ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Учебно-методическое пособие

по дисциплине

«Анализ и кодирование информации»

Автор
Гнедина О. А.

Ростов-на-Дону, 2019

Аннотация

Учебно-методическое пособие предназначено для студентов очной формы обучения направления 09.03.04 «Программная инженерия».

Авторы

ст. преподаватель
каф. ПОВТиАС,
Гнедина О.А.

Оглавление

1. ЛАБОРАТОРНАЯ РАБОТА №1: АНАЛИЗ ЧАСТОТНЫХ ХАРАКТЕРИСТИК ДВОИЧНЫХ ФАЙЛОВ

1.1. Цель работы

Освоение работы с двоичными файлами и методов получения частотных характеристик двоичных файлов.

1.2. Теоретическая часть

Двоичный файл отличается от текстового тем, что данные в нем представлены во внутренней форме. Поскольку при внутреннем представлении используется двоичная система счисления, то файлы и называются двоичными. По существу, двоичный файл является аналогом внутренней (оперативной, физической) памяти – неограниченным массивом байтов с возможностью непосредственного обращения (произвольного доступа) к любой его части.

Такая модель файла полностью совпадает с системой представлений, принятой в Си для работы с памятью на низком (физическом уровне).

– физическая память имеет байтную структуру – единицей адресации является байт;

– любая переменная занимает фиксированное количество байтов, определяемое ее типом. Операция sizeof возвращает эту размерность;

– указатель на переменную интерпретируется как ее адрес в памяти. Преобразование типа указателя к void* позволяет интерпретировать его как адрес, а преобразование к char* - как указатель на массив байтов.

Исходя из этих принципов, функции двоичного ввода-вывода *fread* и *fwrite* переносят содержимое памяти в двоичный файл байт в байт без каких либо преобразований. Функции используются для перенесения данных из файла в память программы (чтение) и обратно (запись).

```
int     fread  (void *buf, int size, int nrec, FILE *fd);
int     fwrite (void *buf, int size, int nrec, FILE *fd);
```

Особенностью этих функций является то, что для них безразличен (неизвестен) характер структуры данных в той области памяти, в которую осуществляется ввод-вывод (указатель void*

buf). Функция fread читает, а функция fwrite пишет в файл, начиная с текущей позиции, массив из nrec элементов размерностью size байтов каждый, возвращая количество успешно прочитанных (записанных) элементов.

Чтобы воспользоваться этими функциями, необходимо обеспечить преобразования переменных к «массиву байтов», используя указатели для задания адресов и операцию *sizeof* для вычисления размерности:

```
// Прочитать  целую переменную и следующий за ней
// динамический массив из n переменных типа double
int n;              // в целой переменной– размерность массива
fread(&n, sizeof(int),1,fd);        // указатель на переменую int
double   *pd = new double[n];
fread(pd, sizeof(double),n,fd);    // преобразование к void* -
 неявное
```

Произвольный доступ базируется на понятии адреса в двоичном файле. Поскольку на физическом уровне двоичный файл представляется как «неограниченно растущий» массив байтов, то под адресом понимается порядковый номер байта, начиная с 0.

С каждым открытым файлом связывается такой параметр как текущая позиция (текущий адрес) - номер байта, начиная с которого будет выполняться очередная операция чтения-записи. При открытии файла текущая позиция устанавливается на начало файла, после чтения-записи порции данных перемещается вперед на размерность этих данных. Для дополнения файла новыми данными необходимо установить текущую позицию на конец файла и выполнить операцию записи. *Т*екущая позиция в файле является адресом размещения переменной в нем, но получить этот адрес можно *перед, а* не *после* ее чтения оттуда.

Текущую позицию можно читать и устанавливать с помощью функций позиционирования, которые превращают последовательный файл в файл произвольного доступа. Функция *long ftell(FILE *fp)* возвращает текущую позицию в файле. Если по каким-то причинам текущая позиция не определена, функция возвращает -1.

Функция *int fseek(FILE *fp, long pos, int mode)* устанавливает текущую позицию в файле на байт с номером pos. Параметр mode определяет, относительно чего отсчитывается текущая позиция в файле, и имеет символические и числовые значения (установленные bstdio.h):

```
#define SEEK_SET 0    // Относительно начала файла
                      // начало файла - позиция 0
#define  SEEK_CUR 1   // Относительно текущей позиции,
                      // >0 - вперед, <0 - назад
#define SEEK_END 2    // Относительно конца файла
                      // (значение pos - отрицательное)
```

Функция *fseek* возвращает значение 0 при успешном позиционирование и -1 при ошибке. Различные значения параметра mode определяют различные способы адресации данных в файле:

– значение SEEK_SET определяет абсолютную адресацию данных в файле от его начала. Заметим, что функция ftell возвращает текущую позицию в абсолютном значении;

– значение SEEK_END за начало координат берет конец файла;

– значение SEEK_CUR дает способ относительной адресации от текущего положения указателя в файле. Таким образом, задается *расстояние* в байтах от текущей переменной до адресуемой. Если это *расстояние* само находится в файле, то оно обычно носит название смещения.

С помощью функции fseek можно определить размер файла. Определить текущую длину файла можно простым позиционированием:

```
long      fsize;
fseek(fl,0L,SEEK_END); // Установить позицию на конец файла
fsize = ftell(fd);          // Прочитать значение текущей позиции
```

При открытии или создании нового файла необходимо указать режим работы с файлом как с двоичным. Среди множества режимов можно выделить два: создание нового файла для записи всех данных, либо начальной структуры данных и открытие существующего файла с уже имеющейся структурой данных для чтения, записи и добавления. Последний режим наиболее точно соответствует модели двоичного файла как неограниченно расширяемого прямо адресуемого массива байтов.

```
// Открыть существующий как двоичный для чтения и записи
```

```
FILE *fd; fd = fopen("a.dat","rb+wb");
// Создать новый как  двоичный для записи  и чтения
fd = fopen("a.dat","wb+");
```

Задачей данной лабораторной работы является анализ частоты встречаемости различных символов в файле. Ниже приведен пример подсчета частоты встречаемости символов в текстовой строке. Как видно из примера, для упрощения процедуры подсчета частоты встречаемости символов, вместо проверки номера символа в массиве, сам символ используется в качестве указателя на элемент массива. Делается это за счет приведения типов: *map[(unsigned char)str[i]]++.* Так как один байт определяет значения в диапазоне от 0 до 255, то для хранения информации будет достаточно линейного массива из 255 элементов.

```
#include <stdio.h>
#include <string.h>

#define map_size 256

int main()
{
   int map[map_size] = {0};

   const char * str = "Текстовая строка для анализа";
   int str_len = strlen(str);

   int i;
   for(i = 0; i < str_len; i++)
   {
      map[(unsigned char)str[i]]++;
   }

   for(i = 0; i < map_size; i++)
   {
      if(map[i])
      printf("'%c' => %d\n", (char)i, map[i]);
   }

   return 0;
}
```

1.3. Задание к лабораторной работе

1. Реализовать функцию чтения и записи небольших файлов в оперативную память (стек программы).
2. Подсчитать количество различных символов в файле и вывести гистограмму по встречаемости различных символов.
3. Вывести на экран частоты встречаемости всех символов в заданном текстовом файле.

1.4. Вопросы для самоконтроля

1) Чем отличаются процедуры открытия двоичного и текстового файлов?
2) Как осуществляется адресация в двоичном файле?
3) Как определить размер файла?
4) Какой функцией можно целиком прочитать в память небольшой файл?
5) Что такое – операция приведения типов?

2. ЛАБОРАТОРНАЯ РАБОТА №2: ОЦЕНКА КОЛИЧЕСТВА ИНФОРМАЦИИ В ФАЙЛЕ И АНАЛИЗ ИЗБЫТОЧНОСТИ КОДИРОВАНИЯ

2.1. Цель работы

Практическое знакомство с методами оценки количества информации и пределами сжатия сообщений.

2.2. Теоретическая часть

Количество информации, приходящейся на один элемент сообщения, называется удельной информативностью или энтропией.

$H=$

$$\frac{I}{n} = -\sum_{i=1}^{m} P_i \log P_i = \sum_{i=1}^{m} P_i \log \frac{1}{P_i}.$$

Количество информации и энтропия являются логарифмическими мерами и измеряются в одних и тех же единицах. Основание логарифма определяет единицу измерения количества информации и энтропии. Двоичная единица соответствует основанию логарифма, равному двум, и называется битом. Один бит - это количество информации в сообщении в одном из двух равно-

вероятностных исходов некоторого опыта.

Из формулы Шеннона следует, что количество информации, содержащейся в сообщении, зависит от числа элементов сообщения *n*, алфавита *r* и вероятностей выбора элементов P_i.

Зависимость *I* от *n* является *линейной*.

Отметим некоторые свойства энтропии:

1. Энтропия является величиной вещественной, ограниченной и неотрицательной, то есть *H*>0. Это свойство следует из выражения.

2. Энтропия минимальна и равна нулю, если сообщение известно заранее, то есть если P_i=1, а

$$P_1 = P_2 = \ldots = P_{i-1} = P_{i+1} = \ldots = P_n = 0$$

.

3. Энтропия максимальна, если все состояния элементов сообщений равновероятны.

H=$H_{\max}$

$$P_1 = P_2 = \ldots = P_i = \ldots = P_m = P = \frac{1}{m}$$

, если .

Величина максимальной энтропии:

$$H_{\max} = -\sum_{i=1}^{m} \frac{1}{m} \log \frac{1}{m}$$

4. Энтропия бинарных (двоичных) сообщений может изменяться от нуля до единицы (двоичный алфавит, следовательно, *r*=2.)

H= $-\sum_{i=1}^{2} P_i \log P_i = -P_1 \log P_1 - P_2 \log P_2$.

Информационное содержание одного сообщения в потоке, в наиболее общем случае, определяется как:

$$r = \mathbb{E}H(M_t | M_{t-1}, M_{t-2}, M_{t-3}, \ldots)$$

Обозначим как *R* логарифм числа символов в алфавите сообщений:

$$R = \log |M|$$

Абсолютная избыточность может быть определена как разность этих двух величин:

$$D = R - r_{\frac{D}{R}}$$

Соотношение $\frac{D}{R}$ называется относительной избыточностью и дает математическую оценку максимальной степени сжатия, на которую может быть уменьшен размер файла.

2.3. Задание к лабораторной работе

На основе программы полученной в результате выполнения первой лабораторной работы реализовать программное средство позволяющей по заданному файлу оценить следующие параметры:

- энтропию источника сообщения;
- количество информации в сообщении;
- абсолютную избыточность кодирования;
- теоретическую максимальную степень сжатия сообщения.

2.4. Вопросы для самоконтроля

1) Что такое энтропия?

2) Как зная частоты встречаемости символов определить величину энтропии источника сообщения?

3) Каковы основные свойства энтропии?

4) Как определить количество информации в сообщении?

5) Что такое избыточность кодирования и как её можно определить?

6) Чем отличается абсолютная избыточность от относительной избыточности?

7) Как определить теоретическую максимальную степень сжатия сообщения?

3. ЛАБОРАТОРНАЯ РАБОТА №3: ПЕРЕКОДИРОВАНИЕ ТЕКСТОВЫХ ФАЙЛОВ

3.1. Цель работы

Практическое знакомство с особенностями хранения текстовой информации в двоичных файлах, принципах использования различных кодовых таблиц и их особенностей.

3.2. Теоретическая часть

На сегодняшний день большое количество пользователей при помощи компьютера обрабатывает текстовую информацию, которая состоит из: букв, цифр, знаков препинания и других элементов.

Обычно для кодирования одного символа, используется 1 байт то есть 8 бит. Принцип данного кодирования заключается в том, что каждому символу (букве, знаку) соответствует свой двоичный код от 00000000 до 11111111. Текстовая информация может быть представлена в десятичном коде от 0 до 255.

Использование различных кодовых таблиц

Институт стандартизации США *(ANSI - American National Standard Institute)* ввел в действие систему кодирования *ASCII (American Standard Code for Information Interchange - стандартный код информационного обмена США).* В системе *ASCII* закреплены две таблицы кодирования - *базовая* и *расширенная.* Базовая таблица закрепляет значения кодов от 0 до 127, а расширенная относится к символам с номерами от 128 до 255.

Первые 32 кода базовой таблицы, начиная с нулевого, отданы производителям аппаратных средств (в первую очередь производителям компьютеров и печатающих устройств). В этой области размещаются так называемые *управляющие коды,* которым не соответствуют никакие символы языков, и, соответственно, эти коды не выводятся ни на экран, ни на устройства печати, но ими можно управлять тем, как производится вывод прочих данных.

Начиная с кода 32 по код 127 размещены коды символов английского алфавита, знаков препинания, цифр, арифметических действий и некоторых вспомогательных символов. Базовая таблица кодировки *ASCII* приведена в таблице 1.1.

Таблица 1.1. Базовая таблица кодировки ASCII

32	пробел	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	

Аналогичные системы кодирования текстовых данных были разработаны и в других странах. Так, например, в СССР в этой области действовала система кодирования *КОИ7 (код обмена информацией, семизначный).* Однако поддержка производителей оборудования и программ вывела американский код *ASCII* на уровень международного стандарта, и национальным системам кодирования пришлось «отступить» во вторую, расширенную часть системы кодирования, определяющую значения кодов со 128 по 255. Отсутствие единого стандарта в этой области привело к множественности одновременно действующих кодировок. Только в России можно указать три действующих стандарта кодировки и еще два устаревших.

Так, например, кодировка символов русского языка, известная как кодировка *Windows 1251,* была введена «извне» — компанией Microsoft , но, учитывая широкое распространение операционных систем и других продуктов этой компании в России, она глубоко закрепилась и нашла широкое распространение (таблица 1.2). Эта кодировка используется на большинстве локальных компьютеров, работающих на платформе Windows .

Таблица 1.2. Кодировка Windows 1251

128 Ђ	144 ђ	160	176 °	192 А	208 Р	224 а	240 р
129 Ѓ	145 ‘	161 Ў	177 ±	193 Б	209 С	225 б	241 с
130 ‚	146 ’	162 ў	178 І	194 В	210 Т	226 в	242 т
131 ѓ	147 “	163 Ј	179 і	195 Г	211 У	227 г	243 у
132 „	148 ”	164 ¤	180 ґ	196 Д	212 Ф	228 д	244 ф
133 …	149 •	165 Ґ	181 µ	197 Е	213 Х	229 е	245 х
134 †	150 –	166 ¦	182 ¶	198 Ж	214 Ц	230 ж	246 ц
135 ‡	151 —	167 §	183 ·	199 З	215 Ч	231 з	247 ч
136 €	152	168 Ё	184 ё	200 И	216 Ш	232 и	248 ш
137 ‰	153 ™	169 ©	185 №	201 Й	217 Щ	233 й	249 щ
138 Љ	154 љ	170 Є	186 є	202 К	218 Ъ	234 к	250 ъ
139 ‹	155 ›	171 «	187 »	203 Л	219 Ы	235 л	251 ы
140 Њ	156 њ	172 ¬	188 ј	204 М	220 Ь	236 м	252 ь
141 Ќ	157 ќ	173 -	189 S	205 Н	221 Э	237 н	253 э
142 Ћ	158 ћ	174 ®	190 s	206 О	222 Ю	238 о	254 ю
143 Џ	159 џ	175 Ї	191 ї	207 П	223 Я	239 п	255 я

Другая распространенная кодировка носит название *КОИ8 (код обмена информацией, восьмизначный)* — ее происхождение относится ко временам действия Совета Экономической Взаимопомощи государств Восточной Европы (таблица1.3). Сегодня кодировка *КОИ8* имеет широкое распространение в компьютерных сетях на территории России и в российском секторе Интернета.

Таблица 1.3. Кодировка КОИ-8

128	144	160	176	192 ю	208 п	224 Ю	240 П
129	145	161 Ё	177	193 а	209 я	225 А	241 Я
130	146	162	178	194 б	210 р	226 Б	242 Р
131	147	163 ё	179 Ё	195 ц	211 с	227 Ц	243 С
132	148	164	180	196 д	212 т	228 Д	244 Т
133	149	165	181	197 е	213 у	229 Е	245 У
134	150 √	166	182	198 ф	214 ж	230 Ф	246 Ж
135	151 ≈	167	183	199 г	215 в	231 Г	247 В
136	152 ≤	168	184	200 х	216 ь	232 Х	248 Ь
137	153 ≥	169	185	201 и	217 ы	233 И	249 Ы
138	154	170	186	202 й	218 з	234 Й	250 З
139	155 ⌠	171	187	203 к	219 ш	235 К	251 Ш
140	156 ⌡	172	188	204 л	220 э	236 Л	252 Э
141	157 ²	173	189	205 м	221 щ	237 М	253 Щ
142	158 ·	174	190	206 н	222 ч	238 Н	254 Ч
143	159 ÷	175	191 ё	207 о	223 ъ	239 О	255 Ъ

Международный стандарт, в котором предусмотрена кодировка символов русского алфавита, носит название кодировки *ISO (International Standard Organization — Международный институт стандартизации).* На практике данная кодировка используется редко (таблица 1.4).

Таблица 1.4. Кодировка ISO

В *ISO* не определены	160	176 А	192 Р	208 а	224 р	240 №
	161 Ё	177 Б	193 С	209 б	225 с	241 ё
	162 Ђ	178 В	194 Т	210 в	226 т	242 ђ
	163 Ѓ	179 Г	195 У	211 г	227 у	243 ѓ
	164 Є	180 Д	196 Ф	212 д	228 ф	244 є
	165 S	181 Е	197 Х	213 е	229 х	245 s
	166 І	182 Ж	198 Ц	214 ж	230 ц	246 і
	167 Ї	183 З	199 Ч	215 з	231 ч	247 ї
	168 J	184 И	200 Ш	216 и	232 ш	248 j
	169 Љ	185 Й	201 Щ	217 й	233 щ	249 љ
	170 Њ	186 К	202 Ъ	218 к	234 ъ	250 њ
	171 Ћ	187 Л	203 Ы	219 л	235 ы	251 ћ
	172 Ќ	188 М	204 Ь	220 м	236 ь	252 ќ
	173 -	189 Н	205 Э	221 н	237 э	253 §
	174 Ў	190 О	206 Ю	222 о	238 ю	254 ў
	175 Џ	191 П	207 Я	223 п	239 я	255 џ

Если проанализировать организационные трудности, связанные с созданием единой системы кодирования текстовых данных, то можно прийти к выводу, что они вызваны ограниченным набором кодов (256). В то же время, очевидно, что если, например, кодировать символы не восьмиразрядными двоичными числами, а числами с большим количеством разрядов, то и диапазон возможных значений кодов станет на много больше. Такая систе-

ма, основанная на 16 разрядном кодировании символов, получила название универсальной - **UNICODE**. Шестнадцать разрядов позволяют обеспечить уникальные коды для 65 536 различных символов — этого поля достаточно для размещения в одной таблице символов большинства языков планеты.

Перекодировка

Под **перекодировкой** мы будем понимать любое удачно выбранное представление исходной информации, при котором информацию легко можно восстановить обратно и которое занимает меньший физический объем на носителе.

Рассмотрим пример. Представим, что наш источник с равной вероятностью вырабатывает 6 различных сообщений. Для их кодирования приходит на ум очевидное решение использовать двоичные коды чисел от 0 до 5: 000, 001, 010, 011, 100, 101. Так же очевидно, что стоимость кодирования получилась 3 бита, но энтропия при этом $H_0 = \log_2 6 = 2.585 < 3$. Причина в том, что у нас остались неиспользованными 2 кода: 110 и 111.

Можно сделать так, чтобы лишних кодов не было. Для этого используются **коды переменной длины**. Для нашего случая:

0 - 00 1 - 01 2 - 100 3 - 101 4 - 110 5 - 111.

Стоимость кодирования такого кода равна 2.667, что значительно ближе к энтропии.

3.3. Задание к лабораторной работе

1. Для заданного файла определить тип используемой кодовой таблицы.
2. Создать свою кодовую таблицу для русского языка, состоящую из 64 символов.
3. Реализовать алгоритм перекодирования файла из исходного формата (ASCII, Windows-1251, KOI-8, UNICODE) в 6-битный формат в соответствии с предложенной кодовой таблицей.

3.4. Вопросы для самоконтроля

1) Как кодируются текстовые данные?
2) Что такое кодовая таблица?
3) Что такое таблица ASCII кодов?
4) Какие кодовые таблицы могут быть использованы для кодирования текстов на русском языке?
5) В чем особенность таблицы UNICODE?
6) Что такое перекодировка?

7) Как можно уменьшить объем файла, за счет перекодировки?

4. ЛАБОРАТОРНАЯ РАБОТА №4: РЕАЛИЗАЦИЯ БАЗОВОГО АЛГОРИТМА RLE

4.1. Цель работы

Практическое знакомство с алгоритмами сжатия информации без потери качества.

4.2. Теоретическая часть

Алгоритм RLE является самым быстрым, простым и понятным алгоритмом сжатия данных и при этом иногда оказывается весьма эффективным. Именно подобный алгоритм используется для сжатия изображений в файлах *PCX*.

Он заключается в следующем: любой последовательности повторяющихся входных символов ставится в соответствие набор из трех выходных символов: первый-байт префикса, говорящий о том, что встретилась входная повторяющаяся последовательность, второй-байт, определяющий длину входной последовательности, третий-сам входной символ-*<prefix,length,symbol>.* Лучше всего работу алгоритма пояснить на конкретном примере.

Например: пусть имеется (шестнадцатиричный) текст из 20 байт:

05 05 05 05 05 05 01 01 03 03 03 03 03 03 05 03 FF FF FF FF

Выберем в качестве префикса байт FF. Тогда на выходе архиватора мы получим последовательность

FF 06 05 FF 02 01 FF 06 03 FF 01 05 FF 01 03 FF 04 FF

Ее длина-18 байт, то есть достигнуто некоторое сжатие. Однако, нетрудно заметить, что при кодировании некоторых символов размер выходного кода возрастает (например, вместо 01 01 - FF 02 01). Очевидно, одиночные или дважды (трижды) повторяющиеся символы кодировать не имеет смысла - их надо записывать в явном виде. Получим новую последовательность:

FF 06 05 01 01 FF 06 03 05 03 FF 04 FF

длиной 13 байт. Достигнутая степень сжатия: 13/20*100 = 65%.

Нетрудно заметить, что префикс (маркер) может совпасть с одним из входных символов. В этом случае входной символ может быть заменен своим “префиксным” представлением, например:

FF то же самое, что и FF 01 FF (три байта вместо одного).

Поэтому, от правильного выбора префикса зависит качество самого алгоритма сжатия, так как, если бы в нашем исходном тексте часто встречались одиночные символы FF, размер выходного текста мог бы даже превысить входной. В общем, случае в качестве префикса следует выбирать самый редкий символ входного алфавита.

Таким образом, при выполнении лабораторной работы на первом этапе необходимо найти самый редко встречаемый в кодируемом файле символ, для того, чтобы использовать его в качестве маркера. Следует отметить, что в качестве маркера может быть использован любой символ, который ни разу не встретился в файле.

4.3. Задание к лабораторной работе

1. Написать функции чтения и записи данных в двоичный файл.
2. Реализовать алгоритм поиска префикса в байтовом массиве.
3. Реализовать алгоритм кодирования данных посредствам RLE кода.
4. Предложить и реализовать алгоритм декодирования RLE кода.

4.4. Вопросы для самоконтроля

1) Что понимается под серией байт?

2) Какова должна быть минимальная длина серии байтов, чтобы она могла быть эффективно сжата с помощью RLE кода?

3) Какие символы можно использовать в качестве префикса при кодировании сообщения с использованием RLE кода?

4) Что делать, если в исходном сообщении при его кодировании встретился символ равный префиксу RLE кода?

5) В каком формате следует открыть файл на чтение и запись для кодирования его данных с помощью RLE кода?

6) Как можно заранее определить длину декодируемого сообщения?

5. ЛАБОРАТОРНАЯ РАБОТА №5: РЕАЛИЗАЦИЯ АЛГОРИТМА ЛЕМПЕЛЯ-ЗИВА

5.1. Цель работы

Практическое знакомство с алгоритмами сжатия информации без потери качества.

5.2. Теоретическая часть

Алгоритм названный так по первым буквам фамилий авто-

ров и году своего опубликования. Классический алгоритм предельно прост, и может быть описан следующим образом: "если в прошедшем ранее выходном потоке уже встречалась подобная последовательность байт, причем запись о ее длине и смещении от текущей позиции короче чем сама эта последовательность, то в выходной файл записывается ссылка (смещение, длина), а не сама последовательность".

Пример 1: Строка "КОЛОКОЛ_ОКОЛО_КОЛОКОЛЬНИ" с помощью алгоритма LZ77 будет закодирована строкой "КОЛО(-4,3)_(-5,4)О_(-14,7)ЬНИ".

Пример 2: Строка "АААААА" с помощью алгоритма LZ77 будет закодирована "А(-1,6)".

Основу алгоритма составляют четыре базовых принципа:

1. Каждая очередная закодированная последовательность символов добавляется к ранее закодированным символам таким образом, что вместе с ними она образует разложение всей исходной последовательности на несовпадающие между собой фразы.
2. Разложение хранится в памяти и используется в дальнейшем в качестве словаря.
3. Кодирование осуществляется при помощи указателей на фразы из уже сформированного словаря фраз.
4. Кодирование является динамической процедурой, ориентированной на блоки. Сам процесс кодирования может быть дополнен скользящими окнами, содержащими текущий словарь фраз и окно просмотра.

Скользящий (*sliding*) словарь V объемом |V|=2...32 кБ. В качестве словаря используются |V| байт сжимаемого текста, *предшествующие* текущему кодируемому символу в позиции *pos* (отсюда и название "скользящий"-словарь как бы скользит вдоль по тексту от его начала к концу, поддерживая самую свежую информацию о его содержании). Перед началом процесса компрессии словарь пуст (см. рис 1 а).

Под строкой будем понимать любую непрерывную последовательность символов (букв), начинающаяся с определенной позиции в словаре или в тексте, длиной $|s| \le s_{max}$. s_{max} обычно выбирается из диапазона 16...256 байт. Словами в словаре выступают любые строки длины не более s_{max}, начинающиеся в любой позиции словаря. Таким образом, в словаре всегда присутствует $|V| - s_{max}$ слов.

Если очередная входная строка s текста совпадает со строкой из словаря, то она заменяется на указатель *ptr* вида *ptr* = <1, *distance, length*>. Если же строки s в словаре не оказалось, генерируется код *chr* вида *chr* = < 0, *symbol*>, где *symbol*-текущий символ исходного текста.

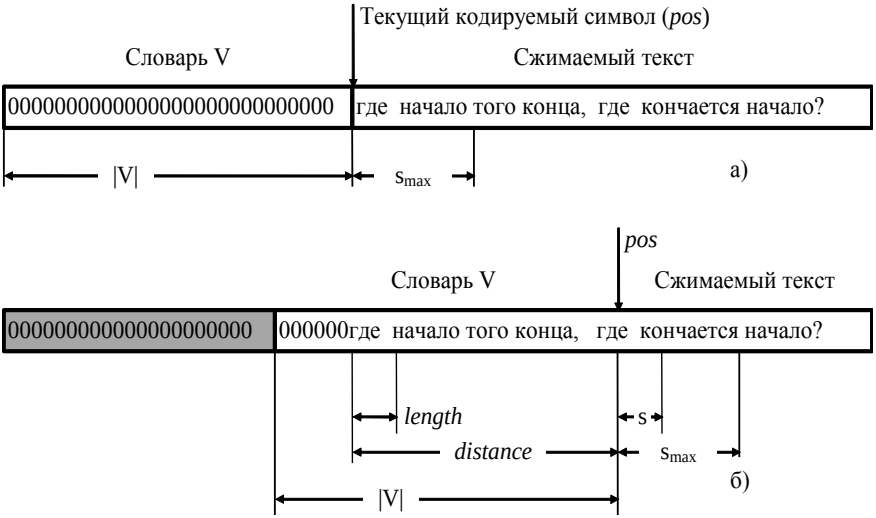


Рисунок 1 - Формирование и использование словаря.

На рис 1. б) показан вид словаря и кодируемого текста на 23-ем символе, когда алгоритм обнаружил совпадение слова "где " с таким же словом в словаре длиной *length* = 4 байта, на расстоянии *distance*=23 байта от текущей позиции *pos*. Будет сгенерирован код *ptr*=<1,23,4>.

Префикс, как видно, нужен для того, чтобы отличить код *ptr* от кода *chr*. Именно префикс вносит в алгоритм кодирования LZ77 некоторую дополнительную информацию, из-за которой возможно возрастание избыточности. Длина ptr-кода

$|ptr| = 1 + \lceil \log(|V|) \rceil + \lceil \log(|s_{max}|) \rceil$ (бит),

chr-кода

$|chr| = 1 + \lceil \log(|symbol|) \rceil$ (бит).

Очевидно, что если длина ptr-кода в байтах превышает *length*, то такое кодирование лишь увеличит избыточность, поэтому в алгоритм вводят порог (threshold), равный обычно 2-3 байтам и, если совпадающая строка короче этого порога, она за-

писывается в явном виде посредством chr-кодов.

Обратное восстановление текста осуществляется значительно быстрее и проще процедуры кодирования. Поскольку к моменту декодирования очередного символа весь предыдущий текст уже известен, то декодеру не требуется передавать словарь. Словарь строится декодером автоматически по тому же алгоритму, на основе уже полученных символов. Если декодер обнаруживает, что поступил chr-код, он просто копирует *symbol* в выходной поток и в словарь, передвигая затем словарь на одну позицию вслед за текстом. Если же поступил ptr-код, декодер копирует *length* символов, начиная с позиции *distance* в словаре. Напомним, что *distance* отсчитывается от конца словаря к началу.

5.3. Задание к лабораторной работе

1. Написать функции чтения и записи данных в двоичный файл.
2. Реализовать алгоритм кодирования данных LZ77.
3. Предложить и реализовать алгоритм декодирования LZ77 кода.

5.4. Вопросы для самоконтроля

1) В чем основное отличие кода LZ77 от RLE?
2) Что представляет собой словарь в классическом алгоритме Лемпеля-Зива и как он может быть реализован практически?
3) Нужен ли в алгоритме LZ77 префикс?
4) Указатель на начало слова в словаре имеет положительную или отрицательную величину и как он может быть записан?
5) Какая длина словаря будет оптимальной для кодирования двоичных файлов и от чего она зависит?

6. ЛАБОРАТОРНАЯ РАБОТА № 6-7: ЧТЕНИЕ И АНАЛИЗ ГРАФИЧЕСКИХ ФАЙЛОВ ФОРМАТА BMP. РАЗЛОЖЕНИЕ ЦВЕТНЫХ ИЗОБРАЖЕНИЙ НА СОСТАВЛЯЮЩИЕ ЦВЕТА; СЖАТИЕ ИЗОБРАЖЕНИЙ ФОРМАТА BMP ДОПУСКАЮЩЕЕ ПОТЕРЮ КАЧЕСТВА ИЗОБРАЖЕНИЯ С ПРЕДВАРИТЕЛЬНОЙ ОБРАБОТКОЙ И ИСПОЛЬЗОВАНИЕМ КОДОВ RLE И ЛЕМПЕЛЯ-ЗИВА

6.1. Цель работы

- Изучение особенностей хранения информации о графических изображениях и способы сжатия информации.
- Изучение особенностей хранения информации о графических изображениях.

6.2. Теоретическая часть

BMP (от англ. Bitmap Picture) – формат хранения растровых изображений, разработанный компанией Microsoft. Глубина цвета в данном формате может быть 1, 4, 8, 16, 24, 32, 48 бит на пиксель. При этом для глубины цвета меньше 16 бит используется палитра с полноцветными компонентами глубиной 24 бита.

При использовании формата DIB (англ. Device Independent Bitmap, аппаратно-независимый растр) программист может получить доступ ко всем элементам структур, описывающих изображение, при помощи обычного указателя. Но эти данные не используются для непосредственного управления экраном, так как они всегда хранятся в системной памяти, а не в специализированной видеопамяти. Формат пикселя в оперативной памяти может отличаться от того формата, который должен заноситься в видеопамять для индикации точки такого же цвета. Например, в DIB-формате может использоваться 24 бита для задания пикселя, а графический адаптер в этот момент может работать в режиме HiColor с цветовой глубиной 16 бит. При этом ярко-красная точка в аппаратно-независимом формате будет задаваться тремя байтами 0x0000ff, а в видеопамяти — словом 0xF800. При копировании картинки на экран система будет тратить дополнительное время на преобразование кодов цвета из 24-битного формата в формат видеобуфера.

Формат DDB (англ. Device Dependent Bitmap, аппаратно-зависимый растр) всегда содержит цветовые коды, совпадающие с кодами видеобуфера, но храниться он может как в системной, так и в видеопамяти. В обоих случаях он содержит только коды цвета в том формате, который обеспечит пересылку изображения из ОЗУ в видеопамять при помощи простого копирования.

BMP-файл состоит из четырёх частей:

1) Заголовок файла BITMAPFILEHEADER;
2) Заголовок изображения BITMAPINFOHEADER;
3) Палитра (может отсутствовать);
4) Данные изображения.

Смещение	Длина поля	Описание поля
Заголовок файла		
0	2	Код 4D42h - Буквы 'BM'
2	4	Размер файла в байтах
6	2	0 (Резервное поле)
8	2	0 (Резервное поле)
10	4	Смещение, с которого начинается само изображение
Заголовок BITMAP (Информация об изображении)		
14	4	Размер заголовка BITMAP (в байтах) равно 40
18	4	Ширина изображения в пикселях
22	4	Высота изображения в пикселях
26	2	Число плоскостей, должно быть 1
28	2	Бит/пиксел: 1, 4, 8 или 24
30	4	Тип сжатия
34	4	0 или размер сжатого изображения в байтах.
38	4	Горизонтальное разрешение, пиксел/м
42	4	Вертикальное разрешение, пиксел/м
46	4	Количество используемых цветов
50	4	Количество "важных" цветов.
Палитра (Карта цветов для N цветов), если есть		
54	4*N	Палитра

BITMAPFILEHEADER

Эта структура содержит информацию о типе, размере и представлении данных в файле. Размер структуры – 14 байт.

```
typedef unsigned long   DWORD;
typedef unsigned int    WORD;
typedef signed long     LONG;
typedef unsigned int    UINT;

// Заголовок файла
typedef struct tagBITMAPFILEHEADER
{       UINT    bfType;
        DWORD   bfSize;
        UINT    bfReserved1;
        UINT    bfReserved2;
```

```
        DWORD   bfOffBits;
} BITMAPFILEHEADER;
```

Поля структуры:

bfType – тип файла, символы «BM» (в HEX: 0x42 0x4d);

bfSize – размер всего файла в байтах;

bfReserved1 и bfReserved2 – зарезервированы, должны содержать нули;

bfOffBits – содержит смещение в байтах от начала структуры BITMAPFILEHEADER до непосредственно битов изображения.

BITMAPINFOHEADER

Стандартный вариант заголовка. Размер структуры – 40 байт.

```
// Заголовок Bitmap
typedef struct tagBITMAPINFOHEADER
{       DWORD   biSize;
        LONG    biWidth;
        LONG    biHeight;
        WORD    biPlanes;
        WORD    biBitCount;
        DWORD   biCompression;
        DWORD   biSizeImage;
        LONG    biXPelsPerMeter;
        LONG    biYPelsPerMeter;
        DWORD   biClrUsed;
        DWORD   biClrImportant;
} BITMAPINFOHEADER;
```

Поля структуры:

biSize – размер данной структуры в байтах. Формат BMP со временем дополнялся и по значению этого поля определяется версия формата;

biWidth – ширина изображения в пикселях. Для Win98/Me и Win2000/XP: если поле biCompression содержит BI_JPEG или BI_PNG, здесь указана ширина распакованного изображения;

biHeight – высота изображения в пикселях. Если содержит положительное значение – изображение записано в порядке снизу-вверх (нулевой пиксель в нижнем левом углу). Если значение отрицательное – изображение записано сверху-вниз.

biPlanes – количество цветовых плоскостей и в формате BMP содержит единицу;

biBitCount – количество бит на пиксель. Может принимать следующие значения:

0 – имеет смысл для Win98/Me/2000/XP. Число бит на пиксель определяет формат JPEG или PNG.

1 – изображение монохромное. Член bmiColors структуры BITMAPINFO содержит два элемента. Каждый бит изображения представляет один пиксель; если бит равен нулю — пиксель имеет цвет первого элемента таблицы bmiColors, иначе — цвет второго.

4 – шестнадцатицветное изображение. Пиксели определяются 4-х битными индексами, каждый байт изображения содержит информацию о двух пикселях — старшие 4 бита для первого, оставшиеся — для второго.

8 – в палитре содержится до 256 цветов, каждый байт изображения хранит индекс в палитре для одного пикселя.

16 – если поле biCompression содержит значение BI_RGB, файл не содержит палитры. Каждые два байта изображения хранят интенсивность красной, зелёной и синей компоненты одного пикселя. При этом старший бит не используется, на каждую компоненту отведено 5 бит:

0RRRRRGGGGGBBBBB.

Если поле biCompression содержит значение BI_BITFIELDS, палитра хранит три четырёхбитных значения, определяющих маску для каждой из трёх компонент цвета. Каждый пиксель изображения представлен двухбайтным значением, из которого с помощью масок извлекаются цветовые компоненты. Для WinNT/2000/XP – последовательности бит каждой компоненты должны следовать непрерывно, не перекрываясь и не пересекаясь с последовательностями других компонент. Для Win95/98/Me – поддерживаются только следующие маски: 5-5-5, где маска синей компоненты 0x001F, зелёной 0x03E0, красной 0x7C00; и 5-6-5, где маска синей компоненты 0x001F, зелёной 0x07E0, красной 0xF800.

24 – палитра не используется, каждая тройка байт изображения представляет один пиксель, по байту для интенсивности синего, зелёного и красного канала соответственно.

32 – если поле biCompression содержит значение BI_RGB, изображение не содержит палитры. Каждые четыре байта изображения представляют один пиксель, по байту для интенсивности синего, зелёного и красного канала соответственно. Старший байт каждой четвёрки обычно не используется, однако позволяет хранить данные альфа-канала.

Если поле biCompression содержит значение BI_BITFIELDS, в палитре хранятся три четырёхбайтных цветовых маски – для

красной, зелёной и синей компоненты. Каждый пиксель изображения представлен четырьмя байтами. WinNT/2000: маски компонент не должны перекрываться или пересекаться. Windows 95/98/Me: система поддерживает только один режим сжатия, полностью аналогичный режиму без компрессии BI_RGB – старший байт каждой четвёрки используется в качестве альфа-канала, следующие три отведены для синего, зелёного и красного канала соответственно:

0xAARRGGBB.

biCompression – тип сжатия для сжатых изображений. Если изображение не сжато, то поле равно 0.

biSizeImage – размер изображения в байтах. Может содержать ноль для BI_RGB-изображений. Win98/Me/2000/XP: если biCompression содержит BI_JPEG или BI_PNG, biSizeImage указывает размер BI_JPEG или BI_PNG буфера изображения.

biXPelsPerMeter – горизонтальное разрешение в пикселях на метр для целевого устройства. Приложение может использовать это значение для выбора из группы ресурсов изображения, наиболее подходящего для текущего устройства. Для DPI 96, которое принято в Microsoft для мониторов, оно будет равно 3780 (если считать по формуле (96 / 25,4) * 1000).

biYPelsPerMeter – вертикальное разрешение в пикселях на метр для целевого устройства.

biClrUsed – количество используемых цветовых индексов в палитре. Если значение равно нулю – изображение использует максимально доступное количество индексов, в соответствии со значением biBitCount и методом сжатия, указанным в biCompression.

Если содержит ненулевое значение и biBitCount меньше 16, biClrUsed указывает количество цветов, к которым будет обращаться драйвер устройства или приложение. Если biBitCount больше или равен 16, biClrUsed размер палитры, используемой для оптимизации работы системных палитр. Если biBitCount равен 16 или 32, оптимальная палитра следует сразу после трёх четырёхбайтных масок.

В упакованном изображении массив пикселей следует сразу после структуры BITMAPINFO, biClrUsed должен содержать ноль, либо реальный размер палитры.

biClrImportant – количество элементов палитры, необходимых для отображения изображения. Если содержит ноль – все индексы одинаково важны.

Важно:

Изображение сохраняется построчно СНИЗУ-ВВЕРХ. Для хранения каждой строки выделяется кратное 4 количество байт. В незначащих байтах хранится мусор.

Старшему биту или тетраде соответствует самый левый пиксел. При хранении изображения TrueColor каждому пикселу соответствуют три последовательные байта, хранящие составляющие цвета B, G, R (не R, G, B).

6.3. Задание к лабораторной работе №6

1. Написать программу способную прочитать заголовок BMP файла и вывести его на экран.
2. Разложить 24 битное изображение, представленное в файле формата BMP на цветовые составляющие и сохранить их в виде отдельных файлов.
3. Разложить изображение на битовые срезы и сохранить их в виде отдельных файлов.

6.4. Задание к лабораторной работе №7

1. Провести анализ полученных изображений цветовых плоскостей и битовых срезов.
2. Сжать изображения при помощи стандартных алгоритмов RLE и LZ77.
3. Провести анализ полученных результатов.
4. Модифицировать алгоритм RLE с учетом особенностей хранения информации о цвете каждого из пикселей изображения.

6.5. Вопросы для самоконтроля

1) Как организована структура для хранения информации об изображении в BMP файле?

2) Какие типы BMP файлов Вы знаете?

3) Что такое палитра?

4) Как определить глубину цвета в изображении?

5) В каком формате хранится информация о цвете каждой из точек изображения?

7. ЛАБОРАТОРНАЯ РАБОТА №8 АНАЛИЗ И КОДИРОВАНИЕ АУДИО-ФАЙЛОВ ФОРМАТА WAV

7.1. Цель работы

Практическое знакомство с кодированием аудио данных и

изучение возможных способов сжатия данных.

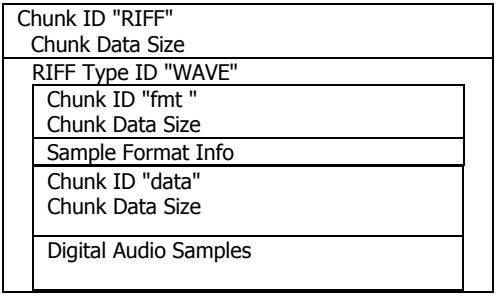
7.2. Теоретическая часть

Формат WAV изначально использовался в системе Windows для сохранения цифровых аудиоданных. Это самый известный и широко поддерживаемый формат благодаря популярности платформы Windows и большому количеству написанных для неё программ. Почти любая современная программа, работающая со звуком, может прочитать или записать формат WAV, поэтому этот формат очень интересен для разработчиков программного обеспечения. Далее подробно описываются структуры данных формата WAV.

Поскольку формат WAV-файла пришел от операционной системы Windows, в которой традиционно использовались процессоры Intel, все значения данных формата хранятся как Little-Endian, т. е. самый младший значащий байт идет первым.

WAV-файлы могут содержать строки текста, например метки секций, информационные комментарии и т. д. Строки сохраняются таким образом, что первый байт указывает количество байт текста ASCII в строке.

WAV-файл использует стандартную RIFF-структуру, которая группирует содержимое файла из отдельных секций (chunks) - формат выборок аудиоданных, аудиоданные, и т. п. Каждая секция имеет свой отдельный заголовок секции и отдельные данные секции. Заголовок секции указывает на тип секции и количество содержащихся в секции байт. Такой принцип организации позволяет программам анализировать только необходимые секции, пропуская остальные секции, которые не известны или которые не требуют обработки. Некоторые определенные секции могут иметь в своем составе подсекции (sub-chunks). Например, как можно увидеть на диаграмме, описывающий основной формат WAV-файла, секции "fmt " и "data" являются подсекциями секции "RIFF".



Если секция содержит нечетное число байт данных (невыравненное до 2 байт), то добавляется дополнительный нулевой байт данных в конец данных секции. Этот дополнительный байт не учитывается в размере секции заголовка, таким образом, программа всегда должна учитывать выравнивание для расчета смещения начала следующей секции.

Заголовки WAV-файла используют стандартный формат RIFF. Первые 8 байт файла - стандартный заголовок секции RIFF, который имеет ID секции "RIFF" и размер секции, равный размеру файла минус 8 байт, используемых для RIFF-заголовка. Первые 4 байта данных в секции "RIFF" определяют тип ресурса, который можно найти в секции. WAV-файлы всегда используют тип ресурса "WAVE". После типа ресурса (ID "WAVE") идут все секции звукового файла, которые определяют аудиосигнал.

Смещение	**Размер**	**Описание**	**Значение**
0x00	4	Chunk ID	"RIFF" (0x52494646)
0x04	4	Chunk Data Size	(file size) - 8
0x08	4	RIFF Type	"WAVE" (0x57415645)
0x10	Wave chunks (секции WAV-файла)		
Значения полей секции RIFF			

Существует довольно много типов секций, заданных для файлов WAV, но большинство WAV-файлов содержат только две из них - секцию формата (**"fmt "**) и секцию данных (**"data"**). Это именно те секции, которые необходимы для описания формата выборок аудиоданных, и для хранения самих аудиоданных. Хотя официальная спецификация не задает жесткий порядок следования секций, наилучшей практикой будет размещение секции формата перед секцией данных. Многие программы ожидают именно такой порядок секций, и он наиболее разумен для передачи аудиоданных через медленные, последовательные источники наподобие Интернет. Иначе если формат придет после данных, то перед стартом воспроизведения необходимо считать и запомнить все аудиоданные, только после получения формата запускать воспроизведение.

Все секции формата RIFF и соответственно секции Wave сохраняются в следующем формате:

Смещение	Размер	Описание
0x00	4	Chunk ID
0x04	4	Chunk Data Size
0x08	Chunk Data Bytes	
Формат секций RIFF и Wave		

Секция формата - "fmt "

Секция формата содержит информацию от том, как сохранены аудиоданные и как они должны воспроизводиться. Информация включает в себя тип используемой компрессии, количество каналов, скорость выдачи выборок (sample rate), количество бит в выборке (bits per sample) и другие атрибуты.

Смещение	Размер	Описание	Значение
0x00	4	Chunk ID	"fmt " (0x666D7420)
0x04	4	Chunk Data Size	16 + extra format bytes
0x08	2	Compression code	1 - 65,535
0x0a	2	Number of channels	1 - 65,535
0x0c	4	Sample rate	1 - 0xFFFFFFFF
0x10	4	Average bytes per second	1 - 0xFFFFFFFF
0x14	2	Block align	1 - 65,535
0x16	2	Significant bits per sample	2 - 65,535
0x18	2	Extra format bytes	0 - 65,535
0x1a	Дополнительные данные формата (Extra format bytes) *		

Идентификатор секции (Chunk ID) и объем данных (Data Size) - идентификатор секции всегда "fmt " (0x666D7420) и объем данных равен размеру стандартного формата WAV (16 байт) плюс размер всех дополнительных байт формата, необходимых для поддержки специфических форматов звука, если он не содержит несжатых данных PCM. Обратите внимание, что идентификатор секции "fmt " оканчивается на символ пробела (0x20).

Код сжатия (Compression Code) - первое слово данных формата указывает на тип сжатия, используемого для данных звука. В таблице содержится список кодов сжатия, используемых

в настоящее время.

Количество каналов (Number of Channels) - количество каналов указывает, сколько отдельных аудиосигналов закодировано в секции данных звука (wave data chunk). Значение 1 означает монофонический сигнал, 2 означает стерео, и т. п.

Скорость выборок (Sample Rate) - число выборок аудиосигнала, приходящихся на секунду. На эту величину не влияет количество каналов.

Среднее количество байт в секунду (Average Bytes Per Second) - величина, показывающая, сколько байт за секунду данных должно быть пропущено через цифроаналоговый преобразователь (D/A converter, DAC) во время воспроизведения файла. эта информация полезна, чтобы определить - могут ли данные поступать от источника с нужной скоростью, чтобы не отставать от воспроизведения. Эта величина просто вычисляется по формуле:

AvgBytesPerSec = SampleRate * BlockAlign

Выравнивание блока (Block Align) - количество байт на одну выборку. Эта величина может быть вычислена по формуле:

BlockAlign = SignificantBitsPerSample / 8 * NumChannels

Количество используемых бит на выборку (Significant Bits Per Sample) - величина указывает количество бит, формирующих каждую выборку сигнала. Обычно эта величина 8, 16, 24 или 32. Если число бит не выравнено по байту (не делится нацело на 8), количество используемых байт на выборку округляется вверх к наименьшему количеству байт. Неиспользуемые биты устанавливаются в 0 и игнорируются. Такие форматы (с числом бит на выборку, некратным 8) встречаются редко.

Дополнительные данные формата (Extra Format Bytes) - величина указывает, сколько далее идет дополнительных данных, описывающих формат. Она отсутствует, если код сжатия 1 (uncompressed PCM file), но может присутствовать и иметь любую другую величину для других типов сжатия, зависящую от количества необходимых для декодирования данных. Если величина не выравнена на слово (не делится нацело на 2), должен быть добавлен дополнительный байт в конец данных, но величина должна оставаться невыровненной.

Секция данных - "data"

Секция данных Wave (Wave Data Chunk) содержит данные цифровых выборок аудиосигнала, которые можно декодировать с

использованием формата и метода компрессии, указанных в секции формата Wave (Wave Format Chunk). Если код компрессии 1 (несжатый PCM, Pulse Code Modulation), то данные представлены в виде сырых, непреобразованных (raw) величин выборок.

WAV-файлы обычно содержат только одну секцию данных, но секций может быть несколько, если они содержатся в секции списка Wave (Wave List Chunk "wavl").

Смещение	**Длина**	**Тип**	**Описание**	**Значение**
0x00	4	char[4]	chunk ID	"data" (0x64617461)
0x04	4	dword	chunk size	зависит от количества выборок и компрессии
0x08	данные выборок (sample data)			

Аудиовыборки многоканального цифрового аудио сохраняются как чередуемые (interlaced) данные, которые просто означают последовательные аудиовыборки нескольких каналов (таких как стерео и каналы окружения surround). Выборки каналов сохранены последовательно друг за другом, перед тем как произойдет переход к следующему времени выборки. Это сделано с целью возможности последовательного проигрывания файла даже тогда, когда еще не весь файл прочитан целиком. Это удобно, когда проигрывается большой файл с диска (который не может быть размещен целиком в памяти) или файл передается в последовательном потоке данных через сетевое соединение (например Интернет). Значения в диаграмме ниже были бы сохранены в WAV-файле в порядке, как они перечислены в столбце значений (от начала до конца).

Когда выборки представлены 8 битами, они определены как значения без знака (unsigned). Все другие битовые размеры указываются как величины со знаком (signed). Например, выборка 16 бит может иметь значение в диапазоне от -32768 до +32767, где средняя точка (напряжение сигнала равно 0) соответствует значению 0.

Как уже было указано ранее, все секции RIFF (включая секции WAVE "data") должны быть выровнены по размеру на слово (2 байта). Если данные выборок содержатся в нечетном количестве байт, в конец данных выборок должен быть добавлен выравнивающий нулевой байт. За заголовке секции "data" размер не должен учитывать этот выравнивающий байт.

Пример: Чтение заголовка WAV файла.

```
#include <stdio.h>
#include <tchar.h>
#include <conio.h>
#include <math.h>

// Структура, описывающая заголовок WAV файла.
struct WAVHEADER
{
    // WAV-формат начинается с RIFF-заголовка:

    // Содержит символы "RIFF" в ASCII кодировке
    // (0x52494646 в big-endian представлении)
    char chunkId[4];

    // 36 + subchunk2Size, или более точно:
    // 4 + (8 + subchunk1Size) + (8 + subchunk2Size)
    // Это оставшийся размер цепочки, начиная с этой позиции.
    // Иначе говоря, это размер файла - 8, то есть,
    // исключены поля chunkId и chunkSize.
    unsigned long chunkSize;

    // Содержит символы "WAVE"
    // (0x57415645 в big-endian представлении)
    char format[4];

    // Формат "WAVE" состоит из двух подцепочек: "fmt " и "data":
    // Подцепочка "fmt " описывает формат звуковых данных:

    // Содержит символы "fmt "
    // (0x666d7420 в big-endian представлении)
    char subchunk1Id[4];

    // 16 для формата PCM.
    // Это оставшийся размер подцепочки, начиная с этой позиции.
    unsigned long subchunk1Size;

    // Аудио формат, полный список можно получить здесь http://audiocoding.ru/wav_formats.txt
    // Для PCM = 1 (то есть, Линейное квантование).
    // Значения, отличающиеся от 1, обозначают некоторый формат сжатия.
    unsigned short audioFormat;

    // Количество каналов. Моно = 1, Стерео = 2 и т.д.
    unsigned short numChannels;

    // Частота дискретизации. 8000 Гц, 44100 Гц и т.д.
```

```
        unsigned long sampleRate;

        // sampleRate * numChannels * bitsPerSample/8
        unsigned long byteRate;

        // numChannels * bitsPerSample/8
        // Количество байт для одного сэмпла, включая все каналы.
        unsigned short blockAlign;
        // Так называемая "глубиная" или точность звучания. 8 бит, 16 бит и
т.д.
        unsigned short bitsPerSample;

        // Подцепочка "data" содержит аудио-данные и их размер.

        // Содержит символы "data"
        // (0x64617461 в big-endian представлении)
        char subchunk2Id[4];

        // numSamples * numChannels * bitsPerSample/8
        // Количество байт в области данных.
        unsigned long subchunk2Size;

        // Далее следуют непосредственно Wav данные.
    };

    int _tmain(int argc, _TCHAR* argv[])
    {
        FILE *file;
        errno_t err;
        err = fopen_s(&file, "Slipknot - Three Nil.wav", "rb");
        if (err)
        {
            printf_s("Failed open file, error %d", err);
            return 0;
        }

        WAVHEADER header;

        fread_s(&header, sizeof(WAVHEADER), sizeof(WAVHEADER), 1, file);

        // Выводим полученные данные
        printf_s("Sample rate: %d\n", header.sampleRate);
        printf_s("Channels: %d\n", header.numChannels);
        printf_s("Bits per sample: %d\n", header.bitsPerSample);

        // Посчитаем длительность воспроизведения в секундах
        float fDurationSeconds = 1.f * header.subchunk2Size / (head-
er.bitsPerSample / 8) / header.numChannels / header.sampleRate;
```

```
    int iDurationMinutes = (int)floor(fDurationSeconds) / 60;
    fDurationSeconds = fDurationSeconds - (iDurationMinutes * 60);
    printf_s("Duration: %02d:%02.f\n", iDurationMinutes,
fDurationSeconds);
    fclose(file);

    _getch();
    return 0;
}
```

7.3. Задание к лабораторной работе

1. Написать программу, которая выводит на экран основную информацию о WAV файле, количестве и размере секций.
2. Сжать WAV файл алгоритмами RLE и LZ77. Оценить размер сжатого файл и степень сжатия.
3. Предложить способ сжатия информации в WAV файле с потерями, который мог бы позволить сжать информацию с большей степень сжатия, чем была получена при выполнении второй задачи.

7.4. Вопросы для самоконтроля

1) К какому типу файлов относятся файлы формата WAV?

2) Что такое RIFF заголовок?

3) Из каких основных секций состоит WAV файл?

4) Как в файлах формата WAV хранятся данные об уровне аудио сигнала?

5) Сколько бит может быть отведено для хранения информации об одной выборке для одного канала?

6) Что такое средняя точка?