ФУНКЦИИ С ПЕРЕМЕННЫМ КОЛИЧЕСТВОМ ПАРАМЕТРОВ

В языке Си допустимы функции, количество параметров (их тип) у которых при компиляции функции могут быть неизвестными. Количество и типы параметров становятся известными только в момент вызова функции, когда явно задан список фактических параметров. При определении и описании таких функций, спецификация формальных параметров заканчивается запятой и многоточием:

```
тип имя (спецификация_явных_параметров, ,. );
```

где *тип* - тип возвращаемого функцией значения; *имя* - имя функции;

спецификация_явных_параметров - список спецификаций параметров, количество и типы которых фиксированы и известны в момент компиляции. Эти параметры можно назвать обязательными.

Каждая функция с переменным количеством параметров должна иметь хотя бы один обязательный параметр. После списка явных (обязательных) параметров ставится запятая, а затем многоточие, извещающее компилятор, что дальнейший контроль соответствия количества и типов параметров при обработке вызова функции проводить не нужно. Сложность в том, что у переменного списка параметров нет даже имени, поэтому не понятно, как найти его начало и конец.

Каждая функция с переменным списком параметров должна иметь механизм определения их количества и их типов. Принципиально различных подходов к созданию этого механизма два. Первый подход предполагает добавление в конец списка реально использованных (необязательных) фактических параметров специального параметра-индикатора с уникальным значением, которое будет сигнализировать об окончании списка. При таком подходе в теле функции параметры последовательно перебираются, и их значения сравниваются с заранее известным концевым признаком. Второй подход предусматривает передачу в функцию сведений о реальном количестве фактических параметров. Эти сведения о реальном количестве используемых фактических параметров можно передавать в функцию с помощью одного из явно задаваемых (обязательных) параметров. В обоих подходах - и при задании концевого признака, и при указании числа реально используемых фактических параметров переход от одного фактического параметра к другому выполняется с помощью указателей.

Доступ к адресам параметров из списка. Следующая программа включает функцию с изменяемым списком параметров, первый из которых (единственный обязательный) определяет число действительно используемых при вызове необязательных фактических параметров.

```
#include <stdio.h>
/*Функция суммирует значения своих параметров типа int */
long summa(int k,...) /* k - число суммируемых параметров  */
{
   int *pick = &k; /* Настроили указатель на параметр k*/
   long total_= 0; for(; k; k--)
   total += *(++pick); return total;
}
void main()
```

```
{
   printf("\n summa(2, 6, 4) = %d",summa(2,6,4));
   printf("\n summa(6, 1, 2, 3, 4, 5, 6) = %d",
   summa(6,1,2,3,4,5,6));
}
```

Результат выполнения программы:

summa(2, 6, 4) = 10
summa(6, 1, 2, 3, 4, 5, 6} = 21

Для доступа к списку параметров используется указатель pick типа int *. Вначале ему присваивается адрес явно заданного параметра k, т.е. он устанавливается на начало списка параметров в памяти. Затем в цикле указатель pick перемещается по адресам следующих фактических параметров, соответствующих неявным формальным параметрам. С помощью разыменования *(++pick) выполняется выборка их значений. Параметром цикла суммирования служит аргумент k, значение которого уменьшается на 1 после каждой итерации и, наконец, становится нулевым. Особенность функции - возможность работы только с целочисленными фактическими параметрами, так как указатель pick после обработки значения очередного параметра "перемещается вперед" на величину **sizeof(int) и** должен быть всегда установлен на начало следующего параметра.

Недостаток функции - возможность ошибочного задания неверного количества реально используемых параметров.

Следующий пример содержит функцию для вычисления произведения переменного количества параметров. Признаком окончания списка фактических параметров служит параметр с нулевым значением.

```
#include <stdio.h> /*   Функция вычисляет произведение параметров:*/
double prod(double  arg, ...)
{
   double aa = 1.0; /* Формируемое произведение*/
   double *prt = &arg; /* Настроили указатель на параметр arg */
   if (*prt == 0.0) return 0.0;
   for (; *prt; prt++) aa *= *prt;
   return aa;
}
 void main()
{
   double prod(double, ...);/*  Прототип функции  */
   printf("\n prod(2e0, 4e0, 3e0, 0e0)   =  %e", prod(2e0,4e0,3e0,0e0)) ;
   printf("\n prod(1.5, 2.0, 3.0, 0.0)   =  %f", prod(1.5,2.0,3.0,0.0)) ;
   printf("\n prod(1.4, 3.0, 0.0, 16.0, 84.3, 0.0)=%f",
      prod(1.4,3.0,0.0,16.0,84.3,0. 0)) ;
   printf(   "\nprod(0e0)   =  %e",prod(0e0));
 }
```

Результат выполнения программы:

```
  prod(2e0, 4e0, 3e0, 0e0) = 2.400000e+01
  prod(1.5, 2.0, 3.0, 0.0) = 9.000000
  prod(1.4, 3.0, 0.0, 16.0, 84.3, 0.0) = 4.200000
  prod(0e0) = 0.000000e+00
```

В функции prod() перемещение указателя prt по списку фактических параметров выполняется всегда за счет изменения prt на величину **sizeof(double).** Поэтому все фактические параметры при обращении к функции prod() должны иметь тип **double.**

В вызовах функции проиллюстрированы некоторые варианты задания параметров. Обратите внимание на вариант с нулевым значением параметра в середине списка. Параметры вслед за этим значением игнорируются. Недостаток функции - неопределенность действий при отсутствии в списке фактических параметров параметра с нулевым значением.

Чтобы функция с переменным количеством параметров могла воспринимать параметры различных типов, необходимо в качестве исходных данных каким-то образом передавать ей информацию о типах параметров. Для однотипных параметров возможно, например, такое решение - передавать с помощью дополнительного обязательного параметра признак типа параметров. Определим функцию, выбирающую минимальное из значений параметров, которые могут быть двух типов: или только **long**, или только **int**. Признак типа параметра будем передавать как значение первого обязательного параметра. Второй обязательный параметр указывает количество параметров, из значений которых выбирается минимальное. В следующей программе предложен один из вариантов решения сформулированной задачи:

```
#include <stdio.h>
void main() {       /* Прототип функции: */
  long minimum(char, int , ...);
  printf("\n\tminimum('l',3,10L,20L,30L) = %ld", minimum('l',3,10L,20L,30L));
  printf("\n\tminimum('i',4,11, 2, 3, 4) = %ld", minimum('i',4,11,2,3,4)) ;
  printf( "\n\tminimum{'k', 2, 0, 64) =%ld", minimum('k',2,0,64));
}
/* Определение функции с переменным списком параметров */
long minimum(char z, int k,...) {
   if (z == 'i') {
     int *pi = &k + 1; /* Настроились на первый необязательный параметр */
     int min = *pi; /* Значение первого необязательного параметра */
     for(; k; k-, pi++) min = min > *pi ? *pi : min;
     return (long)min;
   }
   if (z == 'l') {
     long *pl = (long*)(&k+1);
     long min = *pl; /* Значение первого необязательного параметра */
     for(; k; k--, pl++) min = min >*pl ? *pl : min;
     return (long)min;
   }
   printf("\nОшибка!   Неверно задан 1-й параметр: ") ;
   return 2222L;
}
```

Результат выполнения программы:

```
minimum('1', 3, 10L, 20L, 30L) = 10
minimum('i', 4, 11, 2, 3, 4) = 2
Ошибка! Неверно задан 1-й параметр:
minimum('k',2,0,64) = 2222
```

В приведенных примерах функций с изменяемыми списками параметров перебор параметров выполнялся с использованием адресной арифметики и явным применением указателей нужных типов. К проиллюстрированному способу перехода от одного параметра к другому нужно относиться с осторожностью. Дело в том, что порядок размещения параметров в памяти ЭВМ зависит от реализации компилятора. В компиляторах имеются опции, позволяющие изменять последовательность размещения параметров. Стандартная для языка Си на IBM PC последовательность: меньшее значение адреса у первого параметра функции, а остальные размещены подряд в

соответствии с увеличением адресов. Противоположный порядок обработки и размещения будет у функций, определенных и описанных с модификатором **pascal,** Этот модификатор и его антипод - модификатор cdecl являются дополнительными ключевыми словами, определенными для ряда компиляторов. Не останавливаясь подробно на возможностях, предоставляемых модификатором pascal, отметим два факта. Во-первых, применение модификатора pascal необходимо в тех случаях, когда функция, написанная на языке Си, будет вызываться из программы, подготовленной на Паскале. Во-вторых, функция с модификатором pascal не может иметь переменного списка параметров, т.е. в ее определении и в ее прототипе нельзя использовать многоточие.

МАКРОСРЕДСТВА ДЛЯ ПЕРЕМЕННОГО ЧИСЛА ПАРАМЕТРОВ.

Для обеспечения мобильности программ с функциями, имеющими изменяемые списки параметров, в каждый компилятор стандарт языка предлагает включать специальный набор макроопределений, которые становятся доступными при включении в текст программы заголовочного файла stdarg.h. Макрокоманды, обеспечивающие простой и стандартный (не зависящий от реализации) способ доступа к конкретным спискам фактических параметров переменной длины, имеют следующий формат:

```
void va_start(va_list param, последний_явный_параметр);
type va_arg(va_list param, type);
void va_end(va_list param);
```

Кроме перечисленных макросов, в файле stdarg.h определен специальный тип данных va_list, соответствующий потребностям обработки переменных списков параметров. Именно такого типа должны быть первые фактические параметры, используемые при обращении к макрокомандам va_start(), va_arg(), va_end(). Кроме того, для обращения к макросу va_arg() необходимо в качестве второго параметра использовать обозначение типа *(type)* очередного параметра, к которому выполняется доступ. Объясним порядок использования перечисленных макроопределений в теле функции с переменным количеством параметров (рис. 5.2). Напомним, что каждая из функций с переменным количеством параметров должна иметь хотя бы один явно специфицированный формальный параметр, за которым после запятой стоит многоточие. В теле функции обязательно определяется объект типа **va_Hst.** Например, так:

```
va_list factor;
```

Определенный таким образом объект factor обладает свойствами указателя. С помощью макроса **va_start()** объект factor связывается с первым необязательным параметром, т.е. с началом .списка неизвестной длины. Для этого в качестве второго аргумента при обращении к макросу **va_start()** используется последний из явно специфицированных параметров функции (предшествующий многоточию):

```
va_start (factor, последний явный параметр);
```

Рассмотрев выше способы перемещения по списку параметров с помощью адресной арифметики, мы понимаем, что указатель factor сначала "нацеливается" на адрес последнего явно специфицированного параметра, а затем перемещается на его длину и тем самым устанавливается на начало переменного списка параметров. Именно поэтому функция с переменным

списком параметров должна иметь хотя бы один явно специфицированный параметр.

Теперь с помощью разыменования указателя factor мы можем получить значение первого фактического параметра из переменного списка. Однако нам не известен тип этого фактического параметра. Как и без использования макросов, тип параметра нужно каким-то образом передать в функцию. Если это сделано, т.е. определен тип *type* очередного параметра, то обращение к макросу

```
va_arg (factor, type)
```

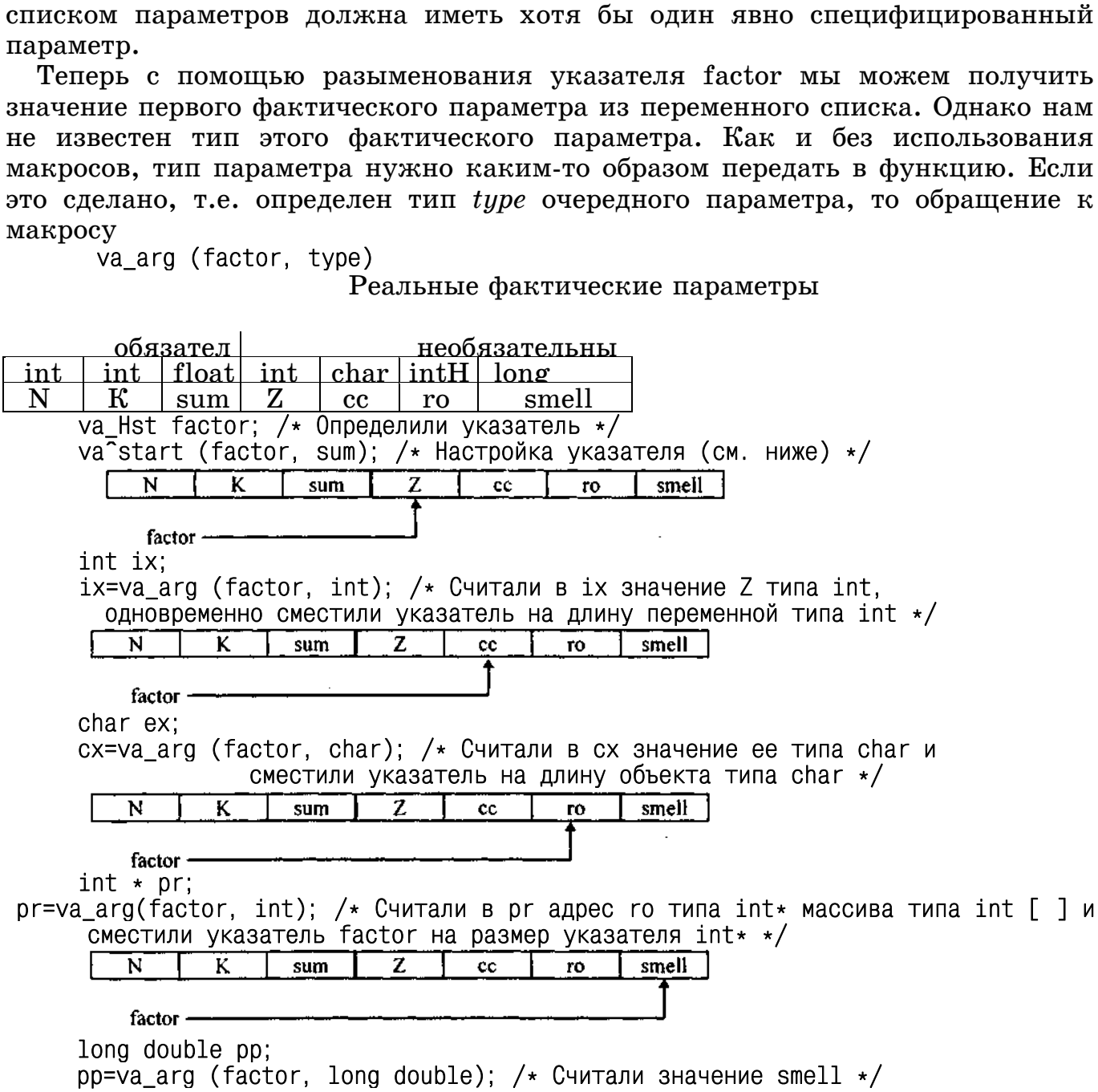

Рис. 5.2. Схема обработки переменного списка параметров с помощью макросов стандартной библиотеки

V

позволяет, во-первых, получить значение очередного (сначала -первого) фактического параметра типа *type*. Вторая задача мак-рокоманды va_arg() - заменить значение указателя factor на адрес следующего фактического параметра в списке. Теперь, узнав каким-то образом тип, например *typel,* этого следующего параметра, можно вновь обратиться к макросу:

va_arg (factor, *typel)*

Это обращение позволяет получить значение следующего фактического параметра и переадресовать указатель factor на фактический параметр, стоящий за ним в списке, и т.д.

Примечание. Реализация компиляторов Turbo C++ и Borland C++ запрещает использовать с макрокомандой **va_arg( )** типы **char, unsigned char, float.** Данные типа **char и unsigned char** преобразуются при передаче в функцию с переменным количеством параметров в тип

int, а данные типа **float** приводятся к типу **double.** Именно поэтому при выводе с помощью **printf()** данных как типа **float,** так и типа **double** используются одни и те же спецификаторы %f и %e. Вывод данных типа **char и unsigned char можно** выполнять и по спецификации %d, и по спецификации %c.

Макрокоманда **va_end()** предназначена для организации корректного возврата из функции с переменным списком параметров. Ее единственным параметром должен быть указатель типа **va_list,** который использовался в функции для перебора параметров. Таким образом, для наших иллюстраций вызов макрокоманды должен иметь вид:

va_end (factor**);**

Макрокоманда **va_end()** должна быть вызвана после того, как функция обработает весь список фактических параметров.

Макрокоманда **va_end()** обычно модифицирует свой аргумент **(**указатель типа **va_list),** и поэтому его нельзя будет повторно использовать без предварительного вызова макроса **va_start( ).**