отрезка [0, 1]. В табл. 1 приведены результаты счета на вещественной оси по методу Рунге — Кутта и методу непрерывного аналитического продолжения.

В табл. 1 x и y соответствуют значениям аргумента и функции, полученным по методу Рунге — Кутта четвертого порядка точности с вычислительной схемой:

$$\Delta y = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5),$$

$$k_1 = hf(x, y),\ k_2 = hf\left(x + \frac{h}{2},\ y + \frac{k_1}{2}\right),$$

$$k_3 = hf\left(x + \frac{h}{2},\ y + \frac{k_2}{2}\right),\ k_4 = hf(x+h,\ y+k_3),\ y' = f(x, y),\ y(x_0) = y_0.$$

В табл. 1 $\tilde{x}$ — точки, полученные с помощью метода непрерывного аналитического продолжения, в которых значение функции $y(x)$ подозрительно на полюс.

2. На комплексной плоскости решение задачи проводилось для начальных условий: $w(0) = 1$, $w'(0) = 0$.

Счет проводился на неравномерной сетке узлов

$z_{kj} = x_k + iy_j$, $x_k = x_{k-1} + h_k$, $x_0 = 0$, h_k — шаг, $y_j = y_0 + j\tau$, $y_0 = 0$, $\tau = 0,\ 1$, причем при переходе на каждый новый $j+1$ слой в качестве w_0 и w_0' были взяты соответственно значения $w(z_{0j})$, $w'(z_{0j})$.

На плоскости (z), $\mathrm{Re}\, z \geqslant 0$, $\mathrm{Im}\, z > 0$ найден первый полюс функции $w(z)$, $z = 1{,}837332 + 0{,}1\,i$. Результаты счета приведены в табл. 2.

ЛИТЕРАТУРА

1. Еругин Н. П.— Докл. АН БССР, 1958, т. 2, № 4.
2. Davis H. T., Scott W., Springer G., Resch D.— J. Northwestern University Studies, 1956, № 3.
3. Яблонский А. И.— Весці АН БССР. Сер. фіз. тэхн. навук, 1959, № 3.
4. Березин И. С., Жидков Н. П. Методы вычислений, т. 2.— М., 1950.
5. Воробьев А. П.— Дифференц. уравнения, 1965, т. I, № 1.

Поступила в редакцию 25.01.78.

Кафедры дифференциальных уравнений и общего программирования

УДК 681.3.06

Г. А. ДРОБУШЕВИЧ, ТО ТУАН

О СТРУКТУРНОМ ПРОГРАММИРОВАНИИ НА ЯЗЫКЕ АССЕМБЛЕР

В последние годы наметился значительный интерес к технологии программирования — совокупности знаний о способах и средствах получения программного продукта [1]. Литература, посвященная технологии структурного программирования на языках высокого уровня довольно обширна, однако в сравнительно немногочисленных статьях по структурному программированию на языке Ассемблер рассматривается возможность реализации с помощью макросредств языка Ассемблер необходимых управляющих структур типа *IF — THEN — ELSE — ENDIF, DO — WHILE (UNTIL) — ENDDO*. В действительности, основной концепцией структурного программирования является концепция программирования сверху вниз [2] наряду с концепцией структурного кодирования, т. е. кодирования с использованием только трех базовых управляющих структур [3]. По мнению Деннинга, структурное программирование есть метод проектирования программ по нисходящей схеме поэтапного усовершенствования с использованием при кодировании блок-схемных образцов с одним входом и одним выходом. К этому определению

структурного программирования нужно добавить еще требование, заключающееся в том, что программные проекты должны быть организованы в соответствии с концепцией «бригады с главным программистом», предложенной и обсужденной Вайкером и Миллзом [4]. На наш взгляд, наиболее удачным является определение Хоора [5], по которому структурное программирование представляет собой систематическое применение абстракций как средства компенсации количественной ограниченности человека [6] для управления массой сложных деталей в процессе программирования.

Трудность создания и отладки Ассемблер-программ общеизвестна. Практика нашей работы показывает, что процесс обучения студентов языку Ассемблер не менее сложен, чем процесс программирования на нем. Предложенный нами структурный подход к обучению языкам программирования и языку Ассемблер, в частности, является попыткой совместить методологию обучения и методологию структурного программирования и, таким образом, систематизировать процесс обучения и формирования структурного мышления [7].

В качестве примера структурного программирования сверху вниз на языке Ассемблер рассмотрим задачу суммирования n двухбайтных упакованных чисел из массива M (входные данные вводятся с перфокарт). Очевидна следующая общая структура требуемой программы.

1. <НАЧАЛО>
2. <ВВОД ИСХОДНЫХ ДАННЫХ>
3. <ВЫЧИСЛЕНИЕ СУММЫ>
4. <ВЫВОД РЕЗУЛЬТАТА>
5. <ОСТАНОВ>
6. <ОБЪЯВЛЕНИЕ РАБОЧИХ ПОЛЕЙ>
7. <КОНЕЦ>

Для облегчения записи этого алгоритма будем использовать специально разработанный нами макроязык *ASTRU*. Пусть имеется следующая программа.

```
SUMAP2  PROC  MAIN      основная процедура
        INPUT (R3)      ввод n в регистр R3
        INM   M,(R3)    ввод исходного вектора размера n
   <вычисление суммы>
OUTPUT  ('СУММА=', S)   в S размещается сумма
STOP                    останов
DCL     (M, 100PL2),(S, PL4)  объявление
ENDPROC SUMAP2          конец процедуры
```

Теперь будем конкретизировать блок <ВЫЧИСЛЕНИЕ СУММЫ> (рис. 1). Прямоугольник наглядно изображает цикл, который должен повторяться n раз. Очевидно преимущество предлагаемого способа изображения схемы программ перед традиционной блок-схемой, которая содержит много лишних сложных фигур и линий.

На уровне 4 производится уточнение объекта M_i записью $0(2, R_2)$, которая является явным заданием адреса памяти с указанием длины. На этом же уровне возникает новый объект R_2, являющийся наглядным обозначением общего регистра 2. Новый объект приведет к новым операторам, которые вводятся на уровне 5. Оператор $R_2=A(M)$ пересылает адрес массива M в регистр R_2. На уровне 7 вводятся макрокоманды *REPEAT* и *ENDREP*, которые должны реализовать требуемый цикл. Если учитывать, что число n хранится в регистре $R3$, то уровень 7 должен выглядеть следующим:

```
S=0
R2=A(M)
REPEAT TIMES=R3
   S=S+0(2, R2)
   R2=R2+2
ENDREP
```

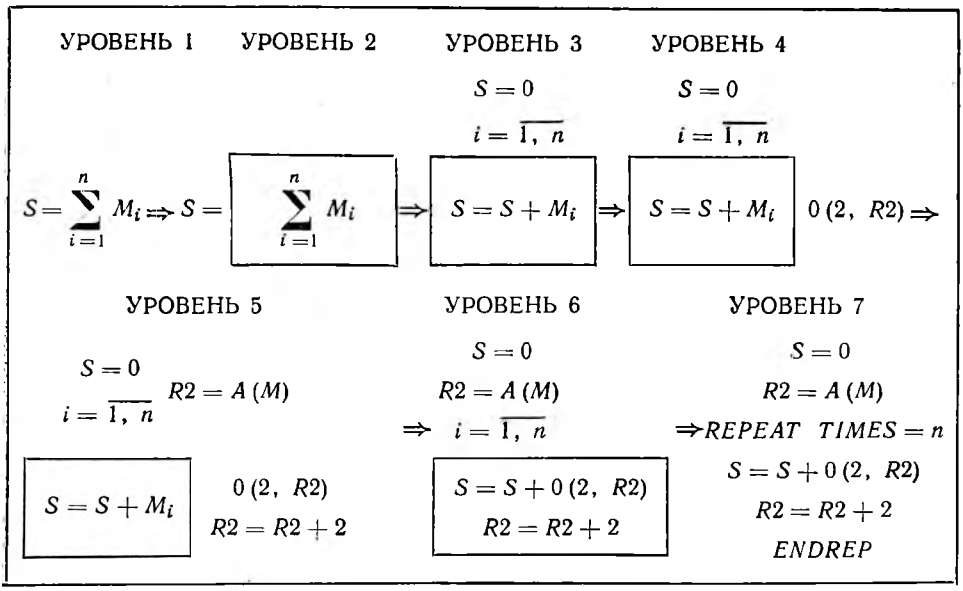

Рис. 1. Технологический процесс реализации блока <ВЫЧИСЛЕНИЕ СУММЫ>

После добавления необходимых имен и разделителей получается следующий фрагмент на языке *ASTRU*:

```
PAC S,=,0
FIX R2,=,A(M)
REPEAT TIMES=R3
    PAC S,=,S,+,0(2, R2)
    FIX R2,=,R2,+,2
ENDREP
```

Полученную программу на языке *ASTRU* можно легко переводить в программу на языке Ассемблер. Язык *ASTRU* полностью реализован макросредством языка Ассемблер, и полученную *ASTRU*-программу можно выполнить на машине (в вычислительной среде ДОС/ЕС). Отметим, что блок вычисления суммы можно реализовать с помощью более обобщенной по сравнению с *REPEAT* макрокомандой *DO*:

```
PAC S,=,0
DO R2, FROM=A(M), BY=2, TIMES=R3
    PAC S,=,S,+,0(2, R2)
ENDDO
```

Опыт структурного создания Ассемблер-программ по нисходящей схеме с помощью языка *ASTRU* привел нас к новому подходу к обучению языкам программирования и языку Ассемблер, в частности. Основной идеей структурного обучения является стратегия обучения сверху вниз, которая начинается с языка высокого уровня (возможно с некоторого подмножества естественного языка), а затем постепенно происходит переход (спуск) на язык менее высокого уровня, и так далее до изучаемого языка программирования. Таким образом, для каждого изучаемого языка L необходимо создать целое семейство иерархически связанных языков, $\{L_k\}$, $k=\overline{1,\ n}$, n — некоторое целое положительное число, а обучение языку L следует провести сверху вниз, начиная с языка L_1. Общая схема структурного обучения языку L с помощью семейства $\{L_k\}$ иллюстрируется на рис. 2.

Довольно часто на практике обучение осуществляется по восходящей схеме. Если изучаемый язык достаточно сложен (язык Ассемблер), традиционное обучение неэффективно, поскольку им трудно управлять и

очень трудно автоматизировать, к тому же между обучаемыми и ЭВМ возникает психологический барьер. Следует отметить, что традиционный подход к обучению порождает плохой стиль программирования в отличие от структурного стиля, который предполагает постепенное развитие программы от некоторой абстрактной формы по нисходящей схеме. На рис. 3 показан структурный процесс обучения программированию арифметических выражений.

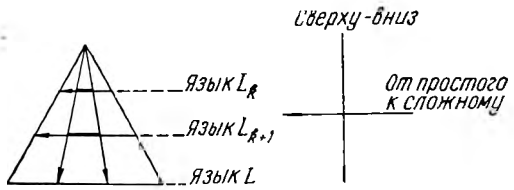

Рис. 2. Схема структурного обучения языку L

На уровне 2·1 понятие «Регистр» вводится как конечное множество ключевых слов $R = \{R0, R1, \ldots, R15\}$. Очевидно, что команды L (загрузка), A (сложение) и ST (обратная загрузка) усваиваются в таком процессе легче, чем механическое запоминание их без связи с конкретным применением (обучение снизу вверх из глубины машинных команд

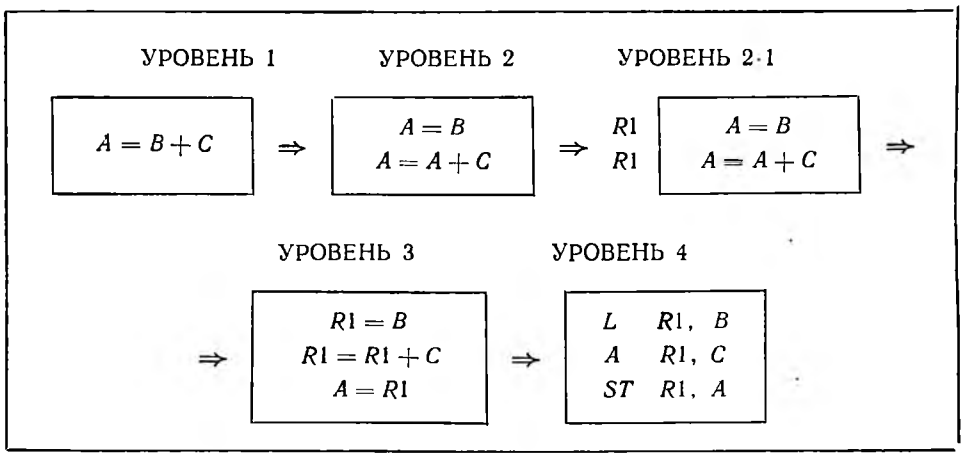

Рис. 3. Структурный процесс обучения программированию арифметических выражений.

и представлений информации). Частично описанный ранее язык *ASTRU* служит удобным средством на пути к постепенному овладению языком Ассемблера. На первых этапах обучения удобные макрокоманды языка ASTRU способствуют быстрому устранению психологического барьера между обучаемыми и ЭВМ. Чтобы переходить на следующий уровень обучения, нужно просто раскрыть соответствующие макрокоманды и изучить их макрорасширения, которые могут содержать другие макрокоманды. Каждый обучаемый в зависимости от успеваемости может задержаться на текущем уровне обучения, продвигаться вперед на несколько уровней. Следует отметить, что переход с одного уровня на другой обычно является частичным на определенных участках текущего уровня. Управляемость, систематичность и упорядоченность предлагаемого подхода компенсируют дополнительное время на прохождение различных уровней иерархии. После усвоения языка L, семейство $\{L_k\}$ не отбрасывается, а систематически применяется программистами в своей практике изготовления L-программ по технологии структурного программирования. Таким образом, предлагаемый многоцелевой подход к обучению является попыткой введения технологии в область обучения. Структурное обучение и структурное программирование успешно внедряются на факультете прикладной математики.

ЛИТЕРАТУРА

1. Глушков В. М., Вельбицкий И. В.— Управляющие системы и машины, 1976, № 6, с. 75.
2. Wirth N.— Computing Surveys, 1974, v. 6, № 4, p. 248.
3. Dijkstra E. W.— Communs. ACM, 1968, v. 11, № 3, p. 147.
4. Baker F. T., Mills H. D.— Datamation, 1973, v. 19, № 3, p. 58.
5. Knuth D. E.— Computing Surveys, 1974, v. 6, № 4, p. 263.
6. Дейкстра Е. В.— В кн.: Структурное программирование.— М., 1975, с. 7.
7. Иодан Э. Структурное проектирование и конструирование программ.— М., 1979.

Поступила в редакцию 20.02.79.

Кафедра общего программирования

УДК 517.544

Л. П. ПРИМАЧУК

НЕЛИНЕЙНАЯ ЗАДАЧА СОПРЯЖЕНИЯ НА РАЗОМКНУТОЙ ДУГЕ ДЛЯ ДВУХ ГОЛОМОРФНЫХ ФУНКЦИЙ

Постановка задачи. Пусть $L=ab$ — ограниченная простая гладкая разомкнутая дуга с концами a и b. Введем следующие обозначения: $D=\bar{C}\setminus L$; $H(L)$ — класс функций, удовлетворяющих на L условию Гельдера; $B^{m_1,m_2}(D)$ — класс аналитических в D векторов $\Phi(z)=(\varphi_1,\ \varphi_2)$, ограниченных во всякой замкнутой области, не содержащей точек a и b, причем $\varphi_i(z)$ имеет m_i нулей в D, а $\ln(\varphi_i(z))$ в окрестности точек a и b имеет слабую особенность; $B^{0,0}(D)=B(D)$; $B^{m_1,m_2}(D)\subset B^{m_1,m_2}(D)$ — подкласс вектора из $B^{m_1,m_2}(D)$, ограниченных в D.

Требуется найти аналитический вектор $\Phi(z)=(\varphi_1(z),\ \varphi_2(z))\in B^{m_1,m_2}(D)$ по следующему нелинейному краевому условию на L:

$$\varphi_1^+(t)\,\varphi_2^-(t)=f_1(t),\ \varphi_2^+(t)\,\varphi_1^-(t)=f_2(t), \tag{1}$$

где $f_i\in H(L)$ и $f_i(t)\neq 0,\ \forall\, t\in L$.

Задача (1) для одной неизвестной функции изучалась в работах [2—5].

Решение задачи (1) в классе B(D). Лемма 1. Вектор

$$\Psi(z)=(\Psi_1,\ \Psi_2)=\exp\frac{\sqrt{(z-a)(z-b)}}{4\pi i}\int\limits_L\frac{\ln(f_1(\tau)\,f_2(\tau))}{\sqrt{(\tau-a)(\tau-b)}}\,\frac{d\tau}{\tau-z}\times$$
$$\times\left(\exp\frac{1}{4\pi i}\int\limits_L\ln\left(\frac{f_1(\tau)}{f_2(\tau)}\right)\frac{d\tau}{\tau-z},\ \exp\frac{-1}{4\pi i}\int\limits_L\ln\left(\frac{f_1(\tau)}{f_2(\tau)}\right)\frac{d\tau}{\tau-z}\right) \tag{2}$$

является решением задачи (1) в классе $B(D)$.

Доказательство следует из непосредственной подстановки (2) в краевое условие (1) и свойств интеграла типа Коши на разомкнутом контуре.

Лемма 2. Пусть $\Phi(z)=(\varphi_1,\ \varphi_2)\in B(D)$ — решение задачи (1). Тогда вектор $\left(\frac{\varphi_1(z)}{\psi_1(z)},\ \frac{\varphi_2(z)}{\psi_2(z)}\right)$ удовлетворяет краевому условию (1) с $f_i(t)\equiv 1$, $i=1,\ 2$. Доказательство очевидно. Из леммы 1 и 2 следует

Лемма 3. Общее решение задачи (1) в классе $B(D)$ имеет вид

$$\Phi(z)=(\varphi_1(z)\psi_1(z),\ \varphi_2(z)\psi_2(z)), \tag{3}$$

где вектор $\Psi(z)=(\psi_1(z),\ \psi_2(z))$ определяется формулой (2), а вектор $\Phi(z)=(\varphi_1(z),\varphi_2(z))$ является общим решением задачи

$$\varphi_1^+(t)\,\varphi_2^-(t)=1,\ \varphi_2^+(t)\,\varphi_1^-(t)=1. \tag{4}$$