Лекция 9-10
Пример проектной работы «Упрощенный язык программирования С0»

При изучении программирования очень полезно знакомство не только с общими принципами и алгоритмами функционирования типовых компонентов программного обеспечения, но и с деталями их реализации.

В качестве примера проектной работы с использованием модульного программирования рассмотрим проект учебной системы программирования С0 – «Си-ноль» - максимально упрощенный язык С [71].

Целью проектирования системы программирования С0 является закрепление знаний языка С и языка ассемблера или первоначальное знакомство с ними, а также понимание связи языка высокого уровня с машинным языком.

Компилятор С0 переводит исходную программу с языка С0 на язык ассемблера IBM PC. Выбор языка ассемблера в качестве выходного (*объектного*) языка позволил значительно упростить компилятор, сделать его однопроходным и более понятным, избежав многочисленных технических деталей генерации машинного кода.

Компилятор получился простым и обозримым (всего лишь около 700 строк на языке С), ***см. папку-приложение к Лекции 9.***

9.1. Возможности и применение компилятора С0

9.1.1. Входной язык С0

Пример 9.1. Программа "Коды символов". Задача: для каждого вводимого символа выводить на экран его десятичный и восьмеричный коды, перед которыми выводить знак равенства. Закончить работу при нажатии клавиши Esc или Ctrl-Break.

Если после запуска программы набирать, например, текст: " язык С0",
то на экране появится строка:

=32=4я=239=357з=167=247ы=235=353к=170=252 =32=40C=67=1030=48=60

После нажатия клавиши <Esc>, которой соответствует код 27, программа прекращает работу.

Программа 9.1. С0-программа "Коды символов"

```
kod (x,baza)
```

Можно написать и короче, в стиле, характерном для С:

```
{ int y;
  if ((y = x / baza) != 0)
     kod (y,baza);
  putchar (x % baza + 48);
}
int c;
main ()
{  c=0;
   while (c != 27)
   {  c = getchar ();
      putchar (61);
      kod (c,10);
      putchar (61);
      kod (c,8);
   }
}
```

```
kod (x,baza)
{ if (x >= baza)
     kod (x/baza, baza);
  putchar (x % baza + 48);
}
int c;
main ()
{ while ((c=getchar())!=27)
   {  putchar (61);
      kod (c,10);
      putchar (61);
       kod (c,8);
   }
}
```

Пояснения к программе "Коды символов". Рекурсивная функция kod выводит на экран (точнее, в стандартный выходной файл) целочисленное значение x в системе счисления с заданным основанием baza ($2 \le baza \le 10$).

Оператор putchar (x % baza + 48); выводит на экран младшую цифру числа x, имеющую значение x%baza (% - остаток от деления). Если x содержит более одной цифры, то перед этим с помощью рекурсивного вызова kod(x/baza,baza); выводится значение старших разрядов числа x, равное x/baza.

Например, если x=723 и baza=10, то сначала рекурсивный вызов функции kod выводит старшие разряды числа x, равные x/10=72, затем выводится младшая цифра x, равная x%10=3.

Переменная y в функции kod используется для демонстрационных целей, без нее можно обойтись. В правом столбце и аналогичной C0-программе функции putn в библиотеке языка C0 (приложение1) показано более короткое решение.

Число 61 используется в этой программе как код символа '=', 48 - код цифры ноль '0'.

Входной язык C0 является подмножеством языка C и содержит данные только целочисленного типа, без массивов. Разрешаются глобальные и локальные переменные, функции с параметрами, рекурсия.

Параметры функций могут быть только входными (ввиду отсутствия операции получения адреса). Поэтому результат работы функции передается только в виде ее значения или присваивается глобальной переменной. В отличие от языка C, параметры не описываются (в языке C нет и стандартной функции putn).

Имеются следующие операторы: оператор-выражение, составной оператор, сокращенный условный оператор (**if** без **else**), цикл с предусловием, оператор возврата.

Оператор-выражение представляет собой выражение, заканчивающееся точкой с запятой:

выражение;

Условный оператор используется для организации ветвлений. В языке С0 разрешен только *сокращенный* условный оператор:

if (выражение) оператор

Для организации повторяющихся действий используется *цикл с предусловием:* **while** (выражение) оператор

Выражение в операторах **if** и **while** считается истинным, если оно не равно нулю. Внутри операторов **if** и **while** разрешено использовать только один оператор. Если в этом месте требуется написать последовательность из нескольких операторов, используется *составной оператор*, т. е. последовательность операторов, заключенная в операторные скобки { и }:

{ оператор_1 ... оператор_n }

Эти скобки превращают последовательность операторов в один составной оператор. В некоторых языках для этой цели используются служебные слова begin и end.

Оператор возврата завершает выполнение подпрограммы (функции) и задает значение функции (если она обладает значением):

return [выражение];

В выражениях допускаются:

- арифметические операции: +, -, *, /, %,
- сравнения: ==, !=, <, >, <=, >=,
- присваивание =,
- скобки ().

Отсутствующие в языке С0 логические операции можно заменить арифметическими. Если, например, X и Y имеют значения 0 или 1, то

! X = 1 – X, X && Y = X * Y, X || Y = X + Y - X * Y,

а в выражениях-условиях операторов **if** и **while**, поскольку они проверяются по принципу «0 – ложь, не 0 – истина», X || Y можно заменить на X + Y.

Примечания.

1. Служебные слова **int**, **if**, **while**, **return** пишутся строчными буквами.

2. Длина имени не ограничена, но учитываются не более 8 символов.

3. Отсутствующий оператор **return** в конце описания функции вставляется автоматически.

Грамматика языка С0 на метаязыке МБНФ [67] имеет следующий вид.

```
программа              ::=  {описание-переменных | описание-функции}...
описание-переменных    ::=  int имя [,имя]...;
описание-функции       ::=  имя ( [имя[,имя]...] )
                            { [описание-переменных]...[оператор]... }
оператор  ::=  [выражение]; | { [оператор]... } |
               if (выражение) оператор | return [выражение];
```

		while (выражение) оператор
выражение	::=	терм [{ + | - | * | / | % | < | > | <= |>= | == | != | = } терм]...
терм	::=	число | имя | имя ([выражение[,выражение]...]) |
		- терм | (выражение)
имя	::=	буква [буква|цифра]...
число	::=	цифра ...
буква	::=	A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
		a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
цифра	::=	0|1|2|3|4|5|6|7|8|9

Эта грамматика описывает не только синтаксис, но и лексику языка С0. К лексике относятся четыре последних правила.

Стандартные функции языка С0. Библиотека std.asm (приложение 1) присоединяется к программе на этапе ее ассемблирования и содержит следующие стандартные функции (все они кроме putn имеются и в языке С).

getchar ()	Вводит с клавиатуры символ с отображением на экране и выдает в качестве значения его код. Реагирует на Ctrl-Break.
getch()	Вводит с клавиатуры символ без отображения на экране и выдает в качестве значения его код. Реагирует на Ctrl-Break.
putchar (x)	Выводит на экран символ с кодом x. Не имеет значения.
putn (x)	Выводит на экран целое значение x. Не имеет значения.

9.1.2. Использование компилятора С0

Для трансляции и выполнения С0-программы используется следующая последовательность команд операционной системы Windows или MS DOS:

c0.exe	трансляция
masm p.asm (или tasm p.asm)	ассемблирование
tlink p.obj	компоновка
p.exe [<входной_файл] [>выходной_файл]	выполнение

Компилятор С0 вводит исходную программу из файла p.c0, помещая результат трансляции на языке ассемблера в файл p.asm. Затем производятся ассемблирование, редактирование связей компоновщиком tlink.exe (из Turbo C или Turbo Pascal) и выполнение программы. Объектный и исполняемый модули транслируемой программы обычно получаются в файлах p.obj и p.exe.

Для ассемблирования необходимо наличие библиотеки стандартных функций - файла std.asm, который присоединяется псевдокомандой INCLUDE к транслируемой программе на этапе ее ассемблирования. Эта команда вставляется транслятором С0.

Транслятор С0 переносит строки исходной программы в получаемую из нее ассемблерную программу в виде строк комментария, начинающихся символом ";". За каждой такой строкой размещается объектный код, т. е. команды, полученные в результате ее трансляции (точнее, после ее ввода).

Чтобы облегчить поиск ошибок, сообщения об ошибках в исходной программе также вставляются транслятором С0 в объектный код в виде строк комментария, содержащего номер (тип) ошибки и символ "^", указывающий на текущую позицию предшествующей исходной строки в момент обнаружения ошибки. В конце объектной программы вставляется итоговое сообщение о количестве обнаруженных ошибок, дублируемое на экране.

Ассемблер при запуске запрашивает имена выходных файлов для бъектной программы, листинга (протокола) трансляции и таблицы перекрестных ссылок. На эти три запроса обычно можно отвечать клавишей <Enter>.

Если же ассемблер выдаст предупреждения (Warning) или сообщения об ошибках (Error), необходимо повторить ассемблирование, указав на этот раз какое-либо имя файла для листинга трансляции, например, p.lst. Просмотрев содержимое этого файла, можно определить, в каких строках ассемблерной программы и соответствующих им строках С0-программы обнаружены ошибки. Обычно эти ошибки вызваны использованием имен несуществующих функций, поскольку компилятор С0 «не знает» имена стандартных функций и не проверяет наличие вызываемых функций.

Сообщения об ошибках транслятора С0

1. Число больше 32767
2. Слишком много имен
3. Требуется int или имя функции
4. Многократно описанное имя
5. Требуется '('
6. Требуется имя
7. Требуется ')'
8. Не удалось открыть входной или выходной файл
9. Недопустимый символ
10. Требуется ','
11. Требуется '{'
12. Требуется ';'
13. Требуется '}'
14. Имя не описано
15. Неверный ограничитель в выражении
16. Неверный тип операнда выражения
17. Несоответствующие типы операндов
18. Неверный тип левого операнда присваивания
19. Нет имени функции
20. Неверный вызов функции
21. Деление на ноль

9.2. Язык ассемблера IBM PC

Рассмотрим лишь минимальный набор средств языка ассемблера IBM PC, достаточный для понимания работы учебного транслятора C0.

В языке ассемблера строчные и заглавные буквы считаются одинаковыми (в данной книге команды для наглядности обычно выделяются заглавными буквами). Символ подчеркивания "_" рассматривается как буква. Используется строчный формат команд: каждая команда пишется с начала новой строки. Команда в общем случае состоит из четырех частей (полей): метки, названия, операндов и комментария, и имеет вид:

```
метка      название   операнд,...,операнд      ; комментарий
```

Поля обычно не содержат пробелов и разделяются пробелами. Любое поле может отсутствовать. Название команды состоит из одной или нескольких букв и определяет ее смысл. Операнды уточняют смысл команды и присутствуют только тогда, когда есть название. Они разделяются запятыми. Метка - это имя, смысл которого зависит от вида команды.

Любой текст от точки с запятой до конца строки рассматривается как комментарий, в частности, вся строка может состоять только из комментария.

Имеются три типа команд: машинная команда, псевдокоманда и макрокоманда (здесь не рассматривается).

Машинная команда - это буквенная запись машинной команды IBM PC, обозначающая некоторую операцию. Название машинной команды определяет вид операции (ниже кратко описаны все используемые операции).

Машинная команда имеет не более двух операндов. Обычно первый из них служит *приемником* информации (сокращенно - прм), а второй - ее *источником* (ист).

Например, команда вида MOV прм,ист обозначает пересылку (move - переслать) из источника в приемник: ист --> прм.

Команда ADD прм,ист обозначает сложение (add - прибавить) приемника с источником и запись суммы в приемник:

прм + ист --> прм

Например, присваивание A = B + 9 можно выполнить следующими командами с использованием двухбайтового регистра AX:

```
M1:  MOV   AX,B          ; B      --> AX
     ADD   AX,9          ; AX+9   --> AX
     MOV   A,AX          ; AX     --> A
```

Операнды *приемник* и *источник* могут быть *адресом* переменной (области памяти), например A, B, или названием *регистра*, например, AX; *источник* может быть также *непосредственным операндом* - числом, участвующим в операции, как например, в команде ADD AX,9.

Адрес переменной (области памяти) может также задаваться в виде смещение[BP], где *смещение* записывается как целое число, например 4[BP]

обозначает адрес, равный содержимому регистра BP, сложенному с заданным смещением 4: BP+4.

Перед машинной командой можно поставить (в том числе в отдельной строке) метку с двоеточием, как M1 в команде MOV приведенного примера. Эта метка обозначает адрес данной команды и используется для переходов к ней из других мест программы командами перехода, например: JMP M1.

Псевдокоманда (директива) не обозначает какую-либо операцию компьютера при выполнении программы. Это - указание ассемблеру, влияющее на трансляцию программы. Метки перед псевдокомандами не содержат двоеточия.

Псевдокоманда DW (define word - определить слово) вида

```
метка  DW  ?
```

пишется в том месте программы, где необходимо разместить область памяти из двух байт (*слово*) для хранения, например, целочисленного значения. Метка обозначает адрес этого значения и соответствует имени переменной.

Например, переменные A и B в приведенном выше примере можно было разместить в памяти псевдокомандами

```
A  DW  ?
B  DW  ?
```

Следующая псевдокоманда создает область из 100 слов:

```
метка  DW  100 DUP (?)
```

Запись 100 DUP (?) обозначает 100-кратное повторение операнда (?).

Псевдокоманда ASSUME (предполагать) указывает ассемблеру имена сегментов команд, стека и данных программы (программе необходимо поместить их адреса в регистры сегментов CS, SS, DS):

```
ASSUME CS:сегмент-команд, SS:сегмент-стека, DS:сегмент-данных
```

Используются также следующие ***псевдокоманды***.

```
                INCLUDE  имя-файла          ; вставить заданный файл
имя-сегмента    SEGMENT                     ; начало сегмента
имя-сегмента    ENDS                        ; конец  сегмента
имя-процедуры   PROC     FAR                ; начало процедуры (дальней)
имя-процедуры   ENDP                        ; конец  процедуры
                END      точка-входа              ; конец  программы
```

В архитектуре процессора IBM PC предусмотрены двухбайтовые *регистры*: AX, BX, DX - для выполнения операций; BP – база для вычисления адреса; SP - указатель стека (адрес последнего записанного байта); CS, SS, DS - для адресов сегментов. AH - старший байт регистра AX; DL - младший байт регистра DX. Регистры флагов – однобитовые регистры, содержащие характеристики результата команды: знак, равенство/неравенство нулю, наличие переполнения и т.п.

Машинные команды

Пересылки:

```
   MOV      прм, ист          ; ист      -->   прм
   LEA      регистр, адрес    ; адрес    -->   регистр
   POP      регистр           ; стек     ==>   регистр   (вытолкнуть)
   PUSH     регистр           ; регистр ==>  стек   (втолкнуть)
Операции:
   ADD      прм, ист    ; сложение              прм + ист   --> прм
   SUB      прм, ист    ; вычитание             прм - ист   --> прм
   NEG      прм         ; изменение знака       -прм        --> прм
   IMUL     ист         ; умножение         AX * ист    --> (DX,AX)
   CWD                  ; слово - в двойное слово AX    --> (DX,AX)
   IDIV     ист         ; деление    (DX,AX) / ист   --> AX   частное
                                     (DX,AX) % ист --> DX   остаток
   INC      прм         ; увеличение на 1       прм+1 --> прм
   DEC      прм         ; уменьшение на 1       прм-1 --> прм
   TEST     AX,AX       ; проверка значения AX и получение флагов
   CMP      AX,BX       ; сравнение AX с BX и получение флагов
Переходы (выполняются по значению флагов):
   JNZ      метка       ; если не равно 0 (после команды TEST)
   JE       метка       ; если =          (после команды CMP)
   JNE      метка       ; если не =       (после команды CMP)
   JL       метка       ; если <          (после команды CMP)
   JG       метка       ; если >          (после команды CMP)
   JLE      метка       ; если <=         (после команды CMP)
   JGE      метка       ; если >=         (после команды CMP)
   JMP      метка       ; безусловный переход
   CALL     имя-проц    ; с возвратом;    адрес-возврата ==> стек
   RET      n       ; убрать из стека адреса возврата и n байт; возврат
```

Лекция 9 Часть 2.

9.3. Объектный код компилятора С0

Разработка структуры объектной программы была самой ответственной и трудной творческой задачей в проектировании транслятора С0.

Объектный код, т. е. результат трансляции С0-программы, представляет собой программу на языке ассемблера IBM PC, состоящую из сегмента кода (команд), сегмента данных и сегмента стека. *Сегмент данных* содержит области для хранения значений глобальных переменных С0-программы, создаваемые псевдокомандами вида:

```
_имя   DW   ?
```

Локальные переменные и параметры каждой функции С0-программы хранятся в сегменте стека в виде кадра, структура которого показана на рис. 9.1. Базовый адрес кадра находится в регистре BP. На рис. 9.2. показан кадр стека функции kod. Каждый элемент стека занимает два байта.

В отличие от глобальных переменных, адреса которых в ассемблерной программе задаются их именами, адреса локальных переменных задаются с помощью смещений относительно регистра BP в виде:

смещение [BP].

В этом случае в объектной программе "Коды символов" адресом переменной с является ее имя, а адрес переменной (параметра) x запишется как 6[BP] адрес переменной y имеет вид -2[BP].

Сегмент команд состоит из процедур на языке ассемблера. Каждая процедура соответствует одной функции C0-программы и имеет такое же имя (перед ним вставляется подчеркивание). Процедуры размещены в таком же порядке, как функции C0-программы.

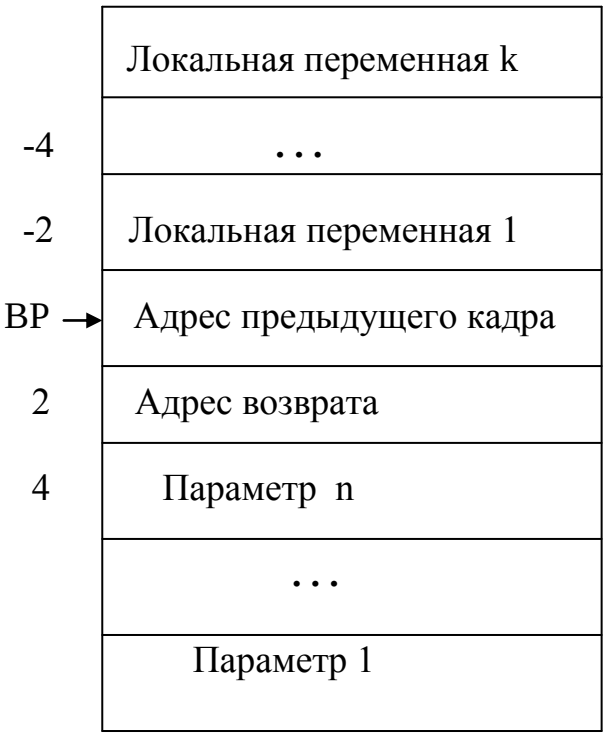

Рис. 9.1. Кадр стека

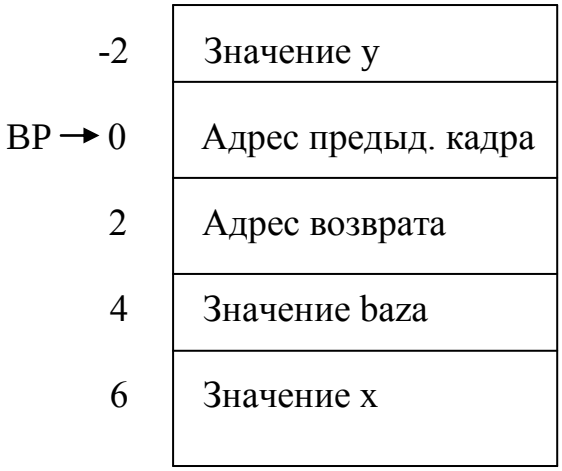

Рис. 9.2. Кадр стека функции kod

Пример 9.2. Структура объектной программы для примера 9.1

```
          ASSUME  CS:KOM_,SS:STEK_,DS:DAN_
;              Сегмент стека
STEK_     SEGMENT STACK
          DW        10000 DUP (?)  ; Область памяти для стека
DNOST_    DW        ?
STEK_     ENDS
;              Сегмент команд
KOM_      SEGMENT
_kod      PROC                ;
         . . .                ; Объектный код функции kod
_kod      ENDP                ;
```

```
_main     PROC                ;
          . . .               ; Объектный код функции main
_main     ENDP                ;
          INCLUDE  std.asm    ; Процедуры библиотечных функций
KOM_      ENDS
;              Сегмент данных
DAN_      SEGMENT
_c        DW   ? ;Область для значения глобальной переменной c
DAN_      ENDS
          END  _main          ; Адрес точки входа в программу
;Компилятор C0 от 10/04/03;
;колич. ошибок 0
```

Структура обычной процедуры (не main):

```
_имя  PROC
; Пролог процедуры       (создание кадра стека)
      PUSH   BP                        ; Сохранение регистра BP
      MOV    BP,SP                     ; Адрес кадра стека
[     SUB    SP,2*кол.лок.переменных   ;Создание лок-х пер-х]
; Объектный код операторов функции
      . . .
; Объектный код оператора return;  обычной функции (не main)
[     ADD    SP,2*кол.лок.перем-х ;Удаление локальных пер-х ]
      POP    BP                     ; Восстановление регистра BP
      RET    2*кол.параметров ;Удаление параметров и возврат
_имя  ENDP
```

Команда PUSH сохраняет значение регистра BP в стеке, MOV пересылает адрес кадра стека в регистр BP, SUB выделяет в кадре стека место для локальных переменных.

Команда ADD удаляет из стека локальные переменные, POP восстанавливает в регистре BP адрес кадра вызывающей функции, RET удаляет из стека параметры функции и выполняет возврат.

Для функции main вместо команды RET используются команды:

```
MOV    AH,4CH
INT    21H
```

Они обеспечивает выход в операционную систему с помощью функции 4CH прерывания номер 21H (типа 21H).

Процедура main имеет такой же пролог, как и обычная процедура, но предварительно должна инициализировать регистры сегментов и указатель стека.

Структура процедуры main:

```
_main  PROC   FAR
       MOV    AX,DAN_   ;Адрес сегмента данных--> регистр DS
       MOV    DS,AX     ;
```

```
        MOV   AX,STEK_  ;Адрес сегмента стека --> регистр SS
        MOV   SS,AX     ;
        LEA   SP,DNOST_ ;Адрес дна стека
; Пролог процедуры       (создание кадра стека)
        . . .
; Объектный код операторов функции main
        . . .
; Объектный код оператора   return;  функции main
[       ADD   SP,2*кол.лок.перем-х ;Удаление локальных пер-х ]
        POP   BP                    ; Восстановление регистра BP
        MOV   AH,4CH                ; Выход в MS DOS
        INT   21H                   ;
_main  ENDP
```

Обозначим объектный код некоторой конструкции С0-программы как объект_код (конструкция).

Ниже в обобщенном виде приведена форма объектного кода разных видов операторов языка С0, которые записаны перед своим объектным кодом в виде комментария языка ассемблера.

; выражение;

```
     объект_код (выражение)
```

; { оператор_1; оператор_2; ... оператор_n; }

```
     объект_код (оператор_1)
     объект_код (оператор_2)
     . . .
     объект_код (оператор_n)
```

; **return** выражение;

```
     объект_код  (выражение)
     ADD     SP,2*кол.лок.переменных
     POP     BP
     RET     2*кол.параметров
```

; **if** (выражение) оператор

```
     объект_код (выражение)
     POP    AX                    ; значение выражения
     TEST   AX,AX                 ; проверка значения
     JNZ    CC_i                  ; истина (не нуль)
     JMP    CC_i+1                ; обход оператора
CC_i:
     объект_код (оператор)
CC_i+1:
```

; **while** (выражение) оператор

```
CC_i:
     объект_код (выражение)
     POP     AX                   ;значение выражения
     TEST    AX,AX                ;проверка значения
```

```
     JNZ     CC_i+1                ;истина
     JMP     CC_i+2                ;выход из цикла
CC_i+1:
     объект_код (оператор)         ;тело цикла
     JMP    CC_i                   ;переход на начало цикла
CC_i+2:
```

Для организации ветвлений и циклов компилятор вставляет в объектную программу метки вида CC_1, CC_2 и т. д., которые здесь обозначены как CC_i. Чтобы вставляемые транслятором метки не могли совпасть с именами функций или глобальных переменных С0-программы, транслятор добавляет к меткам символ подчеркивания “_”, который в языке ассемблера используется наравне с буквами. Этот прием называется *декорированием имен.* Имена функций и глобальных переменных, в свою очередь, приходится декорировать, чтобы они не совпали с именами сегментов, регистров или команд языка ассемблера.

В объектном коде операторов **if** и **while** пару соседних команд перехода нельзя заменить одной командой с противоположным условием, так как в IBM PC условный переход возможен лишь на расстояние не более 128 байтов. Если объект_код (оператор) занимает более 128 байт, возникнет ошибка. В отличие от условного перехода, безусловный переход возможен на любое расстояние.

Объектный код выражения состоит из групп команд, реализующих операции выражения в порядке их выполнения. Результат каждой операции (в том числе значение функции) и значение выражения помещаются в регистр AX.

Если выражение заключено в скобки, например, условие в операторах **if** и **while** или выражение-параметр в вызове функции, его значение затем помещается в стек.

Выражение из одного операнда (без операций) – переменная или константа, переводится в команду вида:

```
                MOV     AX,константа
   или          MOV     AX,адрес-переменной
```

Объектный код операции состоит из следующих шагов.

1. Загрузка 2-го операнда в регистр BX (для присваивания и изменения знака вместо BX используется AX).

2. Загрузка 1-го операнда в регистр AX (не нужен для присваивания и изменения знака).

3. Выполнение операции над AX и BX с записью результата в AX (остаток от деления получается в DX). Функции возвращают значение в регистре AX.

4. Если операция не является последней в выражении, то ее результат помещается в стек.

Указанный порядок загрузки операндов важен, если оба операнда являются результатами других операций и поэтому выбираются из стека (2-й операнд попадает туда позже, чем 1-й операнд, и поэтому должен выбираться в первую очередь).

Операция, операнды которой являются константами, не компилируется, а интерпретируется, т. е. выполняется транслятором.

Объектный код 3-го шага в зависимости от операции имеет вид.

Присваивание = :

```
        MOV       адрес-переменной,AX
```

Операции сравнения (== != < > <= >=):

```
        CMP       AX,BX
        MOV       AX,1          ; Истина
        усл-переход    CC_i
        SUB       AX,AX         ; Ложь - обнуление регистра AX
CC_i:
```

Условие в команде условного перехода совпадает с использованной операцией сравнения:

Операция:	==	!=	<	>	<=	>=
усл-переход:	JE	JNE	JL	JG	JLE	JGE

Арифметические операции:

```
+ :  ADD       AX,BX          ; сложение
- :  SUB       AX,BX          ; вычитание
- :  NEG       AX             ; изменение знака
* :  IMUL      BX             ; умножение
/ или % :          ; частное и остаток целочисленного деления
     CWD            ; преобразование делимого в 4 байта
     IDIV      BX             ; деление
```

Для операции %, если она последняя в выражении, добавляется команда:

```
     MOV       AX,DX
```

Вызов функции имя-функции (выражение_1, ... , выражение_n) :

```
объект_код (выражение_1)      ; параметр_1 --> стек
     . . .
объект_код (выражение_n)      ; параметр_n --> стек
CALL  имя-функции
```

Для ассемблирования необходимо наличие библиотеки стандартных функций - файла std.asm, который присоединяется командой INCLUDE к транслируемой программе на этапе ее ассемблирования (приложение 1). Эта команда вставляется компилятором C0 в конце сегмента кода.

Для наглядности результата трансляции компилятор C0 переносит строки исходной программы в получаемую из нее ассемблерную программу в виде строк комментария, начинающихся символом ";". За каждой такой строкой размещается объектный код, т.е. команды, полученные в результате ее трансляции (точнее - после ее ввода).

Программа 9.2. Объектная программа для C0-программы 9.1 "Коды символов" без двух последних операторов (выведена в две колонки).

```
      ASSUME CS:KOM_,SS:STEK_,DS:DAN_
STEK_      SEGMENT     STACK
           DW    100   DUP (?)
DNOST_     DW    ?
STEK_      ENDS
;kod (x,baza)
KOM_       SEGMENT
;{ int y;
;  if ((y = x/baza)!= 0)
_kod  PROC
      PUSH  BP
      MOV   BP,SP
      SUB   SP,2
      MOV   BX,4[BP]
      MOV   AX,6[BP]
      CWD
      IDIV  BX
      MOV   -2[BP],AX
      PUSH  AX
      MOV   BX,0
      POP   AX
      CMP   AX,BX
      MOV   AX,1
      JNE   CC_1
      SUB   AX,AX
CC_1:
;      kod (y, baza);
      TEST  AX,AX
      JNZ   CC_3
      JMP   CC_2
CC_3:
      MOV   AX,-2[BP]
      PUSH  AX
      MOV   AX,4[BP]
      PUSH  AX
      CALL  _kod
;  putchar (x%baza + 48);
CC_2:
      MOV   BX,4[BP]
      MOV   AX,6[BP]
      CWD
      IDIV  BX
      PUSH  DX
      MOV   BX,48
      POP   AX
      ADD   AX,BX
      PUSH  AX
;}
      CALL  _putchar
      ADD   SP,2
      POP   BP
      RET   4
;int c;
;main ()
_kod  ENDP
_main PROC  FAR
      MOV   AX,DAN_
      MOV   DS,AX
      MOV   AX,STEK_
      MOV   SS,AX
;{  c=0;
      LEA   SP,DNOST_
      PUSH  BP
      MOV   BP,SP
      MOV   AX,0
;   while (c != 13)
      MOV   _c,AX
CC_4:
      MOV   BX,13
      MOV   AX,_c
      CMP   AX,BX
      MOV   AX,1
      JNE   CC_6
      SUB   AX,AX
CC_6:
;   {  c = getchar ();
      TEST  AX,AX
      JNZ   CC_7
      JMP   CC_5
CC_7:
      CALL  _getchar
;      putchar (61);
      MOV   _c,AX
      MOV   AX,61
      PUSH  AX
;      kod (c, 10);
      CALL  _putchar
      MOV   AX,_c
      PUSH  AX
      MOV   AX,10
      PUSH  AX
;   }
      CALL  _kod
      JMP   CC_4
;}
CC_5:
      POP   BP
      MOV   AH,4CH
      INT   21H
;
_main ENDP
      INCLUDE    std.asm
KOM_  ENDS
DAN_  SEGMENT
_c    DW    ?
DAN_  ENDS
      END   _main
; Компилятор C0 от 10/04/03:
; колич. ошибок 0
```

Сообщения об ошибках в исходной программе также вставляются компилятором С0 в объектный код в виде строк комментария, содержащего номер (тип) ошибки и символ "^", указывающий на текущую позицию предшествующей исходной строки в момент обнаружения ошибки. В конце объектной программы вставляется итоговое сообщение о количестве обнаруженных ошибок, дублируемое на экране.

9.4. Общие сведения о компиляторе С0

Компилятор С0 переводит исходную программу с языка С0 на язык ассемблера IBM PC. Выбор языка ассемблера в качестве объектного языка позволил упростить компилятор, сделав его однопроходным, и облегчить понимание использованных методов трансляции, избежав многочисленных деталей генерации машинного кода.

Компилятор написан на языке С и состоит из 18 модулей - подпрограмм (функций), использующих около 30 глобальных переменных (см. приложение).

Таблица имен. Основной глобальной структурой данных компилятора СО является таблица имен транслируемой программы, каждый элемент которой состоит из трех полей: имя, вид именуемого объекта (переменная/ функция) и смещение в кадре стека (для локальных переменных и параметров). Таблица имеет двухуровневую структуру (глобальные имена и локальные имена текущей транслируемой функции) и представляет собой стек в виде массива tabim из четырех частей, границы индексов которых задаются глобальными переменными: kolglb - количество глобальных имен; kolim - общее количество имен и kpar - количество имен после включения параметров (см. рис. 9.4).

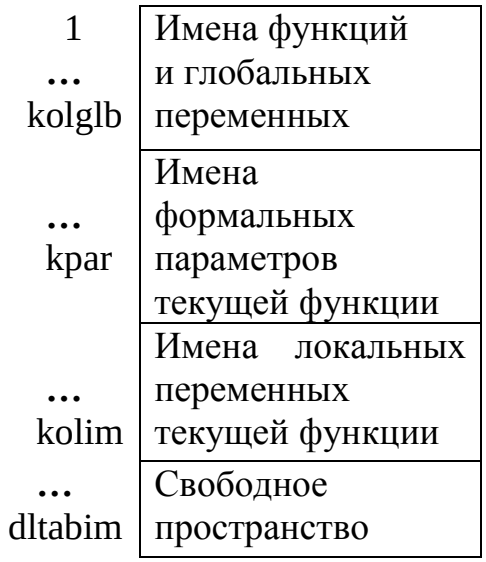

Рис. 9.3.Структура таблицы имен

После трансляции описания текущей функции определенные в ней имена удаляются из таблицы (kolim становится равным kolglb). В глобальную часть таблицы заносятся имена глобальных переменных, определенных между описаниями функций, и имя следующей функции. Затем в таблицу заносятся имена формальных параметров и локальных переменных этой функции, после чего определяются их смещения в кадре стека. Поиск и включение имен в таблицу выполняют подпрограммы pozic и vkluch.

Синтаксический и семантический анализ. Трансляция выражений в С0 производится методом стека с приоритетами. Остальные конструкции языка

(операторы и описания) транслируются методом рекурсивного спуска. Любым из этих методов можно было бы транслировать всю программу.

Конструкцию "программа" языка С0 анализирует главная программа компилятора main, "описание (определение) функции" - подпрограмма oprfun, "описание переменных" - opisper, "оператор" - operator, "выражение" - virag.

Последовательность операторов вида "[оператор] **. . .**}", встречающаяся в описании функции и составном операторе, анализируется подпрограммой poslop. *Синтаксические программы* main, opisper, poslop, operator, virag являются в то же время и *семантическими программами*, т. к. генерируют объектный код для своих конструкций.

Подпрограмма operator соответствует рекурсивному правилу грамматики С0 для конструкции "оператор" (внутри оператора может находиться другой оператор) и поэтому сама является рекурсивной. Это обеспечивает трансляцию многоуровневых вложений составных и условных операторов и циклов.

Структура синтаксической программы (кроме virag) практически однозначно определяется структурой соответствующего правила грамматики.

Пример 9.3. Трансляция оператора **if** языка С0.

оператор ::= **if** (выражение) оператор

Ниже приведен полученный из этой грамматики фрагмент подпрограммы трансляции оператора, написанный на псевдокоде - неформальной версии языка С).

```
operator(...)
{       . . .           /* описание локальных переменных */
 int kmet,nmet;
        . . .
 if (leksema==слово_if) /* if - условный оператор        */
 {  чтение лексемы;
  if (лексема=='(')
  {/* трансляция выражения-условия до правой скобки      */
     virag(')');
     nmet=kmet+1;           /*     номер метки CCi       */
     Вставить в объектный код команды:
          POP AX
          TEST AX, AX
          JNZ _CCi          /* где i номер метки nmet    */
          JMP _CCi+1
     _CCi:
  }
  else  oshibka(5);     /* требуется(                    */
  operator();           /* трансляция оператора          */
                        /* и вывод его объектного кода   */

     Вставить в объектный код метку:
     _CCi+1:
  }
 else
     . . .        /* трансляция других типов операторов */
 }
```

Лексический анализ. Лексический анализ выполняет программа chleks, читает очередную лексему входного текста и присваивает ее значение глобальной переменной leksema, используя для чтения очередного символа подпрограмму chsim.

Подпрограмма чтения символа chsim вводит исходный текст строками длиной до 80 символов. Это делается для того, чтобы после ввода строки можно было бы сразу же вставить ее в виде комментария в объектную программу. В вызывающую программу передается сначала первый символ текущей строки, а затем (при последующих вызовах) - остальные ее символы (по одному). После исходной строки в объектную программу вставляются команды, полученные в процессе чтения и трансляции этой строки, а также сообщения об обнаруженных в ней ошибках. Если бы входной текст вводился посимвольно, этого сделать не удалось бы.

Диагностика и нейтрализация ошибок. Диагностика и нейтрализация ошибок требуется не только транслятору, но и любой программе. При обнаружении ошибки необходимо сообщить о ней. Дальнейшие действия программы могут быть различными.

Самое простое - прекратить работу, но это не всегда приемлемо. По возможности желательно продолжить работу, хотя бы для обнаружения остальных ошибок. Тогда необходимо как-то нейтрализовать ("исправить") ошибку, исходя из наиболее вероятного предположения о характере ошибки.

В большинстве случаев это предположение подтверждается, и программа будет действовать разумно, однако всегда возможны другие варианты ошибки, при которых действия программы будут выглядеть бессмысленными. Поэтому не следует всегда буквально воспринимать сообщения любой программы об обнаруженных ошибках.

В компиляторе C0 подпрограмма oshibka (n) вставляет в выходной файл сообщение об ошибке типа n в виде строки с символом "^", расположенным в позиции текущего символа входной строки, и значением n. Подпрограмма test проверяет принадлежность лексемы множеству допустимых лексем и в случае ошибки пропускает часть теста до появления лексемы, принадлежащей множеству лексем возобновления анализа текста. Множество лексем представляется характеристическими вектором.

9.5. Трансляция выражений в компиляторе C0

В компиляторе C0 метод стека с приоритетами реализован в подпрограмме virag. Задача этой подпрограммы - определение порядка выполнения операций выражения.

```
ограничитель ::= ( | ) | , | ; | = | == | != | < | > |
                 <= | >= | + | - | * | / | %
```

Таблица 9.1.
Приоритеты ограничителей

Ограничитель	Приоритет
(	0
) , ;	1
=	2
== !=	3
> < >= <=	4
+ - (бинарный)	5
* / %	6
- (унарный)	7

К ограничителям относятся знаки операций, скобки, а также запятая, разделяющая фактические параметры вызова функции, и точка с запятой, обозначающая конец оператора-выражения. Ограничителям присвоены приоритеты по старшинству операций, как в языке С (табл. 9.1).

Для определения порядка выполнения операций используется стек из глобальных параллельных массивов: sz, st, sogz, spr с указателем i.

Элемент стека содержит секцию: терм, ограничитель и его приоритет. В компиляторе С0 они хранятся в виде четырех значений: st[i] - тип терма (0 - отсутствующий терм, 1 - число, 3 - имя, 4 - число в стеке выполнения (результат выполнения операции)); sz[i]- значение терма (для имени указывается его позиция в таблице имен, для типов 0 и 4 значение не используется); sogr[i] и spr[i] – ограничитель и приоритет.

Запятая и точка с запятой играют роль закрывающей скобки, соответственно, для фактического параметра и выражения в целом.

Транслятор С0 сочетает компиляцию с интерпретацией. Если операнды являются константами, то операция интерпретируется, т.е. выполняется, и в стек трансляции помещается результат операции. В противном случае операция компилируется, т. е. генерируется (выводится в объектную программу) реализующая операцию группа команд, а в стек трансляции записывается указание о том, что результат операции будет находиться в стеке выполнения (типу терма st[i-1] присваивается 4).

При этом подпрограмма zopreg генерирует различные команды загрузки операнда (терма) в регистр AX или BX в зависимости от типа, т.е. местонахождения терма.

Выталкивание открывающей скобки выполняется подпрограммой vitsk. Если перед открывающей скобкой имеется имя функции, то при ее выталкивании генерируется команда вызова этой функции с предварительной записью в стек ее параметра, если это необходимо.

Выталкивание открывающей скобки подвыражения (перед которой отсутствует имя функции) сводится к продвижению находящейся в верхушке стека информации о значении подвыражения на одну позицию вглубь стека.

Пример 9.4. Трансляция фрагмента программы "Коды символов". На рис. 9.4 показана таблица имен при трансляции описания функции kod, а на рис. 9.5 - кадр стека функции kod.

```
kod (x,baza)
{ int y;
  if ((y = x/baza) != 0)
     kod (y,baza);
  putchar (x%baza + 48);
}
int c;
main ()
{  c=0;
   while (c!=13)
   {  c = getchar ();
      putchar (61);
      kod (c,10);
      putchar (61);  kod (c,8);
   }
}
```

	Имя	Вид объекта	Смещение
kolglb=1	kod	3 (функция)	-
2	x	1 (перемен.)	6
kpar=3	baza	1 (перемен.)	4
kolim=4	y	1 (перемен.)	-2
		...	

Рис.9.4. Таблица имен при трансляции функции kod

Трансляция этой программы происходит следующим образом.

Для трансляции описания функции kod программа main вызывает подпрограмму oprfun, которая заносит в таблицу имя функции kod, ее формальные параметры x, baza и локальную переменную y и определяет их смещения в кадре стека. После этого на весь период трансляции функции kod, таблица имен приобретает вид, показанный на рис. 9.4.

-2	Значение y
0	Адрес предыдущего кадра
2	Адрес возврата
4	Значение baza
6	Значение x

Рис. 9.5. Кадр стека функции kod

На рис. 9.6 приведена трассировочная таблица трансляции оператора

if ((y = x / baza) != 0) kod (y,baza);.

При трансляции этого оператора вызванная из oprfun подпрограмма poslop вызывает подпрограмму operator для трансляции оператора if. В свою очередь, operator вызывает подпрограмму virag для трансляции выражения-условия, а затем (рекурсивно) вызывает подпрограмму operator, которая вызывает virag для трансляции вложенного оператора-выражения.

Программа virag транслирует выражение по секциям. Обработка секции состоит из трех шагов.

Leksema = if	(	(	y =	x /	baza)		
ogr =	(	(	=	/	)		
prior =	0	0	2	6	1		
Sz, st, sogr, spr=	(	(	(	(	(	(	(	(
		(	(	(	(	(	(	Стек
			y =	y =	y =	y =	Стек	
				x /	x /	Стек		
					baza			

Объектный код, генерируемый при выталкивании операции из стека:

```
MOV   BX,4[BP]
MOV   AX,6[BP]
CWD                POP   AX
IDIV  BX           MOV -2[BP],AX
PUSH  AX           PUSH  AX
```

Продолжение таблицы вправо:

!=	0)		
!=	)		
3	1		
(	(	(	Стек
Стек!=	Стек!=	Стек	
	0		

Kod (	y ,	Baza)		;
(	,	)		
0	1	1		
Kod (	kod (	kod (	Стек	
	y	Baza		

```
                                    MOV  AX,-2[BP]     CC_2:
  MOV  BX,0       POP    AX       PUSH  AX
  POP  AX         TEST   AX,AX
  CMP  AX,BX      JNZ    CC_3            MOV   AX,4[BP]
  MOV  AX,1       JMP    CC_2            PUSH  AX
  JNE  CC_1     CC_3:                    CALL  KOD
  SUB  AX,AX
CC_1:
  PUSH  AX
```

Рис. 9.6. Трассировочная таблица трансляции оператора
if ((y = x / baza) != 0) kod (y,baza);

1. Если предыдущий ограничитель не является закрывающей скобкой, то читается терм и записывается в стек (для этого в стек вталкивается новый элемент). Дело в том, что после закрывающей скобки секция никогда не содержит терма. Роль терма для нее играет уже находящаяся в стеке информация о значении заключенного в скобки подвыражения.
2. Читается ограничитель секции. Если он не является открывающей скобкой, то его приоритет сравнивается с приоритетом последнего ограничителя, помещенного в стек (если в стеке есть ограничители). Пока приоритет текущего ограничителя меньше приоритета последнего ограничителя стека, этот более приоритетный ограничитель выталкивается из стека. В случае равенства приоритетов выталкивание происходит только тогда, когда соответствующие операции должны выполняться слева направо (как сложение и вычитание или умножение и деление), а не справа налево, как, например, присваивание. При выталкивании ограничителя из стека реализуется операция, соответствующая этому ограничителю. Если операндами операции являются константы, то она выполняется (интерпретируется). В противном случае операция компилируется, т. е. в выходную программу выводится объектный код операции.
3. Если ограничитель не является закрывающей скобкой, то по окончании цикла выталкивания он записывается в последний элемент стека (новый элемент не создается). Для ускорения работы туда же помещается его приоритет. Закрывающая скобка в стек не записывается. Вместо этого из стека выталкивается соответствующая открывающая скобка. При выталкивании открывающей скобки подвыражения (перед которой терм отсутствует) терм из вершины стека (информация о значении подвыражения) продвигается на один элемент вглубь стека. Если же открывающая скобка находится перед списком параметров вызова функции (ее термом является имя функции), то при ее выталкивании в выходной файл выводится команда вызова этой функции CALL. Если функция имеет параметр-выражение, то к этому моменту в выходную программу уже будет помещен объектный код этого выражения, т. е. команды, вычисляющие значение параметра и вталкивающие его в стек. В этом случае последний терм стека имеет тип 4 (число, помещенное в стек выполнения). В противном случае параметр не содержит операций (представляет собой константу или переменную). Тогда генерируются команды, помещающие его значение в стек. Запятая разделяет параметры вызова функции и служит как бы закрывающей скобкой предыдущего параметра и открывающей скобкой следующего за ней параметра. Поэтому она обрабатывается подобно закрывающей скобке, но открывающая скобка не выталкивается, а остается в стеке. Таким же образом обрабатывается точка с запятой.

9.6. Расширение возможностей языка и транслятора С0

Рассмотрим пример разработки расширения С0. Задача: реализовать в языке С0 операции ++ и – – языка С (увеличения и уменьшения на 1).

Планирование работы

В принципе, план разработки и документирования *изменений* программы похож на план *первоначальной разработки* и документирования программы.

Сначала определяется новая структура входных и выходных данных, затем уточняются глобальные данные и модульная структура программы, внутренние данные и алгоритмы ее модулей, составляется план тестирования и отладки модулей, разрабатываются тесты, драйверы, имитаторы и другие отладочные средства и, наконец, производится отладка.

. В частности, изменения компилятора С0 удобно разрабатывать и реализовать в том же порядке, в каком он проектировался, и в каком дается его описание в разделах 9.1 -9.5. Таким образом, план разработки изменений транслятора состоит из следующих этапов.

1. Разработать изменения *грамматики* входного языка С0. Придумать примеры программ с использованием новых конструкций. Они облегчают разработку программ транслятора и послужат *тестами* для их отладки.

2. Определить, как изменится *семантика* входного языка, т. е. структура *объектного кода* программы в целом, новых конструкций языка, а возможно, и ранее существоваших конструкций. Составить объектный код для разработанных на этапе 1 тестовых примеров С0-программ.

2. Продумать, как изменения входного языка и объектного кода повлияют на *структуры данных* транслятора, в частности, на описание глобальных данных.

3. Уточнить *модульную структуру*, в частности, определить множество модулей (подпрограмм) транслятора, которые потребуется изменить.

4. Разработать *программы* новых и изменяемых модулей.

5. Выписать список всех заменяемых, вставляемых и удаляемых строк транслятора с указанием номеров строк в порядке *возрастания* этих номеров.

Вносить же эти изменения необходимо, наоборот, в порядке *убывания* номеров строк, чтобы ранее сделанные изменения не влияли на номера строк последующих изменений.

5. Разработать *план отладки* (по одному модулю либо группами модулей) и *отладочные средства* (тесты, драйверы, имитаторы, отладочные подпрограммы), необходимые для автономной отладки новых и измененных подпрограмм транслятора.

6. Провести автономную и комплексную отладку измененного транслятора.

Для включения новых операций в *грамматику* С0 необходимо добавить правило: терм ::= ++ имя | имя ++ | -- имя | имя --

Операции увеличения и уменьшения на 1 можно реализовать следующим *объектным кодом*.

; ++ имя	; имя ++	; -- имя	; имя --
INC адрес	MOV AX,адрес	DEC адрес	MOV AX,адрес
MOV AX,адрес	INC адрес	MOV AX,адрес	DEC адрес

Для глобальной переменной адрес - это имя, для локальной переменной адрес записывается в виде: смещение[BP].

Если операция в выражении не последняя, а промежуточная, добавляется команда вталкивания результата в стек: PUSH AX.

В трансляторе C0 добавятся новые *значения лексемы* uvel и umen, что потребует *изменения*: описаний типа tipleks и глобальных переменных tpr, st2, значения nvir множества начальных лексем выражения, подпрограммы чтения лексемы chleks. Новые лексемы uvel и umen будут вставлены в описание типа tipleks между лексемами plus и prisv, чтобы попасть в диапазон операций для вычисления приоритета в строке 387.

В подпрограмму operac добавляется генерация кода для новых операций.

Таким образом, потребуются следующие *изменения текста* транслятора C0.

Заменить строки:

```
79  /* приоритеты: + - * / % == != < > <= >= ( ) , ; ++ -- = */
80  int tpr[16]= { 5,5,6,6,6,3, 3, 4,4,4, 4, 0,1,1,1,7, 7, 2 };

84     mravn,bravn,lskob,pskob,zpt,tchzpt,uvel,umen,prisv,
85     flskob,fpskob,ifsl,intsl,retsl,whilesl};

102  long int st2[27]=            /* st2[i]=2**i (i=0..26)    */
105       8388608,16777216,33554432,67108864};

133
nvir=st2[ident]|st2[chislo]|st2[minus]|st2[lskob]|st2[uvel]|st2[umen];
```

Вставить после строки 458:

```
  char  kop[3];    /* код операции: INC или DEC */
```

Заменить строку:

```
460 if  (*t1<=1 && t2==1 && op!=prisv && op!=uvel && op!=umen)
```

Вставить после строки 489:

```
 else if (op==uvel || op==umen) /* ++ или -- */
 {  kop = (op==uvel)? "INC" : "DEC" ;
    if (*t1 == 0) /*нет 1-го операнда: префиксная оп-ция */
       if(t2==3 && tabim[z2].vidob==1) /*2-й оп-д - пер-я*/
       {  fprintf(fvih,"\t%s\t%s\n",kop,adrper(z2));
          zopreg(z2,t2,"AX");
       }
```

```
         else oshibka(16);       /* неверный тип операнда */
      else
         if(t2==0&&*t1==3&&tabim[*z1].vidob==1)/*имя++|имя--*/
         {  zopreg(*z1,*t1,"AX");
            fprintf(fvih,"\t%s\t%s\n",kop,adrper(*z1));
         }
         else oshibka(16);       /* неверный тип операнда */
 }
```

Вставить после строки 620:

```
else if (*usim=='+' || *usim=='-') /* + или - или ++ или -- */
{  c=*usim;
   if (chsim() == c)         /* ++ или -- */
   {  if (c == '+')   leksema=uvel;         /* ++ */
      else            leksema=umen;         /* -- */
      chsim();   /* чтение символа след. лексемы */
   }
   else
      leksema=leksim[c];     /* + или - */
}
```

Планирование отладки. План отладки программы составляется на основании ее модульной структуры, которая обычно изображается в виде схемы взаимодействия модулей (раздел 8.2.1).

Модульная структура компилятора С0 представлена не в виде схемы, а в форме таблицы, в которой перечислены все вызываемые подпрограммы каждого модуля (приложение 2). Отладку измененного компилятора С0 удобно проводить в два этапа.

1. Автономная отладка подпрограммы *чтения лексемы* chleks. Параллельно и независимо от этого - автономная отладка группы подпрограмм *трансляции выражения*: virag, operac, adrper, zopreg с использованными в них подпрограммами vitsk и adrper. Подпрограмму virag, хотя она и не изменялась, необходимо проверить, т. к. не исключена возможность, что ее надо было бы изменить. Кроме того, без нее трудно убедиться в правильности трансляции выражений с новыми операциями.

2. Затем необходимо тестировать транслятор целиком (комплексно), в том числе убедиться, что не нарушилась его работа на старых тестах (без операций ++ и --).

Разработка тестов. Тестами послужат операторы-выражения с операциями ++ и --. Кроме ++ и --, требуются операции + и -, чтобы проверить правильность чтения этих лексем. Поскольку + и - имеют более низкий приоритет, для проверки учета приоритетов желательны также операции более высокого приоритета, чем ++ и -- (унарная операция изменения знака “-“).

С учетом этих соображений в качестве теста используем следующую последовательность выражений:

++x; y++; -x--; --y; x=-y+++5; x=++y+5; y=--x-5; x=y---5;

Переменная leksema относится к перечислимому типу tipleks и описана в строках 82 – 87 *измененного* компилятора C0 следующим образом:

```
82 enum  tipleks                    /* тип лексемы:              */
83 {osh,ident,chislo,plus,minus,umn,del,ost,ravn,neravn,men,bol,
84     mravn,bravn,lskob,pskob,zpt,tchzpt,uvel,umen,prisv,
85     flskob,fpskob,ifsl,intsl,retsl,whilesl};
86 enum tipleks leksema;            /* текущая лексема           */
```

В соответствии с этим описанием, тип enum tipleks содержит значения 0, 1, 2, …, обозначаемые именами: osh = 0, ident =1, chislo = 2, plus =3, minus = 4, umn = 5, del = 6, ost = 7, ravn = 8, neravn = 9, men = 10, bol = 11, mravn = 12, bravn= 13 , lskob = 14, pskob = 15, zpt = 16, tchzpt = 17, uvel = 18, umen = 19, prisv = 20, flskob = 21, fpskob = 22, ifsl = 23, intsl = 24, retsl = 25, whilesl = 26.

При чтении из файла p.c0 разработанного тестового текста

++x; y++; -x--; --y; x=-y+++5; x=++y+5; y=--x-5; x=y---5;

подпрограмма chleks должна выдавать следующую последовательность лексем (в числовой записи): 18, 1, 17, 1, 18, 17, 4, 1, 19, 17, 19, 1, 17, 1, 20, 4, 1, 18, 3, 2, 17, 1, 20, 18, 1, 3, 2, 17, 1, 20, 19, 1, 4, 2, 17, 1, 20, 1, 19, 4, 2, 17.

Условно будем считать x глобальной переменной, а y - локальной переменной со смещением 2. Если текст теста из файла p.c0 (в отсутствие подпрограммы chleks - в числовом виде) подавать в цикле по одному выражению в подпрограмму virag, то она вместе с подчиненными ей программами должна генерировать в файле p.asm следующий объектный код (записан в три колонки):

```
; ++x;                    INC  WORD PTR 2[BP]     MOV  AX,x
  INC  x                  PUSH AX                 PUSH AX
  MOV  AX,x               MOV  BX,5               MOV  BX,5
; y++;                    POP  AX                 POP  AX
  MOV  AX,2[BP]           ADD  AX,BX              SUB  AX,BX
  INC  WORD PTR 2[BP]     PUSH AX                 PUSH AX
; -x--;                   POP  AX                 POP  AX
  MOV  AX,x               MOV  x,AX               MOV  2[BP],AX
  NEG  AX               ; x=++y+5;              ; x=y---5;
  PUSH AX                 INC  WORD PTR 2[BP]     MOV  AX,2[BP]
  POP  AX                 MOV  AX,2[BP]           DEC  WORD PTR 2[BP]
  DEC  x                  PUSH AX                 PUSH AX
; --y;                    MOV  BX,5               MOV  BX,5
  DEC  WORD PTR 2[BP]     POP  AX                 POP  AX
  MOV  AX,2[BP]           ADD  AX,BX              SUB  AX,BX
; x=-y+++5;               PUSH AX                 PUSH AX
  MOV  AX,2[BP]           POP  AX                 POP  AX
  NEG  AX                 MOV  x,AX               MOV  x,AX
  PUSH AX               ; y=--x-5;
  POP  AX                 DEC  x
```

Разработка драйверов и имитаторов. Далее приведены драйверы и имитаторы для разработанного плана отладки и тестов измененного компилятора С0. Для упрощения драйверов с помощью директивы #include в них вставляются описания глобальных переменных, вынесенные в файл glb.c из текста транслятора (строки 47 -109).

Из модульной структура компилятора С0 (приложение 2) видно, что для автономной отладки функции chleks требуется драйвер и имитаторы вызываемых из нее подпрограмм chsim и oshibka (рис. 9.7).

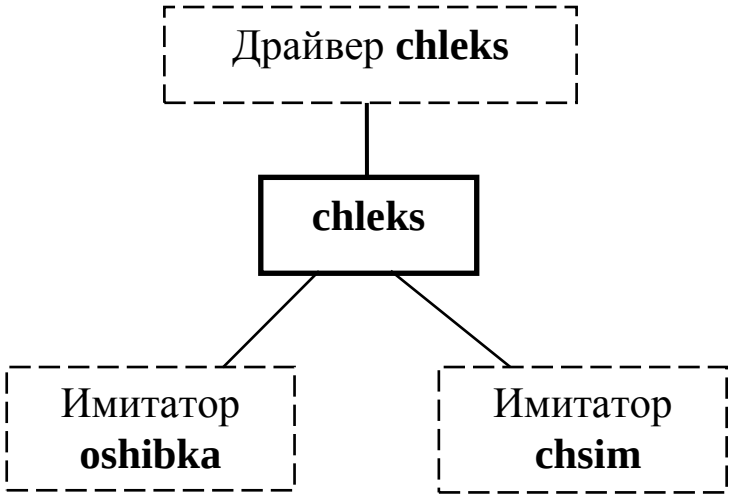

Рис. 9.7. Программа для тестирования модуля chleks компилятора С0

Драйвер для автономной отладки функции chleks разработан на основе фрагментов функции main компилятора С0. Драйвер читает файл p.c0 с помощью подпрограммы chleks и выводит в файл p.asm полученную последовательность лексем в числовом виде.

```
     /*********************************************/
     /*               Драйвер chleks                */
     /*********************************************/
#include <stdio.h>
#include "glb.c"
main()
{
   int  i;
   for (i=0; i<=255; i++) leksim[i]=osh;
   leksim['+']=plus;  leksim['-']=minus;  leksim['*']=umn;
   leksim['/']=del;   leksim['%']=ost;    leksim['=']=prisv;
   leksim['<']=men;   leksim['>']=bol;    leksim['(']=lskob;
   leksim[')']=pskob; leksim['{']=flskob; leksim['}']=fpskob;
   leksim[',']=zpt;   leksim[';']=tchzpt;
```

```
   vhstr[0]=' '; vhstr[1]='\0'; usim=vhstr;
   kolglb=1;
   kolim=2;
   ef=0;
   fvh=fopen("p.c0","r");             /* открыть входной файл  */
   fvih=fopen("p.asm","w");           /* открыть выходной файл */
   if ((fvh==NULL) || (fvih==NULL))
      printf("Файлы не открылись");
   else
   {  while (!ef)
      {
         chleks();
         fprintf(fvih,"%d ",leksema);
      }
      fclose(fvh);
      fclose(fvih);
   }
}
```

Имитатор чтения символа - это упрощенная функция chsim, из которой удалены операторы, вставляющие строки исходного текста в файл p.asm:

```
    /*********************************************/
     /*      Имитатор чтения символа  chsim       */
     /*********************************************/
chsim()
{
   if (feof(fvh)) {ef=1;  return *usim=EOF;}
   return *usim=getc(fvh));
}
```

Поскольку разработанный тест не содержит ошибок, при его чтении функция oshibka не должна вызываться, и ее имитатор можно было бы сделать пустым. Тем не менее, полезно на всякий случай оставить в имитаторе вывод упрощенного сообщения, чтобы зафиксировать случаи возможного «обнаружения» несуществующих «ошибок» входного текста из-за ошибок в отлаживаемой программе chleks:

```
     /*********************************************/
     /*              Имитатор oshibka             */
     /*********************************************/
oshibka(int n)
{
   fprintf(fvih,"Ошибка: ^%d\n",n);
}
```

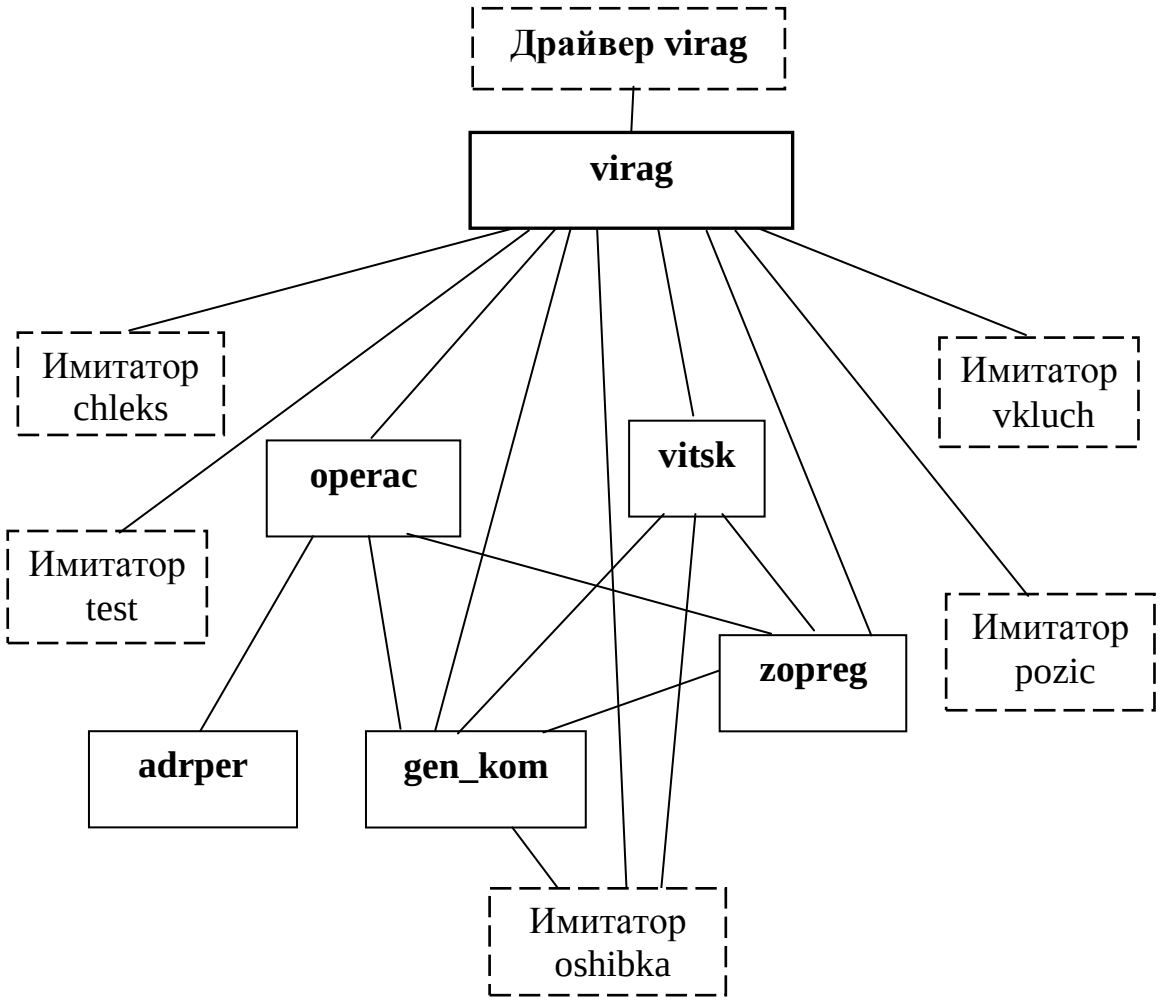

Рис. 9.8. Программа для тестирования модулей трансляции выражения: virag, operac, vitsk, zopreg, gen_kom и adrper компилятора C0

В соответствии с модульной структурой компилятора C0 для тестирования модулей трансляции выражения: virag, operac, vitsk, zopreg, gen_kom и adrper необходим драйвер и имитаторы вызываемых из этих модулей функций: chleks, test, vkluch, pozic и oshibka (рис. 9.8).

Разработанный для отладки входной текст представляет собой последовательность выражений, заканчивающихся точкой с запятой, и поэтому драйвер циклически обращается к модулю virag, пока не кончится файл p.c0. Перед каждым обращением читается первая лексема очередного выражения.

```
     /*********************************************/
     /*                Драйвер virag                */
     /*********************************************/
#include <stdio.h>
#include "glb.c"
main()
{  int  i;
   kolglb=1; kolim=2;
   tabim[1].imja="x"; tabim[1].vidob=1;   /* имя x глобальное */
   tabim[2].smesch=2; tabim[2].vidob=1;   /* имя y локальное  */
```

```
   ef=0;
   fvh=fopen("p.c0","r");              /* открыть входной файл  */
   fvih=fopen("p.asm","w");            /* открыть выходной файл */
   if ((fvh==NULL) || (fvih==NULL))
      printf("Файлы не открылись");
   else
   {  while (!ef)
      {  chleks();
         virag(0L);
      }
      fclose(fvh);
      fclose(fvih);
   }
}
```

В отсутствие функции чтения лексемы входной текст содержит подготовленную вручную последовательность числовых значений лексем, а имитатор функции chleks просто вводит очередное число.

```
     /*********************************************/
     /*              Имитатор chleks              */
     /*********************************************/
chleks()
{  fscanf(fvh,"%d ",&leksema);
   return leksema;
}
```

Разработанные тесты: ++x; y++; -x--; --y; x=-y+++5; x=++y+5; y=--x-5; x=y---5; задаются в файле p.asm в виде числовых значений лексем, разделенных пробелами и/или символами новой строки: 18 1 17 1 18 17 4 1 19 17 19 1 17 1 20 4 1 18 3 2 17 1 20 18 1 3 2 17 1 20 19 1 4 2 17 1 20 1 19 4 2 17.

При этом вид ожидаемого объектного кода определен из условия, что x является глобальной переменной, а y - локальной переменной со смещением 2.

Драйвер virag моделирует соответствующую таблицу имен, содержащую под индексом 1 глобальную переменную x, а под индексом 2 - локальную переменную y, имеющую смещение 2:

```
   kolglb=1; kolim=2;
   tabim[1].imja="x"; tabim[1].vidob=1;   /* имя x глобальное */
   tabim[2].smesch=2; tabim[2].vidob=1;   /* имя y локальное  */
```

При обработке разработанного тестового текста обращения к имитатору pozic для поиска позиции (индекса) очередного имени будут происходить в том порядке, в котором в этом тексте расположены имена x и y. Под видом «найденных» индексов имен x и y имитатор pozic будет выдавать размещенные в таком же порядке числа из глобального массива zi.

```
     /*********************************************/
     /*              Имитатор pozic               */
     /*      позиция имени id в таблице tabim     */
     /*********************************************/
int nt = 0;  /* 1 2 3 4 5 6 7 8 9 10 11 12  N имени */
int zi[20] =  { 1,2,1,2,1,2,2,1,1,2, 1, 2   };

     /* В тесте x y x y x y y x x y  x  y          */
pozic(int tabl)        /* вид таблицы: kolglb/kolim */
{  return zi[++nt] ;
}
```

Для разработанного теста имитатор vkluch вызываться не должен, но на тот случай, когда это все-таки произойдет из-за ошибок в отлаживаемой программе, в имитаторе предусмотрен вывод соответствующего сообщения.

```
     /*********************************************/
     /*              Имитатор vkluch              */
     /*     включение имени id в таблицу имен     */
     /*********************************************/
vkluch()
{
   fprintf(fvih,"\n vkluch \n"); /* если по ошибке вызовут */
}
```

Имитатор oshibka можно использовать такой же, как для отладки chleks. Имитатор test оставлен пустым:

```
     /*********************************************/
     /*   Имитатор test: проверка лексемы,        */
     /*   сообщение об ошибке и ее нейтрализация  */
     /*********************************************/
test(long s1, long s2, long n)
/* s1 - множество допустимых лексем (логич. шкала)  */
/* s2 - множество дополнительных лексем для пропуска*/
/* n  - номер ошибки                                */
{
}
```

Приведенный пример имеет в определенной степени демонстрационный характер для иллюстрации возможной методики и приемов автономной отладки программных модулей.

9.7. Дополнительные упражнения

1. Составить С0-программы для решения следующих задач.

а) Подсчитать количества повторений во входном тексте сочетаний "РФ" и "РТ".

Указание: использовать переменную "символ, предшествующий текущему".

б) Вывести входной текст на экран с нумерацией строк.

в) Вывести входной текст на экран, заменяя строчные латинские буквы заглавными.

Указание: использовать тот факт, что в коде ASCII коды строчных латинских букв возрастают по алфавиту через 1, и таким же свойством обладают коды заглавных букв.

г) Входной текст состоит из слов, разделенных пробелами. Подсчитать количество слов и их среднюю длину.

Указание: использовать переменную "символ, предшествующий текущему".

д) Входной текст состоит из слов, разделенных пробелами. Найти максимальную длину слова.

е) Входной текст состоит из слов, разделенных пробелами. Подсчитать количество слов, начинающихся с буквы 'a'.

Указание: использовать переменную "символ, предшествующий текущему".

ж) Входной текст состоит из слов, разделенных пробелами. Подсчитать количество слов, кончающихся буквой 'a'.

Указание: использовать переменную "символ, предшествующий текущему".

з) Вывести таблицу кодов символов, лежащих в диапазоне, задаваемом двумя вводимыми символами.

и) Подсчитать во входном тексте количество слов, у которых первый символ с совпадает с последним.

Указание: использовать переменную "символ, предшествующий текущему".

к) Составить функцию ввода целого числа со знаком getn(), которая пропускает пробелы и символы новой строки перед числом, читает число и выдает его значение в качестве значения функции. Пример использования этой функции: x=getn();

Указание: использовать в качестве аналога функцию chislo() из программы вычисления выражения (программа 4.1 из раздела 4.3.4).

л) Дано натуральное число N и N целых чисел. Найти сумму заданных целых чисел. Для ввода чисел использовать функцию getn() из задачи 6, вариант 10.

м) Переписать на язык С0 программу 2.3 вычисления выражения из подраздела 2.10.1.

2. Предложить реализацию расширения языка и транслятора С0:

а) Запись числа в восьмеричной системе счисления, начиная с цифры ноль, например: 027 = 2*8 + 7 = 23.

б) Запись числа в шестнадцатеричной системе счисления, начиная с 0x или 0X, например: 0x4C = 4*16 + 12 = 76.

в) Запись служебных слов как малыми, так и большими буквами.

г) Задание имен входного и выходного файлов при запуске транслятора.

д) Использование поразрядных операций: конъюнкции &, дизъюнкции | и отрицания ~, как в языке С.

е) Использование логических операций: конъюнкции &&, дизъюнкции || и отрицания !, как в языке С.

ж) Выдача текстовых сообщений об ошибках вместо числовых.

з) Использование оператора цикла с предусловием, как в языке С:

do *оператор* **while** (*выражение*);

и) Использование условного оператора, как в языке С:

if (*выражение*) *оператор* [**else** *оператор*]

к) Использование оператора цикла **for**, как в языке С:

for (*выражение* ; *выражение* ; *выражение*) *оператор*

Указание: удобно расположить объектные коды выражений в том порядке, как они записаны в операторе **for**, а необходимый порядок их выполнения обеспечить за счет меток и команд перехода, как делается при трансляции операторов **if** и **while**.

л) Использование оператора переключателя, как в языке С:

switch (*выражение*)

{ [**case** *число*: [*оператор*] **…**]**…**

default: [*оператор*] **…**

}

м) Описание локальных переменных в начале составного оператора - блока, как в языке С.

Указание: удобно выделять и освобождать память для локальных переменных составного оператора в начале кадра стека, используя отрицательные смещения.

н) Символьные константы вида: '*' = 42, 'A' = 65 и т. д.

о) Комментарии вида: /* [*символ*]**...** */, как в языке С.

п) Строчные комментарии, как в языке С++, вида:

//[*символ*]**...** конец-строки

р) Задание начальных значений в определении глобальных переменных, например: **int** a=5, b, x=-78;

3. Составить объектный код на языке ассемблера заданного оператора языка С0.

а) **while** (100>a) a = (b-6)*a + c;

где a, b - параметры; c - глобальная переменная.

б) **if** (f(c+d) < 3*d) s = 20;

где f - функция; c, s - глобальные переменные; d – локальная переменная.

в) **return** a*(a>b+4) - 8*(a<=b+4);

где a - глобальная переменная; b - локальная переменная.

г) p(5-x, a+10*(b-c/8));

где p - функция; a - глобальная переменная; x - параметр; b, c - локальные переменные.

4. Составить трассировочную таблицу компиляции на язык ассемблера оператора языка С0: pr (b+4, (a+7*b)/c);

где pr - функция; a - глобальная переменная; b - параметр; c - локальная переменная.

5. Построить дерево синтаксического анализа (грамматического разбора) заданного оператора языка С0:

а) **while** (x!=5) x = x - y/(z+10);

б) **if** (x==y+1) putchar(c+v/8);

9.8 Правила документирования программы

Не забывайте золотого правила документирования: оформляйте свои программы в таком виде, в каком Вам хотелось бы видеть программы, написанные другими!

Приведем более конкретные рекомендации по документированию программ в виде требований к оформлению программ в рамках проектной работы.

Пояснительная записка к проектной работе (отчет) должна соответствовать общим требованиям к оформлению курсовых / дипломных работ и действующим стандартам [26]. .

Отчет содержит подробное описание программы и включает следующие основные разделы: задание, описание применения, описание программы и заключение. Затем располагается список использованной литературы.

В заключении приводятся сведения о состоянии проектной работы (степени выполнения задания), ее возможных применениях на практике и направлениях дальнейшего развития.

В приложениях к отчету приводятся: подробные экранные формы пользовательского интерфейса (окна, меню, кнопки и т.п.), текст программы,

описание контрольных примеров / тестов, распечатки результатов тестирования и различные дополнительные или справочные сведения о программе.

Кроме перечисленных стандартных разделов программной документации, для учебных целей отчет по требованию руководителя может содержать информацию о технологии и этапах разработки программы, дневник выполнения работы и сведения о ее трудоемкости.

Ниже даны краткие рекомендации по выполнению основных этапов работы и содержанию соответствующих разделов пояснительной записки, в которых отражаются результаты этих этапов.

1. Описание применения

Дается описание программы с точки зрения её применения, составленное как инструкция для пользователя программы. Данный раздел включает подразделы: постановка задачи (назначение и функции программы), интерфейс пользователя (обращение к программе, входные данные, выходные данные, сообщения, основные экранные формы; форма вызова и параметры для подпрограмм и т.п.).

Постановка задачи – это точная формулировка решаемой в программе задачи, описание назначения и функций программы с учетом случаев отсутствия решения и возможных ошибок в исходных данных.

Обязательно согласуйте постановку задачи с пользователем (или преподавателем) и убедитесь, что правильно ее понимаете!

В подразделе "Обращение к программе" описывается способ вызова программы и передачи исходных данных и результатов.

При описании входных и выходных данных указывается, что входит в эти данные, в каком виде и где они располагаются. Границы входных величин, например, максимальное число вершин графа, подбираются экспериментально так, чтобы время выполнения программы не превышало 30 сек. (не слишком задерживать проверку работы).

Дается краткое описание основных экранных форм (подробности приводятся в приложениях). Отдельно перечисляются все возможные сообщения программы. При необходимости разъясняется их смысл и требуемые ответные действия. Сообщения об ошибках отделяются от сообщений, появляющихся при нормальной работе программы.

2. Описание программы

Данный раздел описывает внутреннее "устройство" программы и содержит подразделы: метод решения задачи, структура программы и описание модулей.

Метод решения задачи описывается в самом общем виде со ссылками на литературу. При необходимости его можно объединить с постановкой задачи или с описанием соответствующих модулей.

Структура программы включает перечень программных модулей с информацией о том, какие модули вызывает каждый модуль (оформляется в виде схемы взаимодействия модулей), характеристикой прочности и сцепления модулей (см. раздел 8.1); описание глобальных данных; другую информацию об организации программы в целом.

Желательно, чтобы разработанные модули, имеющие самостоятельное значение, можно было использовать как библиотечные программы в других задачах. Поэтому модули преобразования данных обычно сами не занимаются их вводом-выводом: входные и выходные данные передаются в виде параметров. К таким модулям относится, прежде всего, модуль решения основной задачи проекта.

Ввод и вывод данных (в частности, сообщений об ошибках) производятся либо в главной программе, либо оформляются как отдельные модули.

Остальные модули обычно сообщают об ошибках в исходных данных или о невозможности решить задачу не человеку на экран, а вызывающей программе с помощью кода завершения, передаваемого как выходной параметр или значение функции.

Обычно нулевой код свидетельствует об успешном завершении программы, а ненулевое значение рассматривается как номер ошибки.

Описание модулей приводится в определенном порядке, например, в алфавитном порядке их имен. Каждый модуль описывается лаконично по единому плану, например: форма вызова (заголовок функции), перечень используемых глобальных данных, функция (назначение), перечень входных данных, перечень выходных данных, перечень рабочих (промежуточных) данных, используемый метод, алгоритм на псевдокоде или в виде схемы.

3. Общие правила оформления

Пояснительная записка оформляется на листах формата А4 рукописным или печатным способом в соответствии с действующими стандартами [26] (см, пример в [70]). В качестве образца полезно также использовать техническую литературу. Ниже даны только самые краткие рекомендации по оформлению для предотвращения наиболее распространенных ошибок.

Текст пишется лаконично, технически грамотно, допускаются только общепринятые стандартные сокращения слов. Метод решения задачи и работа алгоритма (программы) описываются неопределенно-личными предложениями: "вводится..., вычисляется..., выводится..." или в третьем лице: "алгоритм (программа) вводит..., вычисляет..., выводит...". Не следует писать от первого лица: "(мы) вводим..., вычисляем..., выводим...".

Схемы, диаграммы, графики и т. п. оформляются как рисунки и помещаются либо на одних листах с текстом, либо на отдельных листах, вставляемых в текст, ближе к первой ссылке на них.

Рисунки и таблицы нумеруются отдельно в каждом разделе и могут иметь название. Длинная таблица может продолжаться на следующих листах.

Алгоритмы представляются в виде схемы или на псевдокоде - неформальном языке, сочетающем операторы языка программирования с математическими формулами и предложениями естественного языка.

Чтобы не разрывать алгоритмы при переносе на следующий лист, можно оформлять их с нумерацией, подобно рисункам. Распечатки текстов программ приводятся в приложениях. Программы записываются в наглядной ступенчатой форме и снабжаются комментариями.

В тексте делаются ссылки по номерам на формулы, рисунки, таблицы и алгоритмы. В отчете приводится список использованной литературы, на которую в тексте есть ссылки в квадратных скобках по номерам из списка (как в данном учебнике).

В заключение перечислим основные *принципы разработки* программ.

1. *Простота и ясность* в выборе методов решения задач, структур данных и алгоритмов для их реализации, структуры программы, ее внешнего интерфейса с пользователем и внутреннего интерфейса между компонентами (модулями). Не усложняйте программу без необходимости, без уверенности, что это усложнение окупится определенными преимуществами.

2. *Общность и универсальность*: проще решить задачу раз и навсегда, чтобы не возвращаться к ней в различных частных случаях.

3. *Эволюционность*: не пытайтесь сразу создать окончательную версию программы; более эффективен итерационный подход - цикл развития: разработка, опытная эксплуатация и доработка.

4. *Стандартизация* и единообразие (унификация) во всем: в обозначениях (языках), стиле программирования, методах проектирования, тестирования, отладки и документирования важны всегда и особенно в коллективной работе.

5. *Автоматизация*: не делайте вручную то, что можно поручить компьютеру. Пусть за вас работает машина.