МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»

Электронные методические указания к выполнению лабораторных работ по дисциплине «Организация ЭВМ и систем»

Санкт-Петербург
Издательство СПБГЭТУ «ЛЭТИ»
2013

УДК 004.424:004.422.63(075.8)

Электронные методические указания к выполнению лабораторных работ по дисциплине «Организация ЭВМ и систем»/ Сост.: Андреева А.А., Грушвицкий Р.И., Кочетков А.В., Манирагена В., Мурсаев А.Х., Павлов С.М., Чугунов Л.А.. — СПб.:Изд-во СПбГЭТУ «ЛЭТИ», 2013. –81с.: ил.

Описывается цикл лабораторных работ.

Предназначено для студентов, обучающихся по направлению 09.03.01 «Информатика и вычислительная техника» по профилю «Организация и программирование интеллектуальных систем»

Утверждено
редакционно-издательским советом университета
в качестве методических указаний

Оглавление

Введение

Методические указания к лабораторным работам по указанным дисциплинам предназначены для предварительного ознакомления студентами с составом аппаратных и программных средств, предназначенных для ввода, обработки и вывода различной информации на доступные устройства компьютера. Указания содержат в необходимых случаях задания, рекомендации по выполнению лабораторных работ и содержанию отчётов.

Лабораторная работа №1.

ИССЛЕДОВАНИЕ ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ РАЗЛИЧНЫХ ФОРМАТОВ ДАННЫХ

Цель работы: знакомство с внутренним представлением различных типов данных, используемых компьютером при их обработке.

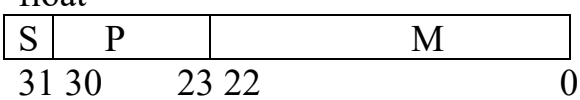
1.1. Общие положения

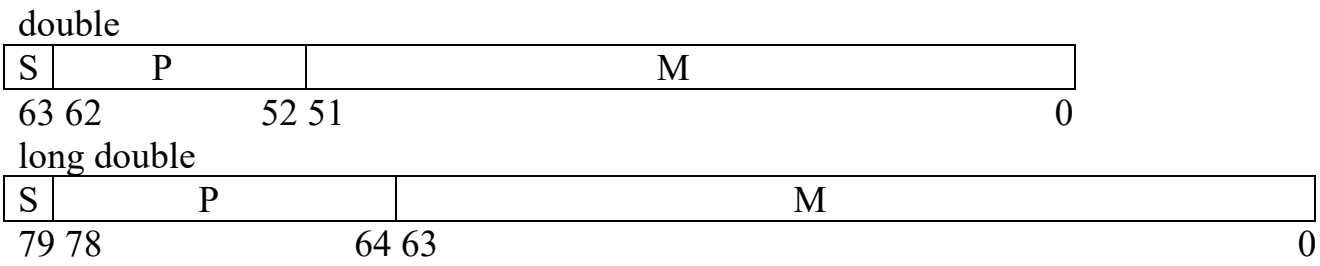
При программировании на языке C++ используются 11 стандартных типов данных. Среди них можно выделить 3 группы:

1. данные символьные и целого типа беззнаковые (с фиксированной запятой);
2. данные символьные и целого типа со знаком (с фиксированной запятой), значения которых хранятся в двоичном дополнительном коде;
3. данные вещественного типа (с плавающей запятой (точкой)).

Числовое значение данных первой группы занимает всю разрядную сетку (количество двоичных разрядов от 8 до 32), отведенных под конкретный тип. Знак числа данных второй группы занимает старший (левый) разряд, а остальную часть разрядной сетки (от 7 до 31 двоичных разрядов) занимает числовое значение данных, отведенных под конкретный тип. Формат хранения данных третьей группы описывается IEEE 754 - стандартом в виде значения мантиссы (M) со знаком (S) и значения порядка (P). Число бит для хранения мантиссы и порядка зависит от типа данных с плавающей запятой.

float





Вещественное число в памяти хранится с нормализованной мантиссой, значение которой в десятичном эквиваленте лежит в диапазоне от 1 до 2. Причём 2 не входит в границу диапазона. Если в процессе выполнения какой-либо операции над данными с плавающей запятой значение мантиссы выходит из указанного диапазона, то в конце операции выполняется нормализация результата путем приведения значения мантиссы к указанному диапазону с соответствующим изменением значения порядка. При этом значение старшего бита мантиссы должно оказаться равным единицы. Если значение порядка превышает допустимое значение, то вырабатывается признак переполнения разрядной сетки. Если значение мантиссы равно нулю или в процессе выполнения операции значение порядка становится меньше допустимой величины, то в результате выполнения операции сформируется так называемый «машинный ноль», то есть код, у которого значение всех бит равно нулю. Но если мантисса всегда нормализована, то старший её бит, то есть единицу, можно и не хранить в памяти. Стандартом предложено этот бит не хранить в памяти и тем самым увеличить точность представления вещественных чисел в 2 раза. Эта единица присутствует неявно, то есть скрыта от глаз наблюдателя и называется неявной единицей (implicit one). Отбрасывание старшей цифры мантиссы выполняется для форматов float и double, но не выполняется для long double.

Порядок числа в соответствии с указанным форматом хранится «сдвинутым», то есть к его действительному значению добавляется в зависимости от формата такое число, чтобы порядок P был всегда неотрицательным. Для формата float прибавляется 127, для чисел формата double прибавляется 1023, а для формата long double добавляется 16383. Всегда неотрицательный порядок упрощает выполнение операции сравнения порядков и арифметических операций над ними, а также избавляет от необходимости выделять один бит для хранения знака порядка.

Например, число 15.375 (1111.011 в двоичной системе счисления) в формате **float** IEEE-стандарта получается следующим образом:

1.921875 (1.111011 в двоичной системе счисления) - это значение нормализованной мантиссы;

3 (11- в двоичной системе счисления) это степень несмещённого двоичного порядка (15.375=1.921875*8);

0-это знак числа.

Учитывая отбрасывание неявной единицы и сдвиг порядка, получаем внутреннее представление числа:

S=0;

P=3+127=130 (10000010 в двоичной системе счисления);

M=11101100000000000000000.

Таким образом, внутреннее представление числа 15.375 в формате float будет:

0	10000010	11101100000000000000000

31 30 … 23 22 … 0

Это же число 15.375 в формате **double** записывается так:

S=0;

P=3+1023=1026 (10000000010 в двоичной системе счисления);

M=1110110000000000000000000000000000000000000000000000.

0	10000000010	11101100000000000...00000000000000000000000

63 62 … 52 51 … 0

Это же число 15.375 в формате **long double** записывается так:

S=0;

P=3+16383=16386 (100000000000010 в двоичной системе счисления);

M=11110110000000000000000…0 (единица не отбрасывается).

0	100000000000010	11110110000000000…00000000000000000000000000000000

79 78 … 64 63 … 0

Приведём ещё несколько примеров, но уже без подробных комментариев.

Для более компактной формы записи используем шестнадцатеричную систему счисления:

-16.5 float: C1 84 00 00 h
double: C0 30 80 00 00 00 00 00 h
long double: C0 03 84 00 00 00 00 00 00 00 h

- 0.0625 float: BD 80 00 00 h
double: BF B0 00 00 00 00 00 00 h
long double: BF FE A0 00 00 00 00 00 00 00 h

1.2. Предварительная подготовка к работе

1. Ознакомиться с внутренними форматами представления данных.
2. Вспомнить поразрядные логические операции в языке C++, указатели и операции над ними, структуры типа union и функции работы с файлами.

Для вывода данных символьного и целого типов в двоичном коде достаточно использовать поразрядные логические операции, операции сдвига, условный оператор и оператор цикла.

Для вывода данных вещественного типа потребуется применение более сложных действий. И если для вывода данных типа float достаточно применить указатель, то для других вещественных типов потребуется использование структуры типа union или функций работы с файлом.

1.3. Порядок выполнения работы

1. В зависимости от номера варианта задания разработать алгоритм ввода с клавиатуры требуемых типов данных и показать на экране их внутреннее представление в двоичной системе счисления.

2. Написать и отладить программу на языке C++, реализующую разработанный алгоритм. Программа должна

- иметь дружественный интерфейс
- выводить на экран информативное сообщение при вводе некорректных данных
- предложить повторный ввод пока не будут введены корректные данные

3. В соответствии с заданием дополнить разработанный ранее алгоритм блоками для выполнения преобразования двоичного полученного кода исходного типа данных и последующего вывода преобразованного кода в двоичной системе счисления и в формате исходного данного.

Варианты заданий приведены в таблице 1.1.

Табл.1.1

№ варианта	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
unsigned char	*		*									*							
Char		*									*								
unsigned int				*									*		*			*	
int					*					*									
short int						*			*					*					*
unsigned long							*									*			
long								*									*		
float			*				*		*				*				*		
double		*		*		*		*		*		*		*		*		*	
long double	*				*						*				*				*
Вид преобраз.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Виды преобразований:

1. Установить в заданное пользователем состояние определённое количество бит, номера которых, как и всё остальное, вводится с клавиатуры.

2. Инвертировать значения определённого количества бит, номера которых, как и их количество, вводится с клавиатуры.

3. Установить в заданное пользователем состояние определённое количество рядом стоящих бит, номер старшего бита, как и всё остальное, вводится с клавиатуры.

4. Установить в заданное пользователем состояние определённое количество рядом стоящих бит, номер младшего из которых, как и всё остальное, вводится с клавиатуры.

5. Поменять местами заданные пользователем группы рядом стоящих бит, номера старших разрядов этих групп и количество бит в группе, вводится с клавиатуры.

6. Выполнить циклический сдвиг в заданную сторону на некоторое вводимое с клавиатуры количество разрядов.

7. Поменять местами значения рядом стоящих бит в парах. Количество пар и номер старшего разряда левой пары задаётся с клавиатуры.

8. Выполнить циклический сдвиг в заданную пользователем сторону на заданное количество разрядов в пределах определённой группы разрядов, количество которых и номер старшего разряда в группе задаются с клавиатуры.

9. Инвертировать значения всех бит кроме тех, количество и номера которых задаются с клавиатуры.

10. Установить в заданные пользователем значения некоторые разряды, количество которых и номера разрядов задаются с клавиатуры.

11. Выполнить зеркальную перестановку в группе рядом стоящих разрядов, количество которых и номер старшего разряда в группе задаются с клавиатуры.

12. Выполнить в пределах группы бит путём сдвига вправо все биты, значение которых равно единице и влево все биты, значение которых равно нулю.

Количество бит и номер старшего разряда в группе задаются с клавиатуры.

13. Выполнить зеркальную перестановку в группе рядом стоящих разрядов, количество которых и номер младшего разряда в группе задаются с клавиатуры.

14. Инвертировать значения рядом стоящих бит, количество которых и номер старшего разряда задаются с клавиатуры.

15. Выполнить циклический сдвиг в заданную пользователем сторону на некоторое количество разрядов в пределах определённой группы разрядов, количество которых и номер младшего разряда в группе задаются с клавиатуры.

16. Выполнить путём сдвига вправо все биты, значение которых равно нулю и влево все биты, значение которых равно единице.

17. Поменять местами заданные пользователем группы рядом стоящих бит, номера младших разрядов этих групп и количество бит в группе, вводится с клавиатуры.

18. Поменять местами значения рядом стоящих бит в парах. Количество пар и номер младшего разряда правой пары задаётся с клавиатуры.

19. Поменять местами значения бит в заданном количестве пар бит. Номера бит в парах задаются с клавиатуры.

1.4. Содержание отчёта

Отчет по лабораторной работе должен содержать:

- титульный лист;
- задание на лабораторную работу;
- блок-схему алгоритма с пояснениями;
- текст программы;
- тестовые примеры запуска программы;
- указание программной среде компилировалась и выполнялась программа;

1.5. Контрольные вопросы

1. В каком коде хранятся целые числа со знаком?
2. Чем отличаются процессы сдвига влево и вправо для чисел со знаком и беззнаковых?
3. Как представляется корректный двоичный код числа типа float (double, long double), имеющего в десятичном виде наименьшее положительное значение, отличное от нуля?
4. Как выглядит десятичный код числа типа float (double, long double), имеющего наименьшее положительное значение, отличное от нуля?
5. В каком порядке следует выполнять действия для получения дополнительного кода двоичного целого числа из прямого кода?
6. Чем отличается циклический сдвиг двоичного кода от логического сдвига?
7. Чем отличается логический сдвиг двоичного кода от арифметического сдвига?
8. Как изменяется значение числа при арифметическом сдвиге на 1 двоичный разряд влево?
9. Как изменяется значение числа при арифметическом сдвиге на 1 двоичный разряд вправо?
10. В каком порядке следует выполнять действия для получения прямого кода двоичного целого числа из дополнительного кода?

Лабораторная работа №2.

ИССЛЕДОВАНИЕ ВИДЕОСИСТЕМЫ (ТЕКСТОВЫЙ РЕЖИМ)

Цель работы: изучение работы с видеосистемой в текстовом режиме, освоение приемов использования цветовой палитры: изменение цвета символов и фона на всем экране и в отдельном окне.

2.1. Общие положения

Аппаратные средства для вывода информации на экран включают специальную электронную плату (видеоадаптер, либо адаптер дисплея, либо просто адаптер) и монитор (или просто экран). Конструктивно видеоадаптеры - это весьма сложные устройства, управляемые собственным микропроцессором, сравнимым по мощности с центральным процессором компьютера. Несмотря на огромное разнообразие фирм-производителей видеоадаптеров, имеется несколько стандартов, которым все эти продукты удовлетворяют.

В самом общем виде видеоадаптер состоит из двух основных частей: контроллера и видеопамяти (видеобуфера). Помимо этих обязательных узлов, наиболее совершенные видеоадаптеры имеют в своем составе ряд дополнительных узлов, например специализированные контроллеры быстрой манипуляции содержимым видеобуфера (так называемые контроллеры графики). Основное назначение видеобуфера - хранение образа информации экрана. Видеоадаптер 25 и более раз в секунду формирует изображение на экране. Так как человеческий глаз не способен уловить такое быстрое мелькание кадров, создается иллюзия неподвижного изображения на экране монитора. Изображение на экране строится из небольших точек - так называемых пикселов (pixel - Picture ELement). Число пикселов в строке и число самих строк различно для разных типов видеоадаптеров.

Память, необходимая для хранения полного образа экрана, называется видеостраницей. Часто общий объем видеопамяти намного превышает объем страницы. В этом случае появляется возможность хранить в видеобуфере не одну, а несколько страниц. Та видеостраница, которая постоянно "освежается" в данный момент, называется текущей. Видеоадаптер способен выполнять переключение текущей видеостраницы. Объем видеопамяти и число возможных страниц, зависит от конкретного адаптера.

Управление параметрами видеосистемы может выполняться на двух уровнях:

1. на уровне портов видеоадаптера;
2. обращением к функциям BIOS.

В данной части описывается управление лишь некоторыми из параметров видеосистемы: определение типа и характеристик видеоадаптера и монитора, зада-

ние формы курсора и его позиции на экране, выбор режима, видеостраницы и палитры.

2.2. Видеорежимы и их краткая характеристика

Интегральной характеристикой особенностей работы адаптера является совокупность поддерживаемых им режимов. Поведение адаптера в том или ином режиме является фактическим стандартом и полностью характеризует все особенности адаптера, доступные для программиста средства управления адаптером и т.п. Режимы принято нумеровать, начиная с нуля. Чем совершеннее видеоадаптер, тем больше режимов он поддерживает. Как правило, более совершенные адаптеры полностью совместимы со своими предшественниками и с точки зрения прикладной программы отображает информацию точно так же, как и его предшественник. Некоторые режимы работы видеоадаптеров описаны в табл. 2.1.

Табл. 2.1. Режимы работы видеоадаптеров

Режим	Тип	Размер шрифта	Максимальное число страниц	Разрешение		Адрес видеобуфера	Тип Видеоадаптера
				графика	Текст		
0,1	Текст	8x8	8	_	40x25	B8000h	CGA, EGA, VGA,MCGA
	Текст	8x14	8		40x25	B8000h	EGA,VGA
	Текст	8x16	8	-	40x25	B8000h	MCGA
	Текст	9x16	8	_	40x25	B8000h	VGA
2,3	Текст	8x8	4	-	80x25	B8000h	CGA
	Текст	8x8	8	-	80x25	B8000h	EGA,VGA
	Текст	8x8	8		80x43	B8000h	EGA
	Текст	8x8	8		80x50	B8000h	VGA
	Текст	8x14	8	-	80x25	B8000h	EGA,VGA
	Текст	8x16	8		80x25	B8000h	MCGA
	Текст	9x16	8	-	80x25	B8000h	VGA
4,5	Граф.	8x8	1	320x200	40x25	B8000h	CGA, EGA, VGA, AT&T MCGA
6	Граф.	8x8	1	640x200	80x25	B8000h	EGA,VGA
7	Текст	8x14	4		80x25	B0000h	EGA,VGA
Dh	Граф.	8x8	8	320x200	40x25	A0000h	EGA,VGA
Eh	Граф.	8x8	4	640x200	80x25	A0000h	EGA,VGA
Fh	Граф.	8x14	2	640x350	80x25	A0000h	EGA,VGA
10h	Граф.	8x14	2	640x350	80x25	A0000h	EGA,VGA
llh	Граф.	8x16	1	640x480	80x30	A0000h	MCGA, VGA
12h	Граф.	8x16	1	640x480	80x30	A0000h	VGA

При всем многообразии режимов работы видеоадаптеров их можно объединить в две группы: текстовые и графические. Переключение из текстового режима в графический и наоборот означает полное изменение логики работы видеоадаптера с видеобуфером.

Если видеоадаптер включен в текстовый режим, он рассматривает экран как совокупность так называемых текселов (texel - Text Element) (рис. 2.1).

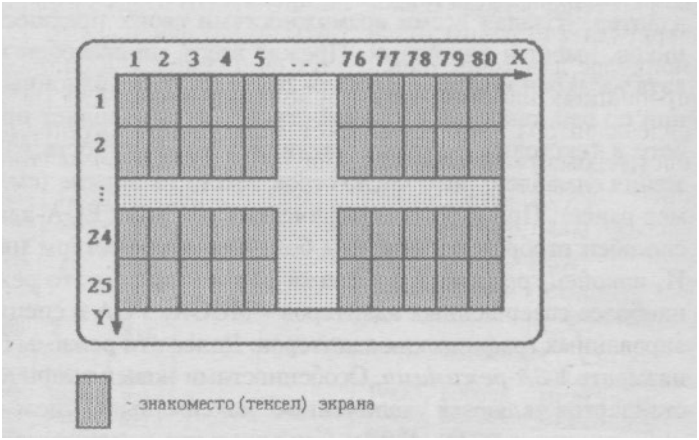


Рис. 2.1. Представление экрана в текстовых режимах "25 строк х 80 столбцов"

Каждому знакоместу экрана (текселу) в текстовом режиме соответствуют два байта памяти видеобуфера. Байт по четному адресу хранит ASCII-код символа, а следующий за ним байт по нечетному адресу кодирует особенности отображения символа на экране: цвет пикселов, из которых формируется очертание символа (Foreground Color), цвет всех остальных пикселов знакоместа или цвет фона символа (Background Color), мерцание символа и необходимость повышения яркости символа при отображении. Этот байт называется байтом атрибута. Закрепление битов байта атрибута приведено на рис. 2.2.

Рис. 2.2. Назначение битов байта атрибута

R, G, B -соответственно красный (Red), зеленый (Green), синий (Blue) цвета (1 - цвет включен; 0 - цвет выключен)

Задавая различные числовые значения байту атрибута в видеобуфере, можно управлять цветом символов и цветом фона, на котором эти символы отображаются. Например, если значение байта атрибута равно 112, то выводится немерцающий символ черного цвета на сером фоне. Действительно, биты RGB цвета символа для данного кода атрибута равны нулю. Биты цвета фона равны 1, и на мониторе для точек фона будут смешиваться в необходимых пропорциях красный, синий и зеленый цвета. Для цветного видеоадаптера - это серый цвет. Повышение интенсивности цвета символа выполняется путем установки бита с номером 3 в 1. Светло-серый цвет - это белый цвет, поэтому на экране цветного монитора при работе видеоадаптера в текстовом режиме могут быть белые буквы, но не может быть белый фон. Например, символы, код атрибута которых в видеопамяти равен 15, будут отображаться белыми пикселами на черном фоне. В принципе, если задать цвета фона и символа одинаковыми, символы будут невидимыми, например красный символ на красном фоне (атрибут 0x44), что можно использовать в адаптерах, у которых мерцание символа с помощью бита 7 не реализовано.

Видеоадаптеры типов EGA и VGA имеют некоторые особенности использования бита интенсивности, которые будут рассмотрены несколько позже.

Видеопамять адаптера при работе в текстовых режимах доступна непосредственно из программы. Это значит, что любая ячейка видеобуфера может быть прочитана программой так же, как и обычная ячейка оперативной памяти. И как в обычную ячейку памяти, в видеобуфер возможна запись значений из программы. Адреса ячеек видеопамяти начинаются для разных типов адаптеров с разных границ, приведенных в табл. 2.1. Если адаптер работает в текстовых режимах "40 столбцов x 25 строк", то для хранения полного образа экрана (видеостраницы) требуется 25 x 40 x 2 = 2000 байт видеопамяти. В режимах "80 столбцов x 25 строк" видеостраница занимает уже 25 x 80 x 2 = 4000 байт. Минимальная конфигурация видеоадаптера CGA имеет обычно 16К байт видеопамяти, что позволяет хранить 8 страниц текста в режимах 0 или 1 и 4 страницы в режимах 2 или 3.

Вывод на монитор содержимого видеобуфера происходит, начиная с некоторого начального адреса, называемого смещением до видеостраницы. Страница 0 имеет нулевое смещение. Страница 1 в режиме "80 строк x 25 столбцов" начинается с адреса, смещенного на 4096 байт (l000h) относительно начального адреса видеопамяти, страница 2 - со смещения 8192 байт (2000h) и т.д. Если изменить значение смещения, произойдет переключение страницы, т.е. на экране возникнет образ другой страницы видеопамяти. Иногда переключение видеостраниц в текстовом режиме используется для реализации динамических изображений.

Видеоадаптер при работе в текстовом режиме периодически считывает со-

держимое ячеек видеобуфера и по коду символа и байту атрибута формирует пикселы, образующие в совокупности очертание символа и его фон. При этом байт символа служит индексом для входа в специальную таблицу - так называемую таблицу знакогенератора. Она содержит информацию, по которой видеоадаптер формирует пикселы для изображения того или иного символа. Число строк и столбцов в одной ячейке таблицы различно для различных типов видеоадаптеров. Чем больше строк и столбцов использовано для символа, тем более качественно он изображается на экране.

Число знакомест в одной текстовой строке зависит от видеоадаптера и от режима его работы.

Переключение адаптера в один из графических режимов полностью изменяет логику работы аппаратуры видеосистемы. При работе в графическом режиме появляется возможность управлять цветом любой телевизионной точки экрана или пиксела. Число строк пикселов и число пикселов в каждой строке зависит от режима работы видеоадаптера. Таким образом, экран в графическом режиме представляет собой матрицу пикселов (рис. 2.3).

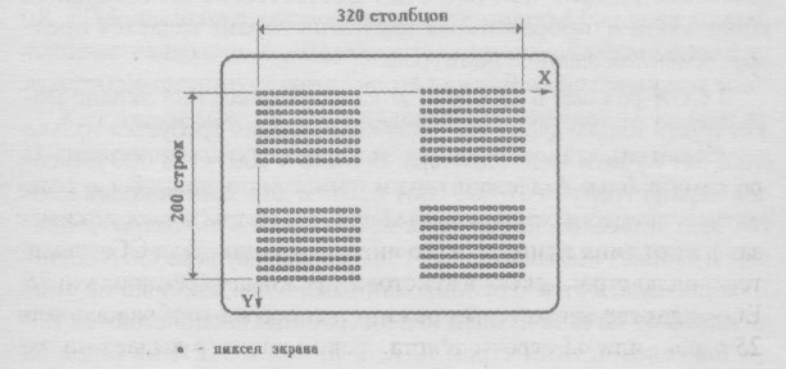


Рис. 2.3. Представление экрана в графических режимах 4,5 (320 столбцов х 200 строк)

2.3. Функции консольного ввода-вывода

Функции консольного ввода-вывода C++ помещены в файле <conio.h>, предназначены для облегчения работы по созданию простейшего оконного интерфейса. Эти функции используют понятие активного окна экрана. Активное окно - это прямоугольная область экрана, в границах которой в данный момент работают функции. Описание активного окна (или, как часто говорят, фрейм) хранится во внутренней структурной переменной C++. Установку параметров активного текстового окна выполняет функция

```
window(int , int ,int , int );.
```

Она описывает активное текстовое окно: первая пара аргументов задает соот-

ветственно номера столбца и строки левого верхнего угла, вторая пара - правого нижнего угла. Строки и столбцы нумеруются, начиная от 1. Поэтому, например, координаты левого верхнего и правого нижнего углов экрана в режимах "25 строк x 80 столбцов" задаются парами (1,1) и (80,25). Расположение осей X и Y на экране показано на рис. 2.1.

Фрейм окна C++ имеет следующую структуру:

```
struct text_info
{unsigned char
winright,  winbottom;   /* столбец, строка правого нижнего угла */
attribute, normattr;    /* атрибуты окна*/
currmode;                /* текущий режим работы видеоадаптера   */
screenheight;            /* полная высота экрана   */
screenwidth;             /* полная ширина экрана   */
curx, cury; };           /* строка, столбец текущей позиции курсора        */
```

Информация об активном окне доступна при выполнении функции

```
gettextinfo( struct text_info *t);
```

При вызове эта функция заполняет поля структурной переменной, описанной по шаблону text_info, указатель t на которую она получает.

Функция window() инициализирует поля координат фрейма окна. Функции textcolor(), textbackground(), textattr() и другие управляют цветом отображаемых символов в окне. Они будут рассмотрены далее.

2.4. Управление курсором

Видеоадаптеры всех типов аппаратно поддерживают курсор, который в текстовых режимах отображается на экране в виде одной или нескольких линий в пределах тексела. Курсор указывает на текущую позицию экрана (строку и столбец тексела), в которую будет записываться или из которой будет читаться средствами BIOS символ. При переключении адаптера в графический режим курсор становится невидимым, но BIOS сохраняет возможность изменять его позицию. Специальные регистры видеоконтроллера хранят текущую позицию и форму курсора.

Программное прерывание 10h BIOS имеет в своем составе специальные функции для установки формы курсора, чтения и установки его координат.

Функция AH = 0lh задает высоту курсора. Регистр CH определяет номер верхней телевизионной линии, а CL - номер нижней линии при изображении курсора. Например, задав значения CH = 0, CL = 13, получим курсор, занимающий все знакоместо. Значения CH = 6, CL = 7 устанавливают форму курсора по умолчанию -

две равномерно мерцающие строки в нижней части тексела. Если CH > CL, курсор будет состоять из двух групп линий вверху и внизу знакоместа с разрывом посередине. BIOS поддерживает одинаковую форму курсора для всех видеостраниц текста. Биты 5 и 6 кода CH управляют мерцанием и могут выключить отображение курсора (табл. 2.2).

Табл. 2.2. Биты управления отображением курсора

Биты регистра CH		Действие, оказываемое на курсор
бит 6	бит 5	
0	0	Видимый, мерцающий равномерно с нормальной скоростью кур-
0	1	Курсор не отображается
1	0	Видимый, мерцающий равномерно с повышенной скоростью кур-
1	1	Видимый, неравномерно мерцающий курсор

BIOS записывает текущую форму курсора в слово по адресу 0040:0060h, при этом младший байт содержит номер нижней строки, а старший - номер верхней строки для отображения курсора.

Управление формой курсора находит ограниченное применение в практике программирования. Однако изменением формы курсора можно отображать различные режимы работы программы, например, режим "Вставка" или режим "Замена". Намного чаще приходится выключать курсор. Это необходимо, когда программа сама отображает курсор.

Среди функций консольного ввода-вывода C++ текущей позицией курсора в окне управляет функция gotoxy(int x, int y);.

Устанавливает курсор в заданную строку y и столбец x в текущем активном окне экрана. Верхний левый угол окна имеет координаты (1,1). При попытке позиционировать курсор за границы окна он останавливается на границе окна. Особенностью функции является то, что координаты задаются относительно левого верхнего угла текущего окна.

Текущую позицию x и y курсора в активном текстовом окне можно узнать при вызове соответственно функций wherex () и wherey().

Эти функции соответственно возвращают номер столбца и номер строки текущей позиции курсора. Кроме того, текущая позиция курсора в окне возвращается в полях curx и cury структурной переменной, заполняемой при вызове функции gettextinfo( ).

2.5. Работа с текстовой информацией

Вывод информации на экран персонального компьютера может выполняться на трех уровнях:

1. на уровне MS-DOS с использованием функций прерывания 21h
2. на уровне BIOS с использованием функций прерывания 10h
3. непосредственным доступом к аппаратным средствам.

Вывод информации на уровне MS-DOS - мобильный, но самый медленный. Функции MS-DOS для вывода информации на экран вызывают драйвер консоли (выполняют вывод в специальный символьный файл CON). Если в системе инсталлирован специальный драйвер (например, ANSI.SYS), могут использоваться дополнительные средства по управлению экраном. Суть расширенного управления состоит в передаче драйверу консоли специальных управляющих строк. Драйвер опознает начало управляющей строки по символу ASCII с кодом 27 (1Bh). Передаваемые на экран вслед за ним символы рассматриваются как параметры команды, которую выполняет драйвер, например, перемещает курсор, устанавливает цвет символа и т.п. Сами управляющие символы не отображаются на экране. Таким образом, использование функций MS-DOS позволяет осуществить вывод через драйвер. Другие достоинства функций MS-DOS - автоматическое позиционирование курсора и скроллинг экрана, реакция на нажатие комбинации клавиш Ctrl-Break. Недостатком является невозможность непосредственного управления курсором и атрибутом символов. На уровне MS-DOS работают функции стандартного вывода, а их прототипы содержатся в файле <stdio.h>.

Вывод на уровне BIOS дает более широкие возможности по управлению экраном. Именно эти функции используются драйверами MS-DOS для вывода информации на экран. Недостатком функций BIOS является невысокая скорость вывода, что особенно заметно при работе в графических режимах. На уровне BIOS работают функции консольного вывода, а их прототипы помещены в файле <conio.h>.

Для приложений, критичных по скорости вывода, приходится выполнять вывод, используя непосредственный доступ к портам и видеопамяти адаптера. Такой способ позволяет достичь максимально возможной скорости вывода, но требует максимальных затрат труда программиста. Функции консольного вывода Turbo C могут по выбору пользователя работать и на самом нижнем уровне, выполняя доступ к видеобуферу при работе в текстовом режиме.

2.6. Скроллинг. Очистка окна и всего экрана

Функции AH = 06 и 07 прерывания 10h BIOS осуществляют так называемый скроллинг (прокрутку) окна экрана. При выполнении скроллинга на одну

строку вверх вся информация в окне перемещается на строку вверх. Внизу окна появляется чистая строка. При выполнении скроллинга на одну строку вниз содержимое окна сдвигается на строку вниз и вверху окна добавляется чистая строка. Значение регистра AL задает число строк, на которое выполняется скроллинг. Если AL=0, выполняется очистка окна. Значения в CH и CL определяют строку и столбец левого верхнего угла окна, в DH и DL -строку и столбец правого нижнего угла. Строки и столбцы нумеруются от 0. Значение в регистре BH задает атрибут добавляемой чистой строки.

Приведем пример Си-функции, выполняющей вертикальный скроллинг окна экрана, заданного строкой и столбцом левого верхнего (l_row, l_col) и строкой и столбцом правого нижнего (r_row, r_col) углов окна. Если переменная direction равна UP, происходит скроллинг на одну строку вверх, если она равна DOWN - скроллинг на одну строку вниз, если ENTIRE -выполняется очистка окна. Добавляется строка с атрибутом attr.

```
#include <CONIO.H>
#include < dos. h >
enum { ENTIRE, UP, DOWN };
void scroll (int direction, char l_row, char l_col, char r_row, char r_col, char attr)
    {
        union REGS r;
         if (direction == DOWN) r.h.ah = 7;
        else r.h.ah = 6;
        r.h.al = lines; // шаг скроллинга
        r.h.ch =  l_row; r.h.cl= l_col; r.h.dh = r_row; r.h.dl= r_col;
        r.h.bh=attr;
         int86(0x10,&r,&r);
    }
```

Если окно занимает весь экран и задается direction = ENTIRE, происходит фактическая очистка всего экрана и его "заливка" цветом, задаваемым атрибутом чистой строки attr. Например, для очистки экрана в режимах с 25 строками и 80 столбцами вызов функции может иметь вид:

```
scroll(ENTIRE,0,0,24, 79,0x07);
```

Для получения цветной рамки по периметру всего экрана можно выполнить два обращения:

```
scroll (ENTIRE, 0,0, 24, 79, color);
scroll(ENTIRE, 1, 1, 23, 78, 0x07);
```

Здесь значение color - атрибут. Цвет рамки будет совпадать с цветом фона символа, так как символом является пробел.

Скроллинг окна средствами BIOS возможен как в текстовых, так и в графиче-

ских режимах работы видеоадаптера. Скроллинг в графических режимах разных адаптеров может иметь некоторые особенности.

Одним из интересных применений скроллинга является построение "взрывающихся" окон (exploding windows). Такие окна "вырастают" на экране из определенного места (середины, одного из углов, сначала заполняется средняя горизонталь, а затем окно раздвигается вверх и т.п.). Секрет построения "взрывающихся" окон прост. Начиная с определенного места окна (например, его центра), выполняется очистка первого, самого маленького окна. После этого по периметру этого окна рисуется рамка. Затем координаты левого верхнего и правого нижнего углов модифицируются, и все повторяется: очистка старой рамки скроллингом, вывод новой, большего размера, модификация координат углов и так далее до тех пор, пока окно не "вырастет" до нормальных размеров.

2.7. Вывод информации в окно экрана

C++ включает большой набор функций ввода-вывода информации в окно экрана. Прототипы этих функций помещены в заголовочном файле <conio.h>. В отличие от функций стандартного ввода-вывода они позволяют управлять цветом выводимых символов и не пересекают пределы активного в данный момент окна. При достижении правой вертикальной границы курсор автоматически переходит на начало следующей строки в пределах окна, а при достижении нижней горизонтальной границы выполняется скроллинг окна вверх.

Функция clreol() стирает в текстовом окне строку, на которую установлен курсор, начиная с текущей позиции курсора и до конца строки (до правой вертикальной границы окна).

Функция clrscr() очищает все текстовое окно. Цвет "заливки" окна при очистке будет соответствовать значению, установленному символической переменной attribute в описании окна (структурная переменная по шаблону text_info). Функции управления цветом фона и символа описаны далее.

Аналог clrscr() – использование system("cls"); (добавить #include <stdlib.h>)

Функция delline() стирает в текстовом окне всю строку текста, на которую установлен курсор.

Функция insline() вставляет пустую строку в текущей позиции курсора со сдвигом всех остальных строк окна на одну строку вниз. При этом самая нижняя строка текста окна теряется.

Функция cprintf(const char *format,...) выполняет вывод информации с преобразованием по заданной форматной строке, на которую указывает format. Является аналогом функции стандартной библиотеки printf (), но выполняет вывод в пределах заданного окна. В отличие от printf () функция cprintf () иначе реагирует

на специальный символ '\n': курсор переводится на новую строку, но не возвращается к левой границе окна. Поэтому для перевода курсора на начало новой строки текстового окна следует вывести последовательность символов CR-LF (0x0d, 0x0a). Остальные специальные символы воздействуют на курсор так же, как и в случае функций стандартного ввода-вывода. Функция возвращает число выведенных байтов, а не число обработанных полей, как это делает функция printf () .

Функция cputs(char *str) выводит строку символов в текстовое окно, начиная с текущей позиции курсора. На начало выводимой ASCII-строки указывает указатель str. Является аналогом функции стандартной библиотеки puts (), но выполняет вывод в пределах заданного окна и при выводе не добавляет специальный символ '\n'. Реакция cputs() на специальный символ '\n' аналогична реакции cprintf(). Функция возвращает ASCII-код последнего выведенного на экран символа. В отличие от puts() в функции отсутствует возврат символа EOF. Другими словами, вывод происходит на экран в любом случае.

Функция movetext(int left, int top, int right, int bottom,int destleft, int desttop) переносит окно, заданное координатами левого верхнего (left, top) и правого нижнего (right, bottom) углов, в другое место на экране, заданное координатами левого верхнего угла нового положения окна. Размеры окна по горизонтали и вертикали сохраняются. Все координаты задаются относительно координат верхнего левого угла экрана (1,1). Функция возвращает ненулевое значение, если перенос заданного окна выполнен. В противном случае возвращается 0. Функция корректно выполняет перекрывающиеся переносы, т.е. переносы, в которых прямоугольная область-источник и область, в которую окно переносится, частично покрывают друг друга.

Функция putch(int ch) выводит символ в текущей позиции текстового окна экрана. Как и для функций cprintf(), cputs(), специальный символ ‘\n' вызывает только переход курсора на следующую строку текстового окна без возврата к его левой вертикальной границе. Остальные специальные символы воздействуют на курсор так же, как и для функций стандартного ввода-вывода. \

Функция puttext(int left, int top, int right, int bottom,void *source) выводит на экран текстовое окно, заданное координатами левого верхнего (left, top) и правого нижнего (right, bottom) углов. Символы и атрибуты располагаются в буфере, адрес начала которого задает указатель source. Другими словами, функция "открывает" (восстанавливает) текстовое окно экрана. Обычно используется вместе с функцией gettext(), выполняющей обратную операцию - запись в буфер source символов/атрибутов, полностью описывающих все знакоместа текстового окна. Функция проверяет по заданным координатам окна, можно ли построить окно на

экране для текущего режима видеоадаптера и корректны ли эти координаты. В случае, когда окно успешно выведено, возвращается ненулевое значение.

Функции highvideo (void), lowvideo (void) и normvideo (void) задают соответственно использование повышенной, пониженной и нормальной яркости для последующего вывода символов на экран.

Описываемые далее функции управляют атрибутом символа. Как отмечено ранее, атрибут задает битами 0-2 код цвета символа, бит 3 определяет повышение яркости, биты 4-6 задают код цвета фона символа, бит 7 определяет наличие или отсутствие мерцания символа. Возможно задать атрибут полностью либо задать только цвет символа или фона. Цвета могут задаваться либо числом, либо с использованием символических констант, значения которых определяет перечислимый тип COLORS (табл. 2.3):

Табл. 2.3.

```
Enum COLORS { /* Цвета нормальной яркости: */
BLACK,          /* черный,          0    */
BLUE,           /* синий,           1    */
GREEN,          /* зеленый.         2    */
CYAN,           /* сине-зеленый,    3    */
RED,            /* красный,         4    */
MAGENTA,        /* красно-синий,    5    */
BROWN,          /* коричневый,      6    */
LIGHTGRAY,      /* светло-серый.    7    */
                /* Цвета повышенной яркости: */
DARKGRAY,       /* темно-серый,     8    */
LIGHTBLUE,      /* ярко-синий,      9    */
LIGHTGREEN,     /* ярко-зеленый,    10   */
LIGHTCYAN,      /* яркий сине-зеленый, 11 */
LIGHTRED,       /* ярко-красный,    12   */
LIGHTMAGENTA,   /* яркий красно-синий, 13 */
YELLOW,         /* желтый,          14   */
WHITE           /* белый.           15   */}
```

Яркие цвета могут задаваться только цвету символа. Кроме того, 7-й бит (бит мерцания) может быть задан как непосредственно в коде байта атрибута, так и с использованием символической константы BLINK, определяемой как код 128. Следует отметить тот факт, что если для цвета фона выбираются цвета с кодами 8-15, это устанавливает в единицу бит мерцания символа в байте атрибута.

Функция textattr(int newattr) устанавливает атрибут для функций, работающих с текстовыми окнами. Атрибут хранится в поле attribute структурной переменной по шаблону text_info, доступной через функцию gettextinfo(). Задаваемый атрибут

может быть или числом, например, 0x70 - атрибут инверсного изображения (черные символы на светло-сером фоне) или формироваться из символических констант, значения которых задает тип COLORS. Например, для задания мерцающих ярко-красных символов на сером фоне атрибут можно сформировать следующим образом:

```
BLINK | (BLACK << 4) | LIGHTRED
```

Тот же результат может быть получен и так:

```
(DARKGRAY << 4)| LIGHTRED
```

Функция textcolor(int newcolor) задает цвет символов, не затрагивая установленный цвет фона. Цвет может быть или числом, или формироваться из символических констант, значения которых определяет перечислимый тип COLORS.

Функция textbackground(int newcolor) задает цвет фона символов, не затрагивая установленный цвет символа. Цвет может быть или числом, или формироваться из символических констант, значения которых определяет перечислимый тип COLORS. Для цвета фона выбор ограничен значениями цветов 0-7. Если для цвета фона выбирается значение 8 - 15, то символы будут мерцать, так как бит мерцания установится в единицу, но цвет фона будет соответствовать значениям 0-7.

2.8. Предварительная подготовка к работе

1. Ознакомиться с организацией и функциональными возможностями различных типов видеосистем.
2. В качестве программной среды при выполнении лабораторных работ 2 – 5 необходимо использовать TurboC ++ для Windows - это модифицированная версия Borland Turbo C ++, которую можно запускать в более новых версиях Windows, таких как Vista, 7, 8 и 10. Более детально смотрите [7,8]
3. Ознакомиться с текстовым режимом отображения информации на экран монитора и стандартными библиотечными функциями C++, обслуживающими этот режим.

2.9. Порядок выполнения работы

1. Написать программу, в которой в окно с координатами (x1,y1,x2,y2) с шагами T (секунд) и S (строк) выводится строка при всех возможных комбинациях цвета фона и цвета символов. Строка содержит обозначение цвета фона и символа. Для каждой комбинации цветов в окне должны выводиться номера или символьные обозначения цветов фона и символов (варианты приведены в табл. 2.4). Цвет окна должен соответствовать цвету фона.

Табл. 2.4. Варианты заданий

Номер варианта	Координаты окна				Обозначение цвета		Шаг		Направление
	Xl	Y1	X2	Y2	Фона	Символа	T	S	
1	10	5	70	15	Номер	Англ	0.3	1	Вверх
2	15	5	65	15	Номер	Англ	0.4	2	Вниз
3	20	5	60	15	Номер	Номер	0.5	3	Вверх
Номер варианта	Координаты окна				Обозначение цвета		Шаг		Направление
	Xl	Y1	X2	Y2	Фона	Символа	T	S	
4	25	5	55	15	Англ.	Англ.	0.6	1	Вниз
5	10	8	70	18	Англ.	Англ	0.7	2	Вверх
6	15	8	65	18	Англ.	Номер	0.8	3	Вниз
7	20	8	60	18	Англ.	Англ.	0.9	1	Вверх
8	25	8	55	18	Англ.	Англ.	1.2	2	Вниз
9	10	10	70	20	Англ.	Номер	1.3	3	Вверх
10	15	10	65	20	Номер	Англ.	1.4	1	Вниз
11	20	10	60	20	Номер	Англ.	1.5	2	Вверх
12	25	10	55	20	Номер	Номер	0.2	3	Вниз
13	10	12	70	23	Англ.	Англ.	0.3	1	Вверх
14	15	12	65	23	Англ.	Англ.	0.4	2	Вниз
15	20	12	60	23	Англ.	Номер	0.5	3	Вверх
16	25	12	55	23	Англ.	Англ.	0.6	1	Вниз
17	10	15	70	20	Англ.	Англ.	0.7	2	Вверх
18	15	15	65	20	Англ.	Номер	0.8	3	Вниз
19	20	15	60	20	Номер	Англ.	0.9	1	Вверх
20	25	15	55	20	Номер	Англ.	1.1	2	Вниз

2. Дополнить программу из п.1 скроллингом окна с направлением в соответствии с вариантом, используя функции прерывания 10h BIOS. Пример скроллинга приведен выше в п.2.6.

3. Две отлаженные программы предъявить преподавателю.

2.10. Содержание отчета

1. Краткие сведения о видеосистемах ПЭВМ, текстовом режиме их работы и функциях обслуживания текстового режима.

2. Задание на лабораторную работу;

3. Алгоритмы и тексты отлаженных программ.

4. Пример запуска программы

2.11. Контрольные вопросы

1. Что входит в понятие "видеосистема"?
2. Какие типы видеосистем вы знаете?
3. Назовите основные характеристики видеосистем?
4. Как влияет размер видеопамяти на характеристики системы?
5. Зачем нужен видеоадаптер?
6. Почему различают текстовый и графический режимы работы видеосистемы?
7. Назовите основные характеристики текстового режима, чем они обусловлены?
8. Что называется окном? Зачем нужны окна?
9. Можно ли на одном экране организовать несколько окон?
10. Какие функции инициализации текстового режима вы знаете?
11. Какие функции обслуживания окон вы знаете?
12. Что такое курсор и как можно им управлять?
13. Зачем нужен байт атрибутов символа?
14. Сколько цветов фона и символов можно одновременно использовать и почему?
15. Какая структура данных используется для хранения цветов?

Лабораторная работа № 3.

ИССЛЕДОВАНИЕ ВИДЕОСИСТЕМЫ (ГРАФИЧЕСКИЙ РЕЖИМ)

Цель работы - изучение работы с видеосистемой в графическом режиме, вывод графика заданной функции с масштабированием и разметкой осей.

3.1. Общие положения

Использование графики в языке C++ - это многошаговый процесс. Прежде всего необходимо определить тип видеоадаптера. Затем устанавливается подходящий режим его работы и выполняется инициализация графической системы в выбранном режиме. После этого становятся доступными для использования функции графической библиотеки graphics.h Turbo C ++ для построения основных графических примитивов: отрезков прямых линий, окружностей, эллипсов, прямоугольников, секторов, дуг и т.д., появляется возможность вывода текста с использованием различных шрифтов.

Использование библиотеки графики намного сокращает объем программирования для вывода основных графических примитивов. C++ "маскирует" многие технические детали управления оборудованием, о которых пользователь должен

быть осведомлен при работе с видеоадаптером через порты или BIOS. Платой за эти удобства является значительное увеличение размера .EXE-файлов. Использование графической библиотеки C++ требует знакомства с моделью графической системы, применяемой компилятором для представления графической системы компьютера. Можно сказать, что сложность овладения деталями аппаратных средств видеоадаптеров сравнима со сложностью освоения графической модели. Однако достоинство графической модели заключается в ее относительной независимости от различных типов видеоадаптеров и открытости для дальнейших расширений. Появление новых типов видеоадаптеров не потребует большой переработки программ, так как все новые особенности аппаратуры будут учитываться в средствах библиотеки C++.

Весь код библиотеки графики разбивается на две части: немобильную, которая зависит от типа видеоадаптера и мобильную.

Немобильная часть представляет собой так называемый .BGI-драйвер (BGI - Borland Graphics Interface). Драйвер является обработчиком прерывания 10h, который должен дополнить системный обработчик до того, как будут использоваться мобильные функции. Перед завершением программы таблица векторов прерывания восстанавливается.

Основные функции, выполняемые .BGI-драйвером, сводятся к установке и обновлению ряда внешних переменных, которые могут изменяться как функциями системного обработчика прерывания 10h (например, при переключении видеорежима, изменении регистров палитры и т.п.), так и мобильными функциями библиотеки графики. C++ включает целую коллекцию драйверов для каждого из типов адаптеров, хранимых обычно в отдельном поддиректории. Система графики является открытой для расширений, так как позволяет использовать и собственные .BGI-драйверы. Сложность состоит в том, что фирма Borland International не раскрывает пока внутреннюю структуру драйвера.

Совокупность внешних переменных библиотеки графики и особенностей поведения мобильных функций образует модель графики C++. Подробно эти элементы модели рассматриваются в следующих подразделах.

3.2. Инициализация и закрытие системы графики

Прежде чем использовать функции графической библиотеки C++, необходимо инициализировать систему графики - загрузить соответствующий адаптеру или режиму .BGI-драйвер, установить в начальные значения внешние переменные и константы, выбрать шрифт и т.д.

Графические режимы, поддерживаемые библиотекой графики, задаются символическими константами, описанными в заголовочном файле <graphics.h> в пе-

речислимом типе graphics_modes. Константы, определяющие видеорежим, приведены в табл. 3.1 вместе с информацией о выбираемом режиме и типе видеоадаптера, который может такой режим поддерживать.

Табл. 3.1. Видеорежимы в библиотеке графики

Константа режима	Характеристика режима	Номер режима	Тип адаптера и драйвер
CGAC0 CGAC1 CGAC2 CGAC3	320x200, палитра 0 320x200, палитра 1 320x200, палитра 2 320x200, палитра 3	4,5	CGA, EGA, VGA, MCGA и др. в режиме эмуляции CGA. Используется CGA.BGI
CGAHI	640x200, 2 цвета	6	
MCGAC0 MCGAC1 MCGAC2 MCGAC3	320x200, палитра 0 320x200, палитра 1 320x200, палитра 2 320x200, палитра 3	4,5	MCGA. Используется MCGA.BGI
MCGAMED	640x200, 2 цвета	6	
MCGAHI	640x480, 2 цвета	11h	
EGALO	640x200, 16 цветов	0Eh	EGA с памятью >128K байт, VGA при эмуляции EGA. Используется EGAVGA.BGI
EGAHI	640x350, 16 цветов	10h	
EGA64LO	640x200, 16 цветов	0Eh	EGA с памятью 64K байт, VGA при эмуляции EGA. Используется EGAVGA.BGI
EGA64HI	640x350, 4 цвета	10h	
EGAMONOHI	640x350, 2 цвета	0Fh	EGA, VGA при эмуляции EGA Используется EGAVGA.BGI
HERCMONOHI	720x348.	7h	
ATT400C0 ATT400C1 ATT400C2	320x200, палитра 0 320x200, палитра 1 320x200, палитра 2	4,5	AT&T. Используется ATT.BGI
VGALO	640x200, 16 цветов	0Eh	VGA. Используется EGAVGA.BGI
VGAMED	640x350, 16 цветов	10h	
VGAHI	640x480, 16 цветов	12h	
PC3270HI	720x350, 1 с.	?	IBM PC 3270. Используется PC3270.BGI
IBM8514LO	640x480, 256 цветов	?	IBM 8514. Используется IBM8514.BGI
IBM8514HI	1024x768, 256 цветов	?	IBM 8514. Используется IBM8514.BGI

Примечание. Символом “?” обозначены режимы, не вошедшие в табл. 2. 1 .

Инициализацию графической модели выполняет функция initgraph().

void far initgraph(int *graphdriver, int *graphmode, char * pathtodriver).

При вызове она инициализирует графическую систему, загружая .BGI-драйвер, определяемый указателем graphdriver, и устанавливая видеоадаптер в графический режим, задаваемый указателем graphmode. Аргумент pathtodriver указывает на ASCII-строку, хранящую спецификацию файла .BGI-драйвера. C++ поддерживает фиксированное число драйверов, каждый из которых, в свою очередь, поддерживает ряд режимов. Как тип драйвера, так и режим могут быть заданы числом или символической константой. Возможные значения для графических режимов даны в табл. 3.1. В табл. 3.2. приведены значения, определяющие графические драйверы при инициализации системы графики. Упомянутые в таблице символические константы определены в перечислимом типе graphics_drivers из заголовочного файла <graphics.h>.

Третий аргумент функции initgraph() задает маршрут поиска файла, содержащего .BGI-драйвер. Если файл не найден в заданной директории, функция просматривает текущий директорий. Если pathtodriver = NULL, драйвер должен располагаться в текущей директории. В случае, когда при вызове initgraph() параметры видеосистемы неизвестны, значение для graphdriver следует задать равным указателю на DETECT.

Благодаря этому функция initgraph() вызывает другую библиотечную функцию – detectgraph() - для определения типа видеоадаптера, подходящего графического драйвера и графического режима максимального разрешения (максимального режима) для активного видеоадаптера системы. Значения для драйвера и максимального режима возвращаются в ячейках памяти, на которые указывают graphdriver и graphmode.

Табл. 3.2. Задание используемого .BGI-драйвера

Символическая константа из graphics_drivers	Значение (в 10 с/с)	Описание
DETECT	0	Запрос автоматического определения типа драйвера
CGA	1	Загрузка драйвера для CGA-адаптера или переключение старших адаптеров в режим эмуляции CGA и загрузка драйвера для CGA-адаптера

MCGA EGA	2 3	Загрузка драйвера для MCGA-адаптера Загрузка драйвера для EGA-адаптера с объемом видеопамяти 128К байт и более и ECD-монитором
EGA64	4	Загрузка драйвера для EGA-адаптера с объемом видеопамяти 64К байт и ECD-монитором
Символическая константа из graphics_drivers	Значение (в 10 с/с)	Описание
EGAMONO	5	Загрузка драйвера для EGA-адаптера с объемом видеопамяти 64К байт и монохроматическим монитором
IBM8514	6	Загрузка драйвера для адаптера IBM 8514 с аналоговым монитором
HERCMONO	7	Загрузка драйвера для адаптера HGC с монохроматическим монитором
ATT400	8	Загрузка драйвера для графического адаптера AT&T с разрешением 400 линий
VGA	9	Загрузка драйвера для VGA-адаптера с аналоговым монитором
PC3270	10	Загрузка драйвера для графического адаптера IBM PC 3270 с аналоговым монитором

Помимо перевода видеоадаптера в заданный графический режим, функция initgraph() динамически распределяет оперативную память для загружаемого драйвера и хранения промежуточных результатов, возникающих при работе некоторых функций графики. После загрузки драйвера initgraph() устанавливает в значения по умолчанию ряд параметров графики: стиль линий, шаблоны заполнения, регистры палитры. С этого момента прикладная программа может использовать любую функцию, прототип которой есть в заголовочном файле <graphics.h>.

Если при выполнении инициализации возникает противоречие между запрашиваемым режимом и типом видеоадаптера, либо отсутствует достаточный объем свободной оперативной памяти и т.п., функция устанавливает код ошибки во внешней переменной, доступной после вызова функции graphresult(). Кроме того, код ошибки передается в точку вызова в ячейке памяти, на которую указывает graphdriver.

Если функции графической библиотеки больше не нужны прикладной программе, следует вызвать функцию closegraph() "закрытия" графического режима и возвращения к текстовому режиму.

closegraph().

Эта функция освобождает память, распределенную под драйверы графики, файлы шрифтов и промежуточные данные и восстанавливает режим работы адап-

тера в то состояние, в котором он находился до выполнения инициализации системы.

Приведем "скелет" программы, выполняющей все необходимые подготовительные действия для использования функций библиотеки графики. Для определения типа видеоадаптера в ней используется функция initgraph() .

```
#include <graphics.h> /* все графические функции используют данный заголовочный файл */
int main(void)
{
int graph_driver;          /* используемый драйвер */
int  graph_mode;                    /*  графический  режим  видеоадаптера  */
int graph_error_code;  /* внутренний код ошибки */
/* Определение типа видеоадаптера, загрузка подходящего .BGI-драйвера и установка максимального режима. Считается, что драйвер находится на диске d: в директории \bc\bgi. */ graph_driver = DETECT;
initgraph(&graph_driver, &graph_mode, "d:\\bc\\bgi” );
/* Определение кода ошибки при выполнении инициализации. */
graph_error_code = graphresult( );
      if(graph_error_code != grOk)       /* всегда следует проверять наличие ошибки !                  */
{
/* Обработка ошибки . return 255; */
return 255;
      }
```

/* Установка в случае необходимости режима, отличающегося от максимального; выбор палитры, цвета, стиля линий, маски заполнения и других параметров, отличающихся от значений по умолчанию. Вывод графических примитивов: прямых линий, окружностей, эллипсов, столбцовых диаграмм и т.п. */

```
/* Закрытие графического режима */
closegraph();
}
```

Наиболее защищенный способ использования функции инициализации требует предварительного уточнения типа адаптера дисплея, активного в текущий момент времени. Для этого либо вызывается функция initgraph() со значением для graphdriver, равным указателю на DETECT, либо явно вызывается функция detectgraph(). Только после определения типа адаптера и его максимального режима выполняются установка нужного пользователю режима и загрузка .BGI-драйвера. Далее приводится описание функции detectgraph().

void detectgraph (int *graphdriver, int *graphmode).

Определяет тип активного видеоадаптера системы и тип подключенного монитора в персональном компьютере. Затем устанавливает тип подходящего для комбинации адаптер/монитор .BGI-драйвера и режим, обеспечивающий максимальное разрешение (максимальный режим). Например, если активным является CGA-адаптер, C++ считает режим 640 x 200 максимальным. Информация о подходящем драйвере и максимальном режиме возвращается в точку вызова в двух переменных, на которые указывают graphdriver и graphmode соответственно.

Прикладная программа может интерпретировать тип драйвера и максимальный режим, сравнивая возвращаемые значения с символическими константами, приведенными в табл. 3.1. и 3.2. В случае, если адаптер не способен работать ни в одном из графических режимов, функция устанавливает внутреннюю переменную кода ошибки в значение, равное grNotDetected (-2). На это же значение будет указывать и graphdriver при завершении функции.

Как отмечалось ранее, функция detectgraph() вызывается автоматически из функции инициализации видеосистемы initgraph(), если последняя вызывается со значением для graphdriver, равным указателю на DETECT. В отличие от функции detectgraph() функция инициализации продолжает свою работу, загружает драйвер и устанавливает максимальный режим, рекомендованный (возвращенный) функцией detectgraph(). Функция detectgraph(), вызванная явно, не производит загрузку драйвера или установку режима. Для этого прикладная программа выполняет обращение к функции инициализации. В случае, если для функции initgraph(), вызываемой после явного обращения к detectgraph(), передаются параметры, возвращенные detectgraph(), получается такой же результат, что и при обращении к initgraph() с параметром graphdriver, равным указателю на DETECT. В этой связи раздельное обращение к detectgraph() и initgraph() имеет смысл лишь в случае, когда предполагается установка режима адаптера, отличающегося от максимального, т.е. если неприемлемы автоматические выбор и установка режима адаптера.

3.3. Обработка ошибок системы графики

Защищенное от ошибок построение программы требует использования функции graphresult() после любого обращения к функциям detectgraph() и initgraph(). Далее следует описание функций обработки ошибок, сообщающих внутренние коды ошибок графической библиотеки (graphresult()) или формирующей строку

диагностического сообщения (grapherrormsg()).

int graphresult(void)

Возвращает значение внутреннего кода ошибки, установленного последним обращением к функциям графической библиотеки. Перед завершением сбрасывает код ошибки в 0. Прикладная программа может интерпретировать возвращаемое значение, сопоставляя его с целым числом либо с символической константой из перечислимого типа graphics_errors, определенного в <graphics.h> (табл. 3.3).

char * grapherrormsg(int errorcode)

Возвращает указатель на ASCII-строку символов, содержащую сообщение об ошибке, соответствующее внутреннему коду ошибки errorcode функций графики Turbo C. Функция grapherrormsg() возвращает указатели на сообщения на английском языке. В принципе несложно выполнить их "перевод" непосредственно, переработав саму функцию grapherrormsg().

Табл. 3.3. Коды ошибок, возвращаемые при выполнении функций графической библиотеки.

Символическая константа из graphics_errors	Значение (в 10 с/с)	Описание
grOk	0	Отсутствие ошибки
grNoInitGraph	-1	Графический интерфейс (.BGI-драйвер) не инсталлирован. Следует выполнить initgraph()
grNotDetected	-2	Не обнаружен видеоадаптер, способный работать в запрошенном (или любом в случае DETECT) графическом режиме
grFileNotFound	-3	Не найден по заданному маршруту и в текущем директории .BGI-файл
grlnvalidDriver	-4	Заданный в качестве .BGI-драйвера файл не соответствует стандарту Turbo C
grNoLoadMem	-5	Недостаточно свободной памяти для загрузки драйвера и хранения промежуточных результатов
grNoScanMem	-6	Нехватка памяти при выполнении графических функций заполнения
grNoFloodMem	-7	Нехватка памяти при выполнении графических функций заполнения
grFontNotFound	-8	Не найден файл описания шрифта
grNoFontMem	-9	Отсутствие памяти для загрузки файла шрифта

grlnvalidMode	-10	Недопустимый графический режим для выбранного .BGI-драйвера
grError	-11	Ошибка функции графики
grIOerror	-12	Ошибка ввода-вывода в графическом режиме
Символическая константа из graphics_errors	Значение (в 10 с/с)	Описание
grlnvalidFont	-13	Файл шрифта, не соответствующий стандарту Borland International
grlnvalidFontNum	-14	Недопустимый номер шрифта
grlnvalidDeviceNum	-15	Недопустимый номер устройства
grlnvalidVersion	-18	Недопустимый номер версии .BGI-драйвера (.BGI-драйвер для версии 1.5)

3.4. Определение и установка графического режима

После того, как проведена инициализация графической системы, может быть установлен другой, не превосходящий максимального, режим видеоадаптера и выбраны цвета для пикселов. Установку режима выполняет функция setgraphmode(). Целая группа функций – getgraphmode(), getmaxmode(), getmodename() , getmoderange() - упрощает работу по определению текущего установленного режима. Две функции позволяют определить ширину и высоту экрана в пикселах для текущего видеорежима: getmaxx() и getmaxy(). Функция restorecrtmode() возвращает видеоадаптер в текстовый режим. Далее следует описание упомянутых функций.

int getgraphmode (void)

Возвращает текущий графический режим, установленный для графической модели функциями initgraph() или setgraphmode(). Возвращаемое значение соответствует номеру режима, установленному для инсталлированного драйвера графики. Возвращаемое значение соответствует числовому значению символических констант режима, перечисленных в табл. 3 .1.

int getmaxmode(void)

Возвращает число, определяющее максимально возможный для инсталлированного .BGI-драйвера режим. Как и в предыдущем случае, возвращаемое значение соответствует номеру режима, установленному для инсталлированного драйвера графики. Возвращаемое значение соответствует числовому значению симво-

лических констант режима, перечисленных в табл. 3.1.

```
int getmaxx(void)
int getmaxy(void)
```

Возвращают максимальные значения координат X и Y для текущего видеорежима. Например, для режима CGA0 getmaxx() возвращает значение 319, а getmaxy() -199. Функции особенно полезны для центрирования изображений и определения таких размеров знакомест при выводе текста в графическом режиме, чтобы текст помещался в заданную область экрана.

```
char * getmodename(int mode_number)
```

Возвращает указатель на ASCII-строку символов, содержащую имя символической константы, соответствующей режиму mode_number. Значение mode_number должно быть в пределах диапазона значений, возвращенных функцией getmaxmode() (для любого драйвера) или getmoderange() (для драйвера Borland International).

```
void setgraphmode(int mode)
```

Устанавливает видеосистему в режим, заданный значением переменной mode, и сбрасывает значения внутренних переменных системы графики в их значения по умолчанию (стиль линий, маска заполнения, шрифт и т.д.). Значение mode соответствует числовому значению символических констант режима, перечисленных в табл. 3.1. При задании недопустимого режима для текущего .BGI-драйвера функция устанавливает внутренний код ошибки -1 (см. табл. 3.3).

Обычно функция используется для обратного переключения в графический режим после того, как видеоадаптер был на время переключен функцией restorecrtmode() в текстовый режим. Однако функция может использоваться для переключения и из одного графического режима в другой, не выходящий за диапазон допустимых режимов.

```
void restorecrtmode(void)
```

Возвращает видеоадаптер в режим, в котором он был до выполнения инициализации системы графики. Как правило, исходным режимом будет текстовый. В том случае, если выполняется временный возврат в исходный (текстовый) режим, перед выполнением функции restorecrtmode() в переменной следует сохранить текущий графический режим, используя, например, обращение к функции getgraphmode(), после чего возможен возврат в графический режим с помощью функции setgraphmode().

3.5. Управление цветами и палитрами

После инициализации системы графики и установки нужного видеорежима возможен выбор необходимых цветов пикселов. Возможности по выбору цветов принципиально различны для CGA-, EGA- и VGA-адаптеров, что обусловлено различной логикой построения аппаратных средств.

Далее приведена спецификация функций библиотеки графики для работы с цветами и палитрами.

int getbkcolor(void)

Возвращает целое число, равное коду цвета фона.

int getmaxcolor(void)

Возвращает максимальное значение кода цвета пиксела минус 1. Это значение позволяет установить максимальное число цветов, которое может отображаться на экране. В зависимости от режима, в котором проведена инициализация системы графики, возвращаемое значение может быть равно 1, 3 или 15.

void setbkcolor (int color)

Устанавливает новый цвет пикселов, имеющих код цвета 0. Новый цвет фона задает значение аргумента color.

void setcolor (int color)

Устанавливает цвет, используемый функциями графического вывода в значение, заданное аргументом color. До того момента, пока цвет не установлен, используется максимальный (из палитры) номер цвета. В случае, если color задает недопустимый номер цвета для текущей палитры, текущий цвет остается неизменным.

3.6. Задание окна экрана. Определение и установка графических координат

Окно экрана в графическом режиме, или графическое окно (viewport), - это прямоугольная область экрана, заданная пиксельными координатами левого верхнего и правого нижнего углов. В графическом окне определены относительные координаты. C++ позволяет выполнять вывод текста и графических примитивов в графическое окно. При этом по желанию пользователя вывод, не вмещающийся в

границы окна, может усекаться. Графическое окно может иметь отличающиеся от других участков экрана цвета фона и пикселов, маску заполнения и другие характеристики.

Для описания окна используется функция setviewport(). Текущие характеристики окна доступны программе через обращение к функции getviewsettings().

```
void far getviewsettings( struct viewporttype *viewport)
```

Заполняет поля структурной переменной по шаблону viewporttype информацией о графическом окне. Описание структурной переменной выполняет вызывающая сторона. Функции передается указатель на описанную переменную. Шаблон viewporttype описан в <graphics.h>:

```
struct viewporttype
{
int left, top;        /* координаты ( столбец, строка) левого верхнего угла* /
int right, bottom; /* координаты (столбец, строка) правого нижнего угла */
int clip;             /* Флаг усечения при выводе (1 - усечение, 0 - нет )*/
 }
```

Левый верхний угол окна рассматривается как начало относительных координат X и Y всеми функциями графического вывода, в том числе и при выводе текста в графических режимах. Сразу после инициализации системы графики графическое окно охватывает весь экран, и, таким образом, началом графических координат по умолчанию является самый левый верхний угол экрана. Основное применение функции - определение и сохранение характеристик текущего графического окна перед переопределением текущего окна для последующего восстановления параметров окна.

```
void setviewport (int left, int top, int right, int bottom, int clip)
```

Описывает новое графическое окно с координатами (столбец, строка) левого верхнего угла left, top, координатами правого нижнего угла right, bottom и значением флага усечения clip. В качестве начала текущих координат для функций графического вывода устанавливается левый верхний угол.

Помимо явного задания окна функцией setviewport(), оно специфицируется и неявно при выполнении функций initgraph(), setgraphmode() и graphdefaults(). При каждом их выполнении в качестве графического окна устанавливается весь экран.

Графические координаты X и Y измеряются в пикселах экрана относительно координат левого верхнего угла текущего окна. Функции графического вывода изменяют эти координаты в соответствии с объемом выведенной на экран инфор-

мации. Текущие координаты в окне доступны через функции getx() и gety(). Установку нужных значений координат текущей позиции выполняют функции moveto() и moverel(). Кроме того, некоторые функции графического вывода позволяют задать текущую позицию (см., например, outtextxy()).

```
int getx (void)
int gety (void)
```

Возвращают текущие координаты X и Y, измеряемые относительно координат левого верхнего угла текущего графического окна.

```
void moveto (int x, int y)
```

Устанавливает новое значение координат текущей позиции. Аргументы x, y задают новые значения координат текущей позиции относительно координат левого верхнего угла текущего графического окна.

```
void moverel(int dx, int dy)
```

Устанавливает новое значение координат текущей позиции. Аргументы dx, dy задают новые значения координат относительно текущих координат графического окна. Другими словами, новая текущая позиция устанавливается в точку, отстоящую от текущей позиции на dx столбцов пикселов по горизонтали и dy строк пикселов по вертикали. Чтобы переместить текущую позицию влево, нужно задать для dx отрицательное значение. Для перемещения текущей позиции по вертикали вверх задается отрицательное значение для dy.

3.7. Вывод текста в графическом режиме видеоадаптера

C++ предоставляет пользователю широкие возможности по выводу текстовой информации в графических режимах. Во-первых, это все функции стандартного вывода.

Библиотека графики позволяет выводить на экран текст различными шрифтами. C++ имеет два типа шрифтов: битовый и сегментированный.

Каждому символу битового шрифта (bit-mapped font) ставится в соответствие матрица пикселов фиксированного размера. C++ использует в качестве битового шрифта таблицу знакогенератора для символов размером 8x8, установленную в компьютере перед инициализацией системы графики. Все изменения таблицы

знакогенератора, сделанные, например, программами русификации, будут сохранены. Это позволяет, применяя функции, выводить текст русскими буквами и в графических режимах.

Другой тип шрифтов, используемый при выводе текста на экран, фактически задает правило "рисования" каждого символа. Он описывается как совокупность отрезков прямых линий, или сегментов. Этим и объясняется название шрифта - сегментированный (stroke font). Программа может задать масштаб для каждого символа, "растягивая" или "сжимая" его по высоте либо ширине. Однако использование сегментированного шрифта для вывода текста несколько замедляет работу видеосистемы.

В C++ доступны четыре сегментированных шрифта: Triplex, Small, Sans-Serif и Gothic. Файлы сегментированных шрифтов располагаются в файлах с расширением .CHR и, подобно .BGI-драйверам, загружаются как оверлеи во время исполнения программы. Библиотека графики дает возможность выводить текст в графических режимах слева направо и снизу вверх. Для вывода символов при использовании любого шрифта может быть задан масштаб знакоместа по отношению к знакоместу шрифта 8x8. При использовании сегментированного шрифта может быть задан размер знакоместа и в относительных единицах масштаба как по вертикали, так и по горизонтали. Кроме того, выводимые строки текста могут выравниваться по-разному: строка может быть "прижата" влево и вверх относительно точки, определенной как текущая точка отсчета, влево и вниз и т.п.

Поведение системы графики при выводе текста в графическом режиме задается целой группой значений внутренних переменных. Их текущие установки доступны после вызова функции gettextsettings().

```
void gettextsettings( struct textsettingstype *texttypeinfo)
```

Заполняет поля структурной переменной по шаблону textsettingstype информацией о текущих шрифте, направлении вывода текста, размере знакоместа относительно шрифта по умолчанию и способе "прижатия" (выравнивания) шрифта в пределах знакоместа. Функции передается указатель texttypeinfo на описанную структурную переменную. Шаблон textsettingstype описывается в <graphlcs.h> так:

```
struct textsettingstype
{
int font;        /* номер шрифта из перечислимого типа font_names[] */
```

```
 int direction;  /* направление вывода текста (гор. или верт.) */
  int charsize;  /* размер знакоместа относительно шрифта 8x8 */
  int horiz;       /* код выравнивания по горизонтали (влево, вправо) */
  int vert;        /* код выравнивания по вертикали (вверх, вниз) */
 }
```

Интерпретация отдельных полей структурной переменной по шаблону textsettingstype приведена в описании функции settextstyle().

```
void settextstyle( int font, int direction, int charsize)
```

Выбирает шрифт, устанавливает направление и размер знакоместа для последующего вывода текстовой информации через функции библиотеки графики outtext() и outtextxy(). Значение font выбирает один из шрифтов Turbo C. Возможные типы шрифтов задаются либо целым числом, либо символической константой из перечислимого типа font_names. В табл. 3.4. приведены доступные шрифты, соответствующие им символические константы, числовые значения и имена файлов шрифтов.

Табл. 3.4. Шрифты, доступные в C++

Символическая константа из font_names	Значение	Описание шрифта	Имя файла шрифта
DEFAULT FONT	0	Битовый шрифт 8x8	-
TRIPLEX_FONT	1	Сегментированный шрифт Triplex	TRIP.CHR
SMALL_FONT	2	Сегментированный шрифт Small	SMAL.CHR
SANS_SERIF_FONT	3	Сегментированный шрифт Sans-Serif	SANS.CHR
T GOTHIC_FONT	4	Сегментированный шрифт Gothic	GOTH.CHR

Значение direction позволяет специфицировать направление вывода. Если direction = HORIZ_DIR, текст будет выводиться горизонтально слева направо. Если direction = VERT_DIR, текст будет выводиться вертикально снизу вверх, а символы будут повернуты на 90 градусов против хода часовой стрелки.

Третий аргумент функции - charsize - задает масштаб каждого символа относительно знакоместа 8x8. Если, например, задать charsize равным 5, то символ будет изображаться в знакоместе 40 x 40. Если charsize равен 0, то для битового шрифта размер знакоместа изменяться не будет. Для сегментированного шрифта размер знакоместа будет определяться значениями, установленными функцией setusercharsize().

В случае ошибки функция settextstyle() устанавливает во внутренней переменной системы графики соответствующий код ошибки. Он доступен программе через обращение к функции graphresult().

Установленные в системе графики значения высоты символа и его ширины можно получить через обращения к функциям textheight() и textwidth().

```
int textheight( char *textstring)
```

Возвращает высоту строки символов в пикселах, на которую указывает textstring. Использует информацию о текущем шрифте и установках масштаба знакоместа. Сама строка, на которую указывает textstring, на экран не выводится. Наиболее часто функция используется для установки нужных промежутков между строками текста, а также при вычислении таких масштабов для символов, которые позволяли бы уместить нужное число строк в фиксированной области экрана.

```
int textwidth ( char far *textstring)
```

Возвращает ширину строки символов в пикселах, на которую указывает textstring. Использует информацию о текущем шрифте и установках масштаба знакоместа. Сама строка, на которую указывает textstring, на экран не выводится. Наиболее часто функция используется для установки нужных промежутков между символами текста, а также при вычислении таких масштабов для символов, которые позволяли бы уместить нужное число строк в фиксированной области экрана. В частности, обращение к функции

```
textwidth("A");
```

возвращает ширину символа 'A' в пикселах.

Еще одна установка системы графики, затрагивающая вывод текста в текстовом режиме, это выравнивание символов. Специальная функция библиотеки графики settextjustify() позволяет изменить установку по умолчанию для выравнивания символов при выводе текста в графических режимах.

```
void far settextjustify(int horiz,int vert)
```

Задает новую установку выравнивания символов текста в графических режимах работы адаптера. Она выполняет выравнивание всей строки символов, вы-

водимой функциями outtext() или outtextxy() относительно некоторой точки, называемой далее точкой отсчета. Координаты точки отсчета задаются либо явно функцией outtextxy(), либо используются текущие значения.

Аргумент horiz может принимать три значения, задаваемых символическими константами: LEFT_TEXT - левая граница строки "прижимается" справа к вертикальной линии, проведенной через точку отсчета; CENTER_TEXT - строка располагается так, что вертикальная линия, проведенная через ее середину, проходит через точку отсчета; RIGHT_TEXT - правая граница строки "прижимается" слева к вертикальной линии, проведенной через точку отсчета.

Аргумент vert также может принимать три значения, задаваемых символическими константами: BOTTOM_TEXT - нижняя граница строки "прижимается" сверху к горизонтальной линии, проведенной через точку отсчета; CENTER_TEXT - строка располагается так, что горизонтальная линия, проведенная через ее середину, проходит через точку отсчета; TOP_TEXT - верхняя граница строки "прижимается" снизу к горизонтальной линии, проведенной через точку отсчета.

Отметим, что при выводе текста вертикально установки выравнивания по горизонтали LEFT_TEXT и RIGHT_TEXT не различаются библиотекой графики Turbo C и аналогичны RIGHT_TEXT.

После того, как заданы все необходимые параметры текста, можно выводить ASCII-строки, используя функцию outtext() или outtextxy().

```
void outtext (char *textstring)
```

Выводит ASCII-строку текста, на начало которой указывает textstring, используя текущие позицию, цвет и установки направления, типа шрифта и выравнивания строки. В случае, когда текст выводится горизонтально и установлено выравнивание LEFT_TEXT, функция outtext() продвигает координату X текущей позиции на значение, равное textwidth(textstring). В остальных случаях координата X текущей позиции остается неизменной. Если текст выводится в графическое окно с включенным усечением, текст усекается на границах окна. Для сегментированных шрифтов усечение производится с точностью до пикселов, для битовых шрифтов оно происходит с точностью до символа. Усечение строки может выполняться по одной границе или по обеим границам сразу в случае, когда задается выравнивание по центру.

```
void outtextxy (int x, int y, char *textstring)
```

Выводит ASCII-строку текста, на начало которой указывает textstring, используя текущие цвет, установки направления, типа шрифта и выравнивания строки. Аргументы x и y явно специфицируют новую текущую позицию, используемую для вывода строки. Координаты X и Y измеряются относительно координат левого верхнего угла текущего графического окна. В случае, когда текст выводится горизонтально и установлено выравнивание LEFT_TEXT, функция outtext() продвигает координату X текущей позиции на значение, равное textwidth(textstring). В остальных случаях координата X текущей позиции остается неизменной. Если текст выводится в графическое окно с включенным усечением, он усекается на границах окна. Для сегментированных шрифтов усечение производится с точностью до пикселов, для битовых шрифтов оно происходит с точностью до символа. В случае, когда установлено выравнивание CENTER_TEXT, но выводимая строка не помещается в текущем графическом окне, функция не выполняет вывод.

Функции способны выводить только нуль-терминированные строки, и для выполнения форматированного вывода в графических режимах выбранными сегментированными шрифтами поступают следующим образом. Сначала, используя функцию стандартного вывода sprintf(), получают нужную форматную строку, а затем выводят ее с помощью функции outtextxy() выбранным шрифтом.

3.8. Вывод графической информации

3.8.1. Параметры и атрибуты графического вывода

Если инициализация системы графики выполнена успешно, становятся доступными функции графической библиотеки для построения основных графических примитивов - отрезков прямых линий, дуг, окружностей, эллипсов, прямоугольников, секторных и столбцовых диаграмм и т.д. Многие из этих фигур могут быть по желанию программиста "залиты" текущим цветом с использованием текущего шаблона или маски заполнения. Специальные функции позволяют запоминать прямоугольные области экрана и затем восстанавливать их в специфицированном месте экрана.

Все функции библиотеки графики, генерирующие вывод информации на экран, работают в пределах текущего графического окна. Для графического вывода используется текущий цвет пиксела, установленный функцией setcolor().

При выводе отрезков прямых линий и графических примитивов система графики позволяет определить такой параметр, как стиль линии. C++ поддерживает

ряд предопределенных стилей линий. Как и в случае маски заполнения, пользователь может описать собственный стиль линии. Для определения текущей установки стиля используется функция getlinesettings(). Выбор подходящего стиля выполняет функция setlinestyle().

```
void getlinesettings (struct linesettingstype *lineinfo)
```

Возвращает информацию об установленном в текущий момент времени стиле "рисования" отрезков прямых линий и графических примитивов. Функция заполняет поля структурной переменной по шаблону struct linesettingstype. Структурную переменную описывает точка вызова и передает в функцию указатель lineinfo на эту переменную. Шаблон struct linesettingstype описан в заголовочном файле <graphics.h> следующим образом:

```
struct linesettingstype
{
                        int linestyle;     /*идентифицирует стиль линии*/
 unsigned upattern;     /*задает определенный пользователем стиль линии.
Имеет значение       только тогда, когда linestyle = USERBIT_LINE*/
                        int thickness;     /*задает толщину линии*/
}
```

Стиль линии задается либо целым числом, либо символической константой по табл. 3.5.

Табл. 3.5. Задание стиля линий

Символическая константа	Значение	Описание стиля линии
SOLID_LINE	0	Сплошная (непрерывная) линия
DOTTED_LINE	1	Линия из точек ....
CENTER_LINE	2	Штрих-пунктирная линия -.-.
DASHED_LINE	3	Штриховая линия
USERBIT_LINE	4	Определенная пользователем линия. Ее шаблон описывает поле upattern в структуре linesettingstype

Толщина линий может быть равна 1 или 3 пикселам. Она задается либо целым числом, либо символической константой из табл. 3.6.

Табл. 3.6. Задание толщины линий в Turbo C

Символическая константа	Значение	Толщина

NORM_WIDTH	1	Задает толщину линии 1 пиксел
THICK_WIDTH	3	Задает толщину линии 3 пиксела

После инициализации системы графики для стиля линии устанавливается непрерывная линия толщиной 1 пиксел. Задание стиля линии выполняет функция setlinestyle().

```
void setlinestyle (int linestyle, unsigned upattern, int thickness)
```

Устанавливает стиль "рисования" отрезков прямых линий и графических примитивов. Аргумент linestyle выбирает стиль линии в соответствии с табл. 3.5., а аргумент thickness - толщину линии по табл. 3.6.

Аргумент upattern используется только в том случае, когда задается отличный от предопределенных стиль линии, т.е. если linestyle равен USERBIT_LINE (4). При этом 16 бит аргумента upattern задают маску линии (светимость 16 подряд расположенных пикселов линии). Если бит в upattern равен 1, пиксел выводится на экран текущим цветом, установленным функцией setcolor(). Определенный пользователем стиль линии действует только в случае, когда устанавливается толщина линии 1 пиксел (NORM_WIDTH). Для толщины 3 пиксела определенный пользователем стиль линии не действует.

При выводе отрезков прямых линий в графическом режиме система графики позволяет задать дополнительно режим вывода линии. Существуют два различных режима, устанавливаемых функцией setwritemode().

```
void setwritemode(int mode)
```

Устанавливает режим вывода отрезков прямых линий в значение, определяемое аргументом mode. Аргумент mode может принимать одно из двух значений, описанных в <graphics.h>: COPY_PUT (0) - пикселы, лежащие на отрезке прямой линии, переопределяют пикселы на экране, и, таким образом, линия на экране имеет текущий цвет; XOR_PUT (1) - пикселы, образующие линию, имеют код цвета, образуемый операцией исключающего ИЛИ (XOR) кода текущего цвета и кода цвета пикселов на экране, через которые линия проходит. В частности, можно стереть выведенную линию с экрана, выполнив вывод линии еще раз.

Следующий параметр системы графики - так называемый коэффициент сжатия, или коэффициент пропорциональности (aspect ratio). Он задает форму пиксе-

ла на экране монитора. Для многих мониторов световое пятно, которое соответствует пикселу, не является строго квадратным, а напоминает по форме эллипс, вытянутый вверх. Как следствие этого, линия, состоящая из одного и того же числа пикселов, расположенная вертикально, выглядит на экране длиннее, чем линия из того же числа пикселов, расположенная горизонтально. По этой же причине вывод прямоугольника, имеющего равные (в пикселах) горизонтальную и вертикальную стороны, не приводит к получению на экране квадрата. Система графики учитывает коэффициент сжатия при выводе сложных графических примитивов - эллипсов, окружностей, дуг, круговых секторов. Благодаря этому они появляются на экране геометрически корректными. По умолчанию после инициализации системы графики автоматически устанавливает коэффициент сжатия в соответствии с характеристиками аппаратуры видеосистемы. Для управления коэффициентом сжатия в предусмотрены две функции: getaspectratio() и setaspectratio().

void getaspectratio (int *xasp, int *yasp)

Заполняет две переменные, описанные точкой вызова, значениями коэффициента сжатия для текущего видеорежима. Возвращаемые значения задают фактически физическую форму пиксела. Для размера пиксела по вертикали (значение, на которое указывает yasp), всегда возвращается 10 000. Если световое пятно на экране, соответствующее пикселу, является квадратным (как для адаптера VGA), то и значение "ширины" пиксела равно 10 000. Для других видеоадаптеров пиксел на экране имеет эллипсообразную форму с большой полуосью, сориентированной по вертикали. Для таких адаптеров в ячейке, на которую указывает xasp, возвращается значение, меньшее 10 000. Зная значения геометрических размеров пиксела, можно так скорректировать параметры для функций вывода, чтобы получить на экране геометрически пропорциональные фигуры, например квадраты. Отметим, что не следует корректировать коэффициент сжатия для вывода окружностей, дуг или секторных диаграмм. Система графики делает корректировку автоматически.

void setaspectratio (int xasp, int yasp)

Устанавливает новое значение коэффициента сжатия, которое будет использоваться системой графики при выводе геометрических примитивов - прямоугольников, дуг, окружностей, эллипсов. Аргумент xasp отображает в условных единицах ширину пиксела на экране, yasp - высоту пиксела. Например, если известно, что высота пиксела на экране в 1.2 раза больше, чем его ширина, геомет-

рически корректный вывод будет получен при задании такого коэффициента сжатия:

setaspectratio(100,120);

Рекомендуемое использование функции - корректировка вывода графической информации при использовании нестандартных мониторов, для которых не может автоматически определить корректное значение коэффициента сжатия, а также корректировка графического вывода для мониторов с некорректной линейностью по вертикали и горизонтали.

Последние из параметров графической системы, влияющие на вывод графической информации, это маска заполнения и стиль заполнения. Маска заполнения позволяет задать способ заполнения отдельных областей экрана. Она определяется восьмибайтовым шаблоном, рассматриваемым как битовая карта 8x8. Заполняемая область также разбивается на блоки (знакоместа) по 8x8 пикселов. Маска "накладывается" на каждое такое знакоместо по следующему правилу: если соответствующий бит в маске заполнения равен 1, то пиксел в знакоместе имеет код текущего цвета; в противном случае пиксел остается неизменным. Для работы с масками заполнения система графики содержит функции getfillpattern() и setfillpattern().

```
void getfillpattern (char * pattern)
```

Заполняет область памяти из 8 байт, описанную точкой вызова, текущим значением маски заполнения. Аргумент pattern указывает на начало описанной области памяти. Маска заполнения может иметь одно из предопределенных значений или описываться пользователем.

```
void setfillpattern (char *upattern, int color)
```

Задает цвет пикселов и маску для заполнения областей экрана. По умолчанию используется белый цвет и маска заполнения, состоящая из матрицы единиц во всех битах. Таким образом, по умолчанию все пикселы заполняемой области имеют белый цвет. Аргумент upattern указывает на начало области из 8 байт, задающих новую маску заполнения. Первый байт задает пикселы самой верхней строки в пределах знакоместа. Старший бит первого байта соответствует самому левому пикселу знакоместа. Аргумент color задает цвет пикселов.

Для удобства пользователей библиотека графики содержит целую группу

предопределенных комбинаций символ/цвет заполнения областей экрана. Пару значений символов/цветов часто называют стилем заполнения (filling style). Для работы с предопределенными стилями используется пара функций getfillsettings() и setfillstyle().

```
void getfillsettings( struct fillsettingstype *fillinfo)
```

Заполняет поля структурной переменной по шаблону struct fillsettingstype информацией о текущей маске и цвете заполнения. Структурную переменную по шаблону struct fillsettingstype описывает точка вызова. Аргумент fillinfo указывает на описанную точкой вызова структурную переменную. Шаблон struct fillsettingstype определен в <graphics.h> так:

```
struct fillsettingstype
{
int pattern; /* идентификатор маски заполнения */
   int color;    /* цвет заполнения       */
}
```

Идентификатором предопределенных масок заполнения служит или целое число или символическая константа (табл. 3.7.).

В случае, когда используется определенная пользователем маска заполнения, поле pattern в структурной переменной, заполняемой функцией getfillsettings(), равно 12.

Табл. 3.7. Задание предопределенных масок

Символическая константа	Значение	Описание стиля заполнения
EMPTY FILL	0	Заполнение цветом фона
SOLID FILL	1	Заполнение текущим цветом
LINE FILL	2	Заполнение символами --, цвет - color
LTSLASH_FILL	3	Заполнение символами // нормальной толщины, цвет - color
SLASH_FILL	4	Заполнение символами // удвоенной толщины, цвет - color
BKSLASH_FILL	5	Заполнение символами \\ удвоенной толщины, цвет - color
LTBKSLASH_FILL	6	Заполнение символами \\ нормальной толщины, цвет - color
HATCH_FILL	7	Заполнение вертикально-горизонтальной штриховкой тонкими линиями, цвет-color
XHATCH_FILL	8	Заполнение штриховкой крест-накрест по диаго-

		нали "редкими" тонкими линиями, цвет - color
INTERLEAVE_FILL	9	Заполнение штриховкой крест-накрест по диагонали "частыми" тонкими линиями, цвет - color
WIDE DOT FILL	10	Заполнение "редкими" точками
Символическая константа	Значение	Описание стиля заполнения
CLOSE DOT FILL	11	Заполнение "частыми" точками
USER FILL	12	Заполнение по определенной пользователем маске заполнения, цвет - color

```
void setfillstyle(int pattern, int color)
```

Выбирает один из предопределенных стилей заполнения. Значение pattern идентифицирует стиль. Возможные значения для pattern приведены в табл. 3.7. Аргумент color задает цвет, используемый для пикселов по заданному шаблону. Данная функция не предназначена для установки определенной пользователем маски заполнения. Для этого используется функция setfillpattern().

3.8.2. Чтение-запись отдельных пикселов

Базовой функцией любой графической библиотеки является функция вывода в заданные координаты пиксела специфицированного цвета. C++ имеет в своем составе две функции манипуляции отдельными пикселами экрана: getpixel() - для определения кода цвета пиксела и putpixel () - для вывода пиксела текущим цветом.

```
unsigned getpixel( int x, int y)
```

Определяет, лежит ли пиксел с координатами (x, y) в текущем графическом окне, и, если лежит, возвращает код цвета этого пиксела. В противном случае возвращается 0.

```
void putpixel(int x, int y, int pixelcolor)
```

Определяет, лежит ли пиксел с координатами (x, y) в текущем графическом окне, и, если лежит, выводит на экран пиксел, код цвета которого равен pixelcolor. В противном случае цвет пиксела не изменяется.

Используя функцию putpixel(), можно "стереть" пиксел, если вывести его с кодом цвета фона.

Типичным применением точечного вывода является формирование сложных изображений, которые не могут быть представлены совокупностью графических примитивов: пиктограмм, фрагментов игрового поля и др. К сожалению, при работе с функцией putpixel() нельзя управлять режимом записи пиксела, т.е. функция putpixel() переопределяет предыдущее содержимое экрана. Это не всегда удобно, особенно в тех случаях, когда требуется добиться видимости изображения на любом фоне, другими словами, выполнить вывод пиксела, используя операцию исключающего ИЛИ с предыдущим содержимым.

3.8.3. Вывод отрезков прямых линий

Целая группа функций библиотеки графики предназначена для вывода отрезков прямых линий. Далее приводится спецификация этих функций. Напомним, что на вывод отрезков прямых линий влияют режим вывода линии и стиль линии.

Выводимые отрезки прямых линий не пересекают границ текущего окна, если при описании окна включен режим "усечения" (clipping).

```
void line( int x1, int y1, int x2, int y2)
```

Выводит отрезок прямой линии между двумя явно специфицированными точками (x1, y1) и (x2, y2), используя текущие цвет, стиль, толщину и режим вывода линии. Координаты (x1, y1) и (x2, y2) задаются относительно левого верхнего угла текущего графического окна. Функция не изменяет текущую позицию.

```
void linerel(int dx, int dy)
```

Выводит отрезок прямой линии между текущей позицией (начало отрезка) и точкой, заданной горизонтальным смещением dx и вертикальным смещением dy от текущей позиции (конец отрезка). При выводе отрезка прямой используются текущие цвет, стиль, толщина и режим вывода линии. После вывода линии функция устанавливает новую текущую позицию, равную координатам конца отрезка.

Установка необходимой текущей позиции может быть выполнена функциями moveto() и movrel().

```
void lineto( int x, int y)
```

Выводит отрезок прямой линии между текущей позицией (начало отрезка) и

точкой, заданной горизонтальной координатой x и вертикальной координатой y (конец отрезка). При выводе отрезка прямой используются текущие цвет, стиль, толщина и режим вывода линии. Координаты (x, y) задаются относительно левого верхнего угла текущего графического окна. После вывода линии функция устанавливает новую текущую позицию, равную координатам конца отрезка.

3.8.4. Вывод основных графических примитивов

Библиотека графики содержит функции для вывода дуги окружности или целой окружности, эллиптической дуги или целого эллипса, кругового сектора, ломаной линии из нескольких отрезков прямой (полигона), прямоугольника, прямоугольной полосы заданного цвета и стиля заполнения, прямоугольника заданной толщины в аксонометрии.

Все примитивы, за исключением полосы и дуги, могут быть выведены контуром или их внутреннее пространство может быть "залито" заданным цветом и заполнено по текущей маске. Для изображения используется линия текущего стиля и толщины, для заполнения - текущие установки стиля заполнения (цвет, маска). Далее приводится описание функций для вывода примитивов.

```
void arc(int x, int y, int stangle, int endangle, int radius)
```

Выводит дугу окружности радиусом radius. Центр окружности задают координаты x, y. Аргументы stangle и endangle задают соответственно начальный и конечный углы (рис. 3.1.) выводимой дуги. Углы задаются в градусах и отсчитываются против хода часовой стрелки. Положению часовой стрелки 3 часа соответствует угол 0 градусов, 12 часов - 90 градусов, 9 часов - 180 градусов, 6 часов - 270 градусов. При задании stangle равным 0 градусов и endangle равным 359 градусов выводится полная окружность. Для вывода дуги используется текущий цвет и только сплошная линия. Толщина линии может быть задана функцией settextstyle() (1 или 3 пиксела). Текущая позиция при выводе дуги не изменяется. Функция автоматически корректирует координаты точек в соответствии с коэффициентом сжатия дисплея.

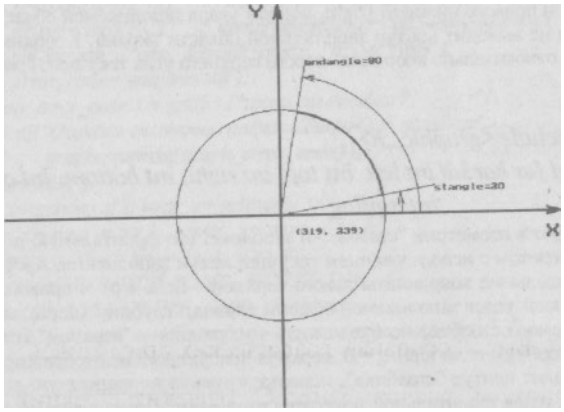


Рис. 3.1. Вывод окружностей и дуг

```
void bar(int left, int top, int right, int bottom)
```

Выводит полосу, заполненную текущим цветом с использованием текущей маски заполнения. Аргументы задают пиксельные координаты левого верхнего (left, top) и правого нижнего (right, bottom) углов заполняемой области экрана. Функция не выводит контур заполняемой области экрана. Координаты углов задаются относительно координат левого верхнего угла текущего графического окна.

```
void bar3d(int left, int top, int right, int bottom, int depth, int topflag)
```

Выводит в изометрии "столбик" и заполняет его фронтальную поверхность текущим цветом с использованием текущей маски заполнения. Аргументы задают: пиксельные координаты левого верхнего (left, top) и правого нижнего (right, bottom) углов заполняемой области экрана; "глубину "(depth) в пикселах изображаемого столбца; необходимость изображения "верхней" поверхности столбца (topflag): если topflag = 0, верхняя поверхность не отображается. Функция выводит контур "столбика", используя только непрерывную линию. Координаты углов фронтальной поверхности задаются относительно координат левого верхнего угла текущего графического окна.

```
void circle( int x, int y, int radius)
```

Выводит окружность заданного аргументом radius радиуса с центром, заданным координатами x и y. Координаты центра определяются относительно координат левого верхнего угла текущего графического окна. Для вывода окружности используется текущий цвет и только сплошная линия. Толщина линии (но не стиль!) может быть задана функцией settextstyle() (1 или 3 пиксела).

Хотя окружность может быть выведена и функцией arc(), использование circle() для этих целей предпочтительнее, так как для полной окружности эта функция более производительная.

Область экрана внутри окружности может быть заполнена функцией floodfill() по текущей маске с использованием текущего цвета.

```
void drawpoly(int numpoints, int polypoints[])
```

"Соединяет" отрезками прямых линий текущего цвета и стиля точки (полигон), координаты которых заданы парами значений. Эти пары расположены в массиве, на который указывает polypoints[] . Аргумент numpoints задает число соединяемых между собой точек. Координаты точек задаются относительно координат левого верхнего угла текущего графического окна. Текущая позиция не изменяется. Для получения замкнутых ломаных линий необходимо задать равными первую и последнюю точки выводимого полигона. Область экрана внутри полигона может быть заполнена с использованием текущего цвета и стиля заполнения функцией fillpoly().

```
void ellipse (int x, int y, int stangle, int endangle, int xradius, int yradius)
```

Выводит эллиптическую дугу или полный эллипс, используя текущий цвет. Аргументы задают (рис. 3.2): пиксельные координаты центра эллипса (x, y); начальный угол дуги (stangle); конечный угол дуги (endangle); радиус эллипса по горизонтали (xradius); радиус эллипса по вертикали (yradius). Функция выводит контур дуги или полный эллипс, используя непрерывную линию. Координаты центра задаются относительно координат левого верхнего угла текущего графического окна. Толщина линии (но не стиль!) может быть задана равной 1 или 3 пикселам функцией settextstyle(). Углы задаются в градусах и измеряются против хода часовой стрелки. Положению часовой стрелки 3 часа соответствует угол 0 градусов, 12 часов - 90 градусов, 9 часов - 180 градусов, 6 часов - 270 градусов. При задании stangle равным 0 градусов и endangle равным 359 градусов выводится полный эллипс. Текущая позиция при выводе дуги не изменяется. Функция автоматически корректирует координаты точек в соответствии с коэффициентом сжатия дисплея. Область экрана внутри эллипса может быть заполнена (функцией floodfill() или fillellipse() по текущей маске с использованием текущего цвета.

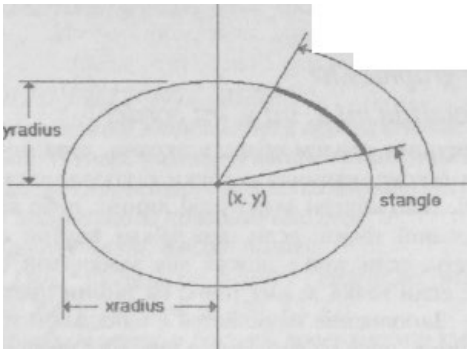


Рис. 3.2. Вывод эллиптических примитивов

```
void fillellipse(int x, int y, int xradius, int yradius)
```

Выводит эллипс, заполненный текущим стилем. Аргументы функции задают (см. рис. 3.2.): пиксельные координаты центра эллипса (x, y); радиус эллипса по горизонтали (xradius); радиус эллипса по вертикали (yradius). Функция выводит контур эллипса текущим цветом, устанавливаемым функцией setcolor(). Координаты центра задаются относительно координат левого верхнего угла текущего графического окна. Текущая позиция при выводе эллипса не изменяется. Функция автоматически корректирует координаты точек в соответствии с коэффициентом сжатия дисплея. Цвет и маска заполнения могут быть заданы с помощью функций setfillpattern() и setfillstyle().

При выполнении операции заполнения функция использует внутренний буфер библиотеки графики Turbo C для хранения промежуточных результатов. Если объем буфера будет недостаточным, функция установит в -6 внутренний код ошибки. Этот код доступен программе через обращение к функции graphresult().

```
void fillpoly(int numpoints, int *polypoints)
```

Выводит контур полигона, заданного numpoints точками. Координаты точек заданы парами, расположенными в массиве, на который ссылается polypoints. Функция соединяет первую и последнюю точки и заполняет область внутри полигона текущим стилем. Функция выводит контур полигона текущим цветом, устанавливаемым функцией setcolor(). Координаты точек задаются относительно координат левого верхнего угла текущего графического окна. Текущая позиция не изменяется. Цвет и маска заполнения могут быть заданы с помощью функций setfillpattern() и setfillstyle().

При выполнении операции заполнения функция использует внутренний буфер библиотеки графики для хранения промежуточных результатов. Если объем буфера будет недостаточным, функция установит в -6 внутренний код ошибки. Этот

код доступен программе через обращение к функции graphresult().

void floodfill (int x, int y, int border)

Заполняет текущим стилем область экрана, ограниченную непрерывной линией с цветом border, начиная с точки с координатами (x, y). Функция заполняет область либо внутри замкнутой линии, либо вне ее. Это зависит от положения начальной точки: если она лежит внутри области, заполняется внутренняя область; если точка лежит вне замкнутой области, заполняется внешняя область; если точка лежит точно на линии цвета border, заполнение не производится. Заполнение начинается с начальной точки и продолжается во всех направлениях, пока не встретится пиксел с цветом border. Цвет border должен отличаться от цвета заполнения, в противном случае будет заполнен весь экран. Цвет и маска заполнения могут быть заданы с помощью функций setfillpattern() и setfillstyle().

void pieslice(int x, int y, int stangle, int endangle, int radius)

Выводит контур кругового сектора и заполняет его внутреннюю область текущим стилем. Контур образован круговой дугой радиусом radius с координатами центра (x, y), проведенной, начиная от угла stangle до угла endangle, и радиусами, соединяющими центр с концевыми точками дуги. Дуга контура выводится текущим цветом, устанавливаемым функцией setcolor() всегда сплошной линией. Толщина (но не стиль!) дуговой линии равна 1 или 3 пикселам и задается функцией setlinestyle(). Стиль линии радиусов может быть любым и управляется функцией setlinestyle(). Особенностью рассматриваемой функции является то, что при задании любого другого стиля линии, отличного от сплошной линии (параметр linestyle в функции setlinestyle(), не равный 0), дуга сектора становится невидимой. Цвет и маска заполнения могут быть заданы с помощью функций setfillpattern() и setfillstyle().

При выполнении операции заполнения функция использует внутренний буфер библиотеки графики для хранения промежуточных результатов. Если объем буфера будет недостаточным, функция установит в -6 внутренний код ошибки. Этот код доступен программе через обращение к функции graphresult(). Углы stangle и endangle выводимой дуги задаются в градусах и измеряются против хода часовой стрелки. Положению часовой стрелки 3 часа соответствует угол 0 градусов, 12 часов - 90 градусов, 9 часов - 180 градусов, 6 часов - 270 градусов. При задании stangle равным 0 градусов и endangle равным 359 градусов выводится полная окружность.

void rectangle(int left, int top, int right, int bottom)

Выводит контур прямоугольника, заданного координатами левого верхнего (left, top) и правого нижнего (right, bottom) углов. Координаты углов задаются относительно координат левого верхнего угла текущего графического окна. Контур выводится линией текущего цвета и стиля. Цвет контура может быть установлен функцией setcolor(). Стиль линии может быть выбран или задан функцией setlinestyle().

void sector(int x, int y, int stangle, int endangle, int xradius, int yradius)

Работает аналогично функции pieslice(), за исключением того, что выводится не круговая, а эллиптическая дуга. Аргумент xradius задает радиус эллипса по горизонтали, а yradius - радиус эллипса по вертикали. При выводе сектора учитывается коэффициент сжатия, и эллиптическая дуга на экране геометрически корректна.

Перечисленными функциями исчерпывается список функций для вывода основных графических примитивов. Дополнительные графические примитивы могут быть построены из стандартных средств C++.

3.9. Предварительная подготовка к работе

1. Ознакомиться с организацией и функциональными возможностями различных типов видеосистем.
2. Ознакомиться с графическим режимом отображения информации на экран монитора и стандартными библиотечными функциями C, обслуживающими этот режим.

3.10. Порядок выполнения работы

1. Разработать программу для вывода на экран графика заданной функции.

Номер	Функция	Диапазон аргумента	
		Начало	Конец
1	$Sin^2(x/2)+Sqrt(x)$	$3\pi/2$	15π
2	$Sin^3(x/2)+Sqrt(x)$	$3\pi/2$	16π
3	$Sin^2(x/4)+Sqrt(x)$	$3\pi/2$	17π
4	$Cos^2(x/2)+Sqrt(x)$	$3\pi/2$	18π

Номер	Функция	Диапазон аргумента	
		Начало	Конец
5	$Cos^3(x/2)+Sqrt(x)$	3π/2	15π
6	$Cos^2(x/4)+Sqrt(x)$	3π/2	16π
7	$Sin^2(x)- Cos^2(x)$	3π/2	7π
8	$Sin^3(x)+ Cos^2(x)$	3π/2	8π
9	$Sin^2(x)+ Cos^3(x)$	π/2	5π
10	$Sin^3(x)+ Cos^3(x)$	π/2	6π
11	$Sin^2(x)- Cos^2(x)$	π/2	7π
12	$Sin^3(x)- Cos^2(x)$	π/2	8π
13	$Sin^2(x)- Cos^3(x)$	π/2	5π
14	$Sin^3(x)- Cos^3(x)$	π/2	6π
15	$Sin^2(x/2)-Sqrt(x)$	π/2	13π
16	$Sin^3(x/2)-Sqrt(x)$	π/2	12π
17	$Sin^2(x/4)-Sqrt(x)$	π	11π
18	$Cos^2(x/2)-Sqrt(x)$	π	10π
19	$Cos^3(x/2)-Sqrt(x)$	π	9π
20	$Cos^2(x/4)-Sqrt(x)$	π	8π

График должен быть отрисован в указанном диапазоне аргумента.

2. Произвести разметку осей и проставить истинные значения точек.

3. Найти максимальное значение функции на заданном интервале и вывести в отдельное окно на экране вместе с графиком.

3.11. Содержание отчета

1. Краткие сведения о видеосистемах ПЭВМ, графическом режиме их работы и функциях обслуживания графического режима.

2. Задание на лабораторную работу;

3. Алгоритмы и тексты отлаженных программ.

4. Пример запуска программы

5. Выводы.

3.12. Контрольные вопросы

1. Зачем нужен графический режим?

2. Почему в видеосистеме используют и текстовый, и графический режимы?

3. Можно ли обойтись только графическим режимом? Если да, то какие характеристики должна при этом иметь ПЗВМ?

4. Как влияет размер видеопамяти на характеристики графического режима?

5. Зачем нужен видеоадаптер в графическом режиме?

6. Назовите основные характеристики графического режима, чем они обусловлены?

7. Существуют ли окна в графическом режиме? Зачем они нужны?

8. Какие функции инициализации графического режима Вы знаете?

9. Какие функции обслуживания графических окон Вы знаете?

10. Есть ли курсор в графическом режиме? Если да, то как можно им управлять?

11. Что такое пиксел? Зачем нужен атрибут пикселя?

12. Какие функции работы с пикселями Вы знаете?

13. Сколько цветов фона и символов можно одновременно использовать в графическом режиме и почему?

14. Какие функции установки цветов Вы знаете?

15. Что называется графическим примитивом и какие функции обслуживания графических примитивов Вы знаете?

Лабораторная работа № 4.

КЛАВИАТУРА IBM PC. ИСПОЛЬЗОВАНИЕ ПРЕРЫВАНИЙ

Цель работы: изучение возможностей работы с клавиатурой, ознакомление со стандартными средствами библиотеки C++ и средствами системы прерываний DOS и BIOS, обслуживающими клавиатуру.

4.1. Общие положения

Подавляющее большинство программ выполняют ввод информации с клавиатуры. Ввод информации в компьютер может быть выполнен на трех уровнях: обращением к функциям MS-DOS; обращением к функциям BIOS; физическим доступом к аппаратным средствам.

Ввод информации на уровне MS-DOS позволяет "пропустить" клавиатурный ввод через инсталлируемые драйверы, обеспечивает отслеживание нажатия комбинации клавиш Ctrl-C (Ctrl-Break), стандартную для MS-DOS обработку ошибок.

Доступ к клавиатуре на уровне BIOS позволяет программе отслеживать нажатие всех, а не только символьных клавиш, выполнять управление аппаратурой клавиатуры и пр. Интерфейсом Turbo C с BIOS является функция bioskey().

Непосредственный доступ к буферу клавиатуры резко повышает производительность программы. В некоторых случаях необходима имитация нажатий клавиш клавиатуры с записью кодов непосредственно в буфер. При этом физически

нажатия клавиш не происходят. Так строятся многие демонстрационные программы, которые открывают или закрывают окна меню, выполняют необходимый выбор, показывают работу программы в "автоматическом" режиме и т.п. На том же самом принципе имитации нажатий клавиш построены программы, способные переносить одним нажатием клавиши целые куски текста из одной программы в любой текстовый редактор. Примером такой программы является входящая в Turbo C резидентная Help-система THELP.COM.

4.2. Аппаратные и программные средства ввода информации с клавиатуры

4.2.1. Аппаратные средства персонального компьютера для ввода информации с клавиатуры

Клавиатура персонального компьютера содержит специальный встроенный микропроцессор. Он при каждом нажатии и отпускании клавиши определяет ее порядковый номер и помещает его в порт 60h специальной электронной схемы - программируемого периферийного интерфейса (ППИ). Далее этот код будем называть скэн-кодом. Скэн-код в первых 7 битах содержит порядковый номер нажатой клавиши, а восьмой бит равен 0, если клавиша была нажата (прямой скэн-код), и равен 1, если клавиша была отпущена (обратный скэн-код). Когда скэн-код записан в порт 60h, схема ППИ выдает сигнал "подтверждения", уведомляя микропроцессор клавиатуры о принятии кода.

Если клавиша остается нажатой дольше некоторого времени задержки (delay value), микропроцессор клавиатуры начинает генерировать с заданной частотой (typematic rate) прямой скэн-код нажатой клавиши. Значения задержки и частоты повторения могут устанавливаться в нужные значения либо через порты клавиатуры, либо через функцию AH = 03h прерывания 16h BIOS. Когда скэн-код принят схемой ППИ, аппаратура компьютера генерирует аппаратное прерывание с номером 9.

Стандартный обработчик прерывания 9 - это программа, входящая в состав BIOS (BIOS ISR). BIOS ISR анализирует скэн-код и по специальным правилам преобразует его. Отметим, что по скэн-коду всегда можно установить, вследствие чего ISR получила управление: из-за нажатия или из-за отпускания клавиши.

4.2.2. Анализ и преобразование скэн-кода

Действия BIOS ISR при нажатии и отпускании одной и той же клавиши различны. Клавиши в зависимости от алгоритма обработки их скэн-кода можно раз-

делить на:

1. шифт-клавиши (Right-Shift, Left-Shift, Alt, Ctrl);
2. триггерные клавиши (NumLock, ScrollLock, CapsLock);
3. клавиши с буферизацией расширенного кода;
4. специальные клавиши (клавиша PrnScr, комбинация Alt-Ctrl-Del, комбинация Ctrl-C (Ctrl-Break)).

За каждой шифт- или триггерной клавишей закреплен свой бит в ячейках памяти по адресам 40: 17h и 40: 18h (табл. 4.1). При каждом нажатии или отпускании шифт-клавиши ISR BIOS инвертирует соответствующий бит. Таким образом, текущее состояние бита шифт-клавиши говорит о том, нажата она в данный момент или отпущена. За триггерными клавишами закреплены два бита: один из них инвертируется только при нажатии клавиши ("фиксирует" состояние "Вкл/Выкл"), другой - при нажатии и отпускании, отслеживая текущее состояние клавиши.

Текущее состояние шифт- и тригтерных клавиш используется BIOS-обработчиком прерывания от клавиатуры при определении правил преобразования скэн-кодов от других клавиш. Большинство клавиш и их комбинаций с шифт-клавишами - это клавиши с буферизацией расширенного кода: при их нажатии в специальный буфер памяти помещается двухбайтовый код, называемый BIOS-кодом клавиши. Младший байт этого кода равен ASCII-коду символа, либо нулю. Старший байт равен скэн-коду клавиатуры, либо так называемому расширенному скэн-коду. Комбинация "ASCII-код/ скэн-код клавиатуры" генерируется в следующих случаях:

- если нажата клавиша клавиатуры, помеченная символом, входящим в ASCII-таблицу (называемая далее ASCII-клавишей). Так как прописные и строчные буквы имеют разный ASCII-код, при генерации BIOS-кода учитывается текущее состояние триггерной клавиши CapsLock и клавиши Shift.

- если нажаты некоторые из ASCII-клавиш в комбинации с нажатой и не отпущенной клавишей Ctrl, а также при нажатии клавиш Backspace, ENTER (Ввод), Tab и Esc (Ключ). В этом случае младший байт BIOS-кода клавиши равен одному из управляющих ASCII-кодов. Это ASCII-коды со значениями 00 - 31, которые не входят в число печатаемых символов, а используются для управления периферийными устройствами. Например, нажатие клавиши ENTER порождает управляющий символ Carriage Return (Возврат каретки), нажатие клавиши Tab порождает

управляющий символ горизонтальной табуляции, комбинация Ctrl-L - управляющий символ Form Feed (Перевод формы), комбинация Ctrl-B - управляющий символ Bell (Звонок). Нажатие комбинации Ctrl-M соответствует также управляющему символу Carriage Return, но полный BIOS-код этой клавиши равен 13/50, а в случае нажатия клавиши ENTER - 13/28.

Табл. 4.1. Состояние шифт- и триггерных клавиш

Бит	Состояние шифт- и триггерных клавиш
	Байт 40:17h
0	Нажата и не отпущена клавиша Right Shift
1	Нажата и не отпущена клавиша Left Shift
2	Нажата и не отпущена клавиша Ctrl
3	Нажата и не отпущена клавиша Alt
4	Зафиксирован скроллинг экрана (ScrollLock - включен)
5	Включена цифровая клавиатура (NumLock - включен)
6	Зафиксирован верхний регистр (CapsLock - включен)
7	Включен режим вставки (хотя клавиша Ins не является триггерной, BIOS-обработчик фиксирует каждое ее нажатие, а код клавиши помещается еще и в буфер клавиатуры)
	Байт40:18h
0	Нажата и не отпущена клавиша Left Ctrl
1	Нажата и не отпущена клавиша Left Alt
2	Нажата клавиша System Request (System)
3	Включен режим Pause(Ctrl-NumLock)
4	Нажата и не отпущена клавиша ScrollLock
5	Нажата и не отпущена клавиша NumLock
6	Нажата и не отпущена клавиша CapsLock
7	Нажата и не отпущена клавиша Ins

Двухбайтовый BIOS-код вида “0 / расширенный скэн-код” генерируется и записывается в буфер клавиатуры при нажатии клавиш Fl - F12, Ins, Del, клавиш управления курсором Home, Up, PgUp, Left, Right, End, Down, PgDn и их комбинации с клавишами Alt, Ctrl, Shift, а также при нажатии комбинации Alt-ASCII-клавиша. Значение расширенного скэн-кода определяется технической документацией BIOS. Правила BIOS таковы, что расширенный скэн-код и скэн-код от клавиатуры не совпадают.

Некоторые нажатия клавиш обрабатываются ISR BIOS особым образом. К их числу относятся:

1) клавиша PrnScr, при нажатии которой ISR BIOS выполняет программное прерывание 5;

2) комбинация Alt-Ctrl-Dei, обнаружив такую комбинацию, ISR BIOS передает управление программе начальной загрузки. Эта программа также входит в состав BIOS;

3) комбинация Ctrl-C (Ctrl-Break); ISR BIOS записывает по абсолютному адресу памяти 00471h значение 80h. Оно используется как флаг, сигнализирующий о желании пользователя остановить выполнение текущей программы. Значение этого флага проверяют при своем выполнении функции MS-DOS, работающие с файлами stdin, stdout, stdprn и stdaux.

Особым образом обрабатывается так называемый Alt-ввод. Если нажимается и удерживается нажатой клавиша Alt и на цифровой клавиатуре набираются цифры, то после отпускания клавиши Alt в буфер клавиатуры помещается двухбайтовый код, старший байт которого равен нулю, а младший байт содержит набранный цифрами код. Если набранный код больше 256, младший байт равен остатку от деления набранного кода на 256. Большинство прикладных программ обрабатывают ASCII-коды, сгенерированные простым нажатием клавиши и Alt-вводом одинаково, несмотря на различные старшие байты двухбайтового кода в буфере клавиатуры. Например, при нажатии клавиши 'Z' в буфер клавиатуры записывается "ASCII-код/скэн-код": 90/44, а при нажатии Alt-90 - "ASCII-код/0": 90/0. Alt-ввод очень удобен для ввода ASCII-символов, не имеющих соответствующих клавиш, например символов псевдографики.

4.2.3. Буфер клавиатуры

Буфер BIOS для записи кодов клавиш занимает 32 байта оперативной памяти с адреса 40:lEh до 40:3Eh. Запись информации в буфер выполняет ISR BIOS прерывания 9, чтение - функции ISR BIOS прерывания 16h. Буфер клавиатуры рассчитан на 15 нажатий клавиш, генерирующих двухбайтовые коды и поэтому имеет 30 байт для кодов клавиш и еще два дополнительных байта, которые резервируются под двухбайтовый код для клавиши ENTER.

Буфер организуется как кольцевая очередь, доступ к которой осуществляется с помощью указателя «головы» (head pointer), адрес которого 40:1Ah, и указателя «хвоста» (tail pointer), адрес которого 40:1Ch. Указатель "хвоста" задает смещение до слова, где будет записан обработчиком прерывания 9 код буферизуемой клавиши, т.е. первое свободное слово буфера. Указатель "головы" задает смещение

слова, которое будет возвращено запросу буферизованного ввода с клавиатуры, сделанного операционной системой или BIOSом.

При каждом нажатии клавиши, для которой генерируется двухбайтовый код, ISR BIOS прерывания 9,'используя текущее значение указателя "хвоста", записывает в память образованный двухбайтовый код. После этого указатель "хвоста" увеличивается на 2. Если указатель "хвоста" перед доступом к буферу указывает на верхнюю границу буфера (на слово 40:3Eh), указатель после записи в буфер "перепрыгивает" на начало буфера, т.е. ему присваивается значение 40:1Eh. Поэтому значение указателя "хвоста" может быть и меньше значения указателя "головы". Это значит, что указатель "хвоста" "перескочил" назад к нижней границе буфера. Когда указатель "хвоста" догонит указатель "головы", наступит переполнение буфера. В этом случае указатель "хвоста" задает смещение до "холостой" позиции. Каждое новое нажатие клавиши игнорируется BIOS-обработчиком; код клавиши не помещается в буфер, и звучит сигнал динамика.

Указатель "головы" используется BIOS-обработчиком прерывания 16h, которое вызывается непосредственно из прикладной программы или функциями MS-DOS ввода с клавиатуры. Если выполняется чтение буфера с разрушением информации (функция AH = 0 прерывания 16h, ISR читает два байта памяти по адресу, смещение которого задает указатель "головы", а сегмент равен 40h. Затем ISR прерывания 16h увеличивает (продвигает) указатель "головы" на 2. Если значения указателей "головы" и "хвоста" равны между собой, буфер пуст. В этом случае ISR прерывания 16 выполняет бесконечный цикл ожидания, условием выхода из которого будет неравенство указателей. При нажатии клавиши, генерирующей двухбайтовый код, этот цикл будет прерван BIOS-обработчиком прерывания 9. В результате указатель "головы" продвинется на 2. После завершения ISR BIOS прерывания 9 возобновится выполнение ISR BIOS прерывания 16h. Так как буфер уже не пуст, произойдет выход из цикла и передача в точку вызова прочитанного в буфере двухбайтового кода. Если выполняется чтение буфера без разрушения информации (функция AH = 1 прерывания 16h), продвижение указателя "головы" не происходит, но прочитанный по этому указателю код передается в точку вызова, если только буфер не пуст. Если буфер пуст, ISR прерывания 16h не выполняет цикл ожидания. Вместо этого флаг переноса CF устанавливается в 1, и обработчик завершает свою работу.

На рис. 4.1. приведены примеры пустого буфера клавиатуры и буфера после ввода с клавиатуры строки "TEST", нажатия клавиш Left и F1 при условии, что текущая программа не выполняла в этот момент ввод с клавиатуры.

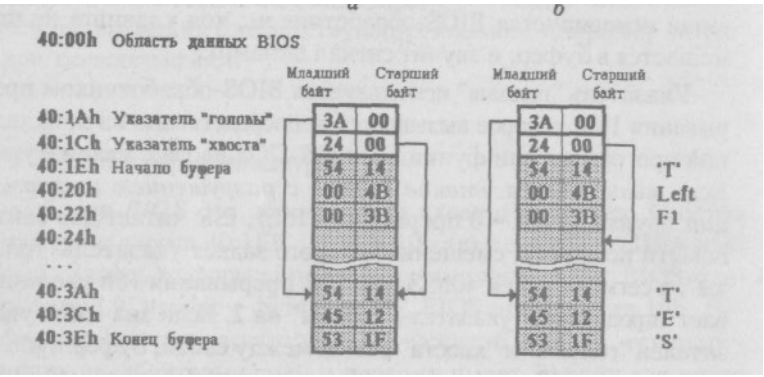


Рис. 4.1. Организация буфера клавиатуры: а - пустой буфер (значения указателей "готовы" и "хвоста" равны); б- тот же буфер после набора на клавиатуре строки TEST, нажатия специальных клавиш Left и F1 (текущая программа не выполняет ввод информации и указатель "головы" не продвигается)

Буфер клавиатуры - это классический пример использования кольцевого буфера для организации асинхронного взаимодействия двух программ по схеме "производитель-потребитель". Одна из программ (ISR BIOS прерывания 9) "производит" информацию или, как говорят, является процессом-производителем. Исполняемая программа через функцию AH= 00h прерывания 16h BIOS "потребляет" информацию или является процессом-потребителем. Асинхронность взаимодействия означает, что запись в буфер новой информации и чтение из него происходят в случайные, не связанные между собой моменты времени. Так как производитель постоянно анализирует наличие переполнения буфера, не происходит переопределения не прочитанных еще потребителем кодов клавиатуры. Другими словами, при переполнении буфера производитель блокируется до тех пор, пока потребитель не прочитает одно или несколько слов из буфера. Если же буфер пуст и выполняется попытка чтения информации, функция AH = 00h прерывания 16h BIOS переходит к бесконечному циклу, условием которого является неравенство между собой указателей "головы" и "хвоста". Фактически текущая программа, выполняющая ввод с клавиатуры, блокируется, не давая "потребить" несуществующую еще информацию.

4.3. Ввод информации с клавиатуры средствами MS-DOS

4.3.1. Функции прерывания 21h MS-DOS для ввода информации с клавиатуры

MS-DOS имеет целую группу функций прерывания 21h для выполнения ввода

информации с клавиатуры. Последовательность действий системы при вводе с клавиатуры такова. Функция MS-DOS вызывает драйвер клавиатуры, передавая ему запрос на ввод одного символа из буфера клавиатуры. Драйвер, выполняя запрос, обращается к нужной функции прерывания 16h BIOS. ISR BIOS прерывания 16h читает из буфера клавиатуры нужное слово и передает в драйвер. Драйвер возвращает байт (обычно младший) в MS-DOS. Таким образом, функции MS-DOS и опирающиеся на них функции библиотеки Turbo C слабо зависят от особенностей аппаратуры, поскольку система от нее изолирована двумя слоями программного обеспечения - драйверами и BIOSом.

Далее приводится характеристика функций MS-DOS, используемых для ввода с клавиатуры.

AH=01h - ввод с ожиданием со стандартного устройства ввода (клавиатуры). Выполняется "эхо" на экран вводимых символов. ASCII-код прочитанного символа помещается в AL. Если нажимается специальная клавиша, в AL возвращается 0, а второе обращение к функции возвращает расширенный скэн-код клавиши.

AH=06h - ввод-вывод с консоли. Если DL = FFh, выполняется ввод со стандартного устройства ввода без ожидания. Если буфер пуст, функция сообщает об этом установленным в 1 флагом нуля (ZF). В противном случае в регистре AL возвращается ASCII-код прочитанного символа.

AH=07h - ввод с консоли с ожиданием без "эха" на экран. ASCII-код прочитанного символа возвращается в AL. Если нажимается специальная клавиша, передаваемое в AL значение равно нулю, а второе обращение к функции возвращает расширенный скэн-код клавиши. Функция не выполняет "фильтрацию" ввода с клавиатуры. Это значит, что нажатие клавиши Backspace не стирает символ на экране, а только сдвигает курсор. Нажатие ENTER не переводит строку, а только перемещает курсор на начало строки.

AH=08h - подобна AH=07h, за исключением того, что если обнаруживается нажатие комбинации клавиш Ctrl-Break, вызывается прерывание 23h.

AH=0Bh - проверка состояния стандартного ввода. Возвращает в регистре AL значение FFh, если буфер клавиатуры не пуст, и 0 в противном случае. Функцию следует использовать перед выполнением функций AH=01h, 07h и 08h для того, чтобы избежать ожидания ввода, если он отсутствует. Кроме того, функция используется как средство проверки того, нажата ли комбинация клавиш Ctrl-Break, если программа долгое время выполняет работу, не связанную с обращением к функциям MS-DOS. Периодическое выполнение функции позволяет аварийно завершить программу, например, в случае ее зацикливания.

AH=0Ch - ввод с клавиатуры с очисткой буфера. Значение в регистре AL со-

держит номер выполняемой функции: 01, 06, 07, 08 или 0Ah. Поведение функции и возвращаемые значения описаны ранее в спецификации функций AH=01, 06, 07,08 или 0Ah.

Рассмотренные функции MS-DOS для ввода с клавиатуры могут вызываться напрямую из программы через функции getinterrupt(), int86(), intr() и т.п., либо неявно другими функциями ввода.

4.3.2. Функции библиотеки C++

int getch (void)

Выполняет ввод с клавиатуры через функцию MS-DOS AH=07h. Она не выполняет "эхо" вывода на экран. В этой связи полезна для организации интерфейса с пользователем, при котором нажатие той или иной клавиши вызывает немедленную реакцию программы без отображения введенного символа на экране.

int getche (void)

Выполняет небуферизуемый ввод с клавиатуры через функцию MS-DOS AH=07h, но в отличие от предыдущей функции обеспечивает вывод введенного символа на экран. Перевод строки происходит при достижении правой вертикальной границы текущего активного окна.

char *getpass(char * prompt)

Выводит на экран ASCII-строку, на начало которой указывает prompt, а затем принимает с клавиатуры без "эха" строку символов. Вводимые символы (не более 7) помещаются во внутреннюю статическую память. Функция возвращает указатель на внутреннюю статическую строку, переопределяемую каждым новым обращением к функции. Основное назначение данной функции - ввод паролей в программе без отображения их на экран.

int kbhit (void)

Проверяет, пуст ли буфер клавиатуры. Если в буфере есть символы, функция возвращает ненулевое значение, в противном случае она возвращает 0. Использует функцию 0Bh MS-DOS. Является удобным средством предотвращения "зацикливания" при ожидании невозможного в данный момент события. Кроме того,

при выполнении функции 0Bh осуществляется проверка нажатия комбинации клавиш Ctrl-Break, что позволяет выполнить аварийное завершение программы.

4.4. Ввод информации с клавиатуры средствами BIOS

Интерфейсом программ в персональном компьютере с клавиатурой является прерывание 16h BIOS. Далее приводится описание его функций.

AH = 00h - чтение с ожиданием двухбайтового кода из буфера клавиатуры. Прочитанный код возвращается в регистре AX: младший байт - в регистре AL, старший - в AH. Если нажата ASCII-клавиша, в AL помещается ASCII-код символа, в AH - скэн-код. При нажатии специальных клавиш AL равен 0, а в AH возвращается расширенный скэн-код.

AH = 0lh - чтение без ожидания двухбайтового кода из буфера клавиатуры. Если буфер пуст, в 1 выставляется флаг нуля ZF. В противном случае в AX возвращается двухбайтовый код из буфера клавиатуры, но продвижение указателя "головы" буфера не производится, т.е. код "остается" в буфере.

AH = 02h - определение состояния шифт- и триггерных клавиш. В регистре AL возвращается содержимое байта по адресу 40:17h (см. табл. 4.1).

Функция AH = 05h не имеет аналогов в библиотеке Turbo C и может использоваться для имитации нажатии клавиш в демонстрационных программах, программах переноса текста и т.д.

Функции AH = 10 - 12h являются аналогами функций 00 - 02h, но предназначены для использования в компьютерах с клавиатурой 101 /102 клавиши.

Функции AH = 00 - 02h прерывания 16h BIOS положены в основу функции **bioskey()** библиотеки Turbo C. Далее следует описание этой функции.

```
int bioskey(int cmd)
```

Обращается в зависимости от значения в cmd к функциям AH = 00 - 02h прерывания 16h. Возвращаемое функцией значение повторяет значение регистра AX при выходе из прерывания.

Функции для работы с вводом/выводом:

getch (void)	Считать клавишу без «эха»
getche (void)	Считать клавишу
kbhit (void)	Проверка, пуст ли буфер
bioskey(int cmd)	Считывание через прерывания

4.5. Предварительная подготовка к работе

1.Ознакомиться с аппаратными средствами компьютера для ввода информации с клавиатуры.

2. Ознакомиться с программными средствами для ввода информации с клавиатуры.

4.6. Порядок выполнения работы

1. Разработать, написать и отладить программу управления перемещением символа (например, "*") в пределах заданного на экране окна. Для управления использовать клавиши из набора: "стрелка вверх" (СтВВ), "стрелка вниз" (СтВН), "стрелка вправо" (СтВП), "стрелка влево" (СтВЛ) или функциональные клавиши Fl - F12 (варианты см. в таблице 4.2). **Для ввода использовать стандартные функции языка C++ (**getch, getche, kbhit, bioskey). Сохранить отлаженную программу.

2. Изменить программу, **заменив стандартные функции библиотеки C++(**getch, getche, kbhit, bioskey) **своими.** Для написания функций используйте заданное прерывание INT 21h или INT 16h (см. таблицу). Если его возможностей не достаточно, то замените его по своему усмотрению. Сохраните отлаженную программу.

3. Две отлаженные программы предъявить преподавателю.

Табл.4.2.

№ варианта	X1	Y1	X2	Y2	Вид движения	Клавиши управления	Номер прерывания
1	10	5	70	15	Постоянное	СтВВ, СтВН	INT 21h
2	15	5	65	15	Пошаговое	СтВП, СтВЛ	INT 21h
3	20	5	60	15	Постоянное	F1, F2	INT 21h
4	25	5	55	15	Пошаговое	Все направления	INT 21h
5	10	8	70	18	Постоянное	F5, F6	INT 16h
6	15	8	65	18	Пошаговое	F1-F4	INT 16h
7	20	8	60	18	Постоянное	F9-F12	INT 16h
8	25	8	55	18	Пошаговое	СтВВ, СтВН	INT 16h
9	10	10	70	20	Постоянное	СтВП, СтВЛ	INT 21h
10	15	10	65	20	Пошаговое	Все направления	INT 21h
11	20	10	60	20	Постоянное	F3, F4	INT 21h
12	25	10	55	20	Пошаговое	F7, F8	INT 21h
13	10	12	70	23	Постоянное	СтВВ, СтВН	INT 16h

14	15	12	65	23	Пошаговое	СтВП, СтВЛ	INT 16h
15	20	12	60	23	Постоянное	F9, F10	INT 16h
16	25	12	55	23	Пошаговое	F11, F12	INT 16h
17	10	15	70	20	Постоянное	СтВВ, СтВН	INT 21h
18	15	15	65	20	Пошаговое	СтВП, СтВЛ	INT 21h
19	20	15	60	20	Постоянное	F5, F10	INT 21h
20	25	15	55	20	Пошаговое	F6, F12	INT 21h

4.7. Содержание отчета

1. Краткие сведения о подсистеме ввода информации с клавиатуры, используемых прерываниях, буфере клавиатуры и функциях обслуживания ввода с клавиатуры.
2. Задание на лабораторную работу;
3. Алгоритмы и тексты отлаженных программ.
4. Пример запуска программы
5. Выводы.

4.8. Контрольные вопросы

1. Что относится к устройствам ввода информации в ЭВМ?
2. Как можно классифицировать устройства ввода?
3. Назовите основные характеристики устройств ввода информации.
4. Зачем нужен буфер клавиатуры?
5. Почему существует ввод с буферизацией и без нее?
6. Какие бывают прерывания?
7. Зачем для ввода данных с клавиатуры используют прерывания?
8. Какое прерывание вырабатывается при нажатии клавиши?
9. Назовите основные характеристики системы прерываний.
10. Почему нужны программные прерывания?
11. Почему для организации ввода с клавиатуры используются два программных прерывания INT 21h и INT 16h?
12. Какие функции библиотеки C++ для ввода с клавиатуры Вы знаете?
13. Какие функции прерывания INT 16h Вы знаете?
14. Какие функции прерывания INT 21h Вы знаете?
15. Можно ли в прикладной программе обойтись без ввода с клавиатуры?

Лабораторная работа № 5.

ИСПОЛЬЗОВАНИЕ АППАРАТНЫХ ПРЕРЫВАНИЙ

Цель работы – знакомство с различного вида аппаратными прерываниями и создание собственных подпрограмм обработки прерываний.

5.1. Общие положения

Микропроцессоры 8086/88 поддерживают механизм прерываний. В самом общем виде это наличие в аппаратуре специальных средств, с помощью которых выполнение текущей программы приостанавливается и процессор переходит к так называемой программе обслуживания прерывания (Interrupt Servise Routine - ISR). Механизм прерываний позволяет организовать выполнение тех или иных функций ядра и быструю реакцию процессора на возникновение каких-то внешних событий: ошибок в арифметических операциях, изменению состояния периферийных устройств и пр.

Микропроцессоры 8086/88 поддерживают 256 прерываний. Каждое из них имеет свой номер и ISR. Адрес точки входа в ISR называется вектором прерывания и хранится в специальной таблице, называемой таблицей векторов прерывания (ТВП). Код ISR может располагаться в любом месте памяти. Поэтому вектор прерывания занимает 4 байта: 2 байта отводится на значение сегментного регистра, устанавливаемое в CS (старшее слово), 2 байта - на значение смещения, устанавливаемое в IP (младшее слово). Вся ТВП занимает 256 * 4 = 1024 байт и располагается в оперативной памяти, начиная с адреса 0000:0000.

При возникновении прерывания процессор помешает в стек 6 байт: текущее значение CS, текущее значение IP (пара этих регистров определяет точку, с которой выполнение прерываемой программы возобновится), а также 2 байта флагов процессора. В CS и IP устанавливаются значения из ТВП, которые задают адрес начала ISR. Прерыванию 0 соответствует вектор прерывания по адресу 0000:0000, прерыванию 1 - по адресу 0000:0004h, прерыванию 2 - по адресу 0000:0008h и т.д.

Сама ISR - это программа, построенная с соблюдением специальных правил:

1)в самом начале она сохраняет в стеке все регистры процессора, которые будут использоваться в этой программе;

2) перед завершением работы программы значения регистров восстанавливаются;

3) последней инструкцией ISR, как правило, является инструкция возврата из прерывания IRET. Выполняя IRET, процессор извлекает из стека шесть слов информации, которые последовательно помещает в регистры IP, CS и регистр флагов, возвращаясь к исполнению прерванной программы.

Часто обработчикам программных прерываний требуется передать какие-то значения, задающие конкретное действие, характеристики ситуации и т.п., и получить какие-то результаты по завершению исполнения ISR. Для такого обмена данными используются внутренние регистры процессора.

Некоторые векторы прерывания в ТВП на самом деле задают не точки входа в ISR, а используются для хранения важной системной информации: адресов данных и таблиц. Кроме того, за некоторые векторы "зацеплены" ISR, не выполняющие никаких действий. Они служат заглушками для подключения дополнительных обработчиков. Так, например, в нормальном состоянии обработчик прерывания 1Ch не выполняет никаких действий и содержит единственную инструкцию возврата из прерывания IRET. Прерывание 1Ch вызывается из пределов ISR таймера (обработчик прерывания 8). Прерывание от таймера, в свою очередь, генерируется 18.2 раза в секунду аппаратурой системного таймера. Есть и другие обработчики - заглушки, вызываемые при функционировании ISR BIOS и MS-DOS.

5.2. Аппаратные прерывания

В процессе функционирования персонального компьютера могут встретиться четыре типа прерываний:

1) аппаратные;
2) программные;
3) исключительные ситуации процессора (processor exceptions);
4) немаскируемые.

Аппаратные прерывания возникают как результат некоторых внешних событий и в их генерации принимает участие специальная микросхема персонального компьютера - программируемый контроллер прерываний, или PIC (Programmable Interrupt Controller). Наиболее часто для этих целей используется одна или несколько микросхем 8259A либо их функциональные эквиваленты. В архитектуре компьютеры IBM PC AT используют PIC, построенный на двух микросхемах 8259A (рис. 5.1).

Микросхема 8259A рассчитана на 8 входов запросов прерываний, обозначаемых IRQ (Interrupt Request). Сигналы на них возбуждают внешние устройства: адаптеры асинхронной последовательной и параллельной связи, плата системного таймера и др. Контроллер прерываний имеет в своем составе ряд программируемых внутренних регистров, определяющих особенности обработки запросов прерываний.

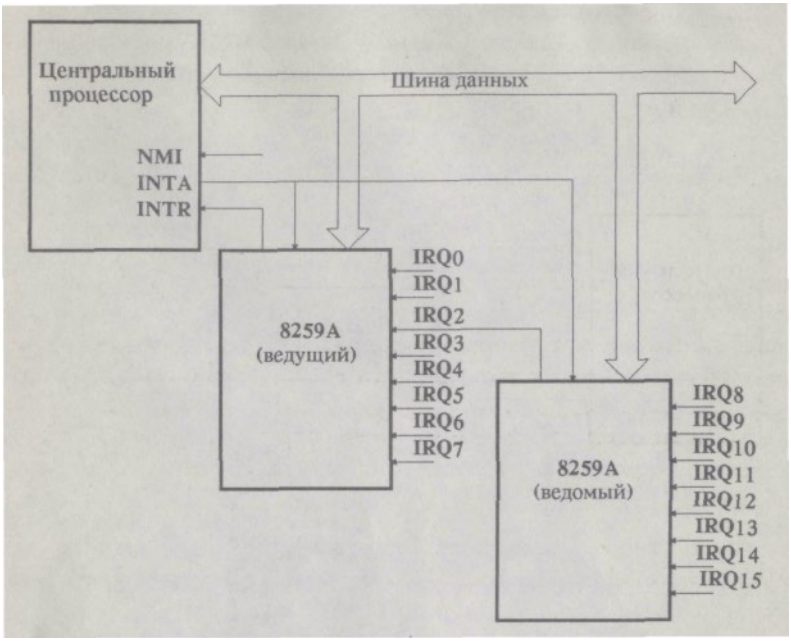


Рис 5.1. Двухкаскадная схема построения контроллера прерываний

Выход ведущей (единственной в однокаскадной схеме) микросхемы 8259A контроллера прерываний подается на специальный вход процессора (INTR). Этот вход процессора является маскируемым: если флаг маскирования прерываний IF равен единице, процессор способен "ощущать" изменение состояния линии INTR (прерывания разрешены); если же IF сброшен в 0, изменения на линии INTR не влияют на работу центрального процессора. Поэтому часто аппаратные прерывания, в формировании которых принимает участие PIC, называют маскируемыми. Если прерывания разрешены и устанавливается высокий потенциал на линии INTR, процессор завершает исполнение текущей инструкции и отвечает двумя циклами сигнала INTA.

Первый цикл сигнала INTA - это, по существу, пустой цикл, который готовит PIC к следующему циклу. Во время второго цикла PIC помещает на шину данных байт, задающий номер аппаратного прерывания. Получив байт номера прерывания, процессор умножает его на 4, формируя смещения до вектора прерываний в ТВП.

Процессор сохраняет в стеке текущее значение регистров флагов CS и IP, затем устанавливает в 0 флаг IF, а в CS и IP - значения из вектора прерывания. В результате управление передается в ISR.

Для того чтобы различать сигналы прерываний от различных внешних устройств, система прерываний IBM PC построена следующим образом. Каждое внешнее устройство подключено к собственной линии запроса прерываний IRQ. При получении сигнала на линии IRQ контроллер прерываний передает в процессор уникальный для данной IRQ байт номера прерывания. Соответствие линий

IRQ и номеров прерывания задается программированием контроллера прерываний. Такое программирование выполняется в ходе начальной загрузки системы специальной процедурой BlOSa и в дальнейшем обычно не изменяется. В принципе, перепрограммирование PIC может выполняться в любой момент и некоторые программы (Windows, OS/2) используют это при своей загрузке. В ходе программирования PIC задаются старшие 5 бит номера прерывания, а младшие 3 бита генерирует микросхема 8259A, определяя двоичный код номера линии IRQ. Ведущая (единственная) микросхема программируется BIOSом так, чтобы передавать в процессор прерывания от 08h до 0Fh. Ведомая 8259A в IBM PC AT настраивается на передачу номеров прерываний от 70h до 77h.

Кроме отображения IRQ на номера прерывания, PIC выполняет упорядочивание по приоритету одновременно возникающих запросов. Обычно наивысший приоритет имеет запрос на линии IRQ0, затем в порядке убывания IRQI, IRQ2, ..., IRQ7. Вход процессора INTR является так называемым "уровнем чувствительным". Это значит, что если процессор ощущает высокий уровень, он всегда начинает цикл обработки прерывания. Если начатая ISR устанавливает IF в единицу (а это, как правило, так и бывает), сохранение сигнала на линии INTR вызовет повторное вхождение в ту же самую ISR, а затем вхождение в третий, четвертый и далее раз до тех пор, пока не переполнится стек. Для того чтобы этого не происходило, контроллер прерываний блокирует генерацию сигнала INTR для текущей активной линии IRQ до тех пор, пока исполняемая ISR не даст явного указания сделать это. Обычно так ISR обозначают свое завершение, посылая в PIC команду завершения прерывания, или EOI (End Of Interrupt). Если ISR не сделает этого, контроллер продолжает блокировать выработку сигнала INTR для всех последующих запросов прерывания как по данной линии, так и по другим, менее приоритетным линиям.

Любая из линий запросов IRQi может быть маскирована. Специальный внутренний регистр PIC хранит битовую маску входов IRQi: бит 0 регистра маски управляет IRQ0 (IRQ8 в ведомой микросхеме 8259A), бит 1 - IRQI (IRQ9), ..., бит 7 - IRQ7 (IRQ15). Если бит равен нулю контроллер генерирует сигнал на линии INTR, если бит равен единице, контроллер не "чувствует" запрос на маскированной битом линии IRQi.

Использование двухкаскадной схемы для построения контроллера прерываний расширяет до 15 чисто обслуживаемых внешних устройств. Для двухкаскадной схемы выход INTR ведомой микросхемы 8259A подается на линию LRQ2 ведущей микросхемы. В результате линии запросов упорядочиваются по приоритету следующим образом: максимальный приоритет имеет IRQ0, затем в порядке

убывания IRQI, IRQ8, ..., IRQ15, IRQ3, ..., IRQ7. Как правило, PIC в ходе начальной загрузки настраивается так, что для линий IRQ0 - IRQ7 генерируются прерывания с номерами 08h - 0Fh соответственно, а для линий IRQ8 - IRQ15 - прерывания с номерами 70h - 77h. Подключение внешних устройств персональных компьютеров к линиям IRQ и, следовательно, закрепление аппаратных прерываний для большинства персональных компьютеров типа IBM PC фактически стандартизовано. В табл. 5.1 приводится закрепление внешних устройств и аппаратных прерываний для IBM PC AT.

Табл.5.1. Использование прерываний в IBM PC AT

Линия запроса	Номер прерывания	Обычное использование
IRQO	8h	Системный таймер
IRQ1	9h	Клавиатура
IRQ2	0Ah	Переадресация от ведомой 8259A
IRQ3	0Bh	COM2 (или COM4)
IRQ4	0Ch	COM1 (или COM3)
IRQ5	0Dh	LPT2
IRQ6	0Eh	Контроллер накопителей на гибком диске
IRQ7	0Fh	LPT1
IRQ8	70h	Таймер реального времени
IRQ9	71h	Прерывание обратного хода луча EGA- и VGA-
IRQ 10	72h	Свободно
IRQ11	73h	Свободно
IRQ12	74h	Свободно
IRQ13	75h	Сопроцессор математики с плавающей точкой
Линия запроса	Номер прерывания	Обычное использование
IRQ 14	76h	Контроллер накопителя на жестком диске
IRQ 15	77h	Свободно

5.3. Немаскируемые прерывания

Процессор, кроме входа INTR, использует еще один вход -вход немаскируемого прерывания, или NMI (NonMaskable Interrupt). Название входа говорит о том, что программное обеспечение не может блокировать восприятие сигнала. Когда на входе NMI появляется сигнал, процессор без помощи PIC генерирует байт номера прерывания, равный двум. В отличие от входа INTR, NMI является "чувствительным к фронту сигнала" (edge sensitive). Генерацию прерывания 2 вы-

зывает изменение состояния линии: с логического нуля на логическую единицу. После того, как прерывание сгенерировано, высокий потенциал линии не способен вызвать очередную генерацию прерываний. Только возврат сигнала в нуль, а затем - в единицу заставит процессор генерировать очередное немаскируемое прерывание.

Сигнал на входе NMI имеет более высокий приоритет, чем INTR, и используется для организации реакции процессора на критические для системы ситуации: обнаружение ошибки четности в данных, хранимых в памяти, выключение питания и т.п.

5.4. Программные прерывания

Когда в программе встречается инструкция INT, процессор исполняет действия, рассмотренные ранее для аппаратного прерывания. Отличие состоит в том, что байт номера прерывания задается самой инструкцией. В этой связи не требуется выполнение циклов INTA. Инструкция INT имеет более высокий приоритет, чем аппаратные и немаскируемые прерывания: если процессор начинает исполнение инструкции INT, он не прерывается сигналами на линиях NMI и INTR. Многие из программных прерываний используются для доступа к ISR BIOSa, операционной системы или инсталлируемых драйверов. Кратко правила взаимодействия с ISR (номер прерывания, описание функции, значения регистров на входе в ISR и после ее завершения, индикация ошибок и т.п.) называют интерфейсом прикладной программы или API (Application Program Interface).

5.5. Исключительные ситуации

Исключительные ситуации - это генерация внутренних прерываний процессором при возникновении необычных условий во время исполнения машинных инструкций. Примером таких ситуаций для микропроцессора Intel 8086/88 является "деление на нуль" (генерируется прерывание 0) и "пошаговое исполнение" (генерация прерывания 1 после завершения текущей инструкции). Число исключительных ситуаций, генерируемых процессорами 80286 и 80386, значительно больше. Для них используются прерывания с номерами 05h и больше (например, для 80386 от 05h до 10h включительно). Многие из этих исключительных ситуаций могут генерироваться только при переключении в защищенный режим работы и связаны с нарушением защиты памяти. Для того чтобы избежать "столкновения" прерываний с одинаковыми номерами, закрепленных за аппаратными прерываниями и исключительными ситуациями защищенного режима, операционная система может выполнить перепрограммирование контроллера прерываний.

5.6. Базовая система ввода-вывода BIOS. Прерывания BIOS. Области данных и таблицы BIOS

Первые 20 прерываний с номерами от 00H до 1Fh закреплены за прерываниями, генерируемыми аппаратными средствами, либо предназначенными для управления аппаратурой персонального компьютера. ISR этих прерываний вместе с некоторыми данными образуют так называемую базовую систему ввода-вывода или BIOS (Base Input-Output System). Все ISR и данные BIOSa записаны в ПЗУ. ISR, входящие в BIOS, представляют собой самый нижний уровень иерархической структуры программного обеспечения (ПО) управления аппаратными средствами компьютера. Они взаимодействуют с аппаратурой на уровне физических сигналов, портов, заданных адресов и в этой связи являются немобильной частью ПО. При появлении новых аппаратных средств приходится перерабатывать BIOS. Поэтому принято различать версии BIOS по дате разработки. Кроме того, для облегчения дополнений BIOSa новые периферийные устройства снабжаются своей секцией ПЗУ, а основной блок BIOS, при загрузке системы проверяет наличие дополнительных секций и "переключает" на них соответствующие прерывания.

Важной особенностью BIOSa является стандартный интерфейс с программой практически для всех персональных компьютеров на базе микропроцессоров семейства Intel. Другими словами, BIOS выполняет роль "экрана" между программами (в частности, программами MS-DOS) и большим разнообразием конкретных аппаратных средств. Например, для вывода символа на экран дисплея независимо от типа дисплея и используемого адаптера необходимо выполнить инструкцию INT 10h с теми же самыми значениями во внутренних регистрах. Все детали интерфейса программы с ВЮ5ом описываются в техническом справочнике BIOS.

При выполнении ISR BIOS для хранения данных используется зарезервированная область памяти, называемая областью данных BIOSa. Она начинается с адреса 40:00h и занимает 256 байт до адреса 40:FFh. Здесь располагается ряд таблиц, копируемых из ПЗУ при начальной загрузке системы и уточняемых по результатам тестирования узлов компьютера. При выполнении функций BIOS многие параметры изменяются. Например, корректируется адрес позиции курсора на экране, номер установленного режима адаптера дисплея и т.п. Другими словами, таблицы в области данных BIOSa отражают текущие параметры и состояние аппаратных средств компьютера.

5.7. Функции библиотеки C++ для доступа к обработчикам прерывания

Библиотечные функции C++, как правило, в конечном итоге обращаются к

ISR BIOS или MS-DOS. В тех случаях, когда необходимо непосредственное обращение к BIOS или MS-DOS, используются специальные функции, описываемые далее.

```
int int86(int intno, union REGS *inregs, union REGS *outregs)
```

Функция загружает внутренние регистры микропроцессора значениями, записанными в объединении по шаблону union REGS, на начало которого указывает inregs, и выполняет прерывание с номером intno. Значения внутренних регистров на выходе из прерывания записываются в объединении по шаблону union REGS, на начало которого указывает outregs. Описание объединений выполняет точка вызова функции. Шаблон union REGS описан в заголовочном файле <dos.h> и представляет собой объединение двух структур:

```
struct WORDREGS
 {
    unsigned int ax, bx, ex, dx, si, di, cflag, flags;
    };

struct BYTEREGS
  {
    unsigned char al, ah, bl, bh, cl, ch, dl, dh;
  };

union REGS
 {
   struct WORDREGS x;
   struct BYTEREGS h;
 };
```

Структура WORDREGS используется для доступа к регистрам как двухбайтовым единицам. Структура BYTEREGS позволяет осуществлять доступ к отдельным байтам РОН. Поле структуры flags позволяет перед вызовом задать, а после вызова прочесть значение регистра флагов. Так как многие функции MS-DOS используют флаг переноса для сигнализации об ошибках в программе-обработчике прерывания, в структуре WORDREGS специально выделено поле cflag для значения флага переноса.

Все функции int...() возвращают значение регистра AX на выходе из ISR. Недостатком функции int86() является возможность доступа лишь к ограниченному числу регистров. При выполнении некоторых функций MS-DOS значения задают-

ся и в сегментных регистрах. В таких (правда, достаточно редких) случаях следует использовать более общую функцию int86x():

```
int int86x(int intno, union REGS *inregs, union REGS *outregs, struct SREGS *segregs)
```

В отличие от int86() перед выполнением прерывания intno дополнительно устанавливаются сегментные регистры из структурной переменной по шаблону SREGS. В функцию передается указатель на эту структурную переменную. По возвращении из ISR в структурную переменную по шаблону SREGS дополнительно копируются значения всех сегментных регистров. Если необходимо выполнить обращение к функции MS-DOS (т. е. прерывание 21h с заданным значением AH), можно использовать функцию intdos(), всегда обращающуюся к прерыванию 21h.

```
int intdos(union REGS *inregs, union REGS *outregs)
```

В отличие от ранее рассмотренных функций данной функции не передается номер генерируемого прерывания, так как всегда генерируется прерывание 21h.

5.8. Предварительная подготовка к работе

1. Ознакомиться с аппаратными средствами системы прерывания.
2. Ознакомиться с программными средствами системы прерывания.

5.9. Порядок выполнения работы

По заданию преподавателя разработать алгоритм и реализовать программу подключения собственной подпрограммы обработки прерывания и использовать её в цепочке со стандартной подпрограммой обработки прерывания от одного из следующих устройств компьютера:

1. системный таймер;
2. клавиатура;
3. контроллер накопителя на гибких магнитных дисках;
4. таймер реального времени;
5. контроллер накопителя на жёстком магнитном диске.

5.10. Содержание отчета

Отчет по лабораторной работе должен содержать:

- титульный лист;
- задание на лабораторную работу;
- блок-схему алгоритма с пояснениями;
-текст программы;
- примеры запуска программы;
- структурная схема аппаратных средств, используемых при выполнении программы с необходимой степенью детализации блоков.

5.11. Контрольные вопросы

1. Что такое таблица векторов прерывания?
2. Что хранится в одной строчке таблицы векторов прерывания?
3. Какая информация сохраняется в стеке автоматически?
4. Какой из регистров процессора указывает на вершину стека?
5. Каким образом определяется точка входа в таблицу векторов прерывания?
6. Какую информацию следует сохранять в стеке в начале выполнения подпрограммы обработки прерывания?
7. Какие действия должна выполнить подпрограмма обработки прерывания перед своим завершением?
8. С какой частотой поступает запрос по линии прерывания IRQ0?
9. Сколько раз может вызываться прерывание от клавиатуры при однократном нажатии и отпускании произвольной клавиши?
10. Каким образом определяется приоритет при одновременном поступлении нескольких запросов на прерывание?
11. Чем определяется время реакции процессора на запрос прерывания?
12. Какие действия автоматически выполняются процессором при выходе из подпрограммы обработки прерывания?

Список литературы

1. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем. Учебник для вузов. СПб.:, Питер, 2006.
2. Организация ЭВМ. 5-е изд./ К.Хамахер, З.Вранешич, С.Заки. – СПб. Питер; Киев: Издательская группа BHV , 2003. – 848 с.: ил.- (Серия «Классика computer science»).
3. Скляров В.А. Применение ПЭВМ. Кн.1. Организация и управление ресурсами ПЭВМ. М.:, Высшая школа, 1992.
4. Ларионов А.М., Горнец Н.Н. Периферийные устройства в вычислительных системах. М.:, Высшая школа,1991.
5. Григорьев В.Л. Видеосистемы ПК фирмы IBM. М.:, Радио и связь,1993.
6. Анисимов А.В., Валов А.А., Герасимов И.В., Петров Г.А., Родионов С.В., Филиппов Е.В., Чугунов Л.А. Основы организации вычислительных комплексов для решения задач автоматизации и управления: Учеб. Пособие / ГЭТУ. - СПб.,1995.
7. https://www.softpedia.com/get/Programming/Coding-languages-Compilers/TurboCplusplus-for-Windows-7.shtml
8. https://ru.wikihow.com/начать-обучение-программированию-на-C-в-Turbo-C%2B%2B-IDE

Содержание

Редактор Г. Г. Петров

Подписано в печать . Формат 60×84 1/16.
Бумага офсетная. Печать офсетная. Печ. л. .
Гарнитура « ». Тираж экз. Заказ

Издательство СПбГЭТУ «ЛЭТИ»
197376, С.-Петербург, ул. Проф. Попова, 5