекса - как правило, натуральное число, но если верхняя граница изменения индекса задана как составное имя в скобках, то текстом будет элемент запаса значений указанного атома. Над этим слоем ортогонов строится общий ортогон, если последний индекс снабжен именем, или значениями индекса служат элементы запаса значений атома. Текстом общего ортогона служит в таком случае имя индекса или указанного атома.

Если элементом массива является поддерево, то под каждый ортогон слоя строится шапка столбцов этого поддерева по правилам оформления простого перечня.

Одномерный массив не имеет шапки строк. Если же количество индексов выше одного, то строится и такая шапка. Для каждого индекса (от первого до предпоследнего) в шапке строк выделяется один столбец, где ортогоны заполняются аналогично "ортогонам слоя" в шапке столбцов. Имя индекса (или указанного атома с запасом значений) помещается в таблицу как заглавие данного столбца.

Наш первый пример на рис. 8 представляет матрицу, которая очевидно составлена в начале второго семестра.

```
                    EXPRESS-INFO
ЛЕГ=KOOLID  КОМПЛ=KOOLID  ЗАПИСЬ=СРЕДНЯЯ ШКОЛА N 1
МЕСТО=TABEL
ПУТЬ=1A,ИВАНОВ,ТОЛЯ
```

ТЕКУЩИЕ ОЦЕНКИ

SEM	AINE											
	ЭСТ.ЯЗ		МАТЕМ.		РУКОД.		ПЕНИЕ		ФИЗ.ВП.		РУССК.	
	Н	КН	Н	КН	Н	КН	Н	КН	Н	КН	Н	КН
1	4	4.25	5	4.50	3	4.00	2	3.60	3	3.33	4	4.67
	5		4		4		2		3		5	
	5				5		5		4		5	
	3						5					
							4					
2	5	5.00	3	4.00	3	3.00	*	0.00	*	0.00	5	5.00
	5		5									
3	*	0.00	*	0.00	*	0.00	*	0.00	*	0.00	*	0.00
4	*	0.00	*	0.00	*	0.00	*	0.00	*	0.00	*	0.00

Рис. 8.

Если же в определении массива поменять местами индексы:

```
*3 TABEL ('ТЕКУЩИЕ ОЦЕНКИ') ARRAY[(AINE),SEM=4],
```

то матрица принимает вид, указанный на рис. 9.

ТЕКУЩИЕ ОЦЕНКИ

AINE	SEM 1		SEM 2		SEM 3		SEM 4	
	H	KH	H	KH	H	KH	H	KH
ЭСТ.ЯЗ	4	4.25	5	5.00	*	0.00	*	0.00
	5		5					
	5							
	3							
МАТЕМ.	5	4.50	3	4.00	*	0.00	*	0.00
	4		5					
РУКОД.	3	4.00	3	3.00	*	0.00	*	0.00
	4							
ПЕНИЕ	2	3.60	*	0.00	*	0.00	*	0.00
	2							
	5							
	5							
	4							
ФИЗ.ВП.	3	3.33	*	0.00	*	0.00	*	0.00
	3							
	4							
РУССК.	4	4.67	5	5.00	*	0.00	*	0.00
	5							
	5							

Рис.9.

Следующий пример на рис. 10 иллюстрирует изображение трехмерного атомарного массива, который описывается по следующему отрывку какой-то легенды:

```
*3 B TEXT SCOPE=[C:F]
*3 T ARRAY[2,A=3,(B)] NAT
```

По этой легенде получается матрица следующей формы:

		A	B			
			C	D	E	F
1	1					
	2					
	3					
2	1					
	2					
	3					

Рис. 10.

Напоминаем, наконец, что простой пример матрицы был приведен в статье [3] как изображение псевдомассива. Следует отметить, что прибавленные в эту таблицу строка и столбец сводок представляют собой исключение: они генерируются функцией составления частотной таблицы (W1SAGED), а ГТ "знает" это обстоятельство.

Литература

1. Пейал Я., Соо В., Томбак М., Язык вывода данных. Труды ВЦ ТГУ, 1982, 49, 42 - 61.
2. Пейал Я., Соо В., Реализация языка вывода данных. Труды ВЦ ТГУ, 1985, 27 - 38.

3. Изотамм А., Диалоговые средства в СУБД РАМА. Наст. сборн.
4. Ээремаа К., Язык управления заданиями системы РАМА. Труды ВЦ ТГУ, 1983, 50, 3 - 22.
5. Изотамм А., Каазик Ю., Томбак М., Язык определения записи. Труды ВЦ ТГУ, 1978, 41, 7 - 64.
6. Каазик Ю., Ээльма П., Ввод и корректировка данных. Труды ВЦ ТГУ, 1978, 41, 75 - 96.

Стандарт схематического языка и схемный редактор

Ю. Кихо, К. Куллман

Идея представления алгоритмов и программ в виде несложных обозримых схем - групп строк, охваченных слева специальными линиями уровня - возникла уже в 1982 году. Порядок выполнения строк внутри таких схем регулируется условиями и выходными стрелками. Первое сообщение о методе схематического программирования опубликовано в 1983 году (см. [1]). Скоро после этого было введено (см.[2]) еще понятие продленной стрелки, позволяющей описывать прерывание сразу нескольких охватывающих схем. С самого начала метод схематического программирования себя хорошо зарекомендовал как при обучении студентов программированию, составлению сложных алгоритмов, (см. [3,4]), так и в качестве основы разработки специальных систем программирования (см. [5,6,7]). Математическое понятие E-схемы как обобщение схемы управления разработано в [8]. После этого в [9,10,11] построены уже достаточно совершенные системы, а метод схем более серьезно внедрен в учебном процессе (см.[12,13]), даже в курсе информатики для начинающих (см. [14]). В настоящее время схематическое программирование как разновидность графических методов программирования (см. [15,16]) приобретает еще большую актуаль-

ность: расширяющиеся графические восможности новых ЭВМ повышают, в частности, плодотворность идеи схематического представления структур программ.

Перспективностью данного подхода к представлению алгоритмов и программ определяется необходимость, а накопившийся опыт применения схематического метода программирования обеспечивает возможность фиксации некоторого стандарта соответствующего схематического языка. В настоящей статье представлен стандарт (SK88), содержащий эталонное описание синтаксиса и семантики схематических программ, а также список сопутствующих терминов на русском, эстонском и английском языках. Кроме того, описана система схематического программирования на базе языка Си. Схемный редактор, как основная часть этой системы, имеет и самостоятельное значение: может быть применен в других новых системах (непосредственно или как прототип редактора схематически представленных структур).

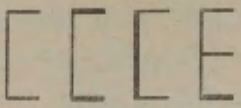
1. Синтаксис схематического языка

Обозначения:

::=	определено как
<>	нетерминальный символ
!	или
()*	необязательная (вертикально) повторяющаяся часть
()+	обязательная (вертикально) повторяющаяся часть
R	символ перевода строки
N	натуральное число (>0)
B	цепочка из 0 или более символов, не содержащая R
<пусто>	пустая цепочка символов

Терминальные символы : - * ? ; : = ~ ( ) ⊢ ⊢

Терминальная графика - линии уровня:



Грамматика:

<схематическая программа> ::= {<инструкция>}*

<инструкция> ::= <элементарная инструкция> ! <схема>

<элементарная инструкция> ::= B <комм>

<комм> ::= <комментарий> R

<комментарий> ::= <пусто> ! --B

<схема> ::= <простая схема> ! <схема выбора>

! <схема цикла> ! <нима>

<простая схема> ::= ┌─ <ком>

│ <тело>

└─ <ком>

<тело> ::= {<член>}*

<ком> ::= B R

<член> ::= <комм> ! <инструкция> ! <условие> ! <терминатор>

<условие> ::= B? <комм>

<терминатор> ::= <сильный терминатор> <комм>

! <слабый терминатор> <комм>

<сильный терминатор> ::= ~ <протяженность>

<слабый терминатор> ::= ~ (<протяженность>)

<протяженность> ::= N

<схема выбора> ::= ┌─ <ком>

? <селектор> <комм>

<варианты>

└─ <ком>

<варианты> ::= { <вход> } +
{ <тело> }

<вход> ::= <строгий вход> ! <нестрогий вход>

<строгий вход> ::= ├─ <детектор> : <комм>

<нестрогий вход> ::= ├─ <детектор> : <комм>

<селектор> ::= B

<детектор> ::= B

<схема цикла> ::= <неогражденный цикл> ! <огражденный цикл>

<неогражденный цикл> ::= ┌─ <ком>
│ <тело>
└─ <ком>

<огражденный цикл> ::= ┌─ <ком>
│ (* <заголовок>)*
├─ <ком>
│ <тело>
└─ <ком>

<заголовок> ::= <общий заголовок> ! <кортежный заголовок>

<общий заголовок> ::= B;B;B <комм>

<кортежный заголовок> ::= <счетчик> B <комм>

<счетчик> ::= <пусто> ! <переменная> =

<переменная> ::= B

<ниша> ::= ┌─ <ком>
│ (B <комм>)*
└─ <ком>

Терминатор ⌟N можно заменить горизонтальной стрелкой, направленной влево и проходящей через N-1 охватывающую его линию уровня. Слабый терминатор ⌟(N) изображается аналогичной пунктирной стрелкой.

2. Семантика схем

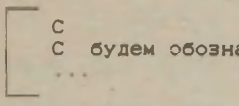
При определении семантики схематической программы исходим из понятия условно-последовательной инструкции (УП-инструкции), семантика которой точно определена в [8] на стр. 59. Ниже приводятся такие правила преобразования схем, при помощи которых можно любую схематическую программу преобразовать в УП-инструкцию. По определению, полученная в результате этих преобразований УП-инструкция семантически равносильна исходной схематической программе. Будем пользоваться следующей формой записи УП-инструкции:

<УП-инструкция> ::= ⌈ {<УП-член>}*

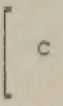
<УП-член> ::= <элементарная инструкция> ! <условие>

! <терминатор> ! <УП-инструкция>

Кроме того, бесконечную периодическую УП-инструкцию (состоящую из повторяющейся группы С УП-членов)



будем обозначать через



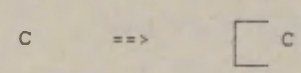
Вводим еще понятия вложенности и избыточности терминатора (см. также [13]). Пусть С - некоторая последовательность членов. Тогда вложенностью в С терминатора ⌊N называется число охватывающих его схем. Образно говоря, вложенность - это число линий | или ⌈, стоящих перед терминатором. Терминатор называется избыточным в С, если его протяженность превышает вложенность. В семантических правилах через С обозначается последовательность УП-членов (предполагается, что все входящие в С подсхемы уже приведены к виду УП-инструкций).

Семантика простой схемы:

Простая схема

выполняется как УП-инструкция, причем слабые и сильные терминаторы не различаются. Последнее означает, что как ⌟N так и ⌟(N) считаются терминаторами (в смысле [8]) с протяженностью N: оба они приводят к завершению выполнения N-ой охватывающей схемы.

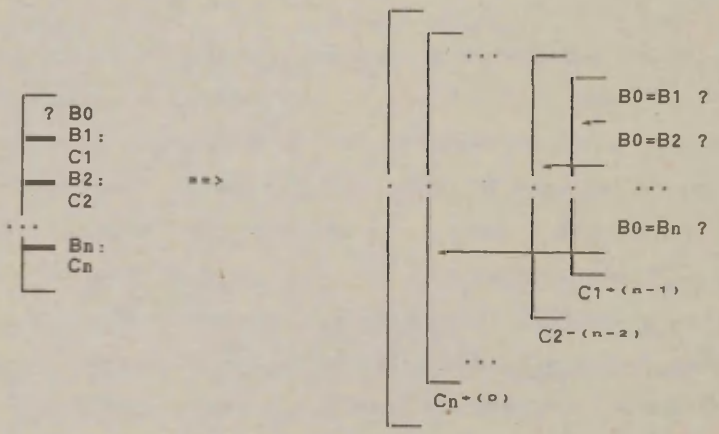
Переобразование схематической программы:



Учитывая, что схематическая программа состоит из элементарных инструкций и схем, написанных друг под другом, то указанное выше правило определяет направленность выполнения схематической программы сверху вниз.

Переобразование схемы выбора:

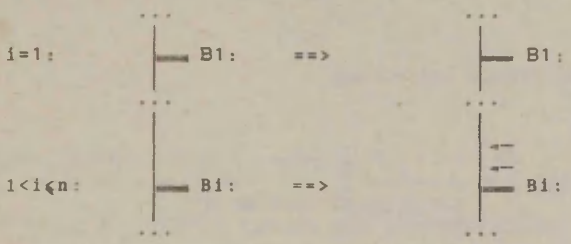
а) редуцирование нестрогих входов



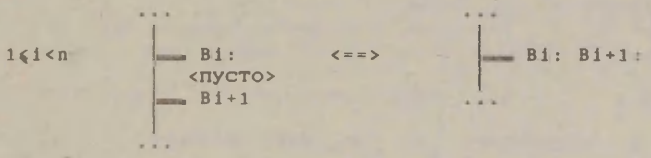
В схеме справа через $Ci^{+(n-i)}$ обозначена последовательность, полученная из Ci увеличением протяженности всех избыточных терминаторов в Ci на величину (n-i).

Условие B0=B1? считается удовлетворенным по определению также в том случае, когда Bi <пусто>.

б) редуцирование строгого входа



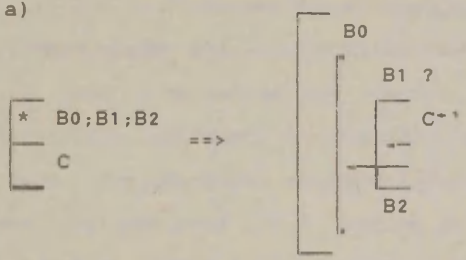
в) семантика пустого варианта



Преобразование огражденного цикла:

В случае общего заголовка:

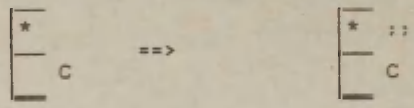
а)



В схеме справа через C^{+1} обозначена последовательность, полученная из C увеличением протяженности всех избыточных

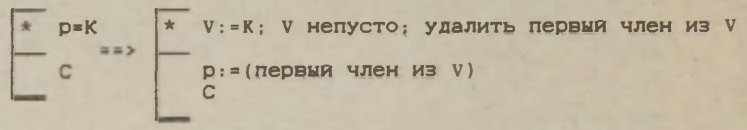
вложенность которых в исходной схеме равняется их протяженности. Такие слабые терминаторы заменены на сильные той же протяженности; точнее, ⌟(N) заменен на ⌟N , если вложенность этого терминатора в C равно N.

б)

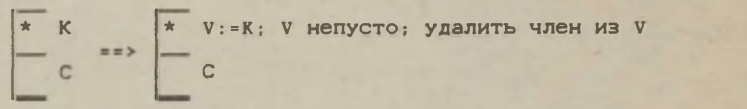


В случае кортежного заголовка:

в)



г)



где p - переменная, K - описание кортежа, V - переменная кортежного типа, V:=K - определить кортеж и присвоить переменной V. В частности, описание кортежа K может быть задано при помощи арифметических выражений a, b и c:

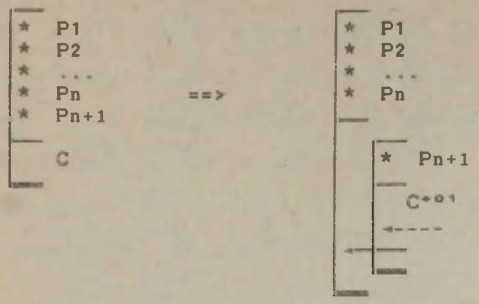
a..b означает кортеж (a,a+1,...,b), причем кортеж пуст, если a>b;

a..b;c означает кортеж (a,a+c,a+2c,...,a+(n-1)c), где n=[(b-a+c)/c], причем кортеж пуст, если n≤0.

Если счетчик не указан, т.е. в случае г), описание кортежа 1..a может быть задано просто в виде арифметического выражения a.

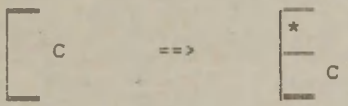
В случае составного заголовка:

д)



В схеме справа через $C^{*^{o1}}$ обозначена последовательность, полученная из С увеличением протяженности всех избыточных терминаторов на единицу, кроме слабых терминаторов, вложенность которых в исходной схеме равняется их протяженности. Протяженность таких слабых терминаторов не изменяется.

Преобразование неограниченного цикла:



Преобразование ниши стандартом не определяется. Семантика ниши точно устанавливается конкретной системой схематического программирования. В схематических алгоритмах нишей изображается подалгоритм, который определяется либо отдельно, либо уточняется позже.

Семантику конкретного (нестандартного) схематического языка желательно также описывать в терминах УП-схем, поскольку их обработка происходит всегда одинаковым образом.

3. Терминология

базовый язык	- baaskeel	- base language
вариант	- variant	- case
вложенность	- tase	- level
вход	- sisend	- entry
выходная стрелка	- väljundnool	- exit arrow
детектор	- detektor	- detector
инструкция	- instruktsioon	- instruction
линия уровня	- tasemejoon	- level line
неограниченный цикл	- vabatsükkel	- free loop
ниша	- niši	- niche
кортежный заголовок	- järjendpäis	- cortege heading
общий заголовок	- üldpäis	- general heading
ограниченный цикл	- piiratud tsükkel	- bounded loop
простая схема	- lihtskeem	- simple sketch
селектор	- selektor	- selector
сильный терминатор	- tugev terminaator	- strong terminator
слабый терминатор	- nõrk terminaator	- weak terminator
схема	- skeem	- sketch
схема выбора	- valikuskeem	- switching sketch
схема цикла	- tsükliskeem	- looping sketch
схематическая программа	- skeemprogramm	- sketchy program
схематический язык	- skeemkeel	- sketchy code
схемный процессор	- skeemprotsessor	- sketch processor
схемный редактор	- skeemtoimeti	- sketch editor
терминатор	- terminaator	- terminator
условие	- tingimus	- condition
элементарная инструкция	- käsk	- statement

4. Схематический язык на базе Си

Используя стандарт SK88 опишем синтаксис и семантику языка схематического программирования SKC, базовым языком которого служит язык Си. Соответствующая система программирования реализована на ПЭВМ ИБМ в виде схемного процессора перевода SKC-программ на язык Си.

Синтаксис SKC получается небольшим изменением стандартного синтаксиса. В последнем необходимо следующее.

1. Удалить понятие <строгий вход>.

2. Переопределить:

<вход> ::= <нестрогий вход>

<элементарная инструкция> ::= <строка операторов> <комм>

! <схемная строка> <комм>

<описание кортежа> ::= В .. В <маг> ! В

<ниша> ::= <имя> <комм>
{<элементарная инструкция>}*
<ком>

<комментарий> ::= <пусто> ! --- <произвольный текст>

3. Добавить определения:

<имя> ::= <идентификатор файла> ! <пусто>

<строка операторов> ::= <пусто> ! В; <строка операторов>

<схемная строка> ::= <условная строка>!<циклическая строка>

<условная строка> ::=

[? (В) <строка операторов> <альтернатива>

<альтернатива> ::= <пусто> ! : <строка операторов>

<циклическая строка> ::= [* <шапка> <строка операторов>

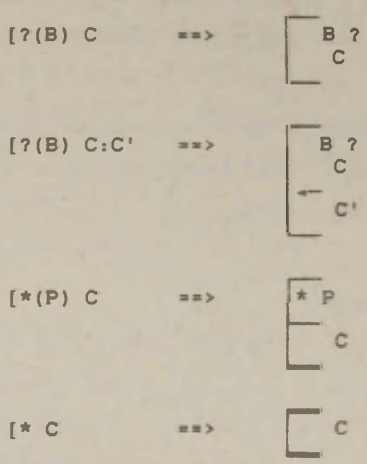
<шапка> ::= (<заголовок>) ! <пусто>

<маг> ::= <пусто> ! ; В

Семантика SKC уточняется следующим образом.

1. Вместо B во всех конструкциях подразумевается выражение на языке Си.

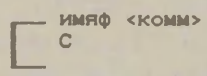
2. Преобразования для схемных строк:



где C и C' - строки операторов, а P - заголовок.

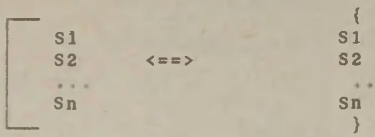
В виде схемных строк записываются небольшие простые схемы специального вида, а также схемы цикла, которые не содержат подсхем, условий и терминаторов.

3. Нима



означает макрокоманду, которая во время трансляции схематической программы заменяется содержимым файла "имяф". Если такого файла нет (или же на месте имяф контрукция <пусто>), то нима заменяется на группу элементарных инструкций C. Таким образом, нима служит в языке SKC средством помаговой разработки программы.

4. Простая схема, тело которой состоит только из строк операторов, равносильна блоку в смысле Си:



(применяется в описаниях структур struct, union и др.)

5. В заголовке кортежного типа счетчик является целой переменной.

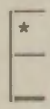
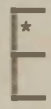
6. Стрелка вида ⌟ (B) выполняется как return(B) в том случае, когда она выводит на уровен 0.

5. Схемный редактор

Схемный редактор разработан в духе известных текстовых редакторов (WordStar, TurboPascal и др.) и реализован для ПЭВМ ИБМ. Но существенной отличительной чертой его является структурность, ориентированность на работу со схемами. Например, (пустая) схема порождается и уничтожается только целиком. Поэлементно, по строкам и символам, можно редактировать лишь входящие в схему тексты - элементарные инструкции, заголовки, условия, комментарии и т.п.

Схемный редактор допускает определенные отклонения от стандарта SK88 при изображении графических элементов на экране (см. таблицу 1). Кроме того, предусмотрена обработка схемных строк (в смысле SKC), а также автоматическая проверка баланса круглых скобок внутри конструкций базового языка. Строгий вход схемы выбора не предусмотрен. Длина строки обрабатываемой программы ограничивается шириной экрана.

Особенности экранного изображения элементов схем

Элемент схемы	По стандарту	На экране
Условие	B ?	? B
Селектор выбора	? B	?? B
Сильный терминатор		
Слабый терминатор		
Ниша		
Неогражденный цикл		
Огражденный цикл		

Основные "схемные" операции редактора присвоены функциональным клавишам Ф2, Ф3, ... , Ф10. В таблице 2 приведена сводка этих операций, причем под уровнем 0 подразумевается область вне схем. Как правило, нажатие на функциональную клавишу порождает новый объект, который вставляется после текущей строки (где находится курсор). Исключение составляют клавиши Ф7 и Ф8, изменяющие лишь статус текущей строки.

Выходная стрелка (терминатор) порождается клавишей Ф9, последующие нажатия на Ф9 увеличивают длину стрелки (напомним, что слабая стрелка просто короче по сравнению с соответствующей сильной стрелкой). Для уменьшения протяженности стрелки предусмотрена клавиша "стирание слева" (backspace). Уменьшение минимальной слабой стрелки стирает ее.

В обмен, для уничтожения объектов используется ^Y (т.е. пара ctrl + Y). В частности, таким способом удаляется и схема, но перед уничтожением схемы необходимо опустошить ее.

Таблица 2

Назначение функциональных клавиш

Клавиша	Допустимое место-нахождение курсора	Порождаемый объект
Ф1	Любое	Меню-справочник
Ф2	Уровень 0 или тело схемы	Простая схема
Ф3	Уровень 0 или тело схемы	Неограженный цикл
Ф4	Уровень 0 или тело схемы	Огражденный цикл
	Начало неогр. цикла	Начало огр. цикла
	Начало огр. цикла	Дополнительный заголовок
Ф5	Уровень 0 или тело схемы	Схема выбора
Ф6	Тело схемы выбора	Дополнительный вход
	Уровень 0 или тело схемы	Схема выбора
Ф7	текст	? текст
	? текст	[? текст
	[? текст	текст
Ф8	текст	[* текст
	[* текст	текст
Ф9	Тело схемы	Слабая стрелка - ⌟(1)
	Слабая стрелка - ⌟(N)	Сильная стрелка - ⌟N
	Сильная стрелка - ⌟N	Слабая стрелка-⌟(N+1)
Ф10	Уровень 0 или тело схемы	Ниша

Редактирование содержимого схемы (элементарных инструкций, условий, заголовков, комментариев и т.п.) мало от-

личается от обработки текста в такой известной системе как WordStar. Отличия связаны, главном образом, с небольшим изменением понятия блока. В данном случае, при выделении частей схематической программы в качестве блоков, не допускается нарушение целостности схем. Выделенный блок членов С может быть превращен в простую схему, простая схема в неограниченный цикл и неограниченный цикл в последовательность членов С; каждое из этих переходов осуществляется нажатием ^KN. Краткий обзор всех операций предоставлен в меню-справочнике на рис. 1.

```
                          SKM-C HELP (Editor)
F2 ┌             ^E-Line up      <BLOCK>          <QUICK>
F9 └             ^X-Line down    ^KB-Put beg      ^QR-Prog.beg
                 ^S-Col.left     ^KK-Put end      ^QC-Prog.end
F3 ┌             ^D-Col.right    ^KR-Read blk     ^QY-Del.line end
F4 │*            ^A-Wrd.left     ^KW-Write blk    ^QE-Ske.beg
   │* A1;A2;A3   ^F-Wrd.right    ^KC-Copy blk     ^QX-Ske.end
   │* I=N1..N2   ^R-Ske.up       ^KV-Move blk     ^QS-Line beg
   │* I=N1..N2;N ^C-Ske.down     ^KY-Del.blk      ^QD-Line end
   │* N          ^W-Roll down    ^KH-Hide.blk     ^QB-Blk.beg
   └             ^Z-Roll up      ^KN-Toggl.blk    ^QK-Blk.end
F8 ┌  [* (P) L   DEL-Del.csr     ^KD-Save&Quit    ^QN-Make.blk
   └             ^G-Del.csr      ^KX-Save&Quit    ^QF-Find
                 ^T-Del.wrd      ^KS-Save         ^QA-Replace
F5 ┌             ^Y-Del.line     ^KQ-Quit          <PARM>
   │??           RET-Line feed   ^KE-Ext.Edit     B-Backward
F6 ┌  Mask:      ^N-Push line                     N-Not ask
   └             ^V-Ins/Over                      G-Global
F7 │?  [?(B)T:F  ^L-Find/Rep                       n-Times
                 ^U-Stop                          C-Comp.line
F10┌  <file>                                      I-In block
   └                                              L-In current line
                 ESC-Quit        --- Remark
```

Рис. 1. Меню-справочник схемного редактора

Обрабатываемая схемным редактором программа сохраняется в обычном текстовом файле, каждая запись в которой представляет одну строку программы и снабжена еще префиксным полем Н для хранения дополнительной информации. Как правило, поле Н

занимает два байта (h1,h2), лишь для строки-терминатора H=(h1,h2,h3). Первый байт характеризует тип строки (см. рис. 2). В общем, h1 содержит "код" графического элемента, который непосредственно предшествует тексту в данной строке.

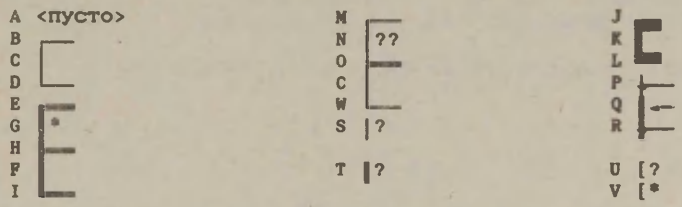


Рис. 2. Кодирование графических элементов в SKC

Второй байт h2 префиксного поля является указателем вложенности текста, следующего за графическим элементом. Слабая стрелка, соответствующая терминатору ⌟(n) имеет префиксное поле

$$h1='Q', h2=n, h3=n-n+1,$$

где n - вложенность терминатора. Другими словами, h2 указывает вложенность "хвоста" стрелки, а h3 вложенность ее "конца" (⌟). Сильная стрелка ⌟n характеризуется так же, как ⌟(n), только h1='P' или h1='R'. На рис. 3. приведен кусок текстового файла схематической программы.

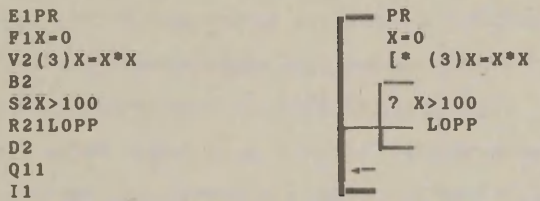


Рис. 3. Кодирование схематической SKC-программы

Описанный выше схемный редактор в настоящее время включен в систему программирования SKM-C. Последняя реализована на ПЭВМ ИБМ с объемом ОЗУ не менее 512 К байтов и обеспечивает обработку схематических программ на языке SKC. Кроме редактора в систему входят: схемный процессор, программа распечатки листинга, компилятор и сборщик TurboC (последние две компоненты - изделия фирмы Borland).

Литература

1. Кихо Ю. О методе схематического программирования. Тез. докл. 5-го сов.-сем. преподавателей математики вузов, 12-14 окт. 1983 Вильнюс. 143-145.
2. Кихо Ю. Схематическое программирование. Труды ВЦ ТГУ, 1984, 50, 52-68.
3. Кихо Ю. Программирование циклов. Тарту, ТГУ,1984, 69.
4. Кихо Ю. Обучение структурам управления на основе единого алгоритмического языка. Тез. докл. респ. конф. "Теоретические и прикладные вопросы математики" 26-27 сент. 1985. Тарту, 56-58.
5. Кихо. Ю. Один путь усовершенствования базового программного обеспечения микро-ЭВМ. Тез. докл. всесоюзн. конф. "Диалог - 84 - микро". Л.1984, 135-137.
6. Кихо Ю. Схематическое программирование на базе Фортрана. Тез. докл. респ. конф. "Теоретические прикладные вопросы математики" 26-27 сент. 1985. Тарту, 59.
7. Абель П., Кихо Ю. Язык схематического программирования САПФИР.Там же. 5.

8. Кихо Ю. Схематическая форма записи структур . Труды ВЦ ТГУ. 1985, 52, 55-57.

9. Кихо Ю., Куллман К. Система схематического программирования для микро-ЭВМ. Тез. докл. II-ой всесоюзн. конф. "Технология программирования", часть 2. Киев 1986, 177-177.

10. Абель П., Кару К., Кихо Ю. Схематическое программирование для микро-ЭВМ. Труды ВЦ ТГУ, 1986, 54, 145-157.

11. Кихо Ю., Сакс Э. Система схематического программирования для ЕС ЭВМ. Труды ВЦ ТГУ, 1988, 55, 65-81.

12. Кихо Ю., Кяхрик Ю. Внедрение метода схематического программирования. Тез. докл. IV-ой зональной конф. "Опыт применения технических средств в учебном процессе", т. II, Вильнюс 1986, 167-168.

13. Кихо Ю., Сакс Э., Эхасалу Я. Разработка системы программирования для студенческого дисплейного класса и опыт ее применения. Тез. докл. совещания-семинара преподавателей математики 31. марта - 2. апр. 1987. Таллинн, 150-151.

14. Кихо Ю. Основы информатики и вычислительной техники. Тарту, ТГУ, 1985, 32.

15. Хлебчевич Г. Применение графических средств в технологиях программирования. Тез. докл. н.-т. конф. Минск, 22-23. окт. 1987. Бел. НИИТИ, 61-63.

16. Robillard P. Schematic Pseudocode for Program Constructs and Its Computer Automation by Schemacode. Comm. of ACM, Vol 29, 1986, 11, 1072-1089.

С о д е р ж а н и е

1 руб.