ЛАБОРАТОРНАЯ РАБОТА 4

РАСПАКОВКА БИТОВЫХ ГРУПП

Рассмотрим решение следующей задачи.

1. **Задание**. Дано 32-битное целое беззнаковое число pack, которое содержит четыре битовые группы, хранящие независимую информацию.

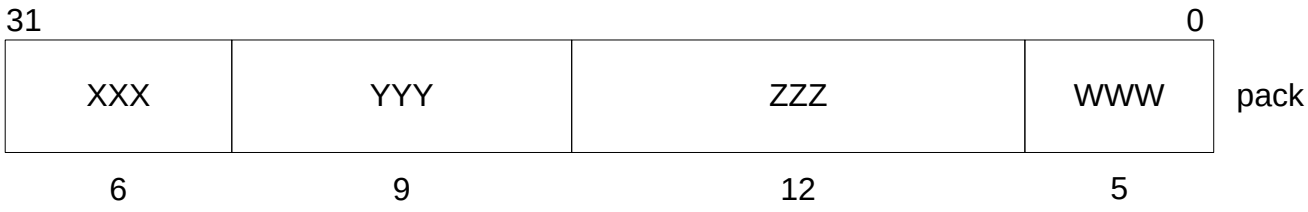

Рисунок 1 – Число pack, содержащие битовые группы.

Написать программу на С и на ассемблере, которая *распаковывает* двоичные группы, содержащиеся в переменной pack. Биты каждого упакованного поля должны располагаться в младших разрядах соответствующей переменной, старшие разряды которой должны содержать нулевые биты. Для размещения каждого поля использовать стандартную битовую группу минимальной длины (байт, слово или длинное слово).

2. Определим исходные данные, необходимые для решения поставленной задачи. Для этого рассмотрим идентификаторы, указанные в задании. Поскольку решается задача распаковки, то идентификатор исходного числа pack и будет исходными данными. По условию задачи исходное число является 32-разрядной беззнаковой величиной, то эта переменная будет определен как unsigned long.

Формат исходных данных.

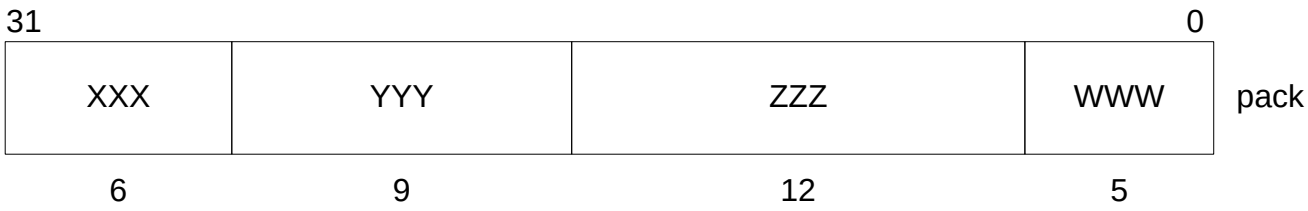

Рисунок 2 – Исходное pack, содержащее несколько битовых групп.

Исходные данные.

pack – переменная, длинное целое беззнаковое число.

3. Определим имена переменных, в которые будут записаны полученные результаты. Поскольку данное тождество нужно вычислить на С и на ассемблере, то необходимо определить два комплекта переменных, в которые будет записан полученный результат: отдельно для С и отдельно для ассемблера.

Исходное число содержит четыре битовые группы, каждая из которых должна храниться изолировано в своей переменной, минимально возможной длины. Поэтому определим идентификаторы x, y, z и w, каждый из которых будет использоваться для хранения поля xxx, yyy, zzz и www соответственно (рис.3). Результирующие переменные будут содержать исходные битовые группы в младших разрядах, а старшие неиспользуемые биты будут сброшены в ноль. Поле www имеет 5 битов, поэтому для хранения этой битовой группы достаточно байтовой переменной типа char. Поле xxx имеет 6 битов и для его хранения достаточно байтовой переменной тира char. Поле zzz имеет 12 битов, для хранения которых достаточно 16 битовой переменной типа short. Для хранения 9-битового поля yyy также достаточно 16-битовой переменной типа short.

Формат требуемого результата.

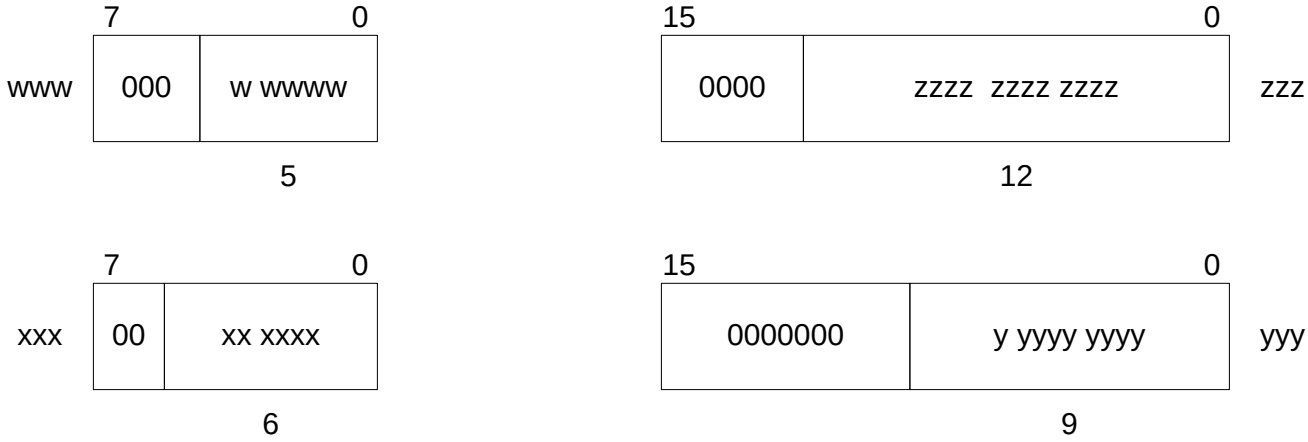

Рисунок 3 – Результирующие переменные, хранящие только одну битовую группу.

Требуемый результат.

x, w – переменные, типа unsigned char, длиной байт;

y, z - переменные, типа unsigned short, длиной слово.

4. Разработка алгоритма

Алгоритм распаковки выделяет из исходного числа битовую группу и размещает ее в отдельной переменной. Распаковка начинается с младшей битовой группы, т.е. с той, которая расположена с правого края числа pack. Алгоритм решения задачи включает такие шаги:

Ввести исходное число с упакованными полями;

Выделить из числа поле www на C;

Выделить из числа поле zzz на C;

Выделить из числа поле yyy на C;

Выделить из числа поле xxx на C;

Выделить из числа поле www на ассемблере;

Выделить из числа поле zzz на ассемблере;

Выделить из числа поле yyy на ассемблере;

Выделить из числа поле xxx на ассемблере;

Вывести значения переменных на С;

Вывести значения переменных на ассемблере.

Окончательная версия алгоритма приводится ниже.

Описание алгоритма на псевдокоде.

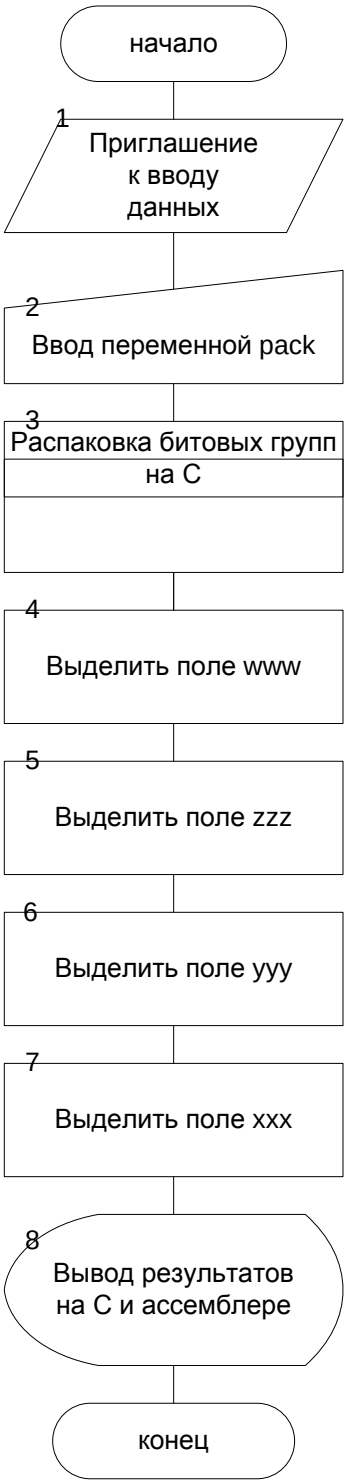

Рисунок 4 – Схема алгоритма распаковки битовых групп

Детальный алгоритм решения задачи представлен ниже (рис.4). В одном блоке указываются те операции, которые можно выполнять последовательно слева направо. Для выполнения проме-

жуточных вычислений используется отдельная вершина в схеме алгоритма. Чтобы ссылаться на отдельные блоки в схеме алгоритма удобно их пронумеровать, как показано на рисунке.

Описание схемы алгоритма.

Блок 1 – "Приглашение к вводу данных". Данный блок выдает на дисплей приглашение к вводу данных с клавиатуры.

Блок 2 – "Ввод переменной pack". Данный блок выполняет ввод исходного упакованного числа с клавиатуры и размещение его в памяти по адресу pack.

Блок 3 – "Распаковка битовых групп на С". Данный блок выделяет битовые группы www, zzz, xxx, yyy из переменной pack на С.

Блок 4 - "Выделить поле www". Данный блок выделяет поле www из исходного числа pack.

Блок 5 – "Выделить поле zzz". Данный блок выделяет поле zzz из исходного числа pack.

Блок 6 – "Выделить поле yyy". Данный блок выделяет поле www из исходного числа pack.

Блок 7 – "Выделить поле xxx". Данный блок выделяет поле xxx из исходного числа pack.

Блок 8 – "Вывод результатов на дисплей". Данный блок выводит на дисплей значения переменных www, zzz, yyy, xxx, www_a, zzz_a, yyy_a, xxx_a, хранящих результаты вычислений на языке С и на ассемблере.

5. Кодирование алгоритма

Теперь можно выполнить кодирование алгоритма распаковки в виде программы на языке ассемблер. Программа представляет собой бесконечный цикл, который начинается в строках 47-48, а заканчивается – в строке 89. В этом цикле выполняется вывод приглашения к вводу данных, вычисление выражения на С, упаковка битовых групп на ассемблере и вывод полученных результатов. Поскольку в данной работе мы имеем дело с битовыми полями, то ввод и вывод будем выполнять в шестнадцатеричном коде, который наиболее нагляден для работы с битами.

Исходная переменная value объявлена как unsigned long в строке 39. Байтовые переменные xxx и yyy объявлены как unsigned char в строке 40 для фрагмента на С (xxx, www) и на ассемблере (xxx_a, www_a). В строке 41 как unsigned short объявлены 16-битовые переменные для хранения результата на С – переменные yyy, zzz, и переменные для хранения результата на ассемблере - переменные yyy_a, zzz_a.

Строки 45-46 выводят заставку, а строки 47- 48 начинают бесконечный цикл.

Блок 1 – "Приглашение к вводу данных" - реализуют функции printf(), расположенные в строках 49-50.

Блок 2 – "Ввод переменной pack" – реализует функция scanf(), расположенная в строке 51. Исходное 32-битовое число вводится в 16-м коде, поэтому в качестве спецификатора формата используется литера X. Данный блок выполняет ввод исходного упакованного числа с клавиатуры и размещение его в памяти по адресу value.

Блок 3 – "Распаковка битовых групп на С" – реализуют команды, расположенные в строках

54-57.

Блоки 4-7 выполняют распаковку битовых групп на ассемблере и реализуются командами в строках 61-76. Вначале в строке 61 исходное число помещается в аккумулятор eax. Операция распаковки каждого поля выполняется однотипно для всех битовых групп и включает следующие этапы:

– крайнее справа битовое поле, имеющее разрядность до 8 или до 16 битов пересылается в результирующую переменную, соответственно длиной 8 или 16 битов. Если битовая группа имеет длину байт, то в результирующую переменную передается содержимое регистра al, а в случае 16-разрядной битовой группы – содержимое регистра ax;

– при помощи битовой операции И в результирующей переменной очищаются старшие биты, содержащие информацию, относящуюся к другому полю;

– при помощи команд сдвига содержимое *всего* исходного числа сдвигается вправо на столько битов, сколько содержит данное битовое поле. В результате этой операции битовое поле, расположенное левее, должно расположиться в исходном числе, начиная с нулевого бита.

Данная реализация алгоритма предполагает, что первой распаковывается младшая битовая группа, то есть та, которая расположена, начиная с нулевого бита. Если это не так, следует после загрузки исходного числа в аккумулятор предварительно сдвинуть его вправо. Для сдвига можно использовать любую операцию сдвига, но удобнее всего сдвиг логический, так как освобождаемые биты заполняются нулями.

Битовая операция И применяется к маске и исходному числу. Маска – это битовая константа, которая имеет единицы в тех разрядах, которые должны остаться неизменными в исходном числе. Остальные биты маски должны быть равны нулю. Например, если нужно оставить в числе 5 младших битов, нужна маска, равная $1\ 1111_2$, имеющая шестнадцатеричное значение 0x1F.

Блок 4 - "Выделить поле www" – реализуют команды, расположенные в строках 63-65. Команда mov в строке 63 передает младший байт исходного числа в результирующую переменную www_a. Следующая команда оставляет в результате только 5 младших битов, для чего при помощи команды and выполняется битовая операция И применительно к маске 11111_2 (0x1F) и результирующему числу. Команда в строке 65 выполняет логический сдвиг исходного числа в аккумуляторе eax на 5 битов вправо, что приводит к размещению соседнего слева поля zzz, начиная с нулевого бита.

Блок 5 – "Выделить поле zzz" - реализуют команды, расположенные в строках 67-69. Команда mov в строке 67 передает младшее слово исходного числа в результирующую переменную zzz_a. Следующая команда оставляет в результате 12 младших битов, для чего при помощи команды and выполняется битовая операция И применительно к маске $1111\ 1111\ 1111_2$ (0xFFF) и результирующему числу. Команда в строке 69 выполняет логический сдвиг вправо аккумулятора eax на 12 битов, что приводит к размещению соседнего слева поля yyy, начиная с нулевого бита.

Блок 6 – "Выделить поле yyy" – реализуют команды, расположенные в строках 71-73. Команда mov в строке 71 передает младшее слово исходного числа в результирующую переменную yyy_a. Следующая команда оставляет в результате 9 младших битов, для чего при помощи команды and выполняется битовая операция И применительно к маске 1 1111 1111_2 (0x1FF) и результирующему числу. Команда в строке 73 выполняет логический сдвиг аккумулятора eax вправо на 9 битов, что приводит к размещению соседнего слева поля xxx, начиная с нулевого бита.

Блок 7 – "Выделить поле xxx" – реализуют команды, расположенные в строках 75-76. Команда mov в строке 75 передает младший байт исходного числа в результирующую переменную xxx_a. Следующая команда оставляет в результате 6 младших битов, для чего при помощи команды and выполняется битовая операция И применительно к маске 11 1111_2 (0x3F) и результирующему числу. Сдвигать исходное число не нужно, так как все поля распакованы.

Блок 8 – "Вывод результатов на дисплей" – выполняют операторы ввода языка C++, расположенные в строках 79-88. Результаты, длиной байт и слово, выводятся в шестнадцатеричном коде. Поскольку необходимо вывести числа, длиной байт и слово в шестнадцатеричном коде, необходимо использовать операторы вывода языка C++ с преобразованием типов. Выполнять вывод байтов из переменных имеющих тип char функциями языка C затруднительно. Для того, чтобы выполнять вывод в 16-м коде в операторе вставки (<<) в поток используется манипулятор hex.

Текст программы с комментариями.

```
 1:  //+==================================================================
 2:  // File unpack.cc
 3:  // Распаковка битовых групп
 4:  //
 5:  // Эта программа распаковывает битовые группы из целого беззнакового
 6:  // числа
 7:  // (C) Дужий В.И., 2012
 8:  //
 9:  // Входные данные:
10:  // Длинное целое число, которое содержит несколько битовых групп
11:  //
12:  //         31----+-----+-----+-----0
13:  //         ! xxx ! yyy ! zzz ! www !  value
14:  //         +-----+-----+-----+-----+
15:  //            6     9     12    5
16:  //
17:  // Выходные данные:
18:  //           7----5------0
19:  //           ! 00 !  xxx !  xxx
20:  //           +----+------+
21:  //                    6
22:  //           15---8------0
23:  //           ! 00 !  yyy !  yyy
24:  //           +----+------+
25:  //                    9
26:  //           15---11-----0
27:  //           ! 00 !  zzz !  zzz
28:  //           +----+------+
```

```
29:  //                    12
30:  //           7----4------0
31:  //           ! 00 !  www !  www
32:  //           +----+------+
33:  //                     5
34:  //-========================================================================
35:  #include <iostream>
36:  #include <iomanip>
37:  using namespace std;
38:
39:  unsigned long  value;
40:  unsigned char  xxx, www, xxx_a, www_a;
41:  unsigned short yyy, zzz, yyy_a, zzz_a;
42:
43:  int main()
44:  {
45:    printf("\n\t\t(C) Дужий В.И., 2012");
46:    printf("\n\tРаспаковка битовых групп");
47:    for (;;)
48:    {
49:      printf("\n\tРаспаковать 32-битовое число Value");
50:    printf("\nПожалуйста, введите 8 16-ых цифр (например, 5a9db8e4) : ");
51:      scanf("%x",&value);
52:  //============================== C ==================================
53:  // Выделить битовые группы из числа Value
54:      www = value & 0x1f;     // выделить 5 младших битов(4-0) из Value
55:      zzz = (value >> 5)  & 0xfff;// выделить 12 битов (16-5)  из Value
56:      yyy = (value >> 17) & 0x1ff;// выделить  9 битов (25-17) из Value
57:      xxx = (value >> 26) & 0x3f ;// выделить  6 битов (31-26) из Value
58:  //============================ Assembler ============================
59:  // Выделить битоые группы из числа Value
60 __asm{
61:      mov    eax,value
62: // выделить 5 младших битов(4-0) из Value
63:      mov    www_a,al
64:      and    www_a,0x1f
65:      shr    eax,5
66: // выделить 12 битов (16-5)  из Value
67:      mov    zzz_a,ax
68:      and    zzz_a,0xfff
69:      shr    eax,12
70: // выделить  9 битов (25-17) из Value
71:      mov    yyy_a,ax
72:      and    yyy_a,0x1ff
73:      shr    eax,$9
74: // выделить  6 битов (31-26) из Value
75:      mov    xxx_a,al
76:      and    xxx_a,0x3f
77: };
78:  // Форматный вывод результатов
79:     cout << hex
80:          << "Битовая группа XXX (C++): " << (int) xxx
81:          << "\nБитовая группа YYY (C++): " << (int) yyy
82:          << "\nБитовая группа ZZZ (C++): " << (int) zzz
83:          << "\nБитовая группа WWW (C++): " << (int) www
84:          << "\n    Битовая группа XXX (Asm): " << (int) xxx_a
85:          << "\n    Битовая группа YYY (Asm): " << (int) yyy_a
86:          << "\n    Битовая группа ZZZ (Asm): " << (int) zzz_a
87:          << "\n    Битовая группа WWW (Asm): " << (int) www_a
```

```
88:             << endl;
89:  };
90:     return 0;
91:  }
```

6. Тестирование программы

Составим тестовые примеры, которые позволят найти ошибки в программе. Для правильного выбора тестовых примеров следует проанализировать поставленную задачу и выбрать значения исходных данных, используя следующие соображения:

– исходное упакованное число должно содержать все биты равные единице. В приведенной ниже таблице это тест 1.

– исходное упакованное число должно содержать чередующиеся нулевые и единичные биты. В приведенной ниже таблице это тесты 2 и 3.

– исходное упакованное число должно содержать все единичные биты только в одной битовой группе и нулевые биты в остальных группах. Всего у нас четыре битовые группы, поэтому таких тестов должно быть четыре. В приведенной ниже таблице это тесты 4-7.

– исходное упакованное число должно содержать произвольные битовые значения в каждой битовой группе, так чтобы в каждой битовой группе была хотя бы одна единица. Таких тестов должно быть несколько. В приведенной ниже таблице это тесты 8-9.

Тестовые примеры.

Номер	Исходное число, value	Ожидаемый результат				Полученный результат				Цель теста
		xxx	yyy	zzz	www	xxx	yyy	zzz	www	
1	FFFF FFFF	3f	1ff	fff	1f					Все биты равны 1
2	AAAAAAAA	2a	155	555	a					Чередующиеся 1 и 0
3	5555 5555	15	aa	aaa	15					Чередующиеся 0 и 1
4	FC000000	3f	0	0	0					Все 1 в битовой группе xxx
5	1F	0	0	0	1f					Все 1 в битовой группе www
6	3FE0000	0	1ff	0	0					Все 1 в битовой группе yyy
7	1FFE0	0	0	fff	0					Все 1 в битовой группе zzz
8	1234 5678									Произвольные биты
9	1A2B 3C4D									Произвольные биты

Протокол тестирования программы приводится ниже.

```
   (C) Дужий В.И., 2012
            Распаковка битовых групп
      Распаковать 32-битовое число Value
Пожалуйста, введите 8 16-ых цифр (например, 5a9db8e4) : FFFFFFFF
Битовая группа XXX (C++): 3f
Битовая группа YYY (C++): 1ff
Битовая группа ZZZ (C++): fff
```

```
Битовая группа WWW (C++): 1f
    Битовая группа XXX (Asm): 3f
    Битовая группа YYY (Asm): 1ff
    Битовая группа ZZZ (Asm): fff
    Битовая группа WWW (Asm): 1f

        Распаковать 32-битовое число Value
Пожалуйста, введите 8 16-ых цифр (например, 5a9db8e4) : AAAAAAAA
Битовая группа XXX (C++): 2a
Битовая группа YYY (C++): 155
Битовая группа ZZZ (C++): 555
Битовая группа WWW (C++): a
    Битовая группа XXX (Asm): 2a
    Битовая группа YYY (Asm): 155
    Битовая группа ZZZ (Asm): 555
    Битовая группа WWW (Asm): a

        Распаковать 32-битовое число Value
Пожалуйста, введите 8 16-ых цифр (например, 5a9db8e4) : 55555555
Битовая группа XXX (C++): 15
Битовая группа YYY (C++): aa
Битовая группа ZZZ (C++): aaa
Битовая группа WWW (C++): 15
    Битовая группа XXX (Asm): 15
    Битовая группа YYY (Asm): aa
    Битовая группа ZZZ (Asm): aaa
    Битовая группа WWW (Asm): 15

        Распаковать 32-битовое число Value
Пожалуйста, введите 8 16-ых цифр (например, 5a9db8e4) : FC000000
Битовая группа XXX (C++): 3f
Битовая группа YYY (C++): 0
Битовая группа ZZZ (C++): 0
Битовая группа WWW (C++): 0
    Битовая группа XXX (Asm): 3f
    Битовая группа YYY (Asm): 0
    Битовая группа ZZZ (Asm): 0
    Битовая группа WWW (Asm): 0

        Распаковать 32-битовое число Value
Пожалуйста, введите 8 16-ых цифр (например, 5a9db8e4) : 1F
Битовая группа XXX (C++): 0
Битовая группа YYY (C++): 0
Битовая группа ZZZ (C++): 0
Битовая группа WWW (C++): 1f
    Битовая группа XXX (Asm): 0
    Битовая группа YYY (Asm): 0
    Битовая группа ZZZ (Asm): 0
    Битовая группа WWW (Asm): 1f

        Распаковать 32-битовое число Value
Пожалуйста, введите 8 16-ых цифр (например, 5a9db8e4) : 3FE0000
Битовая группа XXX (C++): 0
Битовая группа YYY (C++): 1ff
Битовая группа ZZZ (C++): 0
Битовая группа WWW (C++): 0
    Битовая группа XXX (Asm): 0
    Битовая группа YYY (Asm): 1ff
    Битовая группа ZZZ (Asm): 0
```

```
    Битовая группа WWW (Asm): 0

        Распаковать 32-битовое число Value
Пожалуйста, введите 8 16-ых цифр (например, 5a9db8e4) : 1FFE0
Битовая группа XXX (C++): 0
Битовая группа YYY (C++): 0
Битовая группа ZZZ (C++): fff
Битовая группа WWW (C++): 0
    Битовая группа XXX (Asm): 0
    Битовая группа YYY (Asm): 0
    Битовая группа ZZZ (Asm): fff
    Битовая группа WWW (Asm): 0

        Распаковать 32-битовое число Value
Пожалуйста, введите 8 16-ых цифр (например, 5a9db8e4) :^C
```