МЕТОДИКА СОЗДАНИЯ МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ ЭВМ

Министерство высшего и среднего специального образования
Латвийской ССР
Латвийский ордена Трудового Красного Знамени
государственный университет имени Петра Стучки
Вычислительный центр

МЕТОДИКА СОЗДАНИЯ МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ ЭВМ

Межвузовский сборник научных трудов

Под общей редакцией А.Калниньша

Латвийский государственный университет им. П.Стучки
Рига 1980

УДК 681.3.06

В сборнике рассмотрены методические вопросы создания математического обеспечения ЭВМ: отладка программ, специализированные методы доступа к данным, использование средств ОС ЕС для обеспечения АСУ, программное обеспечение обучающих систем.

Сборник рассчитан на научных работников и специалистов, работающих в области создания математического обеспечения ЭВМ, а также может быть полезен студентам физико-математического и экономического факультетов.

Печатается по решению редакционно-издательского совета
ЛГУ им.П.Стучки от 27 июня 1980 года

М 30502-091у Рез.80.2405 000 000
М 812(11)-80

ОДИН ПОДХОД К ОТЛАДКЕ БОЛЬШИХ СИСТЕМ

Я.Я.Бичевский, Ю.В.Борзов

ВЦ ЛГУ им.П.Стучки

I.Основные идеи

Мы рассмотрим один подход к отладке больших систем, который был применен при разработке системы СМОТЛ [I]. Суть этого подхода состоит в следующем.

Строится специальная система отладки (СО), которая является составной частью разрабатываемой системы. Основным средством СО служат макрокоманды, вставляемые в отлаживаемые программы. Притом имеется возможность управлять выполнением макрокоманд: каждая макрокоманда имеет свой приоритет, задаваемый программистом при ее написании. Кроме этого, имеется так называемый текущий приоритет системы, который в каждом сеансе задается извне. Перед выполнением каждой макрокоманды ее приоритет сравнивается с текущим приоритетом системы. Если приоритет макрокоманды меньше или равен текущему приоритету системы, то она выполняется, в противоположном случае - игнорируется.

Таким образом, имеется возможность без каких-либо изменений системы в различных сеансах осуществлять различные режимы отладки. В частности, можно проигнорировать все макрокоманды отладки - эксплуатировать систему в реальном режиме.

2. Ситуация и проблемы

Система СМОТЛ разрабатывалась в 1974-1976 гг. сотрудниками ВЦ Латвийского госуниверситета Василевским М.П., Зариньшем А.К., Страуюмсом У.М. и авторами данной статьи. Система предназначена для автоматического построения полных систем примеров на ЭВМ "Минск-32". Отличительными осо-

бенностями СМОТЛ являются:

- алгоритмы весьма сложны и зачастую носят эвристический характер;

- системе необходимо выполнить работу большого объема (неоднократный просмотр всех путей больших графов и т.п.), следовательно, программы должны быть максимально быстродействующими,

- вся обрабатываемая информация представлена в виде списков.

Функционально система СМОТЛ состоит из управляющего блока (УБ) и десяти рабочих блоков (РБ). (Смотрите рисунок I, на котором передача управления изображена непрерывными, а передача информации - прерывистыми стрелками).

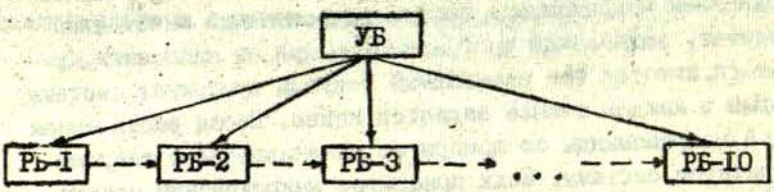


Рис.I.

Система управляется заказом, который вводится с перфокарт. УБ является резидентом системы; он загружает в оперативную память рабочие блоки и передает им управление. Входной информацией для каждого РБ является результат работы предыдущего РБ. Эта информация в виде сложного списка хранится в оперативной памяти. Итак, отметим проблемы, вставшие перед нами:

I. Применимость эвристических алгоритмов проверяется практикой. Только запуская всю систему, мы можем определить пригодность выбранных алгоритмов. Это заставляет нас начать комплексную отладку системы по возможности раньше.

2. Ранняя комплексная отладка требует предусмотреть возможность запуска всей системы при отсутствии некоторых её частей и имитацию работы этих частей. (Идеи об имитации работы отсутствующих блоков почерпнуты из [2]).

3. Ввиду того, что алгоритмы носят эвристический характер, их недостатки могут выявляться в опытной и даже в промышленной эксплуатации. Поэтому необходимо ориентироваться на ситуацию, когда система будет несколько раз изменяться и отлаживаться повторно.

4. При разработке системы должна обеспечиваться возможность получения отладочной информации различной степени детализации. Притом для получения этой информации во время сопровождения системы изменять программы уже недопустимо. Отладочная информация должна быть получена очень простым путем, например, изменением нескольких карт в заказе на работу системы.

3. Неприменимость стандартных средств отладки

Срандартное средство отладки программ - ОП-2, доступное в 1974 году, при всех его достоинствах не удовлетворяет указанным выше требованиям, вытекающим из специфики нашей работы, а именно:

- Степень замедления работы системы, выполняемой под управлением ОП-2, в нашем случае неприемлема. (Отметим только то, что СМОТЛ зачастую использует до 10 минут процессорного времени, а под управлением ОП-2 это составляло бы несколько часов).

- Хотя ОП-2 имеет возможность записи информации в оперативную память и ее распечатки, однако не предполагается работа со списочными структурами данных. (Для задания достаточно простого графа нам потребовалось бы несколько сотен перфокарт, не говоря уже о многочасовом кропотливом труде по кодированию информации в виде графа).

- Ввиду того, что ОП-2 занимает часть оперативной памяти, в принципе невозможно воспроизвести ту же самую ситуацию, которая возникает при работе системы без использования ОП-2.

- Возникают трудности, связанные с выдачей отладочной информации различной степени детализации различными блока-

ми системы. При использовании ОП-2 заказ на отладку ввиду его сложности может составлять лишь разработчик конкретного блока. Это существенно усложняет комплексную отладку системы и получение отладочной информации на стадии опытной эксплуатации системы.

Отметим, что указанные выше проблемы, за исключением, быть может, первой, не решают также отладочные программы ОП-3 для ЭВМ "Минск-32" и ТЕСТРАН для ЕС ЭВМ.

Поэтому для разработки системы СМОТЛ было принято решение разработать свою специальную ОС.

4. Решение проблем

Система отладки реализована следующим образом. В рабочих блоках в виде макрокоманд вставлены обращения к программе, реализующей отладочные операции. Эту программу будем называть отладочным блоком (ОБ). В нашей системе ОБ физически включен в управляющий блок и при работе системы СМОТЛ находится в оперативной памяти. Объем ОБ – 2300 ячеек, включая модули ввода-вывода. В макрокомандах задавались следующие отладочные действия:

1. Ввод с перфокарт закодированного произвольного списка и его размещение в оперативной памяти. Адрес размещения начала списка помещается в переменную, указанную в макрокоманде.

2. Вывод на печать списка, адрес которого указан в переменной, которая, в свою очередь, задается в макрокоманде.

Каждая макрокоманда содержит специальный операнд, называемый приоритетом макрокоманды. Значение этого операнда задает программист при написании макрокоманды, и оно в дальнейшем не меняется. В начале выполнения каждой макрокоманды приоритет этой макрокоманды сравнивается с текущим приоритетом системы. Если приоритет макрокоманды ниже или равен приоритету системы, то макрокоманда выполняется. В противном случае она не выполняется. Приоритет системы задается управляющей картой в заказе на выполнение СМОТЛ. В этой

карте каждому рабочему блоку присваивается определенный приоритет блока (в нашем случае-от 0 до 7). Нулевой приоритет блока означает, что соответствующий рабочий блок в данном сеансе вообще не выполняется (даже не загружается в оперативную память и при работе системы он вообще может отсутствовать). Остальные приоритеты блока означают, что соответствующий рабочий блок должен быть выполнен и при его выполнении текущий приоритет системы устанавливается равный приоритету этого блока. Например, если в данном сеансе третий рабочий блок имеет приоритет 4, то УБ после выполнения первых двух РБ загружает в оперативную память третий РБ, устанавливает текущий приоритет системы, равный 4, и передает управление третьему РБ. В течение работы третьего блока будут выполнены все те макрокоманды отладки, которые имеют приоритет от I до 4, а макрокоманды с приоритетами от 5 до 7, хотя и они в программе могут присутствовать, выполнены не будут.

Ясно, что при разумном использовании приоритетов макрокоманд отладки в зависимости от заказа на выполнение системы СМОТЛ можно получить отладочную информацию различной степени детализации.

Макрокоманды отладки нет нужды менять, и они могут присутствовать в готовой системе. На быстродействие системы СМОТЛ это влияет незначительно, потому, что сравнение приоритета макрокоманды и текущего приоритета системы осуществлено несколькими машинными командами. В частности, полезно оставлять в начале каждого РБ макрокоманды отладки, распечатывающие входную информацию. (Мы им присваивали приоритет 2.) Это намного облегчит в будущем локализацию ошибок.

Отлаженные РБ обычно выполнялись с приоритетом I, а макрокоманды с приоритетом I вообще не использовались. Это позволяло нам выполнить отлаженные блоки без каких-либо отладочных действий.

Теперь рассмотрим имитацию работы несуществующих или неотлаженных РБ. Имитация реализована не замещением РБ мо-

делирующей программой (как в [2]), а включением макрокоманды отладки в начало следующего выполняемого РБ. (При этом имитируемый РБ вообще не выполняется, т.е. ему присваивается нулевой приоритет блока).

Если при выполнении РБ приоритет этой макрокоманды ниже или равен текущему приоритету системы, то она создает в оперативной памяти список, который соответствует результату работы предыдущего (но фактически невыполненного) РБ. (В нашем случае этот список в закодированном виде вводился с перфокарт.) Таким образом получается, что каждый РБ может создавать входные данные для себя и отлаживаться автономно.

Если при выполнении РБ приоритет макрокоманды отладки выше текущего приоритета системы, то она не выполняется. Следовательно, РБ будет обрабатывать информацию, в действительности созданную предыдущим РБ.

Ясно, что этот метод имитации позволяет легко переходить от автономной отладки РБ к комплексной и обратно без всякого изменения разрабатываемой системы: меняется только одна карта, задающая приоритеты выполнения РБ в заказе на работу системы.

Нетрудно заметить, что использование одного и того же текущего приоритета системы для двух различных операций ввода и вывода информации может оказаться в некоторых случаях неудобным. Фактически для каждого типа макрокоманд необходимо было бы задавать текущий приоритет системы независимо. В таком случае значением текущего приоритета системы было бы не одно число, а пара чисел - первое число будет приоритетом ввода, второе - приоритетом вывода. В системе СМОТЛ это не реализовано, однако имеется возможность одной управляющей картой заказа отключить полностью выполнение или всех макрокоманд ввода или всех макрокоманд вывода.

5. Результаты

Разработка описанного выше отладочного блока потребовала примерно 4 человеко-месяца. Но его использование, как оказалось позже, сэкономило, по нашим расчетам, по крайней мере, 3 человеко-года. Дело в том, что разработка и отладка всех РБ проводилась параллельно. Комплексную отладку удалось начать уже через 3 месяца с начала разработки рабочих блоков. Кроме этого, один из РБ не удавалось отладить в течение полугода после того, как все остальные РБ уже функционировали. Несмотря на это, система отладки давала возможность имитировать работу этого РБ, а остальные блоки выполнять в реальном режиме.

Оставленные в РБ макрокоманды отладки позволяют эффективно сопровождать систему.

6. Применимость подхода

Большинство АСУ, транслирующих систем и т.п. имеют свои управляющие блоки и языки управления системой, наличие которых существенно для реализации описанного метода отладки. Следовательно, изменив макрокоманды отладки с учетом специфики обрабатываемой информации, рассмотренный подход вполне применим в более широких масштабах. Свидетельство тому - успешное его использование при разработке системы диалоговой отладки ПЛ/I-программ и системы AUCX в ВЦ ЛГУ им.П.Стучки в 1977-1979 гг.

ЛИТЕРАТУРА

1. Bičevskis J., Borzovs J., Straujums U., Zariņš A., Miller E.F. Jr. SMOTL - A System to Construct Samples for Data Processing Program Debugging. - IEEE Transactions on Software Engineering, 1979, vol.SE-5, No.1, p.60-66.
2. Miller E.F., Lindamood G.E. Structured Programming: Top-down Approach. - Datamation, 1973, vol.19, No.12, p.55-57.

ДИАЛОГОВАЯ СИСТЕМА ПЛ/I-ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ИНСТРУМЕНТАЦИИ ОТЛАЖИВАЕМОЙ ПРОГРАММЫ

Ю.В.Борзов, Р.В.Звирбуле[x], И.Т.Розенштейне, В.В.Сестуле

ВЦ ЛГУ им.П.Стучки

I. Системы диалоговой отладки ПЛ/I-программ.

В течение последних десяти лет значительную популярность получил язык программирования ПЛ/I. Он нашел широкое применение в программировании задач вычислительного характера и задач обработки данных.

Приведем краткий обзор [I] имеющихся диалоговых систем (ДС) для операционной системы OS/360 (соответственно ОС ЕС) конфигурации MVT или MFT с подзадачами, которые позволяют вводить с терминала в вычислительную машину программы на языке ПЛ/I и попереторно выполнять их в диалоговом режиме.

Наиболее известными ДС являются CRJE (соответственно ДУВЗ), ITF (ДИСП), CPS и CHECKOUT.

CRJE обеспечивает ввод текста ПЛ/I-программы и дает возможность для его предоставления на трансляцию обычному компилятору пакетного режима. Выполнение ПЛ/I-программ в диалоговом режиме CRJE не обеспечивает.

ITF и CPS обеспечивают диалоговый режим отладки и выполнения ПЛ/I-программ, однако только для очень ограниченного подмножества языка ПЛ/I, ориентированного на задачи вычислительного характера. Притом это подмножество не согласовано с полным языком ПЛ/I.

Наиболее совершенной системой диалоговой отладки ПЛ/I-программ является TSO с компилятором CHECKOUT. Она обеспечивает выполнение и отладку программ на полном языке ПЛ/I, а компилятор CHECKOUT согласован с оптимизирующим компилятором. С другой стороны, для работы TSO с CHECKOUT необходима оперативная память не менее 512К байт.

---

x) Р.В.Звирбуле в настоящее время работает в Рижском медицинском институте.

Таким образом, ясно, что для малых и средних машин ЕС ЭВМ диалоговых систем отладки ПЛ/I-программ практически не существует.

2. Инструментация программ.

В последние годы разработан ряд автоматических систем отладки программ на основе инструментации [2] отлаживаемой программы. Инструментация представляет собой (обычно автоматическую) вставку в текст отлаживаемой программы обращений к модулям отладки. Эти модули отладки используются для накопления информации о работе отлаживаемой программы, как то: число выполнения каждого оператора, значения заданных переменных, последовательность номеров выполненных операторов, выполнение определенных соотношений между значениями переменных в заданных точках программы, некоторые временные характеристики, и т.п. Во всех этих системах имеется возможность в конце сеанса отладки получить сводные данные об упомянутой информации.

Ни одна из рассматриваемых здесь систем не обрабатывает ПЛ/I-программы. Им также не доступен диалоговый режим выполнения программы.

3. Основная идея.

В текст отлаживаемой ПЛ/I-программы автоматически вставляются обращения к отлаживающему модулю, который связывает отлаживаемую программу с терминалом и, таким образом, обеспечивает диалоговый режим выполнения.

4. Возможности для пользователя.

Не представляет большой сложности реализовать следующие возможности для пользователя:

- передача управления на любой оператор отлаживаемой программы с последующим возвратом управления пользователю у терминала, если достигнут указанный пользователем опера-

тор или выполнено указанное пользователем число операторов; или же возникла авария;
- вывод на терминал текущего значения переменной;
- присвоение заданного пользователем с терминала значения;
- вывод на терминал трассы (последовательности номеров выполненных операторов) программы;
- проверка выполнения заданных пользователем отношений между значениями переменных;
- выдача профиля программы пользователя (списка числа выполнения каждого оператора) в конце сеанса отладки.

5. Инструментация ПЛ/I-программы для диалоговой отладки.

Перед каждым выполняемым оператором программы пользователя автоматически вставляется обращение к отлаживающему модулю и метка. Если, например, в исходной программе пользователя есть оператор A:B=C; с номером 5, проставляемым ПЛ/I-компилятором, тогда после инструментации вместо этого оператора будет проставлено

```
A : CALL  #B(5);  #(5) : B = C;
```

где #B - отлаживающий модуль. Параметр процедуры #B (в данном случае - 5) используется для того, чтобы отлаживающий модуль "знал" номер оператора, который будет выполнен следующим. Метка (в данном случае - #(5)) используется для того, чтобы имелась возможность передавать управление из отлаживающего модуля на любой оператор программы пользователя.

6. Отлаживающий модуль.

Отлаживающий модуль является внутренней процедурой инструментированной программы пользователя. Поэтому передачу управления из отлаживающего модуля в отлаживаемую программу можно реализовать оператором GO TO. Например, если пользователь желает передать управление на оператор с номером 8, тогда в отлаживающем модуле надо выполнить фрагмент:

```
. . .
I=8;
GO TO #(I);
. . .
```

Выполнение следующего оператора программы пользователя реализуется оператором RETURN в отлаживающем модуле.

Для того, чтобы узнать текущее значение переменной, используется оператор PUT STRING( ) DATA( );. Например, пользователь пожелал узнать значение переменной M3. После выполнения в отлаживающем модуле фрагмента

```
. . .
DCL M3 FIXED(3);
DCL A  CHAR(32Ø) VAR;
PUT STRING (A) DATA (M3);
. . .
```

в переменной A окажется, например, 'M3=222', после чего это значение выводится на терминал.

Для присвоения значения переменной используется оператор GET STRING( ) DATA( );. Например, пользователь желает задать значение переменной M3, равное 222. Для этого значение 'M3=222', полученное с терминала, заносится в переменную A, после чего выполняется фрагмент

```
. . .
DCL M3 FIXED(3);
DCL A  CHAR(32Ø) VAR;
GET STRING (A) DATA (M3);
. . .
```

После выполнения такого фрагмента значение переменной M3 будет равным 222.

Проверка соотношений между значениями переменных происходит при помощи выполнения в отлаживающем модуле соответствующих IF-операторов, а профиль и трасса программы накапливается в массивах, определенных в отлаживающем модуле.

7. Последовательность автоматических действий по отладке программы пользователя.

1) Инструментация программы пользователя.
2) Генерация текста отлаживающего модуля в зависимости от заказа пользователя и его присоединение к тексту инструментированной программы пользователя.
3) Компиляция объединенной программы обычным ПЛ/I-компилятором.
4) Редактирование связей объединенной программы. На этой стадии присоединяются модули, реализующие связь отлаживающего модуля с пользователем у терминала.
5) Выполнение полученного загрузочного модуля в диалоговом режиме.

Нетрудно заметить, что по существу диалоговым является только пятое действие.

8. Машинный эксперимент.

В 1978/79 гг. в Вычислительном центре Латвийского государственного университета им.П.Стучки под руководством первого автора была проведена машинная проверка описанного метода. Студентка 5-го курса Р.В.Звирбуле, студентки 4-го курса И.Т.Розенштейне и В.В.Сестуле разработали экспериментальную диалоговую систему отладки ПЛ/I-программ. Программирование системы было также проведено на ПЛ/I и потребовало около 3000 операторов. Получены обнадеживающие результаты. Скорость выполнения программы пользователя в диалоговом режиме достигала порядка 400 операторов ПЛ/I в секунду на ЭВМ ЕС-1022. Занимаемая оперативная память увеличивалась по сравнению с обычным выполнением программы пользователя в допустимых пределах – на 20К байт плюс 3К байт на каждые 100 операторов программы пользователя.

Развитие и доводка системы продолжается. Реализуются возможности проверки утверждений, получения трассы и профиля, не включенные в первую очередь разработки. Проектирует-

ся создание подсистемы диалогового обучения пользованию системой. Только после завершения этих работ и проведения экспериментов по программированию реальных задач с использованием данной системы авторы будут вправе дать окончательную оценку целесообразности ее практического применения.

Описанный метод организации диалогового выполнения ПЛ/I-программ предложен авторами. Однако авторы считают своим долгом отметить, что подобная идея была высказана самостоятельно М.И.Августоном еще в 1976 г.

ЛИТЕРАТУРА

I. Балодис Р.П. Интерактивные средства программирования на базе языка ПЛ/I.-Управляющие системы и машины, 1977, № 3, с.32-39.
2. Huang J.C. Program Instrumentation and Software Testing. Computer, vol.11, No.4, April 1978, p.25-31.

СПЕЦИАЛЬНЫЙ МЕТОД ДОСТУПА AUCE[x]

Я.Я.Бичевский, И.А.Виржбицка[xx], М.Я.Груниере

ВЦ ЛГУ им.П.Стучки

Введение

В настоящее время ЭВМ все шире применяются в типовой обработке больших информационных массивов данных. Наиболее развитая операционная система ОС ЕС позволяет образовать на магнитных дисках наборы данных последовательной, прямой, библиотечной и индексно-последовательной организации (см. [I],[2]). Выбор организации набора данных главным образом определяется необходимой последовательностью обработки данных, точнее, последовательностью обработки записей набора данных.

Если необходима обработка данных в одной определенной последовательности, или же последовательность обработки данных вообще не существенна, то для хранения таких данных обычно используется набор данных последовательной организации.

Если последовательность обработки данных заранее не планируется то для хранения таких данных удобно использовать набор данных прямой организации.

Если необходима обработка данных в определенной последовательности, не исключая возможности индивидуальной обработки отдельных записей, то самой подходящей является индексно-последовательная организация набора данных.

Специальный метод доступа AUCE по своим возможностям является расширением индексно-последовательного метода доступа. При работе с наборами данных индексно-последовательной организации пользователь имеет доступ к записям только по одному ключу, а AUCE дает последовательный и прямой доступ к записям по нескольким ключам. Это необходимо, напри-

---

x) AUCE (АУЦЕ) - река в Латвийской ССР.

xx) И.А.Виржбицка в настоящее время работает в Тресте энергетического строительства.

мер, для решения задач, требующих выдачи информации в различных разрезах.

I. Функциональные возможности

Метод доступа AUCE предполагает создание базы данных, в которую можно включить несколько файлов-массивов (FM). В файле-массиве объединяются данные пользователя, состоящие из записей одинаковой длины и структуры и одного логического назначения. Понятие "FM" для метода доступа AUCE соответствует понятию "набор данных" в ОС ЕС.

Для идентификации записей в рамках одного FM определяются одно или несколько ключевых полей. В качестве ключевого поля можно указать любую непрерывную часть записи. Ключ для выдачи записей формируется пользователем из любой последовательности ключевых полей. При создании нового FM, указываются все ключевые поля, а последовательность указания определяет главный ключ.

Пример 1.

Допустим, что создан FM, состоящий из пяти записей, для которого определены ключевые поля К1 и К2.

		К2		К1	
1.		34		1	
2.		35		1	
3.		36		0	
4.		35		0	
5.		33		1	

Пусть главным ключом является К1;К2. Дополнительно по своему усмотрению пользователь может задавать ключами либо К2;К1, либо К1, либо К2. Значения ключей для примера 1 показаны в следующей таблице.

№ записи \ Ключ	К1;К2	К2;К1	К1	К2
I.	I34	34I	I	34
2.	I35	35I	I	35
3.	036	360	0	36
4.	035	350	0	35
5.	I33	33I	I	33

Метод доступа AUCE позволяет проводить следующие действия.

I. Выдачу записей заданного FM по любому из ключей в последовательности возрастания значений ключа в указанном интервале изменения значений ключа, включающую:

I) выдачу одной записи,

2) выдачу всех записей.

2. Корректировку базы данных, включающую:

I) добавление нового FM в базу данных,

2) удаление существующего FM,

3) корректировку существующего FM.

Для примера I при запросе на выдачу всех записей данного FM в порядке возрастания значений ключа К1;К2, записи будут получены в последовательности 4,3,5,I,2; для ключа К2;К1 - в последовательности 5,I,4,2,3; для ключа К1 - в последовательности 3,4,I,2,5; для ключа К2 - в последовательности 5,I,2,4,3.

I.I. Выдача записей.

Запрос на выдачу записей заранее занесенного в базу данных FM осуществляется из ПЛ/I программы с помощью оператора CALL. Для подготовки FM к работе пользователь должен вызвать модуль AUCEOPN, указывая имя необходимого FM. Для освобождения оперативной памяти, занимаемой конкретным FM, с целью использования ее для другого FM вызывается модуль AUCEFRE, указывая имя освобождаемого FM. AUCEFRE выполняет функции "закрытия" FM.

Выдача очередной записи осуществляется вызовом модуля

AUCERDR с указанием "запроса", в котором представлены: имя FM, ключ и интервал изменения значений ключа. После работы AUCERDR выдается "ответ" о результате завершения работы AUCERDR, который может принимать значения "0", "I" или "2". В случае наличия записей, удовлетворяющих "запрос", выдается очередная запись. То есть: впервые обращаясь к AUCERDR с "запросом" на выдачу последовательности записей данного FM с указанным ключом, пользователь устанавливает "ответу" начальное значение "I", а в результате получает запись с наименьшим (из заданного интервала) значением ключа. "Ответ" получает значение "0". Второй раз обращаясь к AUCERDR с тем же "запросом", он получает запись со следующим по величине значением ключа. "Ответ" получает значение "0" и т.д. При попытке запросить выдачу очередной записи, если весь интервал значений заданного ключа исчерпан, "ответ" получает значение "I". То же самое происходит при попытке запросить выдачу записи из такого интервала значений ключа, который на самом деле не содержит ни одной записи. В случае некорректности "запроса" "ответ" получает значение "2".

Для "закрытия" тех FM, которые не "закрыты" при работе AUCE, пользователь в конце своей программы должен вызвать модуль AUCECLS.

Пример 2.

Допустим, что в базу данных занесены два FM: USPEHI## и АНКЕТА##.

FM по имени USPEHI## состоит из 30 байтовых записей, содержащих информацию об успеваемости студентов некоторого факультета. Для этого FM определены три ключевых поля: U1, U2, U3, сведения о которых представлены в таблице:

Имя ключевого поля	Расположение ключ.поля в записи		Назначение ключевого поля
	Начало(байт)	Конец(байт)	
U1	I	8	№ студ.билета
U2	I4	I4	курс
U3	20	20	успеваемость (+/-)

Главным ключом является U1;U2;U3.

FM по имени ANKETA## состоит из 60 байтовых записей, содержащих анкетные данные студентов. Для этого FM определены два ключевых поля: A1 и A2, сведения о которых представлены в следующей таблице:

Имя ключевого поля	Расположение ключ. поля в записи		Назначение ключевого поля
	Начало (байт)	Конец (байт)	
A1	1	8	№ студ.билета
A2	11	11	комсомольская принадлежность (+/-).

Главным ключом является A1;A2.

Задание: получить анкетные данные неуспевающих комсомольцев первых трех курсов данного факультета, сгруппированные по курсам и в пределах каждого курса - в порядке возрастания номеров студенческих билетов.

Для решения этого задания надо запросить выдачу тех записей FM USPEHI##, у которых U3 имеет значение "-", а U2 - значения "1","2" или "3". Просмотр записей необходимо проводить по ключу U3;U2;U1. Этим будет обеспечена выдача номеров студенческих билетов всех неуспевающих студентов первых трех курсов. В целях получения анкетных данных комсомольцев, для каждого из полученных номеров студенческих билетов необходимо запросить выдачу записи FM ANKETA##, если только ключевое поле A2 имеет значение "+". Ключом для этой цели можно выбирать или A1;A2, или же A2;A1. По соображениям быстродействия рекомендуется выбирать главный ключ (A1;A2).

ПЛ/I программа для решения этого задания будет следующей.

```
PROG: PROC OPTIONS(MAIN);
      DCL  ZAPROS1 CHAR(36)
      INIT('USPEHI##U3--U213U1@@@@@@@@99999999&&');
      DCL  ZAPIS1 CHAR(30);
```

```
      DCL OTVET1 CHAR(1) INIT('1');
      DCL 1ZAPROS2,
           2 IMJA#FM CHAR(8) INIT('ANKETA##'),
           2 A1 CHAR(2) INIT('A1'),
           2 ST#BIL#1 CHAR(8),
           2 ST#BIL#2 CHAR(8),
           2 A2 CHAR(2) INIT('A2'),
           2 PRINADL1 CHAR(1) INIT('+'),
           2 PRINADL2 CHAR(1) INIT('+'),
           2 KONEC CHAR(2) INIT('&&');
      DCL  ZAPIS2 CHAR(80);
      DCL  OTVET2 CHAR(1);
      CALL AUCEOPN(ZAPROS1);
      CALL AUCEOPN(ZAPROS2);
CIKL: CALL AUCERDR(ZAPROS1,ZAPIS1,OTVET1);
           IF OTVET='1' THEN GOTO VSE;
           IF OTVET1='2' THEN GOTO NEVERNO;
           OTVET2='1';
           ST#BIL#1 = SUBSTR(ZAPIS1,1,8);
           ST#BIL#2 = ST#BIL#1;
      CALL AUCERDR(ZAPROS2,ZAPIS2,OTVET2);
           IF OTVET2='1' THEN GOTO CIKL;
           IF OTVET2='2' THEN GOTO NEVERNO;
           Обработка переменной ZAPIS2;
           GOTO CIKL;
NEVERNO: сообщение о некорректности запроса;
VSE: CALL AUCECLS;
     END;
```

I.2. Корректировка.

При корректировке базы данных различаются три действия.

Добавление нового FM в базу данных. Указывается имя FM, длина записи и расположение всех ключевых полей; задаются записи, образующие FM. Перед вводом первого FM необходима специальная инициализация базы данных.

Удаление существующего FM. Указывается имя удаляемого FM.

Корректировка существующего FM, включающая:

1) добавление новых записей (указывается имя FM и задаются добавляемые записи);
2) удаление существующих записей (указывается имя FM и для каждой записи-значение главного ключа);
3) изменение существующей записи (указывается имя FM, значение главного ключа изменяемой записи и новая запись).

Корректировку базы данных проводит модуль AUCE#KOR при помощи специальных карт управления.

2. Алгоритмы

В основу метода доступа AUCE положено использование дерева распределения записей (RSK). RSK содержит информацию о соответствии между физическим размещением записей в базе данных и значениями заданного ключа.

Логически RSK представляет собой дерево, состоящее из вершин, которые размещены по уровням и соединены указателями. Каждая вершина содержит один символ ключа. Количество уровней определено количеством символов в ключе. Вершины первого уровня содержат первые символы ключа, вершины второго уровня - вторые символы ключа и т.д. Ветвь RSK соответствует определенному значению ключа и содержит по одной вершине из каждого уровня. К каждой ветви присоединяется адрес соответствующей записи. Ветви RSK располагаются в порядке возрастания значений ключа.

2.1. Построение RSK.

Построение RSK проводится одновременно с занесением в базу данных нового FM. Для этого необходима следующая информация:

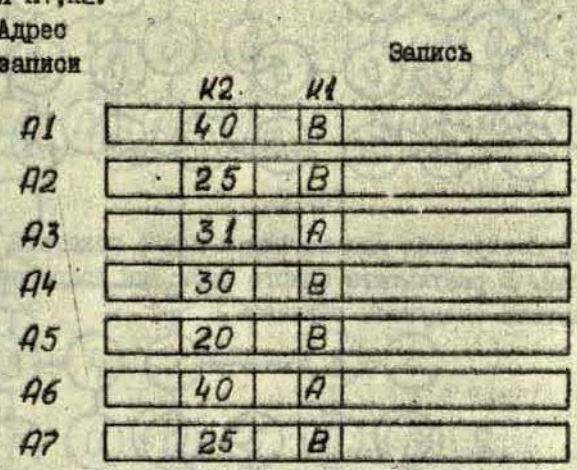
1) главный ключ,
2) значение главного ключа для каждой записи,
3) адреса записей.

В процессе построения RSK делается просмотр всех записей FM. Последовательность просмотра - произвольная (обычно она соответствует физической последовательности

записей). Для каждой рассматриваемой записи создается соответствующая ветвь RSK. Алгоритм построения RSK покажем на примере.

Пример 3.

Допустим, что имеется FM, который содержит семь записей и определены ключевые поля К1 и К2. Главным ключом является К1;К2.



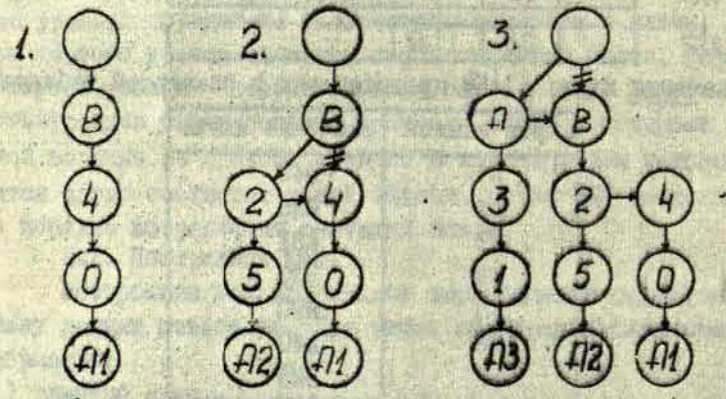
Значения ключа К1;К2 представлены в следующей таблице:

Адрес записи	Значение ключа
A1	B40
A2	B25
A3	A3I
A4	B30
A5	B20
A6	A40
A7	B25

Этому ключу соответствует RSK (рис.1).

Рис.I.

Процесс построения RSK наглядно можно показать по отдельным шагам. В результате одного шага RSK дополняется очередной ветвью следующим образом.



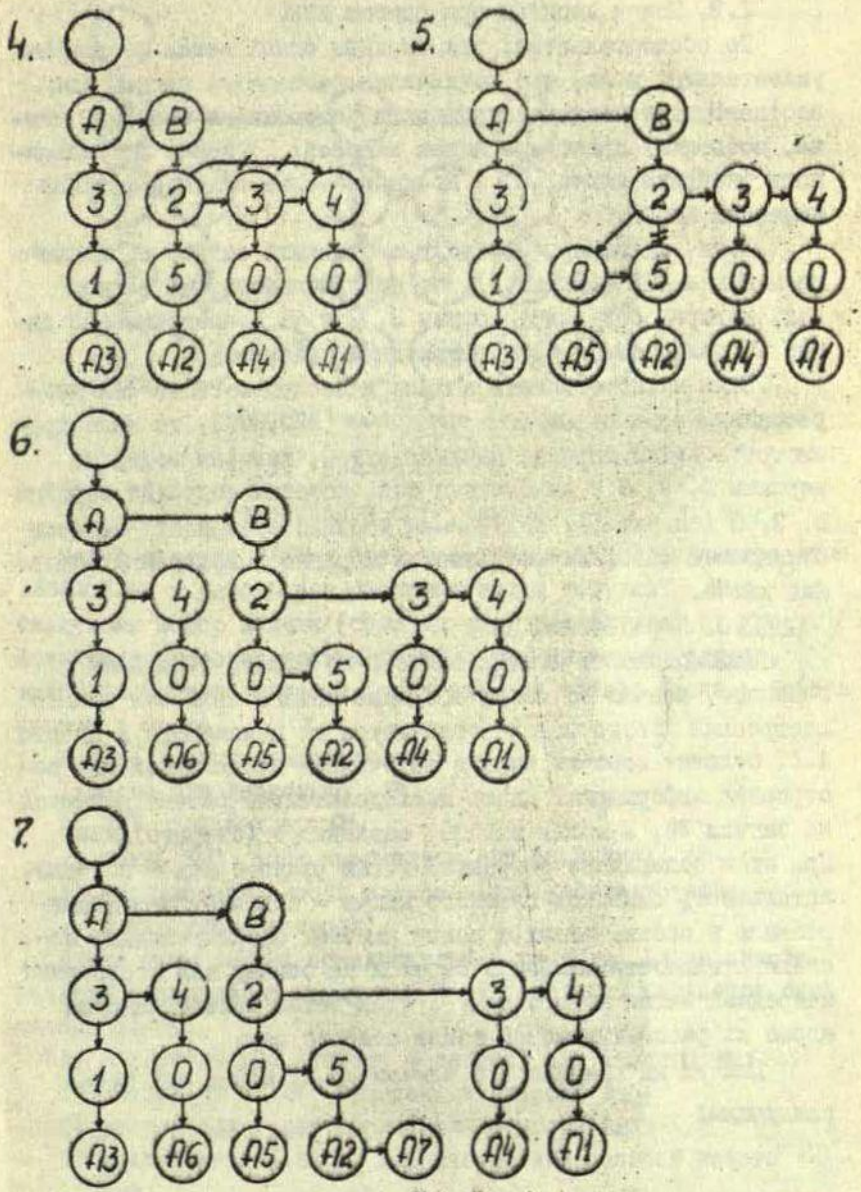

4.
A
B
3
2
3
4
1
5
0
0
A3
A2
A4
A1
5.
A
B
3
2
3
4
1
0
5
0
0
A3
A5
A2
A4
A1
6.
A
B
3
4
2
3
4
1
0
0
5
0
0
A3
A6
A5
A2
A4
A1
7.
A
B
3
4
2
3
4
1
0
0
5
0
0
A3
A6
A5
A2
A7
A4
A1

Рис. 2.

2.2. Поиск записей при помощи RSK.

То обстоятельство, что вершины одной ветви соединены указателями; и то, что логическое размещение ветвей RSK соответствует последовательности возрастания значений ключа, позволяет проводить поиск записей: 1) прямо по указанному значению ключа; 2) в последовательности возрастания значений ключа.

Если, например, необходимо отыскать запись со значением ключа B2O (см.рис.I), то надо пользоваться ветвью RSK, которая содержит вершины B, 2 и O. Присоединенный адрес A5 является адресом необходимой записи.

Если нужно отыскать записи в последовательности возрастания значений ключа в интервале [B2O,B3O], то надо просмотреть ветви дерева, начиная с той, которая содержит вершины B, 2, O и заканчивая той, которая содержит вершины B, 3, O (см.рис.I). Полученная последовательность адресов определяет необходимые записи в порядке возрастания значений ключа.

2.3. Перестройка RSK.

Необходимость выдачи записей по ключу, отличному от главного, влечет за собой создание нового RSK. Сам процесс построения нового RSK не отличается от описанного в пункте 2.I. Отличие состоит лишь в источниках необходимой для построения информации. Здесь последовательно рассматриваются не записи FM, а ветви заранее созданного (старого) RSK. При этом содержание очередной ветви старого RSK - последовательность символов главного ключа - подлежит переформированию в соответствии с новым ключом. Сформированная последовательность символов берется за основу для построения очередной ветви нового RSK. К этой ветви присоединяется адрес из рассматриваемой ветви старого RSK.

Для FM из примера 3 с ключом A2;A1 новое RSK будет следующим:

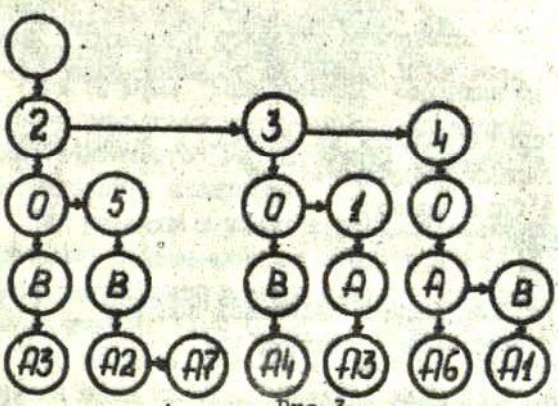


Рис.3.

3. Реализация

3.1. Организация базы данных.

Метод доступа AUCE реализован на языке программирования ASSEMBLER и использует базисный метод доступа BDAM. Физически набор данных (база данных) разделяется на блоки. Логически основной частью базы данных является запись, идентифицируемая адресом, состоящим из номера блока и номера записи в блоке.

В базе данных размещаются:

1) один или несколько FM;
2) RSK для каждого FM;
3) новый RSK, если необходимо его создание;
4) последовательность адресов (для удовлетворения конкретного запроса).

Блоки базы данных объединяются в разделы. Блоки одного раздела связываются между собой указателями. Структура базы данных такова.

I. В базе данных имеются 4 раздела для каждого FM:

1) раздел FM для размещения записей FM;
2) раздел RSK для размещения записей RSK;
3) раздел нового RSK для размещения записей нового RSK;
4) раздел последовательности адресов для размещения упомянутой в пункте 2.2 последовательности адресов.

2. Имеется раздел свободной памяти. В этом разделе объединены те блоки (за исключением нулевого и первого), которые неприсоединены ни к одному из вышеописанных разделов.

3. Раздел содержания расположен на нулевом и первом блоках. Раздел содержит следующее.

I) информацию о базе данных:
   а) длину блока,
   б) число FM,
   в) указать начала раздела свободной памяти;

2) информацию о каждом из отдельных FM:
   а) имя FM,
   б) длину записи,
   в) указатели начала разделов, принадлежащих этому FM,
   г) указатель свободной памяти в пределах FM,
   д) сведения о всех ключевых полях данного FM.

3.2. Виртуализация памяти.

За время работы AUCE записи базы данных запрашиваются неоднократно. В целях максимального использования оперативной памяти и минимальной загрузки дискового устройства применяется виртуализация оперативной памяти [3] следующим образом.

Свободная память, выделенная для задачи пользователя, разделяется на страницы для размещения (содержимого) блоков базы данных.

Чтение записи сводится к поиску местонахождения той страницы, которая содержит необходимый блок. Для ускорения поиска, страницы оперативной памяти объединяются в разделы таким же образом, как блоки базы данных.

I. Четыре раздела для каждого FM.

2. Раздел свободной оперативной памяти.

3. Раздел содержания.

Поиск ведется только по страницам необходимого раздела заданного FM. Если в этом разделе нет страницы, содержащей необходимый блок, то проводится чтение этого блока из базы данных в "заполняемую" страницу оперативной

памяти, после чего "заполняемая" страница присоединяется к необходимому разделу заданного FM.

При выборке "заполняемой" страницы применяется тактика, обеспечивающая максимальное выделение оперативной памяти для "активного" FM. Алгоритм выборки "заполняемой" страницы состоит в следующем.

При наличии страниц в разделе свободной оперативной памяти выбирается одна из этих страниц. Если раздел свободной оперативной памяти пустой, то рассматриваются разделы других FM.

При наличии страниц в любом из разделов других FM выбирается одна из этих страниц. Если все разделы других FM пусты, то рассматриваются другие разделы заданного FM.

При наличии страниц в других разделах заданного FM, выбирается одна из этих страниц. Если все рассмотренные разделы пустые, то выбирается одна из страниц данного раздела заданного FM.

Во время работы пользователя одновременно с несколькими FM происходит распределение оперативной памяти по мере "активности" каждого FM. В примере 3 FM USPEHI** получит больше оперативной памяти, чем "менее активный" FM с ANKETA**.

Заключение

Возможности, которые дает индексно-последовательный метод доступа, метод доступа AUCE реализует для записей с несколькими ключами. При этом AUCE обходится без физического перемещения записей, способствует полному использованию выделенной внешней памяти (за счет учета свободной дисковой памяти), минимальной нагрузке дискового устройства. AUCE максимально использует оперативную память, выделенную для задачи пользователя. Предполагаемый круг применения метода доступа AUCE весьма широк: обработка информации анкетного типа, математическое обеспечение АСУ и др.

Система AUCE разработана в 1979 году в Вычислитель-

ном центре Латвийского государственного университета им.П. Стучки. В настоящее время проводится опытная эксплуатация метода доступа AUCE.

ЛИТЕРАТУРА

I. Единая система электронных вычислительных машин. Операционная система. Управление данными. Руководство программиста Ц51.804.002 Д4, 1979.

2. Система ИБМ/360. Введение в запоминающие устройства прямого доступа и методы организации данных, М., Статистика , 1974.

3. Принципы работы системы IBM/360.(Перевод с английского). /Под редакцией Л.Д.Райкова, М., Мир , 1975.

ИСПОЛЬЗОВАНИЕ ИНДЕКСНО-СТРАНИЧНОГО МЕТОДА В ЗАДАЧАХ ОБРАБОТКИ ДАННЫХ

И.Г.Ермолаева, Л.Л.Федорова

ВЦ ЛГУ им.П.Стучки

Обычная проблема, сопутствующая задачам обработки данных, - это выбор метода организации данных, т.к. каждая задача предъявляет свои специфические требования к организации данных. Реально мы располагаем определенным набором методов, каждый из которых для определенной задачи является наиболее подходящим, т.е. удовлетворяет большему числу требований. Однако оставшиеся невыполненными требования могут быть настолько существенными, что ставят под вопрос возможность реализации задачи и делают необходимым создание надстройки. Примерами надстроек являются разработанные в ВЦ ЛГУ страничная организация памяти, индексно-страничная и т.п.

В задачах обработки данных, требующих к записям прямого и последовательного доступа, обычно используется индексно-последовательный метод организации данных. Но он имеет ряд недостатков. В частности, необходимы специальные процедуры создания файла, которые практически используются один раз, причем перед созданием файла информация должна быть отсортирована в порядке возрастания ключей; отсутствует возможность аннулирования записей и работы с записями переменного формата (имеется в виду операционная система ДОС), длина записи пользователя не может превышать емкость дорожки. Кроме того, использование индексно-последовательного метода требует частой реорганизации файла в связи с заполнением областей переполнения. Этот недостаток особенно ощутим, если файл не статичен и имеет большой объем.

Индексно-страничный метод (I-P) организации данных позволяет преодолеть названные недостатки. Предлагаемый метод является расширением системного индексно-последовательного метода. Сущность I-P метода состоит в том, что в одну запись индексно-последовательного файла помещаются

записи пользователя, ключи которых имеют общую часть, называемую внешним ключом. Остальная часть ключа называется внутренним ключом. Организованные таким образом записи индексно-последовательного файла называются физическими страницами. Записи, имеющие общий внешний ключ, образуют логическую страницу. В общем случае логическая страница - совокупность нескольких физических.

I-P метод работает с записями фиксированной и переменной длины. Можно работать с записями длиной порядка восьми дорожек (имеются в виду диски типа ЕС-5050). Ключи записей имеют фиксированную длину.

Индексно-страничный файл можно создавать двумя способами:

а) обычным, т.е. путем последовательного накопления предварительно отсортированных по ключам записей;

б) путем коррекции пустых страниц.

Последний способ дает возможность зарезервировать память на внешнем носителе и отодвинуть на неопределенный срок реорганизацию файла. При этом способе не требуется предварительной упорядоченности по ключам.

Программное обеспечение I-P метода есть совокупность процедур, позволяющих выполнять операции:

- создание файла двумя вышеназванными способами;
- чтение по ключу записи пользователя;
- последовательное чтение всего файла или отдельных его частей, начиная с ключа, указанного пользователем;
- коррекцию записей файла.

Метод может работать в различных режимах коррекции, а именно: без изменения длины записи пользователя и с изменением длины на большую или меньшую. Имеется возможность удалять и добавлять записи как на существующие логические страницы, так и на новые логические страницы.

Программное обеспечение метода настраивается по целому ряду параметров. Это позволяет получить наиболее оптимальный вариант метода для каждой конкретной задачи и использовать все (или некоторые) имеющиеся системные средства

оптимизации ввода-вывода для индекс о-последовательного файла.

I-P метод реализован в системе программирования ДОС ЕС версии 2.I и был широко использован при проектировании задач автоматизированной системы обработки информации по начислению и выплате пенсий и пособий в Латвийской ССР (АСОИ-ПЕНС).

Рассмотрим применение индексно-отраничного метода на двух конкретных задачах:

- ввод документов с помощью дисплеев;
- создание и ведение архивного выплатного файла.

Задача ввода документов заключается в реализации следующих функций:

- ввода и накопления документов в некотором файле;
- контроля и коррекции введенной информации.

Файл, полученный в результате накопления документов, должен иметь такую организацию, чтобы к записям был возможен прямой и последовательный доступ, так как контроль и коррекция информации выполняются как последовательно, так и выборочно и файл в дальнейшем используется в задачах, где необходимы указанные типы доступа к записям.

Системный метод организации данных, удовлетворяющий этим требованиям, - индексно-последовательный, но его применение в данной задаче неэффективно по следующим причинам:

- документы вводятся одновременно с нескольких дисплеев, следовательно, они неупорядочены;
- документы вводятся различными по величине порциями в течение нескольких дней, что в случае индексно-последовательного файла вызовет необходимость частой реорганизации файла.

Для индексно-отраничной организации названные причины существенного значения не имеют. Данный индексно-отраничный файл организуется таким образом, что в качестве внешнего ключа используется совокупность значений двух реквизитов: номера дня ввода и двух последних цифр регистрационного номера документа. Использование последних двух цифр ре-

гистрационного номера документа во внешнем ключе позволяет регулировать количество документов на логической странице. Внутренний ключ - регистрационный номер документа.

В начале каждого месяца на магнитном диске создается I-P файл с пустыми страницами для всего предполагаемого набора внешних ключей. Этот набор легко определить, если известны дни, когда будет производиться накопление документов.

В задаче создания и ведения архивного выплатного файла требуется постепенно накапливать большой объем информации. При этом поступающие записи имеют переменную длину, и в задачах, использующих этот файл, требуется последовательный и прямой доступ.

Для выбора метода организации данных был собран и тщательно проанализирован целый ряд статистических данных, на основании которых был сделан вывод о целесообразности применения индексно-страничного метода. Ключ записи в этом файле - номер пенсионного дела. При известной средней длине записи оптимальным было признано задать разбиение ключа на внешний и внутренний так, чтобы на логической странице в среднем находилось 28 пенсионных дел.

Использование I-P метода для создания и ведения архивного выплатного файла дает возможность:

1) путем выбора определенной длины физической страницы, способа деления ключа свести к минимуму потери памяти на внешнем носителе;

2) зарезервировать для файла необходимую область памяти на внешнем носителе;

3) избавиться от необходимости предварительной сортировки большого объема информации;

4) предельно упростить эксплуатацию.

ГЕНЕРАТОР ПЕЧАТИ ТАБУЛЯГРАММ

А.Р.Дзерве
ЛГУ им.П.Стучки

I. Введение

При проектировании различных АСУ, необходимо предусмотреть печать разных табуляграмм (контроль на ввод информации, отчетные табуляграммы и т.д.). Обычно реализация программ АСУ пишется на языке ПЛ/I, в котором предусмотрено большое количество средств обработки входящей информации, а также их печать. Но в то же самое время при составлении программ АСУ большое количество из всех операторов программы составляют операторы печати, операторы сравнения для выборки печатаемых реквизитов, обработка тех реквизитов, значения которых должны были выводиться на печать, но которые по тем или иным причинам оказались ошибочными (такие ситуации могут быть при печати контрольных табуляграмм). Это все вместе занимает значительную часть программы. При модификации табуляграммы возможны случаи, когда надо менять почти всю часть печати. Отметим, что многие табуляграммы по своей структуре отличаются между собой незначительно, а иногда тексты в шапках почти совпадают.

Автором предложенный генератор печати табуляграмм является препроцессором процедур печати табуляграмм на языке ПЛ/I. Результатом работы препроцессора являются процедуры двух видов. Процедуры первого вида предусматривают печать шапок табуляграмм, а процедуры второго вида предусматривают печать строк табуляграммы. Эти процедуры генерируются как исходные модули на языке ПЛ/I.

2. Генерация процедуры шапок

Язык генератора ориентирован на то, чтобы сгенерировав процедуры печати можно было легко модифицировать. Для этого описание шапок на языке генератора происходит не по строкам, а по графам и подграфам, а описание печатаемых реквизитов в строке табуляграммы происходит по графам. Подобное описание используется в [I]. Также как в языке, рассматриваемом в [I], в нашем случае каждая графа имеет свой номер уровня. Каждый уровень описывается на отдельных картах. Порядок следования таким образом описанных уровней идет сверху вниз и слева направо . Например, если шапка табуляграммы имеет структуру, показанную на рис.I, тогда уровни необходимо задавать в том порядке, как это показано номерами в шапке на рис.I.

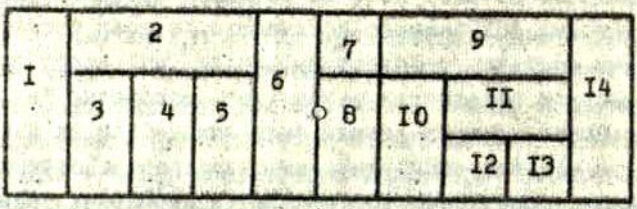


Рис.I. Шапка табуляграммы и порядок следования уровней.

Таким образом, все добавления, удаления и изменения граф и подграф происходят очень легко и удобно. Нет необходимости менять всю шапку, а только те графы, в которых появились изменения. Преимущество такого подхода также и в том, что те шапки, которые незначительно отличаются от уже описанной, легко генерируются при замене карт, которые описывают несовпадающие уровни. Описание шапки происходит в двух этапах. На первом этапе генератору в виде

значений параметров подаются сведения о генерируемой процедуре. Например, с помощью этих параметров можно указать вид процедуры (внутренняя или внешняя процедура ПЛ/1), символы, применяемые для образования вертикальных, горизонтальных (внешних и внутренних) разделителей граф шапки. Также можно указать генератору, необходимы ли в шапке внешние вертикальные разделители или нет. Пользователи могут также указать имя файла печати шапки, команду *PUT* или *WRITE*, с помощью которой будет печататься шапка в генерируемой процедуре, а также имя этой процедуры. Единственный параметр, который необходимо задавать всегда - это высота шапки (остальные параметры имеют значения по умолчанию). Необходимость задания высоты шапки обусловлено тем, что образ печатаемой шапки генерируется в матрице.

На втором этапе описываются сами уровни, т.е., графы и подграфы. Уровни могут быть описаны как построчно, так и поколонно (т.е., текст в уровень заносится вертикально). Имеется возможность описать поля в шапке, значения которых меняются динамически. Описывая уровень, используем формальный язык БНФ. Фигурные скобки означают итерацию.

<уровень> := <глубина уровня> <символ разделитель> <операция> <символ разделитель> { <операция> <символ разделитель> } <символ разделитель>

<глубина уровня> := <целое число без знака>

<символ отделитель> := <символ>

<операция> := <шифр операции> <тело операции>

Под глубиной уровня здесь понимается вложенность подграфы в графу. Например, глубина уровней шапки, изображенной на рис.1, показана на рис.2.

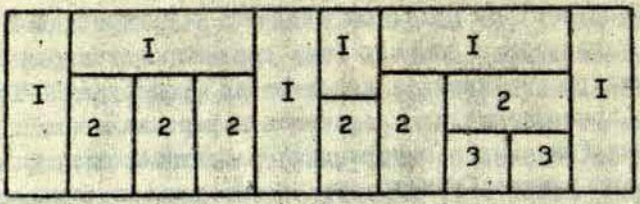


Рис.2. Глубина уровней шапки.

Тело операции - это часть текста уровня, которую нужно поместить в строку или столбец уровня, тело операции может также составлять информация о строке в уровне, значение которого меняется динамически. Шифр операции указывает генератору именно как часть текста (т.е., тело операции) должна поместиться в уровне (горизонтально, вертикально, динамически). Таким образом, одна операция полностью описывает одну строку или колонку. Генератор реализован таким образом, что те операции, которые описывают строки в уровне, размещаются друг за другом в порядке возрастания строк, а операции для описания колонок могут быть размещены независимо от порядка следования других операций в этом уровне. Это разрешается потому, что в шифре операции указывается номер колонки (считая с начала уровня), в которой должен разместиться текст, указанный в теле операции. Символ разделитель служит для отделения операций и также для указания конца описания уровня. Символ разделитель пользователь может указать с помощью параметра на первом этапе описания шапки. Если это не указано, тогда символ разделитель - '¤'.

Рассмотрим пример. Необходимо сгенерировать процедуру печати шапки следующего вида (рис.3).

Для этого необходимо описать шапку следующим образом:

1) ./ MAKE HEAD, H=16, VLIM='¦ ', LLIM='=';
2) ./ HLIM='-'
3) ./ NUMBER NEW1=100,INCR=100
4) I¤ НОМЕНКЛА¤␣ТУРНЫЙ¤␣№␣¤ НОМЕР¤␣№␣¤ТОВАРА¤¤

```
 5) I¤¤
 6) I ¤ [I3]␣¤␣ ДАТА¤␣¤ (Г.М.Д.)¤␣¤¤
 7) ./ OPCHAN VLIM=,*,
 8) ¤¤ ПРИХОД¤[,3] И ␣ ТОВАРА¤¤
 9) ./ OPCHAN VLIM='|'
10) 2¤ УБЫТИЕ¤ [3] И ␣ ТОВАРА¤¤
11) I ¤ [3,2] ␣ КОЛИЧЕСТВО¤¤
12) I ¤ [IØ] ␣¤␣¤␣␣ СУММА¤␣¤␣␣␣ПО¤[-
13)    ¤␣¤␣СКЛАДУ␣ NR¤␣¤[,,
14)    NRSKL P'ZZ9'
15) ¤¤
16) 2¤ [IØ] ШИФР¤␣␣ОРГАНИЗА¤␣␣ ЦИИ¤ ПОСТАВЩИКА¤¤
```

Первые три строки являются первым этапом описания шапки. Параметры H=I6 означает, что высота шапки равна I6; VLIM = ='|' означает, что вертикальный отделитель составляется из символов '|'; LLIM ='=' - символ внешнего горизонтального разделения, HLIM ='-' - символ внутренего горизонтального разделителя. Последние два параметра можно не указывать, так как их значения генерируются по умолчанию. В строке 3 генератору указано, что сгенерированную процедуру необходимо нумеровать. Строки 4-I6 (кроме строк 7 и 9) составляют второй этап описания уровней. Строки 7 и 9 являются дополнением к первому этапу, т.е., служат для изменения некоторых параметров. В примере это VLIM . Строка 4 описывает I уровень. Символ отделитель на I этапе не указан, в таком случае его значение равно '¤'. В первой операции шифр - пустой символ, а тело операции сразу начинается после символа разделителя и продолжается до следующего символа разделителя. Шифр операции - пустой символ - означает, что текст будет помещен в строке уровня. Так как это - первая операция, то в ней указывается длина уровня. В данном случае длина уровня равна числу символов в теле операции, т.е.. 9. 5 строка описывает пустой уровень, т.е., в шапке рядом проходят два разделителя уровней. В 6 строке в первой операций шифр указывает,

это текст необходимо поместить в строку и длина уровня равняется I3. (шифр операции [I3]). Остальные операции описывают последующие строки уровня. В 8 и I0 строках описаны подграфы. Первая операция размещает текст горизонтально, а вторая операция-вертикально, начиная с третьей колонки (шифр операции [, 3]).

В II строке графа описывается с помощью вертикальной операции с шифром [3,2] , где первое число - длина графы, а второе - номер колонки, в которой помещается текст из тела операции.

В I2 строке уровень описан почти так же, как, например, в 4 строке. Отличие только в том, что с помощью операции, шифр которой " [- " (тело операции пустое), можно разделить (перенести) операции на разных картах.

В I3 строке продолжается описание уровня, начатого в I2 строке. Новая операция, которая употреблена здесь - операция с шифром " [,,", указывает, что текущая строка будет описана динамически, т.е., ее значение может меняться при печати шапки в зависимости от значения реквизита NRSKL. (это показано в I4 строке). В I5 строке первый символ - разделитель - указывает на то, что переменные, описывающие эту строку, исчерпаны (вся строка уровня полностью описана), а второй символ разделитель указывает на конец уровня.

Имя генерируемой процедуры определяется по умолчанию, т.е., имя процедуры "GALVA" . Если пользователь желает другое имя, то на первом этапе описания шапки необходимо указывать параметр NAME =' < имя процедуры > '.

3. Генерация процедуры печати строки

При генерировании процедур печати строк, печатаемая строка составляется из значений печатаемых реквизитов и символов разделителей. Строка описывается по графам, т.е., каждая графа пишется на отдельных картах, а каждый реквизит (печатаемый в графе) - на отдельной карте. В проце-

дуре печати строки могут быть случаи ситуаций CONVERSION или даже ERROR при печати наборов данных. Генератором предусмотрено в генерируемой процедуре конструировать блоки обработки таких ситуаций. Также имеется возможность не печатать нулевые значения реквизитов и печатать реквизиты длиннее, чем длина графы, в которой должны располагаться их значения.

Так же, как и в случае шапки, описание строки происходит в двух этапах. На первом этапе можно указать имя процедуры, имя печатаемого файла, имя команды, вертикальный разделитель, обработку ситуаций *CONVERSION* и *ERROR*, отказ печати нулевых значений, а также нумерацию генерируемой процедуры.

На втором этапе описываются реквизиты по графам в строке. Формальное описание реквизита: < описание реквизита > = < идентификатор > [< формат > [< номер графы > [< этаж >]]]. Квадратные скобки в этом описании означают факультативную конструкцию.

< номер графы >: = < целое число без знака >

< номер этажа >: = < целое число без знака >

Идентификатор - это обозначение переменной, значение которой необходимо печатать. Формат указывает, каким образом необходимо печатать значение переменной. Номер графы обозначает графу, в которой необходимо печатать значение переменной (графы нумеруются слева направо). Принято, что, если номер графы не указан, тогда заполняется следующая графа строки. Процедура печати строки вызывается для каждой строки табуляграммы. Но бывают случаи, когда печатаемая запись длиннее, чем ширина табуляграммы. Тогда каждую запись приходится делить по двум или нескольким строкам. Такие строки назовем этажами. Генератор реализован таким образом, что можно указать не только номер графы, но и номер этажа печатаемых реквизитов.

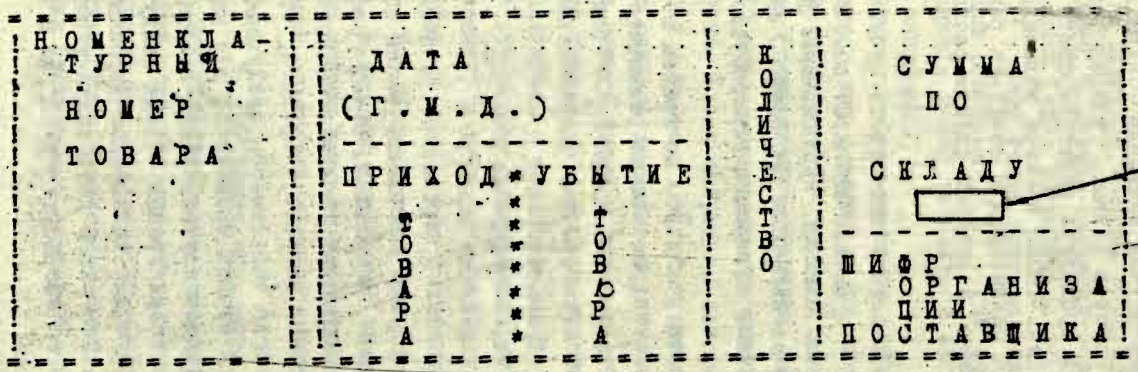


Рис.3. Шапка.

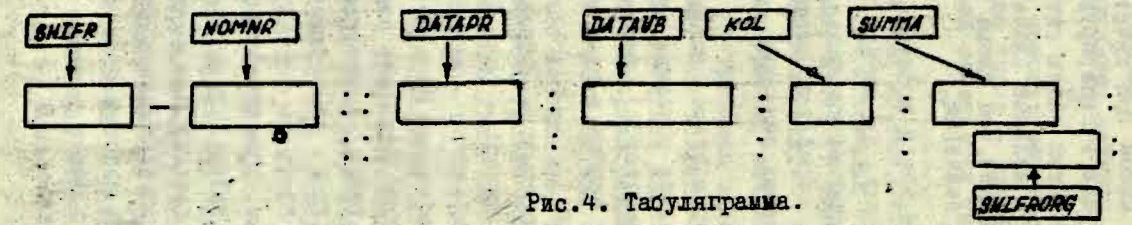


Рис.4. Табуляграмма.

Названия в прямоугольниках обозначают идентификаторы; пустые прямоугольники обозначают месторасположение значений идентификаторов, от которых идут стрелки к указанным прямоугольникам.

Рассмотрим пример. Необходимо генерировать процедуру печати строки с именем STROKA , именем выводного файла PECHSTR , командой печати WRITE , предвидеть обработать ситуацию CONVERSION и для некоторых реквизитов не печатать нулевые значения, а также некоторым реквизитам предусмотреть обработки ситуации ERROR . Шапка табуляграммы изображена на рис.3.

Расположение печатаемых значений реквизитов и идентификаторы реквизитов изображены на рис.4. Для генерации такой процедуры, необходимо строку описать следующим образом:

```
 1) ./ MAKE LINE, NAME='STROKA', FNAME='PECHSTR',
 2) ./ CONV='Y ', PRINT=' W'
 3)    SHIFR F(3)
 4)    ' - ', 1
 5) ./ OPCHAN CONT=' Y ', NUL='N'
 6)    NOMNR P'(5)9', 1
 7) ./ OPCHAN CONT='N ',
 8)    DATAPR A,3
 9)    DATAUB
10)    KOL P'ZZ9'
11)    SUMMA F(7,2)
12)    SHIFRORG (X(3),A),6,2
```

Строки 1-2 являются первым этапом описания строки табуляграммы. Параметр CONV = '4' означает, что необходимо предусмотреть обработку ситуации CONVERSION ; параметр PRINT = ' W ' означает, что команда печати будет WRITE
Строки 3-12, кроме 5 и 7, являются вторым этапом описания процедуры. Строки 5 и 7 (подобно случаю шапки) являются добавлением к первому этапу. Параметр CONT='Y' означает - предвидеть обработку ситуаций ERROR , а NUL='Ч ' - не печатать нулевые значения.
Значение параметра CONT='N' противоположное значение CONT='Y'

В строке 3 "SHIFR" - это идентификатор, "F(3)" - формат печати. В строке помещается литерал '-' в первой графе, т.е., в той же графе, где и значение переменной SHIFR. Таким же образом в строке 4 описанное значение переменной NOMNR с форматом F'(5)9' помещается в первую графу. Так, как 2-ая графа пустая, то описанный реквизит в строке 8 имеет номер графы 3. Переменная DATAUB помещается в графу 4 полностью. Переменная с идентификатором SHIFRON6 помещается в графу 6 на втором этаже, при этом первые три позиции в графе пропускаются и значение отводятся оставшиеся позиции до конца графы.

Генератор обеспечивает полный контроль синтаксических ошибок. После нахождения ошибки, печатается сообщение. Генератор продолжает синтаксический анализ последующих операторов. Не исключается случай, когда после найденной ошибки, генератор может опознать ошибочными те операторы, которые написаны правильно. Генератор реализован на языке ПЛ/I в системе ОС ЕС. Общее количество операторов 2200. Генератор состоит из 3 процедур: управляющая, процедура генерации печати шапки, процедура генерации печати строки. Анализ операторов и генерация процедур происходит за один проход.

Сгенерированные процедуры генератор помещает в задаваемые пользователем наборы данных (один набор данных для шапок, другой - для строк). Компилируя вызывающую процедуру, пользователь может эти процедуры включить в исходный текст оператором % INCLUDE или компилировать их отдельно. В последнем случае эти процедуры с вызывающей процедурой объединяются редактором связей.

Литература

I. Безруков Н.Н. Генерация программы печати шапки документа по описанию структуры шапки. - Программирование, 1979, № 6, с. 92-95.

ПОДМНОЖЕСТВО ТАБЛИЧНОГО ЯЗЫКА ЗАПИСИ АЛГОРИТМОВ "TABELE"

А.Я.Абеле

ЛГУ им. П.Стучки

I. Назначение.

Одним из недостатков использования известных алгоритмических языков при разработке автоматизированных систем обработки данных является разрыв между работниками, занятыми в сфере обработки данных, и программистами, т.е. отсутствие доступного обоим формализованного языка общения, что приводит к программированию ошибочных алгоритмов. Другим, не менее важным, недостатком является трудность модификации программ, необходимость в которой постоянно возникает в связи с частыми изменениями в алгоритмах обработки данных вследствие законодательной деятельности общества. Часто программы морально стареют до их завершения.

Табличные языки являются теми удобными средствами, которые устраняют отмеченные недостатки, т.к. они легко доступны специалистам любой профессии и позволяют легко модифицировать описанные ими алгоритмы. А для программирования самих законодательных актов табличный язык является единственным подходящим средством.

Табличный язык может использоваться отдельно - для связи между работниками, занятыми в сфере обработки данных, и программистами, и, совместно с соответствующим транслятором, - для непосредственного получения рабочих программ. Так как алгоритм на табличном языке легко описывает задачу, указывая, какие действия при выполнении каких условий производятся, то он одновременно является и блок-схемой и описанием программы.

В статье предлагается подмножество табличного языка записи алгоритмов " TABELE ", которое может использоваться как вспомогательный формализованный язык передачи алгоритмов обработки данных программисту. Средства подмножества позволяют описывать несложную, на уровне элементар-

ных полей, внутреннюю обработку данных.

Полный язык "TABELE":

- кроме того, позволяет использовать сложные поля и структуры, операции редактирования, операции ввода-вывода и другие средства, являясь, таким образом, самостоятельным языком описания алгоритмов обработки данных;
- в случае реализации на ЕС ЭВМ транслятора - интепрета - тора, представит удобное средство программирования часто меняющихся алгоритмов, так как модификация програмы сводится к простой замене хранящихся в ЭВМ таблиц языка " TABELE ", содержащих описание алгоритма обработки.

2. Основные элементы и структура логической таблицы.

Условие - это отношение двух определенных частей информации (величин). Условие содержит по крайней мере одну переменную величину и, в зависимости от текущего значения этой переменной, в момент ее проверки может выполниться или не выполниться.

Действие - это команда, которая выполняет операции (арифметические, пересылки и др.) над данными или управляет последовательностью просмотра таблиц.

Правило - это список условий, сцепленный со списком действий. Последний должен быть выполнен, если выполняется первый.

Части логической таблица (рис.I,4) имеют следующие назначения.

Заголовок таблицы	Заголовок правил
Заголовок начала	Заголовок продолжения
Начало условий	Продолжение условий
Начало действий	Продолжение действий

Рис.I. Части логической таблицы.

Заголовок таблицы: идентифицирует таблицу кодом названия таблицы в графе КОД, указывает дату создания графой ДАТ и версию таблицы - графой В, содержит графы СЛЕД и ИНАЧЕ.

Заголовок правил идентифицирует каждое правило кодом его названия в графе П.

Заголовок начала оглавляет столбцы (СТ- строка условии или действия, Д - код действия, I - первый операнд, * - код операции отношения или действия, 2 - второй операнд) для полных условий и действий или для начала условий и начала действий.

Заголовок продолжения оглавляет столбцы по отдельным правилам для связок или для продолжения условий и действий.

В части "начало и продолжение условий" помещаются условные выражения. В части "начало и продолжение действий" помещаются действия.

3. Форматы данных и операндов.

Логические таблицы используют два типа данных - знаковых и числовых.

Знаковое данное - последовательность знаков переменной длины в коде ДКОИ.

Числовое данное - целое десятичное число переменной длины со знаком.

Для записи условий и действий используются следующие форматы операндов.

Формат I. Код таблицы. Имеет длину от 3 до 8 байтов. Начинается символом & ; остальные символы кодирует название таблицы, при этом второй символ обозначает отдельную систему таблиц и не должен меняться в пределах этой системы. Может записываться только в столбцах 2.

Например: &ЗНАДБ2 - вторая таблица НАДБАВКИ системы 3.

Формат 2. Код величины. Имеет длину от I до 8 байтов и начинается с буквы. Фактически это символический адрес

поля, отведенного для этой (постоянной или переменной) величины. Может записываться в столбцах I и 2.

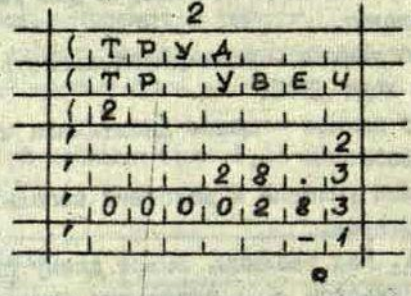
Например: ГРУППНВ - это адрес поля для величины "ГРУППА ИНВАЛИДНОСТИ".

Формат 3. Непосредственная константа. Это есть значение константы, записанной непосредственно в поле операнда. Ее можно записывать только в столбцах 2. Согласно форма - там данных имеются непосредственная знаковая константа и непосредственная числовая константа.

Непосредственная знаковая константа, длиной не более 7 знаков, записывается, начиная со второго байта, и не должна иметь последним знаком пробел. В первом байте по - ля записывается знак (.

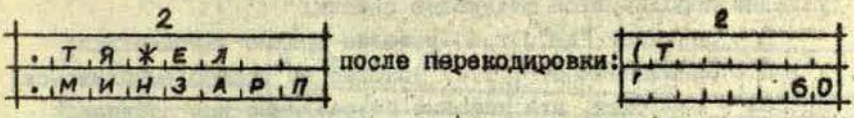
Непосредственная числовая константа, длиной не более 7 цифр, записывается так, чтобы младшая цифра находилась в последнем байте поля операнда. В первом байте заносится знак ' . Точка или запятая допускается, но не учитывается. Перед первой значащей цифрой отрицательного числа ставится знак - .

Например:



Формат 4. Непосредственная символьная константа. Это символическое имя, записанное по правилам знаковой кон - станты, но со знаком . в первом байте, непосредственной константы. Символическое имя в константу перекодируется таблицей кодирования символьных констант. Можно записы - вать только в столбцах 2.

Например:



4. Запись элементов логической таблицы.

Условие записывается в одной строке двумя величинами, соединенными операцией отношения. Код первой величины записывается в столбце I, а код второй величины или значение самой величины или код таблицы, содержащей выражение, - в столбце 2 начала или, при изменении второй величины от правила к правилу, продолжения. Код операций отношения записывается в столбце ж непосредственно перед вторым операндом. Используются шесть операций отношения:

символ	значение
=	равно
#	не равно
<	меньше
⌐	меньше или равно
>	больше
⌋	больше или равно

Строки условий нумеруются, начиная с 0I.

Запись действия занимает одну строку. Начинается она кодом действия и включает один или два операнда. В последнем случае операнды соединяются кодом операции, расположенным непосредственно перед вторым операндом. Если второй операнд изменяется от правила к правилу, он может быть помещен в продолжении таблицы. Строки действий нумеруются, начиная с I.

Правило записывается в продолжении таблицы в виде столбца связок или вторых операндов условий и действий.

Правило в заголовке правил идентифицируется кодом. Для условий используются следующие связки:

Д - означает "да", т.е. условие должно выполняться;

Н - означает "нет", т.е. условие не должно выполняться;

пробел - означает, что условие безразлично для правила.

Для действий связки:

Х - означает, что действие выполняется;

пробел - что действие не выполняется.

Нахождение второго операнда непосредственно в столбце правила для условия означает Д, для действия - Х.

5, Условные выражения.

Условное выражение состоит из условий. В зависимости от выполнения условий, условное выражение может выполняться или не выполняться. Различаются три типа условных выражений:

1) простое условное выражение - состоит из одного условия. Например (рис.4): строка 01 таблицы &1 РЕГПЕР, которое читается: СВЕТ=.КРАСНЫЙ.

2) Составное условное выражение И - простые условные выражения, соединенные с помощью "И".

Например (рис.4): Правило 1 таблицы &1 РЕГПЕР, которое читается: Если СВЕТ=.КРАСНЫЙ И ДВИЖЕНИЕ=.ЕСТЬ,то ....

3) Составное условное выражение ИЛИ - условные выражения типа 1 и (или) 2, соединенные с помощью "ИЛИ". Они представляют собой несколько правил одной и той же таблицы с одними и теми же действиями.

Например (рис.4): правила 1,2 и 3 таблицы &1 РЕГПЕР, которые читаются: Если (СВЕТ=.КРАСНЫЙ И ДВИЖЕНИЕ=.ЕСТЬ) ИЛИ (СВЕТ=.ЖЕЛТЫЙ И ДВИЖЕНИЕ=.ЕСТЬ) ИЛИ (СВЕТ=.ЗЕЛЕНЫЙ И ДВИЖЕНИЕ=.НЕТ); то перейти к таблице &1 ШТРАФ.

6. Действия.

Д	I	М	*	2	Выполняемое действие
ИДТИ				Код таблицы	Безусловный переход к открытой таблице
ВЫПОЛ				Код таблицы	Переход к закрытой таблице с возвратом на следующее действие
СТОП					Выход из системы таблиц
ДАТЬ	Поле откуда		В + - / *	Поле куда	В, $C_т$ пересылка элементарных полей + - сложение - - вычитание / - деление * - умножение Ф - при каждом обращении к закрытой таблице, содержащей только действия, вычисляется ее выражение (формула) и результат помещается в поле куда.
ВЗЯТЬ	Поле куда		С + - / *	Поле откуда Непосред. константа	
			Ф	Код таблицы	
ОКРУГ			n	Адрес поля	Отбрасываются "n" разрядов, с предварительным добавлением 5 к старшему отбрасываемому разряду
ОСТАТ			В	Поле куда	Остаток сразу после деления помещается в поле куда.
ФУНК	ДАТА ВРЕМЯ		В	Поле куда	ДАТА - текущая дата в виде ДД/ММ/ГГ помещается в восьмибайтовое поле куда. ВРЕМЯ - в виде десятичного числа ЧЧММСС помещается в трехбайтовое поле куда.

7. Последовательность просмотра логических таблиц.

Система таблиц, описывающая независимый алгоритм, представляется как набор таблиц со входом в начальную таблицу. Передачи управления (переходы от одной таблицы к другой) всегда происходят к таблице как к целому, а не как к правилу или условию, поэтому логическая таблица всегда рассматривается как целая, что относится и к коррекции системы таблиц.

При одном входе в таблицу может выполняться только один список условий (последовательность которых несущественна) для одного правила, поэтому может быть выполнен только один список действий, соответствующих этому правилу. Действия исполняются в той последовательности, в которой они написаны. Последним действием в правиле обычно должно быть ИДТИ, для перехода к следующей таблице, или СТОП, для выхода из системы таблиц. Если все или большинство правил требуют перехода к одной и той же таблице, то можно для этих правил, вместо ИДТИ для каждого, записать код следующей таблицы в графе СЛЕД заголовка.

Правила в таблице независимы, т.е. не зависят друг от друга. Просмотр правил в таблице производится слева направо до первой выполнившейся. Оставшиеся правила не просматриваются. Если ни одна команда текущей таблицы не была выполнена, т.е. не выполнилось ни одно правило, то следующей должна просматриваться таблица, код которой указан в графе ИНАЧЕ заголовка.

8. Типы таблиц.

" TABELE " использует два типа таблиц - таблиц определения и логических таблиц.

I) Таблицы определения. Для количественной оценки информационной емкости и для перехода на автоматическую обработку закодированного алгоритма ряд таблиц системы используется для определения полей и констант, а также для кодирования непосредственных символьных констант.

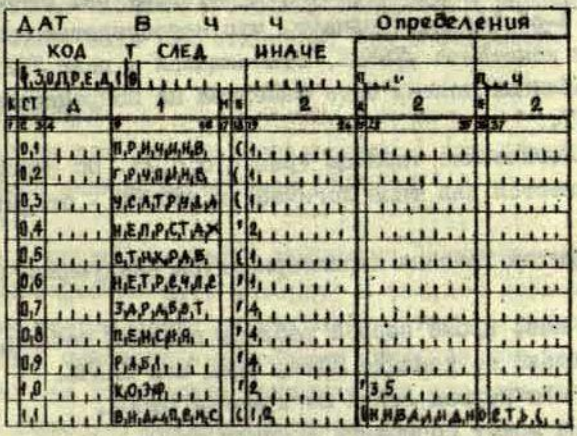


Рис.2. Таблица определения.

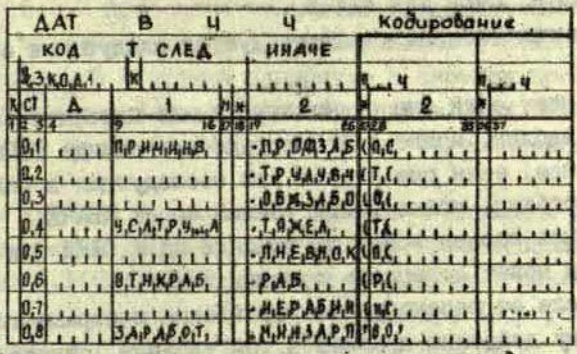


Рис.3. Таблица кодирования.

а) Определение полей и констант (рис.2):
Запись 10-ой строки означает, что в двухбайтовом поле под именем КОЭФ находится число 35.

б) Кодирование непосредственных символьных констант (рис.3).

Непосредственная символьная константа в таблице кодирования связывается с полем, к которому она относится. Запись 4-ой строки означает, что непосредственная символьная константа ,ТЯЖЕЛ, относящаяся к полю УСЛТРУДА, во всех таблицах должна быть заменена на посредственную константу (Т.

2) Логические таблицы (рис.4,5,6,7). По способу обращения имеется два типа логических таблиц - открытая и закрытая.

Открытая таблица характеризуется следующими свойствами:

- В нее, кроме первой таблицы системы таблиц, можно войти только по команде ИДТИ, СЛЕД или ИНАЧЕ.
- В нее нельзя войти по команде ВЫПОЛ.
- Она может содержать команды ВЫПОЛ.
- Она указывает следующую рассматриваемую таблицу командой ИДТИ, СЛЕД или ИНАЧЕ.

Закрытая таблица характеризуется следующими свойствами:

- В нее, если она единственная или первая в закрытой цепочке таблиц, можно войти только по команде ВЫПОЛ.
- В нее, если она вторая или последующая в закрытой цепочке таблиц, можно войти только через ИНАЧЕ.
- В нее нельзя войти по команде ИДТИ, СЛЕД или ИНАЧЕ.
- Она может содержать команды ВЫПОЛ.
- После ее рассмотрения управление возвращается к следующему действию правила в той таблице, откуда было передано управление.
- Значение вычисленного выражения она передает через последнее "поле куда" выполненных действий.

По содержанию имеется три типа таблиц:

- Безусловная таблица – таблица содержит только действия (команды) и не имеет условий. Может быть как открытой, так и закрытой.

- Условная таблица - правила таблицы содержат только условия, с единственным действием ИДТИ. Может быть только открытой.

- Смешанная таблица - таблица содержит и условия, и действия. Может быть как открытой, так и закрытой.

9. Примеры таблиц.

Приводится два примера использования подмножества табличного языка записи алгоритмов "TABELE".

Пример I.

На рис.4 приведена часть алгоритма "Движение на регулируемом перекрестке" суть которого в том, что движение при красном или желтом свете равно как отсутствие движения при зеленом свете приводит к штрафу, а движение при зеленом свете равно как отсутствие движения при красном или желтом свете позволяет продолжать путь.

Пример 2.

На рис.5,6,7 приведена часть алгоритма начисления "Пенсии по инвалидности от профзаболевания", описание которого следует. В описании алгоритма в скобках приводятся использованные в таблицах коды параметров и их значений. Все параметры являются входными, кроме того, параметр ПЕНСИЯ является и выходным.

Пенсии (ПЕНСИЯ - выходная) рабочим и служащим по причине инвалидности (ПРИЧИНВ) вследствие профессонального заболевания (ПРОФЗАВ) начисляются в следующих размерах:

инвалидам I группы (ГРУППНВ) в размере II0 процентов,

инвалидам 2 группы - I00 процентов пенсии по старости (ПЕНСИЯ - входная), инвалидам 3 группы в размере 65 процентов с заработка (ЗАРАБОТ) до 40 рублей в месяц и сверх того, I0 процентов с остального заработка.

Рабочим и служащим, ставшим инвалидами вследствие

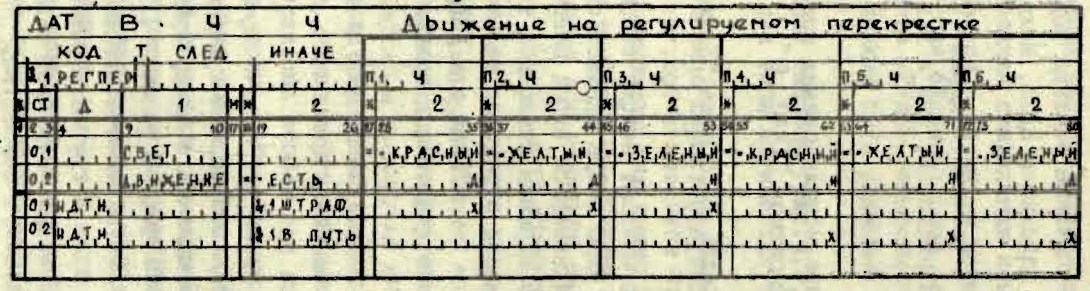


Рис.4. Логическая таблица алгоритма "Движение на регулируемом перекрестке".

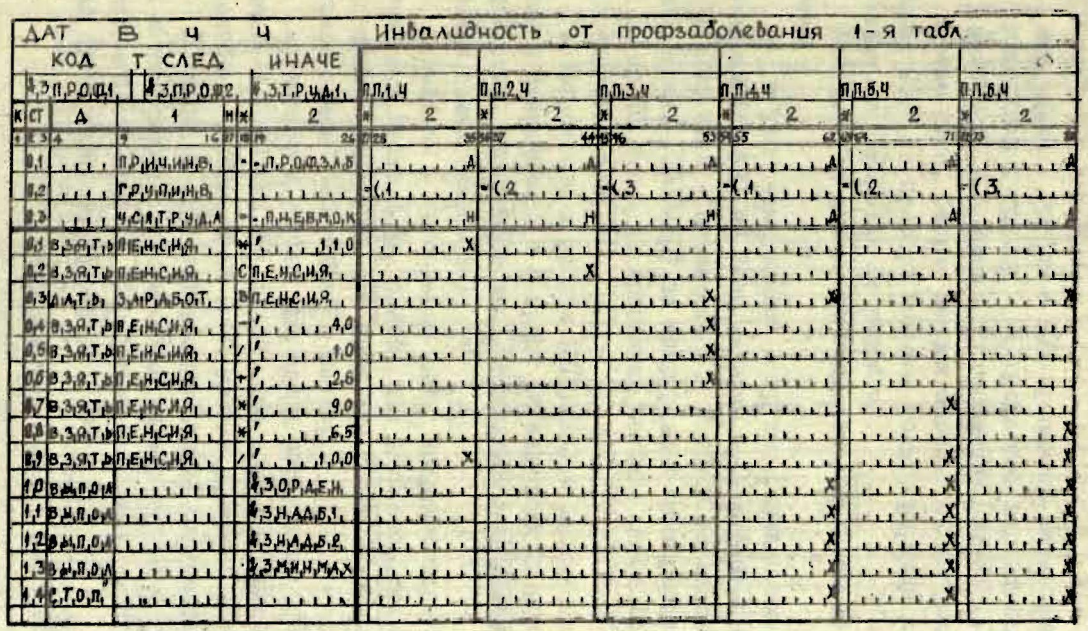


Рис.5. Логическая таблица (первая) алгоритма "Пенсия по инвалидности от профзаболевания".

Рис.6. Логическая таблица (вторая) алгоритма "Пенсия по инвалидности от профзаболевания.

Рис.7. Логическая таблица (третья) алгоритма "Пенсия по инвалидности от профзаболевания.

профессионального заболевания пневмокониозом (ПНЕВМОК), пенсии по инвалидности назначаются в следующих льготных размерах:

инвалидам I группы - 100 процентов заработка;

инвалидам 2 группы - 90 процентов заработка;

инвалидам 3 группы - 65 процентов заработка.

Кроме того, размер пенсии зависит от награждения орденами СССР (в примере показан только переход к закрытой таблице &ЗОРДЕН) и от надбавок.

Учитываются следующие надбавки к пенсиям по инвалидности:

а) на уход - инвалидам I группы (независимо от причины инвалидности) на уход за ними - 15 рублей в месяц;

б) за стаж - инвалидам I и 2 групп вследствие общего заболевания (ОБЩЗАБО) за непрерывный стаж работы (НЕПРСТАЖ): от 10 до 15 лет - 10 процентов пенсии; свыше 15 лет - 15 процентов пенсии;

в) отношение к работе (ОТНКРАБ) и иждивенцы - неработающим инвалидам I и 2 групп (независимо от причин инвалидности), имеющим на своем иждивении нетрудоспособных членов семьи (НЕТРСЧЛС):

инвалидам I группы при одном нетрудоспособном члене семьи - в размере 10 рублей, при двух нетрудоспособных членах семьи - 20 руб., при трех или более нетрудоспособных членах семьи - 30 рублей в месяц;

инвалидам 2 группы при одном нетрудоспособном члене семьи - в размере 10 рублей, при двух или более нетрудоспособных членах семьи - 20 рублей в месяц.

В конце вычисленный размер пенсий подлежит проверке на минимальный и максимальный размер пенсий (в примере показан только переход к закрытой таблице &ЗМИНМАХ, которая производит эту проверку).

ЛИТЕРАТУРА

Эванс И.О. Описание структуры решения задачи при помощи логических таблиц.- В кн.: Современное программирование. /Под ред. Задыхайло И.Б. М., Советское радио, 1967.

СТАНДАРТИЗАЦИЯ ПАРАМЕТРОВ ИНТЕРФЕЙСА ПРОГРАММ АСОД С БАЗОЙ ДАННЫХ, ОС И ОПЕРАТОРОМ ЭВМ

В.К.Пипир, М.Я.Янкевича
ВЦ ЛГУ им.П.Стучки

I.Введение

Специфика ПО (программного обеспечения) АСОД (автоматизированных систем обработки данных), особенно ПО для решения задач учетного характера, определяется тем, что оно состоит из большого числа автономно выполняемых программ (вызов посредством EXEC - предложения языка управления заданиями). Для решения некоторой задачи АСОД необходимо выполнить 5-10 только пользовательских программ. ПО подсистем АСОД состоит из 50-100 таких програмы. Кроме того, в АСОД интенсивно используются сервисные программы ОС (операционной системы).

Задачи АСОД учетного характера, как правило, эксплуатируются в пакетном режиме. Обычно в рамках одного пакета заданий осуществляется запуск всех программ, необходимых для решения некоторой задачи АСОД. Однако на стадии внедрения и в аварийных ситуациях на стадии эксплуатации АСОД, необходимо осуществлять пошаговый запуск программ (вызов в одном пакете заданий одной программы) и пошаговый анализ получаемых результатов.

В условиях большого числа пользовательских программ, интенсивного использования сервисных программ ОС (утилиты, программа СОРТИРОВКА-ОБЪЕДИНЕНИЕ) требуются серьезные усилия на разработку технологии эксплуатации ПО АСОД. Необходимым условием стандартизации и, тем самым, упрощения операций внедрения и эксплуатации ПО АСОД является наличие стандартного интерфейса у всех пользовательских прог-

рами с базой данных, ОС и оператором ЭВМ.

В настоящей работе основное внимание уделяется вопросам стандартизации интерфейса программ АСОД с базой данных, ОС и оператором ЭВМ в рамках операционной системы ОС ЕС и алгоритмического языка ПЛ-I.

Рассматривается пример процедуры языка управления заданиями для шагового вызова пользовательских программ в пакетном режиме.

2. Факторы, определяющие многочисленность программ в АСОД учетного характера

Большое число автономных программ, необходимых для решения даже простейших задач АСОД, объясняется многими факторами. Отметим лишь наиболее существенные из них в плане ввода-вывода информации.

Практически все программы АСОД осуществляют интенсивный обмен данными между оперативной памятью ЭВМ и машинными носителями информации. Для эффективного обмена данными с МД (магнитными дисками) и МЛ (магнитными лентами) и эффективного заполнения данных носителей информацией необходимы существенные ресурсы оперативной памяти ЭВМ. В данном случае имеется в виду использование двух-трех буферов ввода-вывода и размеров физических записей не менее 3К (К = 1024 байта). При использовании двух буферов ввода-вывода скорость обмена с магнитными носителями информации возрастает на 25-30% по сравнению с одним буфером ввода-вывода [1]. При использовании физических записей, превышающих 3К, существенно увеличивается информационная емкость МД и МЛ [2].

Языки программирования высокого уровня также требуют значительных ресурсов оперативной памяти для обеспечения операций ввода-вывода. Так, для эффективного обмена данными с магнитными носителями информации (использование двух буферов ввода-вывода и физических записей, равных 3К) средствами языка ПЛ-I требуется 8-10К оперативной памяти ЭВМ

для НД (наборов данных) с последовательной и региональной организациями и I5-20К для НД с индексно-последовательной организацией [3].

В то же время, размеры разделов (MFT) и зон (MVT) для решения задач АСОД нерационально, с точки зрения загрузки процессора ЭВМ, устанавливать более IØØ-I2ØК. Отсюда следует, что для эффективной загрузки процессора ЭВМ, эффективного обмена данными с внешними носителями информации, увеличения информационной емкости НД и НЛ, в рамках одной программы не следует допускать обработку более пяти-шести НД.

В задачах АСОД необходимы частые группировки, сортировки, объединения и т.п. операции над НД. Такие операции эффективней осуществлять автономными программами, которые, как правило, имеются в развитых ОС, чем реализовать их в подпрограммах пользовательских программ. В свою очередь, в АСОД, построенных с учетом принципов, изложенных в [4, 5], имеется достаточно большое число пользовательсктх программ, используемых при решении нескольких задач, и достаточно большое число подпрограмм, реализующих некоторые, типовые для данной АСОД, алгоритмы.

Перечисленное позволяет утверждать, что алгоритмы задач АСОД рациональнее реализовать большим числом относительно несложных (IØØ-5ØØ операторов алгоритмического языка высокого уровня) программ простой структуры, осуществляющих эффективный обмен данными с носителями информации и оперативной памятью ЭВМ.

3. Принципиальная блок-схема типичной для АСОД программы

На рис.I приведена принципиальная блок-схема некоторой, типичной для АСОД программы. Вопросы стандартизации параметров интерфейса программ с базой данных, ОС и оператором ЭВМ рассмотрим на основе данной блок-схемы.

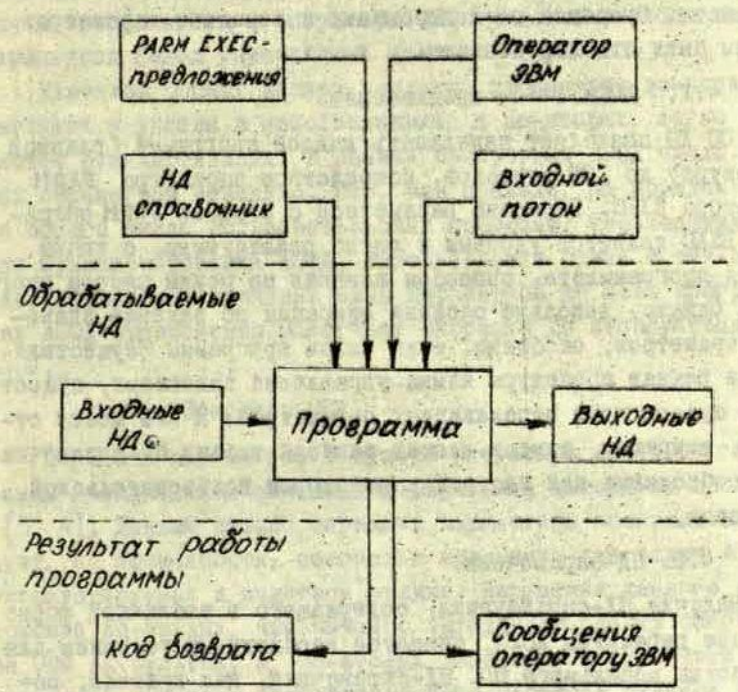


Рис.I. Принципиальная блок-схема программы.

4. Параметры режима работы программы

Алгоритмы разумно построенных программ должны быть способны адаптироваться к постоянно изменяющимся параметрам АСОД. Текущие значения изменяющихся параметров должны передаваться программам извне на начальном этапе их работы. Операции по кодированию и передаче параметров должны быть стандартизованы.

4.1. PARM &EXEC- предложения.

ОС ЕС позволяет передавать каждой программе (главной процедуре) до 100 символов, посредством параметра PARM оператора EXEC. Передача параметров с помощью PARM оператора EXEC является удобным и легко реализуемым, с точки зрения программиста, способом влияния на режим работы программ. Однако, довольно сложные операции по заданию значений параметров, особенно, если вызов программы осуществляется в рамках процедуры языка управления заданиями, существенно сужают круг передаваемых параметров. К ним можно отнести, например, список ключей записей таблиц НД-справочника, необходимых для настройки некоторой пользовательской программы.

4.2. НД-справочник.

Наличие НД-справочника, содержащего в табличной форме основные параметры АСОД, является необходимым условием для разработки мобильного ПО. НД-справочник, как правило, содержит редко изменяющиеся параметры АСОД (описания организационной структуры объектов автоматизации, структуры документации и т.п.). Каждая пользовательская программа, обращаясь к НД-справочнику и считывая необходимые записи, настраивается на текущие значения основных параметров АСОД. Такие параметры определяют некоторый глобальный режим работы программы.

4.3. Входной поток.

Для передачи часто изменяющихся параметров, определяющих одноразовый режим работы программы, необходимо использовать входной поток. ПЛ-I обладает удобными с точки зрения и программиста и оператора ЭВМ средствами ввода таких параметров (ввод управляемый данными - GET DATA).

Ключевой формат записи вводимых параметров во-первых, нагляден и удобен в использовании, а во-вторых, легко поддается стандартизации. В рамках конкретных АСОД число таких параметров невелико. Оно, как правило, на порядок ниже общего числа пользовательских программ, необходимых для функционирования АСОД. Использование одних и тех же идентификаторов ключевых слов параметров во всех программах АСОД существенно облегчает операции по эксплуатации АСОД.

4.4. Оператор ЭВМ.

Передачу параметров с пульта оператора можно считать самым неподходящим способом модификации алгоритма программ [3, 4]. Данный способ передачи параметров программ следует, по возможности, полностью исключить для задач АСОД, эксплуатируемых в пакетном режиме. Нарушение данного положения во-первых, чрезвычайно затрудняет работу оператора ЭВМ за пультом, а, во-вторых, приводит к случайным нарушениям нормального технологического процесса решения задач АСОД.

5. Идентификация DD-предложений.

Важнейшим аспектом стандартизации параметров интерфейса программ АСОД с базой данных, ОС и оператором ЭВМ является операция идентификации файлов и, тем более, DD-предложений. Идентификацию файлов программы и DD-предложений необходимо проводить уже на стадии технического проектирования АСОД. И если при идентификации файлов можно

допускать некоторую свободу в выборе имен, то при идентификации DD -предложений необходимо строго придерживаться единого подхода. Выигрыш от единой системы имен DD-предложений для всех пользовательских программ настолько очевиден при внедрении и эксплуатации АСОД, что остается только сожалеть, что вопросы идентификации DD -предложений, как правило, решаются на уровне программиста без четко установленных правил. Результатом такой практики является то, что для вызова каждой программы АСОД требуется подготовка DD -предложений с присущими только ей одной именами.

ОС ЕС в принципе позволяет осуществлять автономный вызов всех пользовательских программ с помощью одной процедуры языка управления заданиями, передавая ей при вызове имя программы и характеристики НД. Однако такой подход разумен только при единой системе имен DD -предложений и единых принципах кодирования операндной части DD-предложений. В противном случае составить новую процедуру языка управления заданиями проще, чем модифицировать или заново включить необходимые для данной программы DD -предложения. Принципы формирования процедур языка управления заданиями для автономного вызова программ и для решения задач АСОД рассматриваются ниже и в [6]. Однако уже теперь можно утверждать, что при разумном ограничении числа одновременно обрабатываемых НД в одной программе, при единой системе идентификации и единых принципах кодирования операндной части DD -предложений, ОС ЕС позволяет относительно небольшим числом процедур (5-10 на достаточно большую АСОД) языка управления заданиями осуществлять автономный вызов всех программ пользователя. Разработку таких процедур целесообразно осуществлять перед или параллельно разработке ПО. Автономная отладка программ на контрольных примерах должна осуществляться с использованием процедур эксплуатации.

6. Код возврата. Сообщения оператору.

Одним из важнейших параметров интерфейса любой программы в ОС ЕС является код возврата. Код возврата вырабатывается всеми обрабатывающими программами ОС ЕС и подпрограммами обработки ошибок библиотеки ПЛ-I. Кроме того, пользователь средствами языков программирования может вырабатывать собственные коды возврата. При выходе из пользовательской ПЛ-I программы посредством операторов RETURN и END главной процедуры ОС передается код возврата, равный нулю. В противном случае, или с помощью подпрограммы библиотеки ПЛ-I IEHSARC , формируется код возврата с некоторым конкретным значением, не равным нулю.

Ввиду сложности анализа конкретных значений кодов возврата в пакетах заданий [6], будем считать, что пользовательская программа завершила свою работу нормально, если код возврата равен нулю, и аварийно, - если код возврата отличен от нуля.

Удобным средством формирования кода возврата, отличного от нуля, в ПЛ-I программе является оператор STOP (код возврата - 1000). При выходе из программы посредством оператора STOP или после обращения к подпрограмме IEHSARC , на центральный пульт оператора или широкую печать следует выводить сообщения о причине аварийного завершения программы.

7. Вопросы формирования процедур автономного вызова программ АСОД

Язык управления заданиями ОС ЕС позволяет строить простые в использовании и в тоже время мобильные системы эксплуатации ПО АСОД в пакетном режиме. Использование символических параметров при кодировании операндных частей EXEC и DD -предложений позволяет не привязывать жестко ПО АСОД к базе данных, к конфигурации ЭВМ и ОС на стадии разработки АСОД.

Рассмотрим конкретный пример. Пусть имеется каталогизированная процедура PROC1 (рис.2), посредством которой несложно осуществить вызов некоторого множества программ. Все программы данного множества настроены на работу со следующими именами DD -предложений:

SYSPRINT - вывод данных на широкую печать;
ST - ввод записей-таблиц НД-справочника;
SYSIN - ввод параметров одноразового режима работы программы или данных;
F1+F3 - каталогизированные НД на магнитных носителях информации.

```
//PROC1   PROC  LIBP=LOAD,KLST='ØØ2,ØØ5,Ø8Ø',ST=ST,
//  PGMP=BN1ØØ,DF1N=OLD,DF2N=OLD,DF3N=MOD,DSNF1='ND1(Ø)',
//  DSMF2='ND2(Ø)',DSNF3='ND3(Ø)',REG=1ØØK
//STEP1    EXEC    PGM=&PGMP,PARM=&KLST,REGION=&REG
//STEPLIB  DD      DSN=&LIBP,DIBP=SHR
//SYSPRINT DD      SYSOUT=A
//ST       DD      DSN=&ST,DISP=SHR
//F1       DD      DSN=&DSNF1,DISP=(&DF1N,KEEP)
//F2       DD      DSN=&DSNF2,DISP=(&DF2N,KEEP)
//F3       DD      DSN=&DSNB3,DISP=(&DF3N,KEEP)
//SYSIN    DD      DDNAME=STEP1PK
//NZV      EXEC    PGM=COND,COND=(Ø,NE,STEP1)
//  PARM='Ø,PROC1.&PGMP.'
//STEPLIB  DD      DSN=&BIBP,DISP=SHR
//SYSPRINT DD      SYSOUT=A
//AZV      EXEC    PGM=COND,COND=(Ø,EQ,STEP1),
//  PARM='1,PROC1.&PGMP.'
//STEPLIB  DD      DSN=&BIBP,DISP=SHR
//SYSPRINT DD      SYSOUT=A
//AZVA     EXEC    PGM=COND=ONLY,PARM='1,PROC1.&PGMP.'
//STEPLIB  DD      DSN=&BIBP,DISP=SHR
//SYSPRINT DD      SYSOUT=A
```

Рис.2. Текст процедуры PROC1.

Назначение шагов NZV , AZV и AZV описано в [6]. Текст программы COND приведен на рис.3.

```
COND: PROC (PARM) OPTIONS(MAIN);
    DCL PARM CHAR(45) VAR, PS CHAR(43) VAR,
        NZV  CHAR(22) INIT('НОРМАЛЬНОЕ ЗАВЕРШЕНИЕ '),
        AZV  CHAR(21) INIT('АВАРИЙНОЕ ЗАВЕРШЕНИЕ '),
    PS=SUBSTR(PARM,3);
    IF SUBSTR(PARM,1,1)='0' THEN DISPLAY(NZV||PS);
                            ELSE DO;
                                   DISPLAY(AZV||PS);
                                   PUT SKIP EDIT(AZV,PS)(A);
                                 END;
END COND;
```

Рис.3. Текст программы COND.

Процедура PROC1 уже настроена на выполнение работ по программе BN100 . Подготовка необходимого задания (N1) потребует минимума усилий (см.рис.4).

```
//N1      JOB    необходимые операнды
//имя     EXEC   PROC1
//STEP1PK DD
        параметры режима работы
/*
//
```

Рис.4. Задание N1.

При вызове PROC1 произойдет привязка к конкретной программе и НД обработки. Из НД-справочника будут считаны необходимые таблицы, из входного потока параметры одноразового режима работы программы.

Посредством процедуры PROC1 достаточно просто осуществить вызов любой программы из указанного множества. При их вызове необходимо передать процедуре имя программы и присвоить требуемые значения другим символическим параметрам. Нетрудно заметить, что при приведенной в PROC1 форме кодирования операндов DD-предложений, не делается

различия между INPUT , OUTPUT и UPDATE - файлам в программах и способом организации НД на машинном носителе и что число НД обработки на магнитных носителях, необходимых конкретной программе, может колебаться от нуля до трех. Число НД обработки в принципе может быть любым, и они могут быть расположены на любых машинных носителях. Однако в таких случаях разумно на соответствующие классы программ сформировать другие, более приспособленные для простой модификации, процедуры.

При формировании процедур для автономного вызова программ необходимо находить разумный компромисс между числом программ и процедур, с одной стороны, и числом изменяемых символических параметров и DD-предложений при вызове конкретных процедур-с другой стороны.

8. Заключение

При наличии стандартного интерфейса программ АСОД с базой данных, ОС и оператором ЭВМ, ОС ЕС позволяет сократить до минимума число процедур, необходимых для автономного вызова программ, что существенно облегчает операции по внедрению и эксплуатации ПО АСОД. Наличие таких процедур необходимо уже на стадии автономной отладки программ, так как в этом случае существенно упрощается и работы программиста.

Такой подход, однако, накладывает довольно жесткие ограничения на организацию работ коллектива программистов и структур. программ. "Этому вопросу уделяется все большее внимание со стороны организаций, которые начинают понимать, насколько это важно для успешного сопровождения и эксплуатации" [4] .

ЛИТЕРАТУРА

I. Бичевский Я.Я. Измерение скорости ввода-вывода данных в операционной системе ДОС ЕС-В кн.: Разработка АСУ в Латвийской ССР. Рига, ЛГУ им. П. Стучки, 1977, с.32

2. Евсюков Е.П., Колин К.К. Основы проектирования информационно-вычислительных систем. М., Статистика , 1977.

3. Аугустон М.И., Балодис Р.П., Барздинь Я.М., Калниньш А.А. Подмножество PL/1 ОС ЕС. Рига, ЛГУ им.П.Стучки, 1976.

4. Иодан Э. Структурное проектирование и конструирование программ. (Перевод с англ.). М., Мир , 1979.

5. Глушков В.М. Введение в АСУ, Киев, Техника , 1974.

6. Пипир В.К., Янкевица М.Я. Принципы формирования пакетов заданий для эксплуатации программного обеспечения автоматизированных систем обработки данных. Настоящий сборник, с. 72

ПРИНЦИПЫ ФОРМИРОВАНИЯ ПАКЕТОВ ЗАДАНИЙ ДЛЯ ЭКСПЛУАТАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ АВТОМАТИЗИРОВАННЫХ СИСТЕМ ОБРАБОТКИ ДАННЫХ

В.К.Пипир, М.Я.Янкевица

ВЦ ЛГУ им.П.Стучки

I.Введение

Для того, чтобы облегчить работу оператора ЭВМ при эксплуатации АСОД (автоматизированных систем обработки данных) и повысить надежность системы в процессе эксплуатации, необходимо тщательно продумать технологию пропуска программ через ЭВМ. Эксплуатация АСОД обычно проводится в пакетном режиме, т.е. на вход системы подаются задания, оформленные как единое целое. Полные описания заданий средствами языка управления заданиями довольно длинные, поэтому их удобно оформлять в виде процедур. Наиболее часто используемые процедуры следует записать в SYS1.PROCLIB и использовать как каталогизированные процедуры, остальные - как процедуры во входном потоке.

Язык управления задачиями ОС ЕС позволяет строить простые в использовании и в тоже время достаточно универсальные процедуры эксплуатации программ АСОД. Использование символических параметров при кодировании параметров EXEC и DD -предложений позволяет не привязывать жестко программы к конфигурациям ЭВМ, ОС и к базе данных на стадии разработки. Эту привязку можно и нужно осуществлять в момент вызова той или иной процедуры, присваивая символическим параметрам необходимые значения.

Определим необходимые для дальнейшего изложения понятия: обрабатывающий шаг, шаг рестарта и служебный шаг процедуры.

Под обрабатывающим шагом процедуры будем понимать такой шаг, который описывает вызов программы, осуществляющей ввод, преобразование или печать базы данных.

Шагом рестарта назовем такой обрабатывающий шаг, с которого в случае аварийного завершения данного или одного из последующих шагов процедуры возможно осуществление отсроченного рестарта.

Под служебным шагом процедуры будем понимать такой шаг, который описывает вызов программы, осуществляющей вывод на центральный пульт оператора или на широкую печать информации о ходе и результатах вычислений по данной процедуре в целом или по ее составным частям. Обычно служебные шаги располагаются перед шагами рестарта и в конце процедуры. Эти шаги информируют оператора ЭВМ о нормальном прохождении работы, в случае аварийного завершения предыдущих шагов информируют оператора о дальнейших действиях (в том числе - о возможностях и шаге рестарта). В случае аварийного завершения, сообщение об этом выдается и на широкую печать, и тем самым пользователь узнает, что задача завершилась аварийно, если даже оператор ЭВМ этого не заметил или не выполнил инструкцию. Следует иметь в виду, что в системах обработки данных пользователями информации являются товароведы, бухгалтеры и т.д.т т.е. люди, не знакомые с основами программирования, и для них сообщения, выдаваемые операционной системой, не понятны. Служебные шаги в конце процедуры информируют оператора о том, как закончилась работа по процедуре в целом. В случае нормального завершения оператор может отдать результаты пользователю информации, в случае аварийного завершения - должен отдать программисту, который осуществляет сопровождение программного обеспечения АСОД.

При составлении процедур необходимо решить следующие вопросы:

1) определить условия для обхода выполнения шагов процедуры;

2) создать возможность повторения процедуры с необходи-

мого шага рестарта;

3) обеспечить сохранность информации в наборах данных, организованных в поколениях, при повторном вызове процедуры;

4) организовать размещение временных и передаваемых наборов данных.

2. Определение условий для обхода выполнения шага процедуры.

В большинстве случаев в задачах обработки данных последующий обрабатывающий шаг должен быть выполнен только после успешного завершения предыдущих.

Для обхода выполнения шага процедуры в зависимости от того, каким образом закончилось выполнение предыдущих шагов этой же процедуры, служит параметр COND EXEC -предложения. Обрабатывающий шаг процедуры необходимо обходить, если один или несколько предыдущих шагов закончились аварийно. Под аварийным завершением шага процедуры понимается следующее:

1) шаг закончился аварийно в смысле ОС ЕС (возникла ABEND ситуация/;

2) библиотечные PL/1 - программы установили значение кода возврата, отличное от нуля [2].

При формулировке условий обхода выполнения шага необходимо иметь в виду, что если шаг обходится, то его код возврата в последующих шагах ОС ЕС не проверяется.

Для определения условий обхода выполнения последующих шагов необходимо осведомляться только об обходе выполнения шагов из-за недопустимого кода возврата, так как аварийное окончание шага в смысле ОС ЕС само по себе прекращает выполнение остальных шагов той же процедуры, если только специально не запрошено противное (COND=EVEN или COND=ONLY). В одном параметре COND можно указать

не более восьми проверок. Для реальных задач этого вполне достаточно. Если число шагов процедуры больше восьми, необходимо организовать служебные шаги и использовать их для определения условий обхода выполнения последующих шагов.

Для определения условия обхода шага вычисляется дизъюнкция перечисленных в параметре COND условий: если хотя бы одно из перечисленных условий выполняется, шаг обходится.

Пример I.

```
//STEP4 EXEC PGM=PROG1, COND=((0,NE,STEP1), (0,NE,STEP2),
(0,NE,STEP3))
```

Шаг STEP4 будет обойден, если выполнятся следующие условия: код возврата шага STEP1]=0 | код возврата шага STEP2]=0 | код возврата шага STEP3]=0 .

То обстоятельство, что в параметре COND проверяется только дизъюнкция перечисленных условий (и нельзя задавать конъюнкцию), не позволяет в одном служебном шаге обработать аварийное завершение предыдущих шагов. Условие обхода такого шага должно быть следующим: " в предыдущих шагах не было аварий в смысле ОС ЕС" и "код возврата предыдущих шагов равен нулю" (конъюнкция условий).

Поэтому в разработанной нами системе перед шагом рестарта и в конце процедуры всегда имеются три служебных шага первый из них работает в случае нормального завершения предыдущих обрабатывающих шагов (шаг NZV), остальные два - в случае аварийного завершения одного или нескольких предыдущих шагов (шаг AZV в случае недопустимого кода возврата; шаг AZVA - в случае аварийного завершения в смысле ОС ЕС).

Пример 2. Для краткости изложения приведены только параметры, необходимые для обеспечения обхода шагов процедуры и вывода информации о ходе вычислений и шаге реестра.

```
//PROC1   PROC
//STEP1   EXEC  PGM=PGM1
//STEP2   EXEC  PGM=PGM2, COND=(0,NE,STEP1)
//NZV1    EXEC  PGM=COND, COND=((0,NE,STEP1),(0,NE,STEP2))
```

```
//   PARM=Ø,PROC1.STEP2'
//AZV1    EXEC  PGM=COND,COND=(Ø,EQ.NZV1),
//   PARM='1,PROC1,RESTART=PROC1,STEP1'
//AZVA1   EXEC  PGM=COND,COND=ONLY,
//   PARM='1,PROC1,RESTART=PROC1.STEP1'
//STEP3   EXEC  PGM=PGM3,COND=(Ø,EQ,AZV1).
//STEP4   EXEC  PGM=PGM4,COND=((Ø,EQ,AZV1),(Ø,NE,STEP3))
//STEP5   EXEC  PGM=PGM5,COND=((Ø,EQ,AZV1),(Ø,NE,STEP3),
    (Ø,NE,STEP4))
//NZV2    EXEC  PGM=COND,COND=((Ø,EQ,AZV1),(Ø,NE,STEP3))
//   (Ø,NE,STEP4),(Ø,NE,STEP5)),PARM='Ø,PROC1'
//AZV2    EXEC  PGM=COND,COND=((Ø,EQ,AZV1),(Ø,EQ,NZV2))
//   PARM='1,PROC1,RESTART=PROC1,STEP3'
//AZV2    EXEC  PGM=COND,COND=(ONLY,Ø,EQ,AZV2))
//   PARM='1,PROC1,RESTART=PROC1.STEP3'
```

Программа COND [2], вызываемая в служебных шагах, выдает на печать и консоль информацию о ходе и результатах вычислений в обрабатывающих шагах. Через параметр PARM EXEC - предложения программе передается выводимая на печать информация, которая зависит от места применения этой программы.

3. Использование средств отсроченного рестарта

В системах обработки данных, в которых база данных организована в виде некоторой совокупности наборов данных, нельзя обойтись без средств отсроченного рестарта.

Так как аварийное завершение процедуры часто происходит из-за логических ошибок или неконкретности данных, то использовать автоматический рестарт нецелесообразно. Необходимо выяснить причину аварии, устранить ее и после этого возобновить пропуск процедуры. Повторный ввод и пропуск всей процедуры не только означает потерю машинного времени, но, что еще важнее, может привести к неверным результатам (удвоение результатов в некоторых наборах данных; потеря

информации в наборах данных, организованных в поколениях и т.д.).

Все задачи АСОД в плане формирования процедур эксплуатации можно подразделить на две группы:

1) Задачи, не модифицирующие наборы данных и допускающие многократный пропуск программ с одними и теми же входными и выходными данными (печать входных форм, выдача справок и т.п.). Организация отсроченного рестарта процедур, осуществляющих решение таких задач, как правило, не вызывает затруднений (см.пример 2 и рис.1);

2) Задачи, модифицирующие наборы данных или открывающие новые поколения наборов данных. Повторный пропуск процедур, осуществляющих решение таких задач, может привести к искажению или потере информации. При отсроченном рестарте таких процедур необходимо осуществлять модификацию параметров DD -предложений (см.п.4). Для задач, где необходима модификация многих параметров DD -предложений, целесообразна разработка процедур, специально предназначенных для осуществления отсроченного рестарта задач АСОД с некоторых этапов их решения.

Для реализации отсроченного рестарта используется параметр RESTART JOB - предложения, в котором при вызове процедуры задается имя шага рестарта, ранее указанное программой COND . Для того, чтобы выбрать соответствующий обрабатывающий шаг как шаг рестарта, необходимо, во первых, обеспечить, чтобы на начало этого шага сохранилась вся информация, используемая в последующих шагах; во вторых, шаг рестарта необходимо выбрать так, чтобы повторный пропуск с одними и теми же данными не привел бы к неверным результатам. Если аварийное завершение произошло до первого шага рестарта, процедура пропускается полностью заново. Если аварийное завершение произошло между i -тым и $(i+1)$-ым шагом рестарта или i-тым шагом рестарта и концом процедуры, то осуществляется рестарт с i-го шага рестарта.

В нашей системе информация для рестарта содержится в поколениях наборов данных и в постоянных рабочих наборах данных. Постоянные рабочие наборы данных - это такие наборы данных, которые всегда присутствуют на определенном томе магнитного диска и имеют диспозицию (DISP) перед выполнением шага OLD и после закрытия набора данных - KEEP . Выделение памяти для этих наборов данных осуществляется вне процедур специальной программой на этапе подготовки к эксплуатации системы обработки данных. В постоянных рабочих наборах данных содержится промежуточная информация этапов обработки, которая сохраняется до следующего пропуска процедуры или отсроченного рестарта. Программисты, осуществляющие сопровождение системы, могут использовать эту информацию для выявления причин аварийного завершения процедур.

Пример 3.

Рассмотрим часть схемы решения некоторой задачи (рис.I), которая осуществляет следующие функции.

В 1-ом шаге процедуры программа PROGV на основании признаков, введенных с перфокарт, осуществляет выборку записей с последовательного набора данных ARH . ARH - накопительный набор данных, который содержит информацию об однородных, периодически повторяющихся операциях за определенный промежуток времени (например, данные о движении товаров и тары по складам за месяц). Выбранные записи фиксируются во временном последовательном наборе данных &&RAB (например, &&RAB содержит данные по определенным складам). В (i+I)-ом шаге выбранные записи сортируются с использованием программы "Сортировка-объединение" - IERRCOØØ , которая входит в состав общесистемного математического обеспечения ОС ЕС (например, сортировка в порядке возрастания номенклатурных номеров и цен). Рассортированные записи записываются в набор данных RABS , который описывается DD - предложением SORTOUT . В этом примере RABS - постоянный

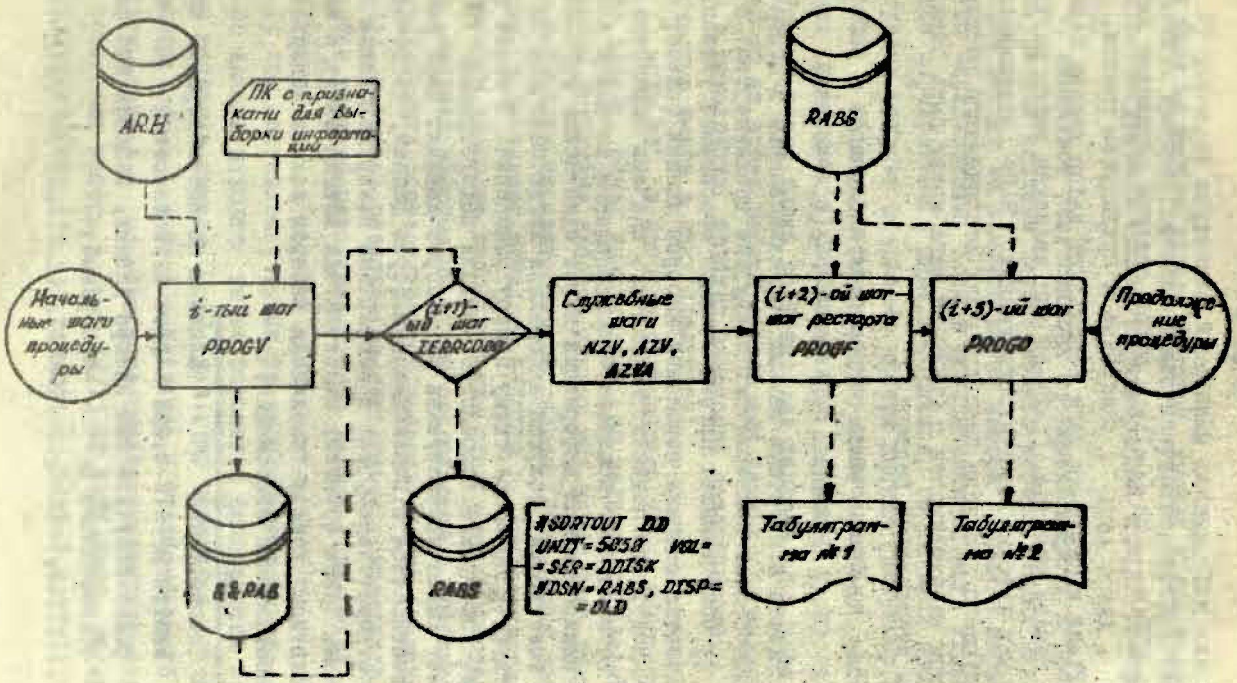


Рис.I. Организация шага рестарта процедуры.

рабочий набор данных, который всегда открыт на томе VOL=SER=DDISK . За (i+1)-ым шагом следуют служебные шаги NZV , AZV , AZVA , описанные в предыдущем пункте. (i+2)-ой и (i+3)-ий шаги используют набор данных RABS для печати отчетов. (i+2)-ой шаг является шагом рестарта. Так как информация в наборе данных RABS сохраняется до следующего пропуска процедуры или отсроченного рестарта, то при необходимости с (i+2)-го шага может быть осуществлен отсроченный рестарт.

4. Обеспечение сохранности информации в наборе данных, организованных в поколениях

Работа с наборами данных, организованных в поколениях, предоставляет пользователю следующие преимущества:

1) обеспечивает автоматическое сохранение указанного количества старших поколений наборов данных (схема: сын-отец-дед), что повышает надежность системы обработки данных;

2) облегчает пользователю наблюдение за базой данных, так как единообразная информация имеет одинаковое групповое имя и в тоже время текущая информация легко отличима от предшествующей по абсолютному номеру поколения. В нашей системе каждая программа посредством вызова подпрограммы FN * печатает имена используемых ею наборов данных. Если работа осуществляется с поколениями наборов данных, то печатается как относительный, так и абсолютный номер поколения наборов данных. Таким образом, пользователь всегда знает, в каких по абсолютному номеру поколениях наборов данных содержалась обрабатываемая информация;

---

* - Подпрограмма FN написана на Ассемблере ОС программистом ВЦ ЛГУ Б.Ариным и дает возможность в PL -программе получить имя набора данных, задавая имя файла, объявленного в PL -программе;

3) упрощает обращение к наборам данных. Так как все поколения наборов данных каталогизированы, то при чтении необходимо указать только параметры DSN и DISP DD-предложений;

4) избавляет пользователя от необходимости в процессе эксплуатации системы переименовать наборы данных или постоянно следит за заменой имени набора данных в параметре DSN DD -предложений. Если в обрабатывающем шаге модифицируется определенного вида информация, то на вход программы поступает текущее поколение набора данных (например, DSN=OP(0)), а на выходе формируется новое поколение набора данных (DSN=OP(+1)), которые при следующем пропуске процедуры на выход опять поступит как текущее поколение (DSN=OP(0)). В то же время работа с наборами данных, организованными в поколениях, требует некоторой осторожности. Необходимо иметь в виду следующее:

I) если шаг, в котором открываются новые поколения наборов данных, начал выполняться, но завершился аварийно, то новые поколения наборов данных уже могут быть открыты и по окончании соответствующей работы (JOB) будут зафиксированы в системном каталоге SYSCITG . При осуществлении повторного пропуска процедуры или отсроченного рестарта шага, входная информация будет содержаться в предшествующих поколениях соответствующих наборов данных (например, OP(-I)), и запись результатов необходимо осуществить в уже открытое текущее поколение набора данных (DSN=OP(0) , DISP=OLD);

2) при составлении процедур необходимо понимать, что в наборе данных SYSCITG новые поколения фиксируются по окончании работы в смысле ОС ЕС (т.е., по окончании соответствующего JOB-а , а не по окончании соответствующего шага. Поэтому, если в i -том шаге процедуры открывается новое поколение набора данных (//F1 DD UNIT=3050, VOL=SER=DDISK,SPACE=(CYL,(10,2)),DSN=OP(+1),DISP=(,CATIG)),

а в (+I)-ом этот же набор данных используется как существующий (т.е. для чтения информации), то обращение к нему должно быть описано следующим образом:

```
//F1 DD DSN=OP(+1),DISP=OLD;
```

3) необходимо, чтобы вся работа с поколениями наборов данных происходила при наличии того резидентного тома, на котором они зафиксированы в системном каталоге.

В нашей системе работа с поколениями наборов данных организована следующим образом. На этапе подготовки к эксплуатации системы осуществляется построение индексов групп поколений наборов данных в системном каталоге, построение моделей DSCB блоков на резидентном томе [I] , каталогизация и открытие первого по абсолютному номеру поколения наборов данных. Последнее мероприятие проводится для унификации работы - чтобы первый пропуск процедуры ничем не отличался бы от последующих (данные считываются из текущего поколения набора данных, модифицируются и заносятся в новое поколение набора данных).

В процессе эксплуатации работа организована двумя способами:

I) в процедурах, которые пропускаются сравнительно редко (несколько раз в месяц), каталогизация и открытие новых поколений наборов данных осуществляется специальной программой перед пропуском соответствующей процедуры. Это дает возможность пропускать процедуру несколько раз без потери информации. Считывание осуществляется с предшествующего поколения набора данных (DSN=OP(-1)) , запись в текущее поколение (DSN=OP(0),DISP=OLD);

2) для ежедневно используемых процедур каталогизация и открытие новых поколений наборов данных осуществляется в рамках процедуры специальным шагом. Так как функции, осуществляемые в этом шаге, крайне простые, то аварийное завершение может произойти фактически только в смысле ОС ЕС.

По окончании такого шага оператору ЭВМ печатается предупреждение, что новые поколения соответствующих наборов данных открыты и повторный пропуск данной процедуры запрещается, а также указывается допустимое имя шага отсроченного рестарта.

5. Размещение временных и передаваемых наборов данных

Использование временных и передаваемых наборов данных непосредственно связано с размещением в процедурах эксплуатации задач АСОД шагов отсроченного рестарта. Все промежуточные результаты обрабатывающих шагов целесообразно размещать во временных и передаваемых наборах данных. Такие наборы данных должны быть по возможности не привязаны к конкретным типам внешних устройств и не содержать специальные запросы томов. Следует лишь помнить, что шагу рестарта не допускается передача временных наборов данных, и что запрашиваемая для временных и передаваемых наборов данных память на устройствах прямого доступа должна быть в распоряжении ОС ЕС.

6. Заключение

Хорошо продуманная система процедур эксплуатации позволяет вводить через входной поток только операторы, вызывающие и модифицирующие процедуры и, при необходимости, данные. Ввод управляющих предложений обрабатывающих программ ОС ЕС (например, программы СОРТИРОВКА-ОБЪЕДИНЕНИЕ) и параметров режима работы пользовательских программ [2] в рамках конкретных процедур целесообразно осуществлять из разделов специальной библиотеки. Такая система позволяет значительно уменьшить входной поток и, тем самым, свести до минимума число ошибок, возникающих при формировании пакетов заданий решения задач АСОД.

При составлении процедур для кодирования параметров необходимо использовать символические параметры [2] . В данной работе для наглядности приведенных примеров DD - предложения даны после замены символических параметров их конкретными значениями.

Не смотря на то, что язык управления заданиями ОС ЕС обладает некоторыми чертами языков программирования высокого уровня, для эффективной организации пропуска заданий через ЭВМ в условиях эксплуатации АСОД этих средств не достаточно. Это в основном относится к средствам организации обхода шага задания (параметр COND EXEC- предложения), в котором возможно задать только дизъюнкцию условий, и к средствам работы с поколениями наборов данных. Хотя в условиях работы с наборами данных, организованными в поколениях, операционная система берет на себя часть функций организации работы, но не дает возможности организовать эффективную защиту данных и допускает неоправданное открытие новых поколений. Чтобы эффективно организовать работу с наборами данных, организованными в поколениях, необходима надстройка пользователя, которую, наверно, следует организовать в виде специального набора данных - "Системного журнала", в котором регистрируются все манипуляции с наборами данных.

Литература

1. Аугустон М.И., Балодис Р.П., Барздинь Я.М., Калниньш А.А. Подмножество PL/1 ОС ЕС. Рига, ЛГУ им. П.Стучки, 1976.
2. Пяпир В.К., Янкевица М.Я. Стандартизация параметров интерфейса программ АСОД с базой данных, ОС и оператором ЭВМ . Настоящий сборник, с. 60

МЕТОДИКА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДИАЛОГОВЫХ АВТОМАТИЗИРОВАННЫХ ОБУЧАЮЩИХ СИСТЕМ

Л.В.Ницецкий, Л.В.Зайцева,
Л.П.Новицкий, У.А.Суковский, В.С.Шитиков
Рижский политехнический институт

В настоящее время в связи с повышением требований к качеству подготовки специалистов в высшей школе и появлением новых технических возможностей (оснащение машинами ЕС и СМ ЭВМ, накопление опыта программирования и разработки больших систем) возникла необходимость более широкого внедрения в учебный процесс различных технических средств обучения, в частности, обучающих и контролирующих систем.

Согласно [4] автоматизированная обучающая система (АОС) - организованный на базе ЭВМ комплекс средств технического, учебно-методического, лингвистического и программного обеспечения, предназначенный для применения в учебных занятиях различного вида и работающий в основном в диалоговом режиме коллективного пользования.

Ряд требований к АОС являются системотехническими и проявляются в первую очередь при разработке средств программного обеспечения.

Системный подход к разработке АОС означает ее проектирование как единого целого и обеспечение высокой эффективности функционирования и использования ресурсов системы с точки зрения взаимосвязи составляющих ее элементов. В частности, для сокращения объема программ во избежание дублирования в начальных стадиях разработки важно выделить ряд общих обеспечивающих программ, например, для управления фрагментами диалога, вывода сообщений, анализа сообщений пользователей и т.д.

Характеристики звеньев АОС необходимо согласовать с характеристиками АСУ вуза (как системы более высокого иерархического уровня) и использовать общие обеспечивающие элементы.

Учитывая большой объем работ по общесистемной отладке АОС и появление коррекций в постановке задачи важно получить действующий вариант системы по возможности раньше. Поэтому допускается разработка упрощенных версий, которые должны дополняться без коренных изменений всей системы в направлении приближения к первоначально поставленной цели и путями ее реализации. Предусматривается возможность замены отдельных подсистем по одной (без переделки наследуемых частей). Например, язык составителей учебного обеспечения должен быть стабильным, чтобы на нем не сказывались такие изменения, как переход от алфавитно-цифровых к графическим дисплеям, учет развитой модели знаний отдельных обучаемых, введение новых способов описания предметной области.

При модернизации и развитии АОС должны сохраняться основные системотехнические характеристики, пока из-за появления принципиально новых технических и программных средств не отпадет цель о разработке новой системы.

Учитывая наличие других АОС и учебно-методического обеспечения к ним, в частности, распространенной системы программирования обучающих курсов (СПОК) [2] важно добиваться совместимости различных систем. Наиболее важно обеспечить возможность передачи учебно-методического обеспечения.

Основным режимом АОС должен быть диалоговый, т.е. обучаемый и обучающая система должны поочередно обмениваться информацией. Это необходимо для того, чтобы индивидуализировать процесс обучения/контроля. Инициатором или ведущим в зависимости от конкретной задачи и обстановки может быть человек или машина. АОС должна быть многорежимной, т.е. должна обеспечивать выполнение различных работ, связанных с учебным процессом, в ряде случаев как в режиме диалога, так и пакетной обработки. АОС должна быть системой коллективного пользования, что обусловлено как способом ее применения (обычно при групповых занятиях), так и соображениями эф|ек-

тивности используемой вычислительной системы, поскольку чем большее количество пользователей одновременно работает с АОС, тем она экономичнее. АОС должна быть системой реального времени, поэтому должна удовлетворять определенным ограничениям на максимально допустимое время реакции системы. Время реакции АОС более 3 с заметно ухудшает качество обучения и контроля. Максимальное количество одновременно работающих пользователей определяется количеством терминалов и необходимостью обеспечить допустимое время реакции. Последнее оценивается при предварительном анализе и экспериментально.

Требования, предъявляемые к обучающей системе, и ее особенности определяют требования к техническому, учебно-методическому и лингвистическому обеспечению АОС, а в итоге - к ее программному обеспечению.

Рассмотрим вопросы, которые приходится решать при разработке программного обеспечения АОС.

В связи с тем, что обучающая система должна быть автоматизированной, возникает проблема выбора технической и программной базы ее реализации, т.е. выбор используемой ЭВМ, операционной системы и языка программирования.

В связи с тем, что АОС должна быть диалоговой, возникает проблема выбора технической базы ведения диалога - терминалов, определения инициатора диалога и языка общения пользователя с системой.

Требование многофункциональности обусловливает необходимость, с одной стороны, определить спектр реализуемых АОС функций, с другой стороны, обеспечить возможность добавления новых функций и изменения старых без перепрограммирования всей системы.

Требование коллективного пользования АОС обусловливает необходимость хранения информации о предыдущих сеансах каждого пользователя и одновременного обслуживания как можно большего числа пользователей.

При этом необходимо, чтобы обеспечение неиспользуемых в данный момент функций АОС (программы и данные) не занимало наиболее дефицитный ресурс вычислительной системы - опе-

ративную память. В то же время необходимо обеспечить выполнение несколькими пользователями одинаковой работы одной копией соответствующей программы (без дублирования ее в оперативной памяти), а при возможности-и одной копией данных.

Необходимость удовлетворения заданной реактивности системы выдвигает требование разработки дисциплины обслуживания запросов с тем, чтобы при каждом виде работы не превышалось максимально допустимое время реакции системы (различное для различных работ).

Рассмотрим последовательно конкретную реализацию всех перечисленных выше требований в АОС КОНТАКТ [I].

Поскольку АОС должна быть рассчитана на широкий круг потребителей, то она должна быть реализована на базе стандартной универсальной ЭВМ или семейства совместимых ЭВМ. Основным семейством являются машины ЕС ЭВМ , как наиболее крупные. Однако предусматривается реализация подмножеств одной и той же системы также на различных моделях СМ ЭВМ, "Искра" и других. В рамках одного семейства ЭВМ важно обеспечить возможность работы АОС при различных объемах оперативной памяти ЭВМ и установить для них оптимальные режимы по количеству пользователей.

Другим важным вопросом, связанным с базой реализации, является альтернатива использования стандартного математического обеспечения данной ЭВМ, либо создание специализированного математического обеспечения (управляющей программы).

Поскольку АОС является разновидностью системы разделения времени, то для улучшения экономических характеристик ЭВМ (увеличение загрузки ЭВМ) целесообразно организовать параллельно с работой диалоговой системы функционирование "фона", например, пакетного режима. Поскольку стандартные системы математического обеспечения имеют хорошо развитые средства организации пакетного режима, с одной стороны, а с другой стороны, современные АОС в большой степени являются экспериментальными, то целесообразно создание АОС на базе стандартных средств математического обеспечения.

Существуют две операционные системы, обеспечивающие работу ЕС ЭВМ в мультипрограммном режиме - ДОС ЕС и ОС ЕС. В качестве

базы была выбрана ОС ЕС как наиболее подходящая для мощных моделей ЕС ЭВМ с объемом оперативной памяти 512 К и более, поскольку машины именно такой конфигурации будут получать наибольшее распространение, по крайней мере в вузах, на которые в первую очередь рассчитана АОС.

В качестве языка программирования для ускоренной реализации первых очередей системы и возможности системотехнических испытаний АОС в целом был выбран язык ПЛ/I [3], который, с одной стороны, позволяет производить практически любые манипуляции, допустимые в ОС ЕС, а. с другой стороны, удобен для кодирования алгоритмов и предоставляет достаточные средства отладки. В тех же случаях, когда средств языка ПЛ/I недостаточно, необходимо программировать на языке Ассемблера ОС ЕС.

Программы обмена с терминалами выделены в виде отдельных модулей, и возможна настройка на конкретный тип терминалов посредством редактирования с другими программами обмена. В случае использования дисплеев программы обмена обеспечивают независимость от формата экрана, в частности, от количества строк на экране (двенадцать для комплекса ЕС-7906 и двадцать для комплекса ЕС-7920).

Важнейшим вопросом, связанным с диалоговым режимом, является выбор инициатора диалога. Поскольку АОС в основном рассчитана на неквалифицированных пользователей - студентов, то в АОС КОНТАКТ выбран диалог, управляемый ЭВМ, при котором система задает вопросы, а пользователи отвечают на них.

В связи с этим возникает проблема кодирования учебно-методического материала для ведения диалога, т.е. разработка языка автора обучающих программ (ЯАОП), а также транслятора с него. ЯАОП должен быть рассчитан на два этапа реализации диалога по обучающим программам (ОП):

I этап - неэффективный по эксплуатационным характеристикам, но быстро реализуемый;

2 этап - эффективный по эксплуатационным характеристикам.

Создание программного обеспечения АОС целесообразно начинать с разработки интерпретатора с ЯАОП и комплекса ОП с тем, чтобы начав опытную эксплуатацию первой очереди системы, проверить.

основные принципы ведения диалога, апробировать и откорректировать имеющиеся ОП. При составлении могут выявиться недостатки языка автора обучающих программ и возникнуть необходимость его расширения.

Поскольку ЯАОП должен быть, в первую очередь, удобен для составителей обучающих программ, то интерпретация непосредственно операторов этого языка может оказаться весьма затруднительной. Поэтому целесообразно транслировать ОП в некоторое промежуточное представление, удобное для интерпретации (рис.I).

Рис. I. Схема обработки обучающей программы.

На первом этапе разработки АОС следует максимально использовать возможности операционной системы, в частности, для трансляции ОП можно применить транслятор с языка ПЛ/I. Это не значит, что ОП придется писать на языке ПЛ/I. Поскольку ПЛ/I предусматривает препроцессорную обработку исходного текста программ, то, используя именно эти возможности языка ПЛ/I, можно осуществить перевод ОП с языка автора на язык ПЛ/I. Затем в результате трансляции и редактирования получить загрузочный модуль ОП, который и может быть использован интерпретатором для обучения в диалоговом режиме.

Чтобы эффективно использовать средства препроцессора, целесообразно написать ряд препроцессорных процедур для преобразо-

вания операторов языка автора ОП в операторы языка ПЛ/I. Каждому оператору языка автора при этом будет соответствовать препроцессорная процедура с тем же именем, а сами операторы иметь вид обращения к процедуре - функции. Например, если язык автора включает операторы QUEST, RIGHT, ELSE, то необходимо наличие препроцессорных процедур с именами QUEST, RIGHT, ELSE. Использование препроцессорных процедур позволяет легко изменять идентификаторы операторов языков автора. Например, вместо QUEST, RIGHT, IF, THEN можно использовать обозначение JAUTAJ, PAREIZI, JA, TAD. При этом можно использовать те же препроцессорные процедуры, лишь изменив их имена.

Необходимость сокращать время составления учебно-методического обеспечения по темам потребовало разделения уровней программирования, а именно: для получения множества вариантов хода диалога преподаватель, ведущий занятия, может устанавливать различные режимы, которые не надо программировать при каждом возможном разветвлении (детальность сообщений, случайный выбор вопросов или последовательность, оптимальная в педагогическом отношении, требование обязательно добиться правильного ответа обучаемого, накопление статистики).

Для преподавателей различных дисциплин, слабо владеющих программированием для ЭВМ, предусматривается заполнение специальных бланков, не требующих точного знания специализированного языка.

Написание транслятора для преобразования ОП в эффективное по эксплуатационным характеристикам внутреннее представление целесообразно начинать после опытной эксплуатации первой очереди, т.е. во второй очереди создания АОС.

Важным вопросом, связанным с диалогом, является лингвистическое обеспечение АОС, основным моментом которого является установление ограничений на язык общения пользователей с системой. С одной стороны, следовало бы допускать любые конструкции языка (сложносочиненные, сложноподчиненные предложения и т.п.), а с другой стороны, для обеспечения эффективности общения целесообразно построение фразы по строго определенным правилам с фиксированным местоположением подлежащего, сказу-

емого, недопустимостью перефразирования. Поэтому (с учетом объемов памяти ЭВМ и реальных возможностей составления программ) необходимо найти некоторый компромиссный вариант.

Поскольку АОС является разновидностью разговорных систем, то следует попытаться учесть особенности разговорной речи обучаемых, например, отрывистых ответов при разработке лингвистического обеспечения АОС . Ограничения на входной язык останутся и в этом случае, но они будут мало заметными для пользователей, так как определяются на основе изучения массива разговорной речи самих пользователей (их "внутренние ограничения").

В зависимости от учебной дисциплины и форматов ожидаемых сообщений пользователей можно получить вариант подсистемы распознавания сообщений оверлейной структуры на заданном объеме оперативной памяти.

Обеспечение многорежимности АОС приводит, во-первых, к тому, что следует определить, какие из необходимых функций целесообразно выполнять в диалоговом режиме, а какие-в пакетном. Существуют функции (например, обучение), которые целесообразно выполнять только в диалогом режиме, а существуют такие, которые имеет смысл выполнять в обоих режимах.

Каждая функция должна реализовываться в виде отдельной программы, которую в дальнейшем будем называть функциональным модулем (ФМ). Кроме того, нужна программа, управляющая выполнением ФМ в диалога и реализующая общее управление ресурсами АОС. Однако, поскольку непосредственный доступ к ресурсам АОС не всегда целесообразен, необходима программа, имитирующая диалог и выполняющаяся в пакетном режиме (имитатор диалога).

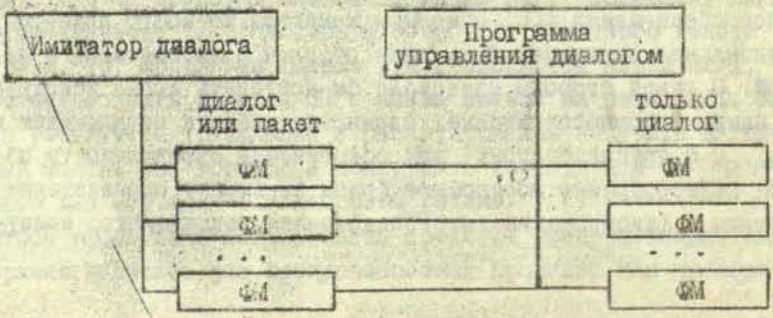


Рис. 2. Структура программного обеспечения АОС.

Программа управления диалогом (ПУД) обслуживает одновременно несколько пользователей, имитатор - только одного.

Функции, которые допускают значительное время реакции системы, например, трансляция обучающей программы или трансляция программы, составленной на одном из языков программирования, можно выполнять вне раздела (зоны), в которой выполняется ПУД.

В диалоговом режиме одновременно может потребоваться выполнение различных функций. Поскольку заранее непредсказуемо сочетание этих функций, то для того, чтобы в оперативной памяти не находились программы, реализующие не нужные в данный момент функции, каждый ФМ оформляется в виде динамически вызываемого загрузочного модуля. В оперативную память загружаются только необходимые модули. Функциональные модули для общения с управляющими программами имеют строго определенные соглашения о связях, в том числе фиксированный список параметров.

Поскольку может не хватить оперативной памяти для хранения требуемых в данный момент ФМ, то возникает необходимость их перезагрузки. Это условие накладывает ограничения на структуру ФМ. Информация, которая может менять свое значение, должна быть оформлена в виде структуры, базированной с помощью переменной локаторного типа, значение которой устанавливает ПУД.

Такой подход позволяет также использовать одну копию ФМ несколькими пользователями, каждому из которых будет выделен свой участок оперативной памяти, для хранения соответствующей ему информации.

Наиболее важным ФМ является интерпретатор обучающего материала, который, в первую очередь, определяет требования к стратегии распределения ресурсов АОС, реализуемой ПУД.

При групповых занятиях с АОС часто возникает ситуация, когда несколько обучаемых работают с одной и той же обучающей программой. При этом следует стремиться к тому, чтобы не дублировались информация, одинаковая для всех пользователей. Для этого ОП следует представлять в виде двух частей:

- часть, которая может быть одновременно использована несколькими обучаемыми (информация, не изменяющаяся во время диалога);

- часть, которая должна быть индивидуальна для каждого обучаемого (информация, меняющаяся во время диалога).

Для обеспечения экономичной по памяти обработки ОП программа управления диалогом выделяет и освобождает участки оперативной памяти требуемой длины по запросам от ФМ. Причем это могут быть запросы на разделяемую память (память мультидоступа) и неразделяемую память (память монопольного доступа).

Каждая функция АОС характеризуется максимально допустимым временем реакции системы. С другой стороны, можно выделить несколько категорий пользователей АОС с различными требованиями к реактивности системы (например, обучаемый, преподаватель, администратор). ПУД обслуживает очереди запросов на ресурсы АОС на основании категории пользователей и используемых ими ФМ.

Для обеспечения эффективности системы ПУД выполняет, кроме того, такие функции, как:

- хронометраж работы системы и установление ограничений на время подготовки ответа;
- ведение досье на каждого пользователя;
- восстановление после сбоев;
- установление стандартных областей для группы пользователей (например, установление одинаковых параметров для контроля знаний для группы одновременно работающих обучаемых);
- настройка на конкретный сеанс (на конкретную конфигурацию выделенных для АОС ресурсов);
- ведение протокола диалога.

Литература

1. Автоматизированная обучающая система КОНТАКТ на базе ЕС ЭВМ. Версия КОНТАКТ/ОС. Метод. указания / Под ред. Л.В. Ницецкого; Рига; Рижск. политехн. ин-т, 1979.

2. Довгялло А.М., Небрат О. П., Платонов Б. А. Обучение с использованием вычислительных машин: современное состояние и перспективы. – Управляющие системы и машины, 1978, № 2, с. 12-20.

3. Аугустон М.И., Балодис Р.П., Барздинь Я.М., Икаунieкс Э.А. Калниньш А.А. Программирование на ПЛ/I ОС ЕС/. М., Статистика.М.,1979.
4. Технические и методические требования к автоматизированным обучающим системам на базе ЕС ЭВМ/Сост.: Ницецкий Л.В., Новиков В.А. Рига, НИИ проблем высшей школы; Рижск.политехн. ин-т, 1979.

АВТОМАТИЗАЦИЯ ОБУЧЕНИЯ РЕШЕНИЮ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

М.В.Витиньш

ВЦ ЛГУ им.П.Стучки

В настоящее время во всем мире уделяется большое внимание созданию автоматизированных систем обучения. Причин тому много, однако основными являются постоянное увеличение числа учащихся на всех ступенях образования и резкий рост объема научной и учебной информации. Поскольку увеличение сроков обучения как средство разрешения проблем современного образования, по-видимому, исчерпало себя, следует обратить внимание на внутренние резервы учебного процесса. Так, автоматизированные системы обучения направлены на то, чтобы способствовать углублению индивидуализации обучения, повышению самостоятельной активности учащихся и сокращению времени преподавателя на выполнение рутинной работы. Они являются логическим развитием идей программированного обучения и применения технических средств обучения. В нашей стране разработаны и используются различные по своему назначению и реализации автоматизированные системы обучения [1]. Среди них следует отметить САДЮ - систему автоматизации диалога и коллективного обучения, и СПОК - систему программирования и поддержания обслуживающих и обучающих курсов [2]. Из зарубежных разработок следует выделить систему PLATO-IV, которая на сегодняшний день считается одной из самых современных систем обучения [3].

В процессе обучения важное место занимает решение задач. Однако известные автоматизированные системы обучения слабо приспособлены для полной автоматизации этой области. Они анализируют решение задачи в основном по конечному результату - ответу, а само решение - получение ответа - упускают из виду. Причиной тому являются сложность языка общения обучаемого с системой и трудности построения алгоритма управления обучаемым. Вместе с тем имеется немало задач, для которых эти проблемы решаются достаточно просто. Это, в первую очередь, задачи, решение которых требует

применения некоторого определенного алгоритма.

Так, в Вычислительном центре Латвийского государственного университета создана автоматизированная система обучения решению систем линейных уравнений методом Гаусса. Этот раздел математики выбран потому, что, во-первых, освоение материала связано с самостоятельной работой обучаемых над решением большого количества упражнений, что при традиционных формах обучения трудно осуществить, во-вторых, для общения обучаемого с системой обучения можно использовать весьма простой и естественный язык, в-третьих, можно построить эффективный алгоритм контроля решения и управления обучаемым и, в-четвертых, применение автоматизированной системы обучения позволит индивидуализировать обучение, повысит самостоятельность обучаемых и освободит преподавателя от утомительной проверки работ обучаемых.

Напомним определения нескольких понятий, которые будут использованы ниже. Так, два уравнения называются равносильными, если они имеют одно и то же множество решений; две системы уравнений называются равносильными, если они имеют одно и то же множество решений. Под системой уравнений треугольного вида будем понимать такую систему уравнений, в которой коэффициенты на главной диагонали равны единице, а слева от нее равны нулю. Напомним также, что сущность метода Гаусса заключается в приведении последовательными целенаправленными преобразованиями исходной системы уравнений к равносильной системе треугольного вида, и решение при этом представляется как последовательность преобразованных систем и ответ. Следовательно, автоматизированная система обучения должна получать от обучаемого решение, проводить контроль равносильности систем, анализировать целенаправленность выполненных преобразований, проверять правильность ответа и сообщать обучаемому как о верно выполненных действиях, так и о допущенных им ошибках и способах их устранения. Таким образом, обучение решению систем линейных уравнений методом Гаусса будет вестись под управлением автоматизированной системы обучения, и вмеша-

тельство преподавателя в процесс обучения потребуется лишь в отдельных случаях.

Техническими средствами реализации данной системы обучения является ЭВМ Единой системы. В качестве индивидуальных пультов пользователей используется комплекс локальных алфавитно-цифровых дисплеев ЕС-7906.

База данных системы размещена на накопителе с прямым доступом ЕС-5052 и содержательно состоит из трех частей: сборника задач, списка пользователей и архива.

Сборник задач в базе данных подготавливается преподавателем и содержит объединенные по вариантам системы линейных уравнений. В сборнике задач могут содержаться до 50 вариантов, которые нумеруются двузначными числами в пределах от 01 до 50. Каждый вариант может иметь от 1 до 15 упражнений, где каждое упражнение - система линейных уравнений. Упражнения каждого варианта также нумеруются двузначными числами от 01 до 15.

Список пользователей, как и сборник задач, подготавливается преподавателем и содержит номера, фамилии и имена пользователей и номера решаемых ими вариантов. Список пользователей в базе данных может содержать сведения не более чем о 50 пользователях, что соответствует их разбиению по классам и группам. Номера пользователям присваиваются преподавателем по его усмотрению или согласно классному или групповому журналу и должны принимать значения от 01 до 50. Номер решаемого варианта для каждого пользователя также определяется преподавателем.

Архив служит для накопления информации о ходе обучения. В нем накапливаются сведения относительно каждого пользователя о том, какие упражнения решал и решил, как много времени затратил на решение каждого из них, какие и сколько ошибок допустил, сколько раз направлялся за помощью к преподавателю. При наличии свободного массива в архиве туда помещаются записи диалогов обучаемых с системой обучения.

Программное обеспечение системы создано в операцион-

ной системе ДОС ЕС на языке ПЛ/I за исключением частей, связанных с использованием базисного метода телекоммуникационного доступа - ВТАМ, которые запрограммированы на языке АССЕМБЛЕР.

Разработанная система обучения функционально состоит из четырех компонент: средства организации и ведения сборника задач, средства организации и ведения списка пользователей, средства общения с обучаемым, анализа сообщения обучаемого и накопления сведений о ходе обучения и средства обработки накопленных сведений о ходе обучения.

Средство организации и ведения сборника задач предназначено для добавления, замены и удаления из базы данных вариантов упражнений. Кроме того, оно обеспечивает отображение текущего состояния сборника задач, а именно: вывод на печать номеров, содержащихся в базе данных вариантов и упражнений, составляющих определенный вариант. Программная реализация данного средства эксплуатируется в режиме пакетной обработки, входная информация поступает с перфокарт и диагностика работы выводится на печать.

Средство организации и ведения списка пользователей по назначению, составу и реализации аналогично средству организации и ведения сборника задач. Единицей добавления, замены и удаления в этом случае являются сведения об одном пользователе. В плане отображения состояния списка пользователей в базе данных обеспечивается только одно: вывод на печать сведений о всех пользователях, содержащихся в списке пользователей.

Средство общения с обучаемым, анализа сообщения обучаемого и накопления сведений о ходе обучения является узловой составной частью системы обучения. Оно непосредственно организует и управляет процессом обучения. Работа обучаемого с системой ведется в режиме диалога и начинается с того, что пользователь на сообщение системы о ее готовности к работе сообщает свой номер. Далее пользователь должен сообщить номер решаемого упражнения. Если от преподавателя не поступили особые указания, то пользователь в первом сеансе работы с системой начинает с первого упражнения, а в

последующих сеансах-либо с нерешенного, либо со следующего упражнения. Система обучения, в свою очередь, сообщает пользователю текст этого упражнения - систему линейных уравнений. Пользователь решает ее и каждую полученную преобразованную систему или же ответ сообщает системе обучения. Та анализирует сообщенную часть решения и либо одобряет действия обучаемого, либо указывает ему на ошибки. В последнем случае пользователь старается ее устранить и исправленную часть решения вновь сообщает системе. Если обучаемый повторно допустил ошибку, то система советует ему обратиться за помощью к преподавателю. Анализ сообщенной преобразованной системы уравнений начинается с контроля ее равносильности предыдущей сообщенной или же исходной системе. В случае неравносильности систем возможны две ситуации. Во-первых, неравносильность систем является следствием неравносильности некоторого уравнения сообщенной системы всем уравнениям предыдущей сообщенной или же исходной системы и, во-вторых, следствием потери уравнения, которое не является линейной комбинацией других уравнений предыдущей сообщенной или же исходной системы. В первом случае система обучения сообщает пользователю, что им допущена арифметическая ошибка и указывает номер неверно полученного уравнения. Во втором случае сообщается только о потере уравнения. Если системы оказались равносильными, то анализируется целенаправленность выполненных преобразований. Критерием целенаправленности принято последовательное получение коэффициентов, равных единице при переменных на главной диагонали и равных нулю при соответствующих переменных ниже ее. В случае невыполнения условий критерия система обучения фиксирует отклонение от метода Гаусса, сообщает об этом обучаемому и указывает, в каком уравнении при какой переменной и какое значение коэффициент следует получить. Если преобразования выполнены целенаправленно, то пользователю сообщается, что преобразование выполнено верно. Результатом анализа сообщенного ответа является сообщение, что упражнение решено верно и следует сообщать но-

мер следующего решаемого упражнения, либо в ответе содержится арифметическая ошибка. Средство общения с обучаемым, анализа сообщения обучаемого и накопления сведений о ходе обучения обеспечивает возможность закончить сеанс работы с системой как по указанию преподавателя, так и по собственной инициативе каждого обучаемого. Оно также осуществляет синтаксический и семантический контроль сообщений обучаемых и подробно информирует их о допущенных ошибках. Кроме того, оно непрерывно накапливает в архиве сведения о ходе обучения и в случае свободного места помещает в него записи диалогов обучаемых с системой обучения.

Средство обработки накопленных сведений о ходе обучения предназначено в основном для их отображения. Оно обеспечивает вывод на печать содержащихся в архиве сведений о ходе обучения и записей диалогов обучаемых с системой. Оно позволяет также удалить из архива не интересующие преподавателя записи диалогов.

Рассмотрим пример работы обучаемого с системой обучения.

Система: СИСТЕМА ОБУЧЕНИЯ ГОТОВА К РАБОТЕ
СООБЩИТЕ НОМЕР ПОЛЬЗОВАТЕЛЯ
Пользователь: НØI

Н указывает, что сообщается номер пользователя, ØI - номер пользователя.

Система: СООБЩИТЕ НОМЕР РЕШАЕМОГО УПРАЖНЕНИЯ
Пользователь: УØI

У - признак сообщения номера решаемого упражнения, OI - номер решаемого упражнения.

Система: РЕШИТЕ СИСТЕМУ УРАВНЕНИЙ

$$x - 4y - 7z = 2$$
$$3x - 8y - 9z = -6$$
$$4x - 5y - 3z = -1$$

Пользователь: П $x - 4y - 7z = 2$
$3K - 8y - 9z = -6$
$4x - 5y - 3z = -1$

П - признак сообщения преобразованной системы, далее

следует сама преобразованная система.

Система: ОШИБКА ЗАПИСИ В СООБЩЕНИИ ПРЕОБРАЗОВАНИЯ
ВТОРОЕ УРАВНЕНИЕ ЗАПИСАНО НЕВЕРНО
ПОСЛЕ КОЭФФИЦИЕНТА НЕ СЛЕДУЕТ ПЕРЕМЕННАЯ

Пользователь: П x - 4y - 7z = 2
3x - 8y - 9z =-6
4x - 5y -3z =-11

Система: АРИФМЕТИЧЕСКАЯ ОШИБКА В СООБЩЕНИИ ПРЕОБРАЗОВАНИЯ
ТРЕТЬЕ УРАВНЕНИЕ НЕРАВНОСИЛЬНО УРАВНЕНИЯМ РЕШАЕМОЙ СИСТЕМЫ
ОБРАТИТЕСЬ ЗА ПОМОЩЬЮ К ПРЕПОДАВАТЕЛЮ

Пользователь: П x - 4y - 7z = 2
4y +12z =-12
4x - 5y - 3z =-1

Система: ПРЕОБРАЗОВАНИЕ ВЫПОЛНЕНО ВЕРНО

Пользователь: П x - 4y - 7z = 2
y - 3z =-3
4x - 5y - 3z =-1

Система: ОШИБКА В ПРИМЕНЕНИИ АЛГОРИТМА
В ТРЕТЬЕМ УРАВНЕНИИ У ПЕРВОЙ ПЕРЕМЕННОЙ СЛЕДУЕТ ПОЛУЧИТЬ КОЭФФИЦИЕНТ Ø

Такая реакция системы объясняется заложенным в нее критерием целенаправленности выполненных преобразований.

Пользователь: П x - 4y - 7z = 2
y + 3z =-3
11y + 25z =-9

Система: ПРЕОБРАЗОВАНИЕ ВЫПОЛНЕНО ВЕРНО

Пользователь: П x - 4y - 7z = 2
y + 3z =-3
-8z = К9

Система: ОШИБКА ЗАПИСИ В СООБЩЕНИИ ПРЕОБРАЗОВАНИЯ
ТРЕТЬЕ УРАВНЕНИЕ ЗАПИСАНО НЕВЕРНО
СВОБОДНЫЙ ЧЛЕН НЕ НАЧИНАЕТСЯ НИ ЗНАКОМ + ИЛИ -,
НИ ЦИФРОЙ ИЛИ .

Пользователь: П x - 4y - 7z = 2
y + 3z =-3
-8z =24

Система: ПРЕОБРАЗОВАНИЕ ВЫПОЛНЕНО ВЕРНО
Пользователь: П $x - 4y - 7z = 2$
$y + 3z = -3$
$z = -3$
Система: ПРЕОБРАЗОВАНИЕ ВЫПОЛНЕНО ВЕРНО

О указывает на сообщение ответа, 5 и 6 - значения первой и второй переменной соответственно.
второй переменной соответственно.

Система: ОШИБКА ЗАПИСИ В СООБЩЕНИИ ОТВЕТА
ОТВЕТ СОДЕРЖИТ МЕНЬШЕ ЗНАЧЕНИЙ, ЧЕМ ПЕРЕМЕННЫХ В СИСТЕМЕ
Пользователь: О 5,6,3
Система: АРИФМЕТИЧЕСКАЯ ОШИБКА В СООБЩЕНИИ ОТВЕТА
ОБРАТИТЕСЬ ЗА ПОМОЩЬЮ К ПРЕПОДАВАТЕЛЮ
Пользователь: О 5,6,-3
Система: УПРАЖНЕНИЕ РЕШЕНО ВЕРНО
СООБЩИТЕ НОМЕР РЕШАЕМОГО УПРАЖНЕНИЯ
Пользователь: К

Признак К означает, что пользователь желает закончить сеанс работы с системой обучения.

Система: СИСТЕМА ОБУЧЕНИЯ ГОТОВА К РАБОТЕ
СООБЩИТЕ НОМЕР ПОЛЬЗОВАТЕЛЯ

Система обучения закончила сеанс работы с пользователем ØI и готова начать работу с другим или возобновить работу с тем же пользователем.

ЛИТЕРАТУРА

I. Савельев А.Я. Автоматизированные обучающие системы на базе ЭВМ. М., Знание, 1977.
2. Алексеенко Е.А., Довгялло А.М., Косая И.Х., Небрат О.П., Платонов Б.А. СПОК-система программирования и поддержания обслуживающих и обучающих курсов.-УСиМ, 1978, № 2, с.127-128.
3. Обучающие машины. Бюллетень иностранной научно-технической информации ТАСС, 1979, № 30, с.15-16.

МЕТОДИКА УСТРАНЕНИЯ ПОСЛЕДСТВИЙ НЕНОРМАЛЬНОГО ЗАВЕРШЕНИЯ ПРОГРАММ ОБРАБОТКИ ДАННЫХ НА ЕС ЭВМ ПОСРЕДСТВОМ ПРОГРАММНО-СХЕМНЫХ РЕШЕНИЙ

А.Я.Абеле

ВЦ ЛГУ им.П.Стучки

Абстрактный алгоритм обработки данных, материализованный в виде конкретного изделия-программы, будет выполняться в конкретной среде (ЭВМ) в соприкосновении с конкретными данными. Эта среда имеет какие-то характеристики достоверности и надежности. Их учитывать обязаны проектные и программно-схемные решения при организации решения задач обработки данных.

Программа обработки данных ни одну запись входного или выходного набора данных не должна вводить или выводить более одного раза, ни одну запись из участвующих в обработке наборов данных не должна терять. Поэтому предоставляемый операционными системами, так называемый, аппарат "контрольных точек" мало пригоден для программ обработки данных, т.к. практически приходится создавать контрольную точку после обработки каждой записи.

Вместо этого программист использует свои, так называемые, программно-схемные решения, которые требуют:

- при выборе схемы обработки данных в основной памяти (ОП) максимально возможно уменьшить затраты времени центрального процессора на обработку одной записи, т.е. увеличить производительность программы;
- специально организовать программу обработки данных так, чтобы она могла выполнить обработку как с начала (с первой записи набора), так и возобновить (не с первой записи) ранее прерванную обработку;
- снабдить программу обработки данных аварийной ветвью.

Обычно программы обработки данных используют протокол обработки, представляющий собой специальное поле ОП, в котором накапливаются представляющие интерес для технологии решения задач обработки данных сведения о данных, обработанных программой, например, идентификатор набора, регис-

трационный номер машинного носителя, количество записей, дата и время создания и очередной обработки набора и т.п. В конце работы программы протокол выводится на печать.

Для идентификации точки прерывания обработки целесообразно в начале поля протокола обработки разместить некоторые счетчики, показывающие, например, номера только что обработанных записей отдельных наборов, и другие элементы, которые необходимы как исходные данные для возобновления прерванной обработки. Абсолютный адрес ячейки ОП, с которой начинается поле протокола обработки, должен быть известен оператору ЭВМ, или как константа, или выведен на ПМ в начале работы программы.

Несмотря на принятые решения, в принципе возможны следующие типы ненормальных завершений любой отлаженной обработки:

- аварийное завершение программы обработки данных по причинам: программа (ошибки программирования, незавершенность алгоритма обработки данных), ЭВМ (отказ оборудования, связанного только с данной программой), данные (ошибочные), оператор ЭВМ (ошибочные действия). При этом вычислительная система (ВС) в целом работоспособна (РС) и снимается только программа, завершившаяся аварийно;
- ОЖИДАНИЕ или так называемое "зависание", когда вследствие технического сбоя ВС приходит в состояние ожидания. При этом ВС в основном остается работоспособной, а только в отдельных случаях неработоспособной (НРС);
- "тяжелый останов" (ТО), когда вследствие технического отказа ВС становится неработоспособной.

После ненормального завершения программы необходимо зафиксировать на бумажном носителе весь протокол обработки или его начальную часть, необходимую для возобновления обработки. Для этого ЕС ЭВМ предоставляет пользователю следующие средства вывода протокола обработки.

Ветвь аварийного завершения (В-АВ) - разработанная пользователем специальная подпрограмма (обеспечиваемая в ДОС и ОС), которая автоматически получает управление по

возникновении причин аварийного завершения. Если эта подпрограмма не может спасти обработку (ее продолжить), то она должна попытаться (перечислены в порядке важности): закрыть файлы, подготовить и вывести на печать информацию для возобновления обработки с промежуточных точек, вывести на печать информацию о причине и месте аварии, снять программу.

Ветвь оператора ЭВМ (В-ПМ) - разработанная пользователем специальная подпрограмма (обеспечиваемая только в ДОС), которая получает управление по инициативе оператора ЭВМ. В ней можно запрограммировать те же функции, что в аварийной ветви.

Чтение ячеек ОП с ПМ (Ч-ПМ) - вывод (обеспечиваемый только в ДОС) на пульт оператора ЭВМ (ПМ) содержание ячеек ОП, начиная с указываемого на ПМ абсолютного адреса протокола обработки в ОП.

Чтение ячеек ОП с инженерного пульта ЭВМ (Ч-ИП) - вывод (не зависящий от операционной системы) на инженерный пульт ЭВМ содержание ячеек ОП, начиная с указываемого на этом же пульте абсолютного адреса протокола обработки в ОП.

Возможность использования средств вывода протокола в зависимости от типа ненормального завершения программы и состояния ВС показана в следующей таблице:

Тип ненормального завершения	Состояние ВС	Средства вывода протокола			
		Ч-ИП	Ч-ПМ	В-ПМ	В-АВ
Аварийное завершение	РС			x) +	+
ОЖИДАНИЕ	РС	+	+	+	
	НРС	+			
ТО	НРС	+			

x) прерывание обработки по инициативе оператора ЭВМ.

В случае ОЖИДАНИЕ-РС, во избежание порчи протокола, оператор ЭВМ, прежде чем использовать средства Ч-ПМ или В-ПМ, должен вывести с поля протокола посредством Ч-ИП необходимый для возобновления обработки минимум информации.

ПРИНЦИПЫ ПОСТРОЕНИЯ И ПЕРСПЕКТИВЫ РАЗВИТИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ АСУ ЛАТМИНВУЗ

З.А.Шулте, М.Г.Кусина

ВЦ ЛГУ им.П.Стучки

Программное обеспечение АСУ ЛАТМИНВУЗ построено на принципе независимости обработки от конкретной структуры данных. Это обусловлено большим количеством часто меняющихся входных и выходных документов. Традиционное написание программ привело бы к необходимости постоянного внесения изменений в программы и использованию структур с перекрытием, реализаций которых требует существенных затрат времени. Реализация принципа независимости программного обеспечения от структуры данных позволила все расчеты, производимые системой, осуществлять сравнительно небольшим количеством программ; основные из них следующие:

- программа ввода и контроля информации;
- программа формирования и коррекции основных массивов;
- программа обмена между основными массивами;
- программа формирования и коррекции дополнительных и архивных программ;
- программа формирования и печати выходных форм.

Большой объем информации, отсутствие достаточного количества накопителей на магнитных дисках, а также структура информационной базы, спроектированная Рижским политехническим институтом для п/с "Абитуриент", "Студент", "Сессия", привели к постановке задачи как файловой. Поэтому перечисленные программы написаны для обработки последовательных файлов с учетом ситуаций, которые могут возникнуть при такой обработке. В случае перехода к какому-либо другому способу хранения данных необходимо будет изменить схему чтения/записи данных, однако блоки обработки данных останутся неизменными.

Основой для работы программы системы служат таблицы описания структуры входной и выходной информации, а также таблицы обработки входных данных. Вышеуказанные таблицы хранятся в индексно-последовательных файлах, которые явля-

ются частью нормативно-справочной информации (НСИ).

Для печати выходных форм необходимо использовать еще несколько файлов НСИ, а именно: кодификаторы, макеты строк и условно-переменную информацию, содержащую планы приема, графики вступительных экзаменов, расписание экзаменов и зачетов и т.д.

Для более ясного представления системы программного обеспечения АСУ ЛАТМИНВУЗ, следует рассмотреть более подробно таблицы описания входной и выходной информации, включающие все массивы системы, все входные документы, а также выходные формы. Структура каждой записи таблицы следующая:

X,	9(8)	, 9(4),	9(8)	, 9(4),	99	, 99	, 99
A	B	C	D	E	F	G	H

A - код удаления;
B - шифр массива или документа;
C - шифр реквизита;
  - шифр кодификатора;
E - начальный символ реквизита;
  - длина реквизита;
  - длина повторяющейся зоны;
H - число повторений.

Последние два поля заполняются сравнительно редко, так как большинство реквизитов в массиве встречаются только один раз, однако для таких реквизитов, как шифры экзаменов и оценки, эти зоны очень полезны.

Во время разработки программного обеспечения системы несколько раз менялась структура массивов, а также входных документов. Использование таблиц описания дало возможность работать с новыми документами и массивами без внесения каких-либо изменений в программы.

Для создания выходного массива и его распечатки в виде выходной формы разработана система команд, содержащая следующие типы действий:

- действия над записями;
- арифметические действия;
- сравнения;
- простые и сложные логические выражения;

- пересылка всего реквизита или его части;
- сортировка записей по заданным ключам.

Эта система команд позволяет распечатывать все выходные формы I очереди АСУ ЛАТМИНВУЗ, а также дает возможность включить дополнительные выходные формы других подсистем. Одним из преимуществ такой системы команд является то, что новую выходную форму может описать человек, не являющийся программистом, но достаточно хорошо освоивший способ настройки программ на конкретный входной документ.

Программа формирования и коррекции основного массива построена так, что одновременно вводятся все входные документы для одного объекта. Это обстоятельство является особо важным, так как массивы расположены на магнитных лентах. Входной поток может содержать любое количество разных документов одной подсистемы, они все будут введены в массив за один сеанс работы программы и одно прокручивание магнитной ленты. Это тоже является ценным качеством программы, так как, например, приказы о студентах обычно идут большими порциями, но они почти всегда разные (приказ об отчислении, о переводе, о стипендии, о поощрении и т.д.).

Программа формирования и коррекции дополнительных и архивных массивов построена аналогично.

Для формирования и пополнения основных массивов различных подсистем на базе уже имеющихся основных массивов используется программа обмена информацией между основными массивами, которая, например, формирует из массива "Абитуриент" массив "Студент1" (I курс) или из массива "Студент1" массив "Студент2" (выпускники).

Опыт создания системы и анализ ее опытной эксплуатации показали возможные пути совершенствования системы. Создаваемое дополнительное программное обеспечение в первую очередь должно решать следующие задачи:

- упрощать эксплуатацию системы;
- облегчать изучение системы и ускорять подготовку для ее эксплуатации;
- выдавать нерегламентированную форму по запросу пользователя.

Соответственно перечисленному, постановка задачи разбивается на следующие этапы:

1. Создание словаря, позволяющего словами русского языка:

- задавать последовательность выполнения действий в процессе счета. Пример: ввод текста "ВВЕСТИ НОВЫЙ КОДИФИКАТОР" должен привести в действие поток программ, обеспечивающих ввод в КОДИК и распечатку на АЦПУ нового кодификатора;
- описывать функции программ, обслуживающих систему;
- задавать алгоритм формирования из входного документа реквизитов основного массива;
- описывать структуру выходного документа (т.е. заголовок, шапку, элементарные строки, итоговые строки);
- задавать алгоритм формирования граф выходного документа.

2. Выяснение, в каком объеме необходимо все это реализовать, с тем, чтобы расход машинного времени и затраты на создание программ компенсировались бы удобствами, представляемыми этими программами.

3. Написание программ, транслирующих запрос и обеспечивающих выполнение основного счета.

Параллельно можно решать задачу ввода такого запроса с дисплея и вывода соответствующей информации на дисплей с тем, чтобы руководящий работник мог непосредственно на своем месте оперативно получать необходимую информацию. Кроме того, для нужд эксплуатации, для облегчения отладки очередной выходной формы или ввода нового входного документа неплохо иметь возможность выводить на дисплей и корректировать НСИ системы.

Вторая и очень важная задача - это вопрос хранения и доступа к данным. С одной стороны, уже сейчас необходимо перейти от хранения основных массивов на МЛ к хранению на МД. Большой объем информации (15 тыс. записей п/с "Студент" только для Рижского политехнического института) затрудняет оперативность выдачи информации.

В настоящий момент можно перейти от хранения информации на МД к хранению ее в последовательном или индексно-последовательном файле на диске, для того, чтобы просматривать во время счета только то подразделение, которое задано в запросе. Это сократит время счета и уменьшит вероятность потерь из-за сбоев в работе внешних устройств.

С другой стороны, рост системы за счет объема решаемых задач приводит к необходимости интегрированного хранения данных. Таким образом, необходимо найти компромиссный вариант. Способ хранения данных и доступ к ним должен обеспечить растущие нужды системы. Но затраты машинного времени на эксплуатацию такой системы, а также затраты людей и средств не должны быть слишком большими.

Тут возможны такие варианты решения:

1. использование подходящего существующего банка данных;
2. создание банка данных, ориентированного на решение задач высшей школы;
3. форма хранения данных остается файловой, но основные массивы расположены на дисках.

Перспективный план создания и развития АСУ ЛАТМИНВУЗ предусматривает завершение I очереди системы в 1980 году. Перечисленные задачи совершенствования системы в плане первой очереди не предусмотрены и будут решаться в следующих пятилетках.

СОДЕРЖАНИЕ

1. Я.Я.Бичевский, Ю.В.Борзов. Один подход к отладке больших систем 3
2. Ю.В.Борзов, Р.В.Звирбуле, И.Т.Розенштейне, В.В.Сестуле. Диалоговая отладка ПЛ/I-программ с использованием инструментации отлаживаемой программы 10
3. Я.Я.Бичевский, И.А.Виржбицка, М.Я.Грунтере. Специальный метод доступа AUCE 16
4. И.Т.Ермолаева, Л.Л.Федорова. Использование индексно-страничного метода в задачах обработки данных 31
5. А.Р.Дзерве. Генератор печати табуляграмм 35
6. А.Я.Абеле. Подмножество табличного языка записи алгоритмов "TABELE" 45
7. В.К.Пипир, М.Я.Янкевица. Стандартизация параметров интерфейса программ АСОД с базой данных, ОС и оператором ЭВМ 60

8. В.К.Пипир, М.Я.Янкевица. Принципы формирования пакетов задания для эксплуатации программного обеспечения автоматизированных систем обработки данных 72
9. Л.В.Ницецкий, Л.В.Зайцева, Л.П.Новицкий, У.А.Суковский, В.С.Шитиков. Методика разработки программного обеспечения диалоговых автоматизированных обучающих систем 85
10. М.В.Витиньш. Автоматизация обучения решению систем линейных уравнений 96
11. А.Я.Абеле. Методика устранения последствий ненормального завершения программ обработки данных на ЕС ЭВМ посредством программно-схемных решений 104
12. З.А.Шулте, М.Г.Кусина. Принципы построения и перспективы развития программного обеспечения АСУ Латвинвуз 107

МЕТОДИКА СОЗДАНИЯ МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ ЭВМ
Межвузовский сборник научных трудов

Редакторы: А.Калниньш, Н.Сарамонова.
Технич.редактор И.Фридберга
Корректор И.Фридберга

Подписано к печати 08.07.1980. ЯТ 12230. Ф/б 60x84/16.
Бумага №1. 7,3 физ.печ.л. 6,8 усл.печ.л. 5,4 уч.-изд.л.
Тираж 500 экз. Зак.№ 1526. Цена 54 к.

Латвийский государственный университет им. П.Стучки
Рига 226098, б. Райниса, 19
Отпечатано на ротапринте, Рига 226050, ул.Вейденбаума,5
Латвийский государственный университет им. П.Стучки