КОМП'ЮТЕРНІ ЗАСОБИ, МЕРЕЖІ ТА СИСТЕМИ

УДК 681.3

В.П. ЗИНЧЕНКО, Н.П. ЗИНЧЕНКО

ИСПОЛЬЗОВАНИЕ МАКРОЯЗЫКА И МАКРОПРОЦЕССОРА

Обоснована эффективность использования макросредств, сформулированы методы построения и обработки макроязыка. Показано, что определяя макрокоманды, можно вводить в прикладное программное обеспечение новые структуры более высокого уровня, что позволяет достичь краткости и простоты управления программной системой. Макрооперации упрощают подготовку, отладку и модификацию прикладных программ, обеспечивают стандартизацию и позволяют автоматизировать инсталляцию прикладных программных систем.

Введение. В прикладном программном обеспечении (ПО) часто необходимо повторять некоторые последовательности действий. В таких случаях можно воспользоваться аппаратом макрокоманд (макро / макрос), которые являются однострочными сокращениями для группы команд. Макрокоманда по существу определяет одну “команду” для представления некоторой последовательности команд в прикладных программах (ПП).

Определяя макрокоманды, можно вводить в прикладное ПО новые структуры более высокого уровня, что позволяет достичь краткости и простоты управления программной системой. Макрооперации упрощают подготовку, отладку и модификацию ПП, обеспечивают стандартизацию и позволяют автоматизировать инсталляцию прикладного ПО.

Постановка задачи. Обосновать эффективность использования макросредств и сформулировать методы построения и обработки макроязыка.

Метод решения поставленной задачи основывается на использовании макроязыка и макропроцессора.

Макрокоманды в своей простой форме представляют собой обозначения последовательности операций. Например, на рис. 1 дважды используется последовательность команд (а). Аппарат макрокоманд IDE Borland C++ [1, 2] позволяет присвоить (а) имя и использовать его вместо (а). Более того, можно определить некоторый макроязык и использовать его вместо (а). Можно определить некоторый макроязык и использовать его при разработке ПП. Обрабатывается такой макроязык *макропроцессором*, который

представляет собой отдельный языковой процессор со своим собственным языком. Форматы макроопределений в различных программных системах различные, но в случае IDE Borland C++ формат определения макрокоманды такой:

MACRO <macro name>
<тело макроопределения>
END;

Директива MACRO определяет следующий за ней идентификатор как *имя макрокоманды*. Далее располагается последовательность команд ("тело макроопределения"), которая заканчивается директивой END.

```
SetPrevPos;          (a)
FixScreenPos;
PageScreenUp;
FixCursorPos;
. . .
SetPrevPos;          (a)
FixScreenPos;
PageScreenUp;
FixCursorPos;
. . .
```

РИС. 1

Если макрокоманда определена, то использование ее имени в ПП (*макровызов*) эквивалентно использованию соответствующей последовательности команд. Если последовательности (а) дать имя "MacP", то программа (рис. 1) будет иметь вид, показанный на рис. 2 (слева – исходный текст, справа – расширение) В этом случае макропроцессор заменит каждую макрокоманду командами (а). Такой процесс замены называется *расширением* макрокоманды.

```
MACRO MacP                 
  SetPrevPos;              
  FixScreenPos;            
    PageScreenUp;          
  FixCursorPos;            
END;                       
. . .                      . . .
MacP                       SetPrevPos;
. . .                      FixScreenPos;
                           PageScreenUp;
                           FixCursorPos;
                           . . .
MacP                       SetPrevPos;
. . .                      FixScreenPos;
                           PageScreenUp;
                           FixCursorPos;
                           . . .
```

РИС. 2

Операторы макрокоманд. Механизм макрокоманд позволяет подставлять последовательности команд вместо макровызовов, где все обращения к макроопределению заменяются идентичными последовательностями команд. Однако такой подход недостаточно гибок, так как в макровызове нет средств модификации кода, который его заменяет. Существенное расширение возможностей макросредств достигается добавлением операндов (параметров) макрокоманд. На рис. 3 показаны похожие, но не идентичные последовательности команд: в (б) используется операнд "1", а в (в) – операнд "2". Можно считать, что они выполняют одну и ту же операцию с переменным параметром (*операнд макрокоманды / формальный параметр)*, который объявляется в той же строке, что и имя мак-

```
. . .
SetPrevPos;
MoveToMark(1);           (б)
CenterFixScreenPos;
. . .
SetPrevPos;
MoveToMark(2);           (в)
CenterFixScreenPos;
. . .
```

РИС. 3

роса. В языке макроассемблера он помечается символом &, что отличает его как символ макроязыка от символов ассемблера. В IDE Borland не предусмотрена работа с макрооперандами, но можно предположить, что если бы это было так, то формат макроопределения мог быть таким (<paramlist> – перечисленные через запятую все операнды макроса):

```
MACRO<macroname>(<paramlist>)
 <тело макроопределения>
END;
```

В таком случае ПП (см. рис. 3) можно переписать так, как показано на рис. 4.

```
MACRO MacG(lab)
  SetPrevPos;
  MoveToMark(lab);

CenterFixScreenPos;
END;
. . .
MacG(1)


. . .
MacG(2)
. . .
```

```
. . .
SetPrevPos;
MoveToMark(1);
CenterFixScreenPos;
. . .
SetPrevPos;
MoveToMark(2);
CenterFixScreenPos;
. . .
```

РИС. 4

Отметим, что макрокоманда может иметь более одного операнда. Каждый операнд при этом должен соответствовать формальному параметру в строке определения имени макроса.

На рис. 5 показана последовательность, в которой операнды и команды различны. Их можно переписать так, как показано на рис. 6.

```
. . .
ScrollScreenDown;
ScrollScreenLeft(0);
FixCursorPos;
. . .
ScrollScreenUp;
ScrollScreenLeft(1);
FixCursorPos;
. . .
```

РИС. 5

Отметим, что существует два основных способа задания операндов – *позиционный* (рис. 6) и *ключевой* (позволяет обращаться к формальным операторам как по именам, так и по позиции).Ссылка на формальные операторы в определении MacM может быть такой: MacM (c = = ScrollScreenUp, l = 1).

```
MACRO MacM(c,l)
   c;
   ScrollScreenLeft(l);
   FixCursorPos;
END;
. . .
MacM(ScrollScreenDown,0)
. . .


MacM(ScrollScreenUp,1)
```

```
ScrollScreenDown;
ScrollScreenLeft(0);
FixCursorPos;
. . .
ScrollScreenUp;
ScrollScreenLeft(1);
FixCursorPos;
. . ..
```

РИС. 6

Условное макрорасширение используется для изменения порядка команд макрорасширения внутри макроса. Например, ПП

```
. . .
ScrollScreenLeft(1);
LiteralChar('-');
. . .
ScrollScreenDown;
ScrollScreenLeft(2);
LiteralChar('*');
. . .
```

РИС. 7

(рис. 7), в которой не только параметры, но и количество команд – переменная величина. Эту ПП можно переписать так, как показано на рис. 8.

Комбинация IF...THEN...ELSE – это *макрометки / символы следования*, которые не включаются в выходной текст макропроцессора. В макроязыке также могут использоваться директивы *условного* и *безусловного перехода* на псевдометку, с которой макропроцессор продолжит обработку ПП. Такие операторы переходов служат для указания выполнения операторов ПП.

Макровызовы внутри макроопределения. Так как макроопределения, по сути, являются "сокращениями" последовательности команд, то полезным свойством была бы возможность выполнять такие "сокращения" внутри самих макроопределений. На рис. 9 показано макроопределение M2, где дважды происходит ссылка на макроопределение M1. Это позволило уменьшить длину макроопределения M2 и сделать его более понятным. Такое использование макросредств приводит к макрорасширениям с несколькими уровнями вложенности (рис. 10). Например, команда M2 могла быть выполнена внутри другого макроопределения.

```
MACRO M1(p,l,ch)
IF p==1 THEN
   ScrollScreen-
Down;
ENDIF
ScrollScreenLeft(2);
LiteralChar('*');
END
. . .
M1(1,2,'-')
. . .

M1(0,1,'*')
. . ..
```

```
. . .
ScrollScreenLeft(1);
LiteralChar('-');
. . .
ScrollScreenDown;
ScrollScreenLeft(2);
LiteralChar('*');
. . .
```

РИС. 8

```
MACRO M1
  SetPrevPos;
  CursorCharRight;
END
. . .
MACRO M2
   M1
   M1
END
. . .
```

РИС. 9

Фактически, макропереходы дают возможность любое число раз обращаться к любому макроопределению и даже к самому себе (*рекурсивные вызовы*).

Макроопределения в макроопределениях. В теле макроопределения можно использовать любые допустимые синтаксисом предложения, в том числе и другие макроопределения. Отметим, что внутреннее макроопределение не будет определено до тех пор, пока не произойдет вызов внешнего макроса. Например, пусть необходимо определить группу макроопределений для обращения к подпрограммам с помощью какой-то определенной по-

следовательности, так как показано на рис. 11, где определена макрокоманда DEF, которая при указании в качестве ее операнда имени подпрограммы определяет соответствующий этому имени макрос.

Рассмотрим метод реализации макроязыка на примере языка макроассемблера [3, 4]. Задачу сформулируем как разработку макропроцессора, который должен распознавать и обрабатывать макроопределения и макрокоманды.

```
MACRO M1
  SetPrevPos;
  CursorChar-
Right;
END
MACRO M2
  M1
  M1
END
. . .

M2
. . .
```

```
MACRO M2
  SetPrevPos;
  CursorChar-
Right;
  SetPrevPos;

CursorCharRight;
END
M2
. . .
```

```
SetPrevPos;
CursorCharRight;
SetPrevPos;
CursorCharRight;
. . ..
```

РИС. 10

Распознавать макроопределения. Макропроцессор должен распознавать макроопределения, выделяемые директивами MACRO и ENDM. Эта задача усложняется тем, что макроопределения могут быть вложенными. Когда это так, то макропроцессор должен правильно распознавать вложения и сопоставлять начало и конец макроса. Весь вложенный текст, в том числе и другие макроопределения, определяет отдельную макрокоманду.

```
MACRO DEF sub
MACRO sub(par)
. . .
   sub(par)
. . .
END
END
```

РИС. 11

Запоминать макроопределения. Процессор должен запомнить определения макрокоманд, которые впоследствии будут использоваться для расширения макровызовов.

Распознавать вызовы. Необходимо распознавать макровызовы, представленные в виде мнемонического кода операции. При этом имена макрокоманд должны обрабатываться так, как обрабатываются коды операций.

Выполнять расширение макрокоманд и подстановку фактических параметров. Вместо формальных параметров макроопределения макропроцессор должен подставить соответствующие операнды макрокоманды. Этот текст, в свою очередь, может содержать как макрокоманды, так и макроопределения.

По формальным параметрам необходимо определить – могут ли они встречаться в качестве кода операции и синтаксиса допустимых параметров. В разных макроязыках встречаются разные варианты реализации подобных ситуаций. Поэтому рассмотрим некоторые варианты возможных реализаций.

Формальные параметры могут встречаться в макроопределении как в команде, так и в коде операции. Для того, чтобы обеспечить возможность конкатенации формальных параметров макроопределения с фиксированными сим-

вольными строками, необходимо определить разделительный символ, обеспечивающий конкатенацию формальных параметров и заданных символьных последовательностей. Гораздо сложнее случай, когда необходимо подставлять значение параметра внутри символьной строки. В таком случае, можно применить конкатенацию по умолчанию двух последовательно друг за другом идущих символьных строк или преобразование формального параметра, заключенного в скобки. Для выполнения функций условных переходов необходимо вычислять некоторые арифметические выражения (например, счетчик), при этом используя псевдопеременные времени компиляции внутри макросов.

Двухпроходный алгоритм. Предположим, что разрабатываемый макропроцессор функционально независим от основного компилятора и его текст должен передаваться этому компилятору. Сначала не разрешим макровызовы и макроопределения внутри макроопределений.

Макропроцессор, как и язык ассемблера, просматривает и обрабатывает строки текста. Если рассматривать макроопределение как единый объект, то можно сделать вывод о том, что строки такого макроопределения не так сильно взаимосвязаны, и макроопределения не могут ссылаться на объекты вне этого макроопределения.

Предположим, что в макроопределении есть строка INC X, и перед этой командой параметр X получил значение 1. Макропроцессор не осуществляет синтаксический анализ, а выполняет текстовую подстановку (вместо “X” подставляется “1”).

Алгоритм будет выполнять 2 просмотра входного текста. В первом проходе будут детерминированы все макроопределения, а во втором – открыты все ссылки на макросы. Во время первого прохода проверяется каждый код операции, макроопределения запоминаются в таблице макроопределений, а копия исходного текста без макроопределений запоминается в памяти, для использования ее на втором проходе. Также во время первого прохода строится таблица имен, которая на втором проходе используется для выделения макроопераций и расширения их до текста макроопределения.

Данные для алгоритма просмотра текста: ВТ – входной текст; ВХТ1, ВХТ2 – выходная копия текста после первого и второго проходов соответственно; МДТ – таблица макроопределений, в которой хранятся тела макроопределений; МНТ – таблица хранения имен макрокоманд, определенных в МНТ; МДТС, МНТС – счетчик для таблицы МДТ и для таблицы МНТ соответственно; АЛА – массив списка параметров для подстановки индексных маркеров вместо формальных параметров перед запоминанием определения.

Алгоритмы обработки макроопределений. Обработка макроопределений двухпроходным способом состоит из алгоритмов первого и второго построчного просмотров входного текста.

Алгоритм первого просмотра (*макроопределения*) проверяет каждую строку входного текста (рис. 12). Если она представляет собой директиву MACRO, то все следующие за ней строки запоминаются в свободных ячейках МДТ. Первая

строка макроопределения – это имя самого макроса, которое заносится в таблицу имен МНТ с индексом этой строки в МДТ. При этом происходит также подстановка номеров формальных параметров, вместо их имен. Если при просмотре встречается команда END, то это означает, что весь текст обработан, и управление можно передавать второму просмотру для обработки макрокоманд.

```
Начало
 МДТС = 0
 МНТС = 0
 ФЛАГ ВЫХОДА=0
 цикл пока (ФЛАГ ВЫХОДА == 0) {
  чтение следующей строки ВТ
  если !(операция MACRO) {
     вывод строки в ВХТ1
     если (операция END) ФЛАГ ВЫХОДА=1
  }
  иначе {
     чтение идентификатора
     запись имени и индекса в МНТ
     МНТС ++
     приготовить массив списка АЛА
     запись имени в МДТ
     МДТС ++
     цикл {
        чтение следующей строки ВТ
        подстановка индекса операторов
        добавление в МДТ
        МДТС ++
     } пока !(операция ENDM)
  }
 }
 переход ко второму проходу
Конец
```

РИС. 12

Алгоритм второго просмотра (*расширение макрокоманд*) проверяет мнемонический код каждого предложения (рис. 13). Если это имя содержится в МНТ, то происходит обработка макропредложения. Из таблицы МНТ берется указатель на начало макроса в МДТ. Макропроцессор готовит массив списка АЛА, который содержит таблицу индексов формальных параметров и соответствующих операндов макрокоманды. Выполняется чтение из МДТ строки и в нее подставляются необходимые параметры. Полученная таким образом строка записывается в ВХТ2. Когда встречается директива END, текст полученного кода передается для компиляции ассемблеру.

Однопроходный алгоритм. Предположим, что допускается реализация макроопределения внутри макроопределений. Основная проблема в том, что внутреннее макро определено только после того, как выполнен вызов внешнего. Для обеспечения использования внутреннего макро необходимо повторять как просмотр обработки макроопределений, так и просмотр обработки макрокоманд. Однако существует решение, которое позволяет выполнить распознавание и расширение в один просмотр.

Алгоритм однопроходного макроассемблера (рис. 14) объединяет два вышеприведенных алгоритма в один. Он является упрощением алгоритма из [5]. Различие состоит в том, что современные средства программирования дают возможность осуществлять вставки и удаления из больших массивов с минимальными затратами процессорного времени, что было невозможно ранее. Кроме того, скорость работы современных процессоров позволяет осуществлять пря-

```
Начало
 ФЛАГ ВЫХОДА = 0
 цикл пока (ФЛАГ ВЫХОДА = = 0) {
  чтение строки из ВХТ1
  НАЙДЕНО = поиск кода в МНТ
  если !(НАЙДЕНО) {
     запись в ВХТ2 строки
     если (операция END) {
        ФЛАГ ВЫХОДА = 1
     }
  }
  иначе {
     УКАЗАТЕЛЬ = индекс из МНТ
     Заполнение списка параметров АЛА
     цикл {
        УКАЗАТЕЛЬ ++
        чтение следующей строки из МДТ
        подстановка параметров
        вывод в ВХТ2
     } пока !(операция ENDM)
  }
 }
 переход к компиляции
Конец
```

РИС. 13

```
Начало
 МТДС = 0
 МНТС = 0
 ФЛАГ ВЫХОДА=0
 цикл пока !(ФЛАГ ВЫХОДА) {
  чтение следующей строки ВТ
  НАЙДЕНО = поиск кода в МНТ
  если (НАЙДЕНО) {
     МДИ = 1
     УКАЗАТЕЛЬ = индекс из МНТ
     Заполнение списка параметров АЛА
     цикл {
        УКАЗАТЕЛЬ ++
        чтение след. строки из МДТ
        подстановка параметров
        вставка во ВТ}
  пока !(операция ENDM)
  иначе если !(операция MACRO)
   {вывод строки в ВХТ1
   если (операция END) ФЛАГ ВЫ-
ХОДА=1}
  иначе {
     чтение идентификатора
     запись имени и индекса в МНТ
     МНТС ++
     приготовить массив списка АЛА
     запись имени в МДТ
     МДТС ++
     цикл {
        чтение следующей строки ВТ
        подстановка индекса операторов
        добавление в МДТ
        МДТС ++
     } пока !(операция ENDM)
  }
 }
Конец
```

РИС. 14

мые вставки и удаления в массивах данных до 64 Кбайт в режиме реального времени.

Таким образом, расширение исходного макроса может быть напрямую вставлено в исходный текст и обработано в расширенном виде. Такая технология позволяет значительно упростить алгоритм обработки макроязыка.

Реализация. Разработанный макропроцессор обрабатывает тексты в режиме препроцессора, т. е. он выполняет полный просмотр входного текста, до того, как передать управление ассемблеру. Макропроцессор также может быть реализован внутри первого прохода ассемблера, что позволяет исключить промежуточные файлы и достичь на порядок большей интеграции макропроцессора и ассемблера путем объединения сходных функций. Например, можно объединить таблицы имен макросов и имен кода операции; специальный признак может указывать на то – макро это или встроенная операция.

Основные преимущества включения макропроцессора в первый просмотр состоят в следующем: большинство функций не надо реализовывать дважды (например, функции ввода-вывода); в процессе обработки нет необходимости создавать промежуточные файлы или массивы данных; можно совмещать средства ассемблера (например, команды EUQ) с макрокомандами. Основные недостатки такие: ПП требует больше оперативной памяти; реализация подобного типа задачи может оказаться на порядок сложнее, чем отдельная реализация ассемблера и макропроцессора.

Выводы. Макроязыки и соответствующие им макропроцессоры представляют собой самостоятельную форму языков программирования. Совместное использование ассемблера и макропроцессора позволяет определять свой язык “высокого” уровня при разработке прикладных программных систем [6].

Существуют четыре основных задачи, решаемых макропроцессором: распознавание макроопределений; хранение макроопределений; распознавание макрокоманд; расширение макрокоманд и подстановка параметров.

Макропроцессор в ассемблере может быть реализован как независимый двух- или однопроходный ассемблер, и как процессор, совмещенный с первым проходом двухпроходного ассемблера (например, MASM [3]).

1. *Сван Т.* Освоение Borland C++4.5. Практический курс. – Киев: Диалектика, 1996. – 544 с.
2. *Страуструп Б.* Язык программирования C++. – СПб. – М.: Бином, 1999. – 991 с.
3. *Абель П.* Ассемблер. Язык и программирование для IBM PC: Пер. с англ. – Киев: Век+, М.: ЭНТРОП, Киев: НТИ, 2003. – 736 с.
4. *Сван Т.* Ассемблер: Освоение Turbo Assembler. – Киев: Диалектика, 1996. – 544 с.
5. *Джордан Дж.* Системное программирование. – М.: Мир, 1989. – 354 с.
6. *Зінченко В.П.* Інформаційна технологія проектних досліджень складних технічних об’єктів // Наукові вісті НТУУ “КПІ”. – 2000. – № 4. – С. 32–42.

Получено 01.07.2005