Borland C++ DOS Support Help Contents

Click the icon above to open all folders. Click an icon below to open a folder or click the underlined text to see a specific topic.

Functions and Global Variables contains a reference to DOS functions and variables.

Run-time Libraries contains an overview of the Borland C++ run-time libraries.

Borland Graphics Interface contains the reference to the Borland graphics functions.

Shortcut: Use the **Search** and **Search All** buttons at the top of the Help window if you are ready to look for an item by name.

Borland C++ DOS Support Help Contents

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Functions and Global Variables contains a reference to DOS functions and variables.

- Functions
- Global Variables

Run-time Libraries contains an overview of the Borland C++ run-time libraries.

Borland Graphics Interface contains the reference to the Borland graphics functions.

Shortcut: Use the **Search** and **Search All** buttons at the top of the Help window if you are ready to look for an item by name.

Borland C++ DOS Support Help Contents

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Functions and Global Variables contains a reference to DOS functions and variables.

Run-time Libraries contains an overview of the Borland C++ run-time libraries.

- Interface Routines
- Memory Routines
- Miscellaneous DOS Routines

Borland Graphics Interface contains the reference to the Borland graphics functions.

Shortcut: Use the **Search** and **Search All** buttons at the top of the Help window if you are ready to look for an item by name.

Borland C++ DOS Support Help Contents

Click any icon to close all folders or click the underlined text to see a specific topic.

Functions and Global Variables contains a reference to DOS functions and variables.

- Functions
- Global Variables

Run-time Libraries contains an overview of the Borland C++ run-time libraries.

- Interface Routines
- Memory Routines
- Miscellaneous DOS Routines

- Borland Graphics Interface contains the reference to the Borland graphics functions.

Shortcut: Use the **Search** and **Search All** buttons at the top of the Help window if you are ready to look for an item by name.

DOS Run-time Libraries

See Also Overview

The static (OBJ and LIB) 16-bit Borland C++ run-time libraries are contained in the LIB subdirectory of your installation. For each of the library file names, the '?' character represents one of the six (tiny, compact, small, medium, large, and huge) distinct memory models supported by Borland. Each model has its own library file and math file, containing versions of the routines written for that particular model.

The following Borland C++ libraries are available for 16-bit DOS-only applications.

File name	Use
BIDSH.LIB	Huge memory model of Borland classlibs
BIDSDBH.LIB	Diagnostic version of the above library
C?.LIB	DOS-only libraries
C0F?.	OBJ MS compatible startup
C0?.OBJ	BC startup
EMU.LIB	Floating-point emulation
FP87.LIB	For programs that run on machines with 80x87 coprocessor
GRAPHICS.LIB	Borland graphics interface
MATH?.LIB	Math routines
OVERLAY.LIB	Overlay development

See Also

DOS-Only Functions

Global Variables

Graphics Routines (BGI)

Interface Routines

Memory Routines

Miscellaneous Dos Routines

DOS Run-time Libraries Overview

The DOS-specific applications use static run-time libraries (OBJ and LIB). These libraries are available only to the 16-bit development tools. Library routines are composed of functions and macros that you can call from within your C and C++ programs to perform a wide variety of tasks. These tasks include low- and high-level I/O, string and file manipulation, memory allocation, process control, data conversion, mathematical calculations, and much more.

Several versions of the run-time library are available. For example, there are memory-model specific versions and diagnostic versions. There are also optional libraries that provide containers, graphics, and mathematics.

DOS Interface Routines

These routines provide operating-system BIOS and machine-specific capabilities for DOS.

Function	Header	Function	Header
absread	(dos.h)	_dos_keep	(dos.h)
abswrite	(dos.h)	freemem	(dos.h)
abswrite	(bios.h)	_harderr	(dos.h)
_bios_keybrd	(bios.h)	harderr	(dos.h)
_bios_printer	(bios.h)	hardresume	(dos.h)
_bios_serialcom	(bios.h)	_hardresume	(dos.h)
bioscom	(bios.h)	hardretn	(dos.h)
biosdisk	(bios.h)	_hardretn	(dos.h)
bioskey	(bios.h)	keep	(dos.h)
biosprint	(bios.h)	randbrd	(dos.h)
_dos_freemem	(dos.h)	randbwr	(dos.h)

DOS Memory Routines

These routines provide dynamic memory allocation in the small-data and large-data models.

Function	Header
allocmem	(dos.h)
brk	(alloc.h)
coreleft	(alloc.h, stdlib.h)
_dos_allocmem	(alloc.h)
_dos_freemem	(alloc.h)
_dos_setblock	(dos.h)
farcoreleft	(alloc.h)
farheapcheck	(alloc.h)
farheapcheckfree	(alloc.h)
farheapchecknode	(dos.h)
farheapfillfree	(dos.h)
farheapwalk	(alloc.h)
farrealloc	(alloc.h)
sbrk	(alloc.h)

Miscellaneous DOS routines

These routines provide sound effects and time delay.

Function	Header
delay	(dos.h)
sound	(dos.h)
nosound	(dos.h)

DOS Global Variables

The following Borland C++ global variables are available for 16-bit DOS-only applications:

_heaplen
_ovrbuffer
_stklen

_heaplen

See Also

Syntax

```
#include <dos.h>
extern unsigned _heaplen;
```

Description

This global variable holds the length of the near heap (Available only for 16-bit DOS applications.)

_heaplen specifies the size (in bytes) of the near heap in the small data models (tiny, small, and medium). _heaplen does not exist in the large data models (compact, large, and huge), as they do not have a near heap.

In the small and medium models, the data segment size is computed as follows:

```
data segment [small,medium] = global data + heap + stack
```

where the size of the stack can be adjusted with _stklen.

If _heaplen is set to 0, the program allocates 64K bytes for the data segment, and the effective heap size is

```
64K - (global data + stack) bytes
```

By default, _heaplen equals 0, so you'll get a 64K data segment unless you specify a particular _heaplen value.

In the tiny model, everything (including code) is in the same segment, so the data segment computations are adjusted to include the code plus 256 bytes for the program segment prefix (PSP).

```
data segment[tiny] = 256 + code + global data + heap + stack
```

If _heaplen equals 0 in the tiny model, the effective heap size is obtained by subtracting the PSP, code, global data, and stack from 64K.

In the compact and large models, there is no near heap, and the stack is in its own segment, so the data segment is

```
data segment [compact,large] = global data
```

In the huge model, the stack is a separate segment, and each module has its own data segment.

See Also
_stklen

_ovrbuffer

See Also

Syntax

```
#include <dos.h>
unsigned _ovrbuffer = size;
```

Description

This global variable changes the size of the overlay buffer. (Available only for 16-bit DOS applications.)

The default overlay buffer size is twice the size of the largest overlay. This is adequate for some applications. Suppose, however, that a particular function of a program is implemented through many modules, each of which is overlaid. If the total size of those modules is larger than the overlay buffer, a substantial amount of swapping will occur if the modules make frequent calls to each other.

The solution is to increase the size of the overlay buffer so that enough memory is available at any given time to contain all overlays that make frequent calls to each other. You can do this by setting the _ovrbuffer global variable to the required size in paragraphs. For example, to set the overlay buffer to 128K, include the following statement in your code:

```
unsigned _ovrbuffer = 0x2000;
```

There is no general formula for determining the ideal overlay buffer size.

See Also

_OvrInitEms

_OvrInitExt

_stklen

See Also

Syntax

```
#include <dos.h>
extern unsigned _stklen;
```

Description

This global variable holds the size of the stack.

_stklen specifies the size of the stack for all six memory models. The minimum stack size allowed is 128 words; if you give a smaller value, _stklen is automatically adjusted to the minimum. The default stack size is 4K.

In the small and medium models, the data segment size is computed as follows:

```
  data segment [small,medium] = global data + heap + stack
```

where the size of the heap can be adjusted with _heaplen.

In the tiny model, everything (including code) is in the same segment,so the data segment computations are adjusted to include the code plus 256 bytes for the program segment prefix (PSP).

```
  data segment[tiny] = 256 + code + global data + heap +stack
```

In the compact and large models, there is no near heap, and the stackis in its own segment, so the data segment is simply

```
data segment [compact,large] = global data
```

In the huge model, the stack is a separate segment, and each modulehas its own data segment.

See Also
_heaplen

DOS-Only Functions

The following library routines are available only for 16-bit DOS-only applications:

absread
abswrite
allocmem
_bios_disk
_bios_keybrd
_bios_printer
_bios_serialcom
bioscom
biosdisk
bioskey
biosprint
brk
coreleft
delay
_dos_allocmem
_dos_freemem
_dos_keep
_dos_setblock
farcoreleft
farheapcheck
farheapcheckfree
farheapchecknode
farheapfillfree
farheapwalk
freemem
harderr
_harderr
hardresume
_hardresume
hardretn
_hardretn
keep
nosound
_OvrInitEms
_OvrInitExt
randbrd
randbwr
sbrk
setblock
sound

Functions And Global Variables

absread

See Also Example Portability

Syntax

```
#include <dos.h>
int absread(int drive, int nsects, long lsect, void *buffer);
```

Description

absread reads specific disk sectors. It ignores the logical structure of a disk and pays no attention to files, FATs, or directories.

absread uses DOS interrupt 0x25 to read specific disk sectors.

drive	drive number to read (0 = A, 1 = B, etc.)
nsects	number of sectors to read
lsect	beginning logical sector number
buffer	memory address where the data is to be read

The number of sectors to read is limited to 64K or the size of the buffer, whichever is smaller.

Return Value

If it is successful, absread returns 0.

On error, the routine returns -1 and sets the global variable errno to the value returned by the system call in the AX register.

See Also
abswrite
biosdisk

abswrite

See Also Portability

Syntax

```
#include <dos.h>
int abswrite(int drive, int nsects, long lsect, void *buffer);
```

Description

abswrite writes specific disk sectors. It ignores the logical structure of a disk and pays no attention to files, FATs, or directories.

If used improperly, abswrite can overwrite files, directories, and FATs.

abswrite uses DOS interrupt 0x26 to write specific disk sectors.

drive	drive number to write to (0 = A, 1 = B, etc.)
nsects	number of sectors to write to
lsect	beginning logical sector number
buffer	memory address where the data is to be written

The number of sectors to write to is limited to 64K or the size of the buffer, whichever is smaller.

Return Value

If it is successful, abswrite returns 0.

On error, the routine returns -1 and sets the global variable errno to the value of the AX register returned by the system call.

See Also
absread
biosdisk

allocmem, _dos_allocmem

See Also Example Portability

Syntax

```
#include <dos.h>
int allocmem(unsigned size, unsigned *segp);
unsigned _dos_allocmem(unsigned size, unsigned *segp);
```

Description

allocmem and _dos_allocmem use the DOS system call 0x48 to allocate a block of free memory and return the segment address of the allocated block.

size is the desired size in paragraphs (a paragraph is 16 bytes). segp is a pointer to a word that will be assigned the segment address of the newly allocated block.

For allocmem, if not enough room is available, no assignment is made to the word pointed to by segp.

For _dos_allocmem, if not enough room is available, the size of the largest available block will be stored in the word pointed to by segp.

All allocated blocks are paragraph-aligned.

allocmem and _dos_allocmem cannot coexist with malloc.

Return Value

allocmem returns -1 on success. In the event of error, allocmem returns a number indicating the size in paragraphs of the largest available block.

_dos_allocmem returns 0 on success. In the event of error, _dos_allocmem returns the DOS error code and sets the word pointed to by segp to the size in paragraphs of the largest available block.

An error return from allocmem or _dos_allocmem sets the global variable _doserrno and sets the global variable errno to:

ENOMEM	Not enough memory

See Also
coreleft
freemem
malloc (in BCW Help)
setblock

bioscom

Example Portability

Syntax

```
#include <bios.h>
int bioscom(int cmd, char abyte, int port);
```

Description

bioscom performs various RS-232 communications over the I/O port given in port.

A port value of 0 corresponds to COM1, 1 corresponds to COM2, and so forth.

The value of cmd can be one of the following:

Value	Meaning
0	Sets the communications parameters to the value in abyte.
1	Sends the character in abyte out over the communications line.
2	Receives a character from the communications line.
3	Returns the current status of the communications port.

abyte is a combination of the following bits (one value is selected from each of the groups):

0x02	7 data bits	0x00	110 baud
0x03	8 data bits	0x20	150 baud
		0x40	300 baud
0x00	1 stop bit	0x60	600 baud
0x04	2 stop bits	0x80	1200 baud
0x00	No parity	0xA0	2400 baud
0x08	Odd parity	0xC0	4800 baud
0x18	Even parity	0xE0	9600 baud

For example, a value of 0xEB (0xE0|0x08|0x00|0x03) for abyte sets the communications port to 9600 baud, odd parity, 1 stop bit, and 8 data bits. bioscom uses the BIOS 0x14 interrupt.

Return Value

For all values of cmd, bioscom returns a 16-bit integer, of which the upper 8 bits are status bits and the lower 8 bits vary, depending on the value of cmd. The upper bits of the return value are defined as follows:

Bit 15	Time out
Bit 14	Transmit shift register empty
Bit 13	Transmit holding register empty
Bit 12	Break detect
Bit 11	Framing error
Bit 10	Parity error
Bit 9	Overrun error
Bit 8	Data ready

If the abyte value could not be sent, bit 15 is set to 1. Otherwise, the remaining upper and lower bits are appropriately set. For example, if a framing error has occurred, bit 11 is set to 1.

With a cmd value of 2, the byte read is in the lower bits of the return value if there is no error. If an error occurs, at least one of the upper bits is set to 1. If no upper bits are set to 1, the byte was

received without error.

With a cmd value of 0 or 3, the return value has the upper bits set as defined, and the lower bits are defined as follows:

Bit 7	Received line signal detect
Bit 6	Ring indicator
Bit 5	Data set ready
Bit 4	Clear to send
Bit 3	Change in receive line signal detector
Bit 2	Trailing edge ring detector
Bit 1	Change in data set ready
Bit 0	Change in clear to send

biosdisk

See Also Example Portability

Syntax

```
#include <bios.h>
int biosdisk(int cmd, int drive, int head, int track, int sector, int
  nsects, void *buffer);
```

Description

biosdisk uses interrupt 0x13 to issue disk operations directly to the BIOS.

drive is a number that specifies which disk drive is to be used: 0 for the first floppy disk drive, 1 for the second floppy disk drive, 2 for the third, and so on. For hard disk drives, a drive value of 0x80 specifies the first drive, 0x81 specifies the second, 0x82 the third, and so forth.

For hard disks, the physical drive is specified, not the disk partition. If necessary, the application program must interpret the partition table information itself.

cmd indicates the operation to perform. Depending on the value of cmd, the other parameters may or may not be needed.

Here are the possible values for cmd for the IBM PC, XT, AT, or PS/2, or any compatible system:

0 Resets disk system, forcing the drive controller to do a hard reset. All other parameters are ignored.

1 Returns the status of the last disk operation. All other parameters are ignored.

2 Reads one or more disk sectors into memory. The starting sector to read is given by head, track, and sector. The number of sectors is given by nsects. The data is read, 512 bytes per sector, into buffer.

3 Writes one or more disk sectors from memory. The starting sector to write is given by head, track, and sector. The number of sectors is given by nsects. The data is written, 512 bytes per sector, from buffer.

4 Verifies one or more sectors. The starting sector is given by head, track, and sector. The number of sectors is given by nsects.

5 Formats a track. The track is specified by head and track. buffer points to a table of sector headers to be written on the named track. See the Technical Reference Manual for the IBM PC for a description of this table and the format operation.

The following cmd values are allowed only for the XT, AT, PS/2, and compatibles:

6 Formats a track and sets bad sector flags.

7 Formats the drive beginning at a specific track.

8 Returns the current drive parameters. The drive information is returned in buffer in the first 4 bytes.

9 Initializes drive-pair characteristics.

10 Does a long read, which reads 512 plus 4 extra bytes per sector.

11 Does a long write, which writes 512 plus 4 extra bytes per sector.

12 Does a disk seek.

13 Alternates disk reset.

14 Reads sector buffer.

15 Writes sector buffer.

16 Tests whether the named drive is ready.

17 Recalibrates the drive.

18 Controller RAM diagnostic.

19 Drive diagnostic.

20 Controller internal diagnostic.

biosdisk operates below the level of files on raw sectors. It can destroy file contents and directories on a hard disk.

Return Value

biosdisk returns a status byte composed of the following bits:

0x00 Operation successful.
0x01 Bad command.
0x02 Address mark not found.
0x03 Attempt to write to write-protected disk.
0x04 Sector not found.
0x05 Reset failed (hard disk).
0x06 Disk changed since last operation.
0x07 Drive parameter activity failed.
0x08 Direct memory access (DMA) overrun.
0x09 Attempt to perform DMA across 64K boundary.
0x0A Bad sector detected.
0x0B Bad track detected.
0x0C Unsupported track.
0x10 Bad CRC/ECC on disk read.
0x11 CRC/ECC corrected data error.
0x20 Controller has failed.
0x40 Seek operation failed.
0x80 Attachment failed to respond.
0xAA Drive not ready (hard disk only).
0xBB Undefined error occurred (hard disk only).
0xCC Write fault occurred.
0xE0 Status error.
0xFF Sense operation failed.
0x11 is not an error because the data is correct. The value is returned to give the application an opportunity to decide for itself.

See Also
absread
abswrite

_bios_disk

See Also Example Portability

Syntax

```
#include <bios.h>
unsigned _bios_disk(unsigned cmd, struct diskinfo_t *dinfo);
```

Description

_bios_disk uses interrupt 0x13 to issue disk operations directly to the BIOS. The cmd argument specifies the operation to perform, and dinfo points to a diskinfo_t structure that contains the remaining parameters required by the operation.

The diskinfo_t structure (defined in bios.h) has the following format:

```
struct diskinfo_t {
    unsigned drive, head, track, sector, nsectors;
    void far *buffer;
};
```

drive is a number that specifies which disk drive is to be used: 0 for the first floppy disk drive, 1 for the second floppy disk drive, 2 for the third, and so on. For hard disk drives, a drive value of 0x80 specifies the first drive, 0x81 specifies the second, 0x82 the third, and so forth.

For hard disks, the physical drive is specified, not the disk partition. If necessary, the application program must interpret the partition table information itself.

Depending on the value of cmd, the other parameters in the diskinfo_t structure may or may not be needed.

The possible values for cmd (defined in bios.h) are the following:

_DISK_RESET — Resets disk system, forcing the drive controller to do a hard reset. All diskinfo_t parameters are ignored.

_DISK_STATUS — Returns the status of the last disk operation. All diskinfo_t parameters are ignored.

_DISK_READ — Reads one or more disk sectors into memory. The starting sector to read is given by head, track, and sector. The number of sectors is given by nsectors. The data is read, 512 bytes per sector, into buffer. If the operation is successful, the high byte of the return value will be 0 and the low byte will contain the number of sectors. If an error occurred, the high byte of the return value will have one of the following values:

0x01	Bad command.
0x02	Address mark not found.
0x03	Attempt to write to write-protected disk.
0x04	Sector not found.
0x05	Reset failed (hard disk).
0x06	Disk changed since last operation.
0x07	Drive parameter activity failed.
0x08	Direct memory access (DMA) overrun.
0x09	Attempt to perform DMA across 64K boundary.
0x0A	Bad sector detected.
0x0B	Bad track detected.
0x0C	Unsupported track.
0x10	Bad CRC/ECC on disk read.
0x11	CRC/ECC corrected data error.

0x20	Controller has failed.
0x40	Seek operation failed.
0x80	Attachment failed to respond.
0xAA	Drive not ready (hard disk only).
0xBB	Undefined error occurred (hard disk only).
0xCC	Write fault occurred.
0xE0	Status error.
0xFF	Sense operation failed.
0x11	is not an error because the data is correct. The value is returned to give the application an opportunity to decide for itself.

_DISK_WRITE — Writes one or more disk sectors from memory. The starting sector to write is given by head, track, and sector. The number of sectors is given by nsectors. The data is written, 512 bytes per sector, from buffer. See _DISK_READ (above) for a description of the return value.

_DISK_VERIFY — Verifies one or more sectors. The starting sector is given by head, track, and sector. The number of sectors is given by nsectors. See _DISK_READ (above) for a description of the return value.

_DISK_FORMAT — Formats a track. The track is specified by head and track. buffer points to a table of sector headers to be written on the named track. See the Technical Reference Manual for the IBM PC for a description of this table and the format operation.

Return Value

_bios_disk returns the value of the AX register set by the INT 0x13 BIOS call.

See Also

absread

abswrite

biosdisk

bioskey

Example Portability

Syntax

```
#include <bios.h>
int bioskey(int cmd);
```

Description

bioskey performs various keyboard operations using BIOS interrupt 0x16. The parameter cmd determines the exact operation.

Return Value

The value returned by bioskey depends on the task it performs, determined by the value of cmd:

0 If the lower 8 bits are nonzero, bioskey returns the ASCII character for the next keystroke waiting in the queue or the next key pressed at the keyboard. If the lower 8 bits are zero, the upper 8 bits are the extended keyboard codes defined in the IBM PC Technical Reference Manual.

1 This tests whether a keystroke is available to be read. A return value of zero means no key is available. The return value is 0xFFFFF (-1) if Ctrl-Brk has been pressed. Otherwise, the value of the next keystroke is returned. The keystroke itself is kept to be returned by the next call to bioskey that has a cmd value of zero.

2 Requests the current shift key status. The value is obtained by ORing the following values together:

Bit 7	0x80	Insert on
Bit 6	0x40	Caps on
Bit 5	0x20	Num Lock on
Bit 4	0x10	Scroll Lock on
Bit 3	0x08	Alt pressed
Bit 2	0x04	Ctrl pressed
Bit 1	0x02	Left arrow + Shift pressed
Bit 0	0x01	Right arrow + Shift pressed

biosprint

Example Portability

Syntax

```
#include <bios.h>
int biosprint(int cmd, int abyte, int port);
```

Description

biosprint performs various printer functions on the printer identified by the parameter port using BIOS interrupt 0x17.

A port value of 0 corresponds to LPT1; a port value of 1 corresponds to LPT2; and so on.

The value of cmd can be one of the following:

0 Prints the character in abyte.

1 Initializes the printer port.

2 Reads the printer status.

The value of abyte can be 0 to 255.

Return Value

The value returned from any of these operations is the current printer status, which is obtained by ORing these bit values together:

Bit 0	0x01	Device time out
Bit 3	0x08	I/O error
Bit 4	0x10	Selected
Bit 5	0x20	Out of paper
Bit 6	0x40	Acknowledge
Bit 7	0x80	Not busy

_bios_printer

Example Portability

Syntax

```
#include <bios.h>
unsigned _bios_printer(int cmd, int port, int abyte);
```

Description

_bios_printer performs various printer functions on the printer identified by the parameter port using BIOS interrupt 0x17.

A port value of 0 corresponds to LPT1; a port value of 1 corresponds to LPT2; and so on.

The value of cmd can be one of the following values (defined in bios.h):

_PRINTER_WRITE	Prints the character in abyte. The value of abyte can be 0 to 255.
_PRINTER_INIT	Initializes the printer port. The abyte argument is ignored.
_PRINTER_STATUS	Reads the printer status. The abyte argument is ignored.

Return Value

The value returned from any of these operations is the current printer status, which is obtained by ORing these bit values together:

Bit 0	0x01	Device time out
Bit 3	0x08	I/O error
Bit 4	0x10	Selected
Bit 5	0x20	Out of paper
Bit 6	0x40	Acknowledge
Bit 7	0x80	Not busy

_bios_keybrd

Example Portability

Syntax

```
#include <bios.h>
unsigned _bios_keybrd(unsigned cmd);
```

Description

_bios_keybrd performs various keyboard operations using BIOS interrupt 0x16. The parameter cmd determines the exact operation.

Return Value

The value returned by _bios_keybrd depends on the task it performs, determined by the value of cmd (defined in bios.h):

_KEYBRD_READ	If the lower 8 bits are nonzero, _bios_keybrd returns the ASCII character for the next keystroke waiting in the queue or the next key pressed at the keyboard. If the lower 8 bits are zero, the upper 8 bits are the extended keyboard codes defined in the IBM PC Technical Reference Manual.
_NKEYBRD_READ	Use this value instead of _KEYBRD_READY to read the keyboard codes for enhanced keyboards, which have additional cursor and function keys.
_KEYBRD_READY	This tests whether a keystroke is available to be read. A return value of zero means no key is available. The return value is 0xFFFF (-1) if Ctrl-Brk has been pressed. Otherwise, the value of the next keystroke is returned, as described in _KEYBRD_READ (above). The keystroke itself is kept to be returned by the next call to _bios_keybrd that has a cmd value of _KEYBRD_READ or _NKEYBRD_READ
_NKEYBRD_READY	Use this value to check the status of enhanced keyboards, which have additional cursor and function keys.
_KEYBRD_SHIFTSTATUS	Requests the current shift key status. The value will contain an OR of zero or more of the following values: Bit 7 0x80 Insert on Bit 6 0x40 Caps on Bit 5 0x20 Num Lock on Bit 4 0x10 Scroll Lock on Bit 3 0x08 Alt pressed Bit 2 0x04 Ctrl pressed Bit 1 0x02 Left Shift pressed Bit 0 0x01 Right Shift pressed
_NKEYBRD_SHIFTSTATUS	Use this value instead of _KEYBRD_SHIFTSTATUS to request the full 16-bit shift key status for enhanced keyboards. The return value will contain an OR of zero or more of the bits defined above in _KEYBRD_SHIFTSTATUS, and additionally, any of the following bits: Bit 15 0x8000 Sys Req pressed Bit 14 0x4000 Caps Lock pressed Bit 13 0x2000 Num Lock pressed Bit 12 0x1000 Scroll Lock pressed Bit 11 0x0800 Right Alt pressed

Bit 10	0x0400	Right Ctrl pressed
Bit 9	0x0200	Left Alt pressed
Bit 8	0x0100	Left Ctrl pressed

_bios_serialcom

Examples Portability

Syntax

```
#include <bios.h>
unsigned _bios_serialcom(int cmd, int port, char abyte);
```

Description

_bios_serialcom performs various RS-232 communications over the I/O port given in port.

A port value of 0 corresponds to COM1, 1 corresponds to COM2, and so forth.

The value of cmd can be one of the following values (defined in bios.h):

_COM_INIT	Sets the communications parameters to the value in abyte.
_COM_SEND	Sends the character in abyte out over the communications line.
_COM_RECEIVE	Receives a character from the communications line. The abyte argument is ignored.
_COM_STATUS	Returns the current status of the communications port. The abyte argument is ignored.

When cmd is _COM_INIT, abyte is a OR combination of the following bits:

Select only one of these:

_COM_CHR7	7 data bits
_COM_CHR8	8 data bits

Select only one of these:

_COM_STOP1	1 stop bit
_COM_STOP2	2 stop bits

Select only one of these:

_COM_NOPARITY	No parity
_COM_ODDPARITY	Odd parity
_COM_EVENPARITY	Even parity

Select only one of these:

_COM_110	110 baud
_COM_150	150 baud
_COM_300	300 baud
_COM_600	600 baud
_COM_1200	1200 baud
_COM_2400	2400 baud
_COM_4800	4800 baud
_COM_9600	9600 baud

For example, a value of (_COM_9600 | _COM_ODDPARITY | _COM_STOP1 |_COM_CHR8) for abyte sets thecommunicationsportto9600 baud,oddparity,1 stop bit, and 8 data bits. _bios_serialcom uses the BIOS 0x14 interrupt.

For all values of cmd, _bios_serialcom returns a 16-bit integer of which the upper 8 bits are status bits and the lower 8 bits vary, depending on the value of cmd. The upper bits of the return value are defined as follows:

Bit 15	Time out
Bit 14	Transmit shift register empty
Bit 13	Transmit holding register empty
Bit 12	Break detect
Bit 11	Framing error
Bit 10	Parity error
Bit	9 Overrun error
Bit	8 Data ready

If the abyte value could not be sent, bit 15 is set to 1. Otherwise, the remaining upper and lower bits are appropriately set. For example, if a framing error has occurred, bit 11 is set to 1.

With a cmd value of _COM_RECEIVE, the byte read is in the lower bits of the return value if there is no error. If an error occurs, at least one of the upper bits is set to 1. If no upper bits are set to 1, the byte was received without error. With a cmd value of _COM_INIT or COM_STATUS, the return value has the upper bits set as defined, and the lower bits are defined as follows:

Bit 7	Received line signal detect
Bit 6	Ring indicator
Bit 5	Data set ready
Bit 4	Clear to send
Bit 3	Change in receive line signal detector
Bit 2	Trailing edge ring detector
Bit 1	Change in data set ready
Bit 0	Change in clear to send

Examples

Using COM1

Using COM2

brk

See Also Example Portability

Syntax

```
#include <alloc.h>
int brk(void *addr);
```

Description

brk dynamically changes the amount of space allocated to the calling program's heap. The change is made by resetting the program's break value, which is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases.

brk sets the break value to addr and changes the allocated space accordingly.

This function will fail without making any change in the allocated space if such a change would allocate more space than is allowable.

Return Value

Upon successful completion, brk returns a value of 0. On failure, this function returns a value of -1 and the global variable errno is set to

ENOMEM	Not enough memory

See Also
coreleft
sbrk

coreleft

See Also Example Portability

Syntax

```
#include <alloc.h>
```

In the tiny, small, and medium models:

```
unsigned coreleft(void);
```

In the compact, large, and huge models:

```
unsigned long coreleft(void);
```

Description

coreleft returns a measure of RAM memory not in use. It gives a different measurement value, depending on whether the memory model is of the small data group or the large data group.

Return Value

In the small data models, coreleft returns the amount of unused memory between the top of the heap and the stack. In the large data models, coreleft returns the amount of memory between the highest allocated block and the end of available memory.

See Also
allocmem
brk
farcoreleft
malloc (in BCW Help)

delay

See Also Example Portability

Syntax

```
#include <dos.h>
void delay(unsigned milliseconds);
```

Description

With a call to delay, the current program is suspended from execution for the number of milliseconds specified by the argument milliseconds. It is no longer necessary to make a calibration call to delay before using it. delay is accurate to a millisecond.

Return Value

None.

See Also
nosound
sleep (in BCW Help)
sound

farcoreleft

See Also Example Portability

Syntax

```
#include <alloc.h>
unsigned long farcoreleft(void);
```

Description

farcoreleft returns a measure of the amount of unused memory in the far heap beyond the highest allocated block.

A tiny model program cannot make use of farcoreleft.

Return Value

farcoreleft returns the total amount of space left in the far heap, between the highest allocated block and the end of available memory.

See Also
coreleft
farcalloc (in BCW Help)
farmalloc (in BCW Help)

farheapcheck

See Also Example Portability

Syntax

```
#include <alloc.h>
int farheapcheck(void);
```

Description

farheapcheck walks through the far heap and examines each block, checking its pointers, size, and other critical attributes.

Return Value

The return value is less than zero for an error and greater than zero for success.

_HEAPEMPTY is returned if there is no heap (value 1).

_HEAPOK is returned if the heap is verified (value 2).

_HEAPCORRUPT is returned if the heap has been corrupted (value -1).

See Also
heapcheck (in BCW Help)

farheapcheckfree

See Also Example Portability

Syntax

```
#include <alloc.h>
int farheapcheckfree(unsigned int fillvalue);
```

Return Value

The return value is less than zero for an error and greater than zero for success.

_HEAPEMPTY is returned if there is no heap (value 1).

_HEAPOK is returned if the heap is accurate (value 2).

_HEAPCORRUPT is returned if the heap has been corrupted (value -1).

_BADVALUE is returned if a value other than the fill value was found (value -3).

See Also
farheapfillfree
heapcheckfree (in BCW Help)

farheapchecknode

See Also Example Portability

Syntax

```
#include <alloc.h>
int farheapchecknode(void *node);
```

Description

If a node has been freed and farheapchecknode is called with a pointer to the freed block, farheapchecknode can return _BADNODE rather than the expected _FREEENTRY. This is because adjacent free blocks on the heap are merged, and the block in question no longer exists.

Return Value

The return value is less than zero for an error and greater than zero for success.

_HEAPEMPTY is returned if there is no heap (value 1).

_HEAPCORRUPT is returned if the heap has been corrupted (value -1).

_BADNODE is returned if the node could not be found (value -2).

_FREEENTRY is returned if the node is a free block (value 3).

_USEDENTRY is returned if the node is a used block (value 4).

See Also
heapchecknode (in BCW Help)

farheapfillfree

See Also Example Portability

Syntax

```
#include <alloc.h>
int farheapfillfree(unsigned int fillvalue);
```

Return Value

The return value is less than zero for an error and greater than zero for success.

_HEAPEMPTY is returned if there is no heap (value 1).

_HEAPOK is returned if the heap is accurate (value 2).

_HEAPCORRUPT is returned if the heap has been corrupted (value -1).

See Also
farheapcheckfree
heapfillfree (in BCW Help)

farheapwalk

See Also Example Portability

Syntax

```
#include <alloc.h>
int farheapwalk(struct farheapinfo *hi);
```

Description

farheapwalk assumes the heap is correct. Use farheapcheck to verify the heap before using farheapwalk. _HEAPOK is returned with the last block on the heap. _HEAPEND will be returned on the next call to farheapwalk.

farheapwalk receives a pointer to a structure of type heapinfo (defined in alloc.h). For the first call to farheapwalk, set the hi.ptr field to null. farheapwalk returns with hi.ptr containing the address of the first block. hi.size holds the size of the block in bytes. hi.in_use is a flag that is set if the block is currently in use.

Return Value

_HEAPEMPTY is returned if there is no heap (value 1).

_HEAPOK is returned if the heapinfo block contains valid data (value 2).

_HEAPEND is returned if the end of the heap has been reached (value 5).

See Also
heapwalk (in BCW Help)

freemem, _dos_freemem

See Also Example Portability

Syntax

```
#include <dos.h>
int freemem(unsigned segx);
unsigned _dos_freemem(unsigned segx);
```

Description

freemem frees a memory block allocated by a previous call to allocmem. _dos_freemem frees a memory block allocated by a previous call to _dos_allocmem. segx is the segment address of that block.

Return Value

freemem and _dos_freemem return 0 on success.

In the event of error, freemem returns -1 and sets errno.

In the event of error, _dos_freemem returns the DOS error code and sets errno.

In the event of error, these functions set global variable errno to

ENOMEM Insufficient memory

See Also
allocmem
_dos_allocmem
free (in BCW Help)

harderr, hardresume, hardretn

See Also Example Portability

Syntax

```
#include <dos.h>
void harderr(int (*handler)());
void hardresume(int axret);
void hardretn(int retn);
```

Description

The error handler established by harderr can call hardresume to return to DOS. The return value of the rescode (result code) of hardresume contains an abort (2), retry (1), or ignore (0) indicator. The abort is accomplished by invoking DOS interrupt 0x23, the control-break interrupt.

The error handler established by harderr can return directly to the application program by calling hardretn. The returned value is whatever value you passed to hardretn.

harderr establishes a hardware error handler for the current program. This error handler is invoked whenever an interrupt 0x24 occurs. (See your DOS reference manuals for a discussion of the interrupt.)

The function pointed to by handler is called when such an interrupt occurs. The handler function is called with the following arguments:

```
handler(int errval, int ax, int bp, int si);
```

errval is the error code set in the DI register by DOS. ax, bp, and si are the values DOS sets for the AX, BP, and SI registers, respectively.

- ax indicates whether a disk error or other device error was encountered. If ax is nonnegative, a disk error was encountered; otherwise, the error was a device error. For a disk error, ax ANDed with 0x00FF gives the failing drive number (0 equals A, 1 equals B, and so on).
- bp and si together point to the device driver header of the failing driver. bp contains the segment address, and si the offset.

The function pointed to by handler is not called directly. harderr establishes a DOS interrupt handler that calls the function.

The handler can issue DOS calls 1 through 0xC; any other DOS call corrupts DOS. In particular, any of the C standard I/O or UNIX-emulation I/O calls cannot be used.

The handler must return 0 for ignore, 1 for retry, and 2 for abort.

Return Value

None.

See Also
peek (in BCW Help)
poke (in BCW Help)

_harderr

See Also Example Portability

Syntax

```
#include <dos.h>
void _harderr(int (far *handler)());
```

Description

_harderr establishes a hardware error handler for the current program. This error handler is invoked whenever an interrupt 0x24 occurs. (See your DOS reference manuals for a discussion of the interrupt.)

The function pointed to by handler is called when such an interrupt occurs. The handler function is called with the following arguments:

void far handler(unsigned deverr, unsigned errval, unsigned far *devhdr);

- deverr is the device error code (passed to the handler by DOS in the AX register).
- errval is the error code (passed to the handler by DOS in the DI register).
- devhdr a far pointer to the driver header of the device that caused the error (passed to the handler by DOS in the BP:SI register pair).

The handler should use these arguments instead of referring directly to the CPU registers.

deverr indicates whether a disk error or other device error was encountered. If bit 15 of deverr is 0, a disk error was encountered. Otherwise, the error was a device error. For a disk error, deverr ANDed with 0x00FF give the failing drive number (0 equals A, 1 equals B, and so on).

The function pointed to by handler is not called directly. _harderr establishes a DOS interrupt handler that calls the function.

The handler can issue DOS calls 1 through 0xC; any other DOS call corrupts DOS. In particular, any of the C standard I/O or UNIX-emulation I/O calls cannot be used.

The handler does not return a value, and it must exit using _hardretn or _hardresume.

Return Value

None.

See Also

_hardresume

_hardretn

_hardresume

See Also Example Portability

Syntax

```
#include <dos.h>
void _hardresume(int rescode);
```

Description

The error handler established by _harderr can call _hardresume to return to DOS. The return value of the rescode (result code) of _hardresume contains one of the following values:

_HARDERR_ABORT	Abort the program by invoking DOS interrupt 0x23, the control-break interrupt.
_HARDERR_IGNORE	Ignore the error.
_HARDERR_RETRY	Retry the operation.
_HARDERR_FAIL	Fail the operation.

Return Value

The _hardresume function does not return a value, and does not return to the caller.

See Also

_harderr

_hardretn

_hardretn

See Also Example Portability

Syntax

```
#include <dos.h>
void _hardretn(int retn);
```

Description

The error handler established by _harderr can return directly to the application program by calling _hardretn.

If the DOS function that caused the error is less than 0x38, and it is a function that can indicate an error condition, then _hardretn will return to the application program with the AL register set to 0xFF. The retn argument is ignored for all DOS functions less than 0x38.

If the DOS function is greater than or equal to 0x38, the retn argument should be a DOS error code; it is returned to the application program in the AX register. The carry flag is also set to indicate to the application that the operation resulted in an error.

Return Value

The _hardresume function does not return a value, and does not return to the caller.

See Also

_harderr

_hardresume

keep, _dos_keep

See Also Example Portability

Syntax

```
#include <dos.h>
void keep(unsigned char status, unsigned size);
void _dos_keep(unsigned char status, unsigned size);
```

Description

keep and _dos_keep return to DOS with the exit status in status. The current program remains resident, however. The program is set to size paragraphs in length, and the remainder of the memory of the program is freed.

keep and _dos_keep can be used when installing TSR programs. keep and _dos_keep use DOS function 0x31.

Before _dos_keep exits, it calls any registered "exit functions" (posted with atexit), flushes file buffers, and restores interrupt vectors modified by the startup code.

Return Value

None.

See Also
abort (in BCW Help)
exit (in BCW Help)

nosound

See Also Example Portability

Syntax

```
#include <dos.h>
void nosound(void);
```

Description

Turns the speaker off after it has been turned on by a call to sound.

Return Value

None.

See Also
delay
sound

_OvrInitEms

See Also Example Portability

Syntax

```
#include <dos.h>
int cdecl far _OvrInitEms(unsigned emsHandle, unsigned firstPage, unsigned
  pages);
```

Description

_OvrInitEms checks for the presence of expanded memory by looking for an EMS driver and allocating memory from it. If emsHandle is zero, the overlay manager allocates EMS pages and uses them for swapping. If emsHandle is not zero, then it should be a valid EMS handle; the overlay manager will use it for swapping. In that case, you can specify firstPage, where the swapping can start inside that area.

In both cases, a nonzero pages parameter gives the limit of the usable pages by the overlay manager.

Return Value

_OvrInitEms returns 0 if the overlay manager is able to use expanded memory for swapping.

See Also
_OvrInitExt
_ovrbuffer (global variable)

_OvrInitExt

See Also Example Portability

Syntax

```
#include <dos.h>
int cdecl far _OvrInitExt(unsigned long startAddress, unsigned long
  length);
```

Description

_OvrInitExt checks for the presence of extended memory, using the known methods to detect the presence of other programs using extended memory, and allocates memory from it. If startAddress is zero, the overlay manager determines the start address and uses, at most, the size of the overlays. If startAddress is not zero, then the overlay manager uses the extended memory above that address.

In both cases, a nonzero length parameter gives the limit of the usable extended memory by the overlay manager.

Return Value

_OvrInitExt returns 0 if the overlay manager is able to use extended memory for swapping.

See Also

_OvrInitEms

_ovrbuffer (global variable)

randbrd

See Also Example Portability

Syntax

```
#include <dos.h>
int randbrd(struct fcb *fcb, int rcnt);
```

Description

randbrd reads rcnt number of records using the open file control block (FCB) pointed to by fcb. The records are read into memory at the current disk transfer address (DTA). They are read from the disk record indicated in the random record field of the FCB. This is accomplished by calling DOS system call 0x27.

The actual number of records read can be determined by examining the random record field of the FCB. The random record field is advanced by the number of records actually read.

Return Value

The following values are returned, depending on the result of the randbrd operation:

0 All records are read.

1 End-of-file is reached and the last record read is complete.

2 Reading records would have wrapped around address 0xFFFF (as many records as possible are read).

3 End-of-file is reached with the last record incomplete.

See Also
getdta (in BCW Help)
randbwr
setdta (in BCW Help)

randbwr

See Also Example Portability

Syntax

```
#include <dos.h>
int randbwr(struct fcb *fcb, int rcnt);
```

Description

randbwr writes rcnt number of records to disk using the open file control block (FCB) pointed to by fcb. This is accomplished using DOS system call 0x28. If rcnt is 0, the file is truncated to the length indicated by the random record field.

The actual number of records written can be determined by examining the random record field of the FCB. The random record field is advanced by the number of records actually written.

Return Value

The following values are returned, depending upon the result of the randbwr operation:

0 All records are written.

1 There is not enough disk space to write the records (no records are written).

2 Writing records would have wrapped around address 0xFFFF (as many records as possible are written).

See Also
randbrd

sbrk

See Also Example Portability

Syntax

```
#include <alloc.h>
void *sbrk(int incr);
```

Description

sbrk adds incr bytes to the break value and changes the allocated space accordingly. incr can be negative, in which case the amount of allocated space is decreased.

sbrk will fail without making any change in the allocated space if such a change would result in more space being allocated than is allowable.

Return Value

Upon successful completion, sbrk returns the old break value. On failure, sbrk returns a value of -1, and the global variable errno is set to

ENOMEM	Not enough core

See Also
brk

setblock, _dos_setblock

See Also Example Portability

Syntax

```
#include <dos.h>
int setblock(unsigned segx, unsigned newsize);
unsigned _dos_setblock(unsigned newsize, unsigned segx, unsigned *maxp);
```

Description

setblock and _dos_setblock modify the size of a memory segment. segx is the segment address returned by a previous call to allocmem or _dos_allocmem. newsize is the new, requested size in paragraphs. If the sement cannot be changed to the new size, _dos_setblock stores the sizeof the largest possible segment at the location pointed to by maxp.

Return Value

setblock returns -1 on success. In the event of error, it returns the size of the largest possible block (in paragraphs), and the global variable _doserrno is set.

_dos_setblock returns 0 on success. In the event of error, it returns the DOS error code, and the global variable errno is set to the following:

ENOMEM	Not enough memory, or bad segment address

See Also
allocmem
freemem

sound

See Also Example Portability

Syntax

```
#include <dos.h>
void sound(unsigned frequency);
```

Description

sound turns on the PC speaker at a given frequency. frequency specifies the frequency of the sound in hertz (cycles per second). To turn the speaker off after a call to sound, call the function nosound.

See Also
delay
nosound

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+						

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+					

DOS Graphics Routines: Borland Graphics Interface (BGI)

A
B
C
D
E
F
G
I
L
M
O
P
R
S
T

The following functions compose the Borland Graphics Interface and are available only for 16-bit DOS applications. Use them to create onscreen graphics with text. They are defined in graphics.h unless indicated otherwise.

A

arc

B

bar

bar3d

C

circle

cleardevice

clearviewport

closegraph

D

detectgraph

drawpoly

E

ellipse

F

fillellipse

fillpoly

floodfill

G

getarccoords

getaspectratio

arc

See Also Example Portability

Syntax

```
#include <graphics.h>
void far arc(int x, int y, int stangle, int endangle, int radius);
```

Description

arc draws a circular arc in the current drawing color centered at (x,y) with a radius given by radius. The arc travels from stangle to endangle. If stangle equals 0 and endangle equals 360, the call to arc draws a complete circle.

The angle for arc is reckoned counterclockwise, with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.

The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.

If you are using a CGA in high resolution mode or a monochrome graphics adapter, the examples in online Help that show how to use graphics functions might not produce the expected results. If your system runs on a CGA or monochrome adapter, pass the value 1 to those functions that alter the fill or drawing color (setcolor, setfillstyle, and setlinestyle, for example), instead of a symbolic color constant (defined in graphics.h).

Return Value

None.

See Also

circle
ellipse
fillellipse
getarccoords
getaspectratio
graphresult
pieslice
sector

bar

See Also Example Portability

Syntax

```
#include <graphics.h>
#include <conio.h>
void far bar(int left, int top, int right, int bottom);
```

Description

bar draws a filled-in, rectangular, two-dimensional bar. The bar is filled using the current fill pattern and fill color. bar does not outline the bar; to draw an outlined two-dimensional bar, use bar3d with depth equal to 0.

The upper left and lower right corners of the rectangle are given by (left, top) and (right, bottom), respectively. The coordinates refer to pixels.

Return Value

None.

See Also
bar3d
rectangle
setcolor
setfillstyle
setlinestyle

bar3d

See Also Example Portability

Syntax

```
#include <graphics.h>
void far bar3d(int left, int top, int right, int bottom, int depth, int
  topflag);
```

Description

bar3d draws a three-dimensional rectangular bar, then fills it using the current fill pattern and fill color. The three-dimensional outline of the bar is drawn in the current line style and color. The bar's depth in pixels is given by depth. The topflag parameter governs whether a three-dimensional top is put on the bar. If topflag is nonzero, a top is put on; otherwise, no top is put on the bar (making it possible to stack several bars on top of one another).

The upper left and lower right corners of the rectangle are given by (left, top) and (right, bottom), respectively.

To calculate a typical depth for bar3d, take 25% of the width of the bar, like this:

```
bar3d(left,top,right,bottom, (right-left)/4,1);
```

Return Value

None.

See Also

bar

rectangle

setcolor

setfillstyle

setlinestyle

circle

See Also Example Portability

Syntax

```
#include <graphics.h>
void far circle(int x, int y, int radius);
```

Description

circle draws a circle in the current drawing color with its center at (x,y) and the radius given by radius.

The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.

If your circles are not perfectly round, adjust the aspect ratio.

Return Value

None.

See Also

arc

ellipse

fillellipse

getaspectratio

sector

setaspectratio

cleardevice

See Also Example Portability

Syntax

```
#include <graphics.h>
void far cleardevice(void);
```

Description

cleardevice erases (that is, fills with the current background color) the entire graphics screen and moves the CP (current position) to home (0,0).

Return Value

None.

See Also
clearviewport

closegraph

See Also Example Portability

Syntax

```
#include <graphics.h>
void far closegraph(void);
```

Description

closegraph deallocates all memory allocated by the graphics system, then restores the screen to the mode it was in before you called initgraph. (The graphics system deallocates memory, such as the drivers, fonts, and an internal buffer, through a call to _graphfreemem.)

Return Value

None.

See Also
initgraph
setgraphbufsize

clearviewport

See Also Example Portability

Syntax

```
#include <graphics.h>
void far clearviewport(void);
```

Description

clearviewport erases the viewport and moves the CP (current position) to home (0,0), relative to the viewport.

Return Value

None.

See Also

cleardevice

getviewsettings

setviewport

detectgraph

See Also Example Portability

Syntax

```
#include <graphics.h>
void far detectgraph(int far *graphdriver, int far *graphmode);
```

Description

detectgraph detects your system's graphics adapter and chooses the mode that provides the highest resolution for that adapter. If no graphics hardware is detected, *graphdriver is set to grNotDetected (-2), and graphresult returns grNotDetected (-2).

*graphdriver is an integer that specifies the graphics driver to be used. You can give it a value using a constant of the graphics_drivers enumeration type defined in graphics.h and listed as follows.

graphics_drivers constant	Numeric value	
DETECT	0	(requests autodetection)
CGA	1	
MCGA	2	
EGA	3	
EGA64	4	
EGAMONO	5	
IBM8514	6	
HERCMONO	7	
ATT400	8	
VGA	9	
PC3270	10	

*graphmode is an integer that specifies the initial graphics mode (unless *graphdriver equals DETECT; in which case, *graphmode is set to the highest resolution available for the detected driver). You can give *graphmode a value using a constant of the graphics_modes enumeration type defined in graphics.h and listed as follows.

Graphics driver	graphics_modes	Value	Column x Row	Palette	Pages
CGA	CGAC0	0	320 x 200	C0	1
	CGAC1	1	320 x 200	C1	1
	CGAC2	2	320 x 200	C2	1
	CGAC3	3	320 x 200	C3	1
	CGAHI	4	640 x 200	2 color	1
MCGA	MCGAC0	0	320 x 200	C0	1
	MCGAC1	1	320 x 200	C1	1
	MCGAC2	2	320 x 200	C2	1
	MCGAC3	3	320 x 200	C3	1
	MCGAMED	4	640 x 200	2 color	1
	MCGAHI	5	640 x 480	2 color	1
EGA	EGALO	0	640 x 200	16 color	4
	EGAHI	1	640 x 350	16 color	2
EGA64	EGA64LO	0	640 x 200	16 color	1
	EGA64HI	1	640 x 350	4 color	1
EGA-MONO	EGAMONOHI	3	640 x 350	2 color	1 *

	EGAMONOHI	3	640 x 350	2 color	2 **
HERC	HERCMONOHI	0	720 x 348	2 color	2
ATT400	ATT400C0	0	320 x 200	C0	1
	ATT400C1	1	320 x 200	C1	1
	ATT400C2	2	320 x 200	C2	1
	ATT400C3	3	320 x 200	C3	1
	ATT400MED	4	640 x 200	2 color	1
	ATT400HI	5	640 x 400	2 color	1
VGA	VGALO	0	640 x 200	16 color	2
	VGAMED	1	640 x 350	16 color	2
	VGAHI	2	640 x 480	16 color	1
PC3270	PC3270HI	0	720 x 350	2 color	1
IBM8514	IBM8514HI	0	640 x 480	256 color	
	IBM8514LO	0	1024 x 768	256 color	

*64K on EGAMONO card
**256K on EGAMONO card

Note: The main reason to call detectgraph directly is to override the graphics mode that detectgraph recommends to initgraph.

Return Value

None.

See Also
graphresult
initgraph

drawpoly

See Also Example Portability

Syntax

```
#include <graphics.h>
void far drawpoly(int numpoints, int far *polypoints);
```

Description

drawpoly draws a polygon with numpoints points, using the current line style and color.

*polypoints points to a sequence of (numpoints * 2) integers. Each pair of integers gives the x and y coordinates of a point on the polygon.

In order to draw a closed figure with n vertices, you must pass n + 1 coordinates to drawpoly where the nth coordinate is equal to the 0th.

Return Value

None.

See Also

fillpoly

floodfill

graphresult

setwritemode

ellipse

See Also Example Portability

Syntax

```
#include <graphics.h>
void far ellipse(int x, int y, int stangle, int endangle, int xradius, int
  yradius);
```

Description

ellipse draws an elliptical arc in the current drawing color with its center at (x,y) and the horizontal and vertical axes given by xradius and yradius, respectively. The ellipse travels from stangle to endangle. If stangle equals 0 and endangle equals 360, the call to ellipse draws a complete ellipse.

The angle for ellipse is reckoned counterclockwise, with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.

The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.

Return Value

None.

See Also

arc

circle

fillellipse

sector

fillellipse

See Also Example Portability

Syntax

```
#include <graphics.h>
void far fillellipse(int x, int y, int xradius, int yradius);
```

Description

Draws an ellipse using (x,y) as a center point and xradius and yradius as the horizontal and vertical axes, and fills it with the current fill color and fill pattern.

Return Value

None.

See Also

arc

circle

ellipse

pieslice

fillpoly

See Also Example Portability

```
#include <graphics.h>
void far fillpoly(int numpoints, int far *polypoints);
```

Description

fillpoly draws the outline of a polygon with numpoints points in the current line style and color (just as drawpoly does), then fills the polygon using the current fill pattern and fill color.

polypoints points to a sequence of (numpoints * 2) integers. Each pair of integers gives the x and y coordinates of a point on the polygon.

Return Value

None.

See Also

drawpoly

floodfill

graphresult

setfillstyle

floodfill

See Also Example Portability

Syntax

```
#include <graphics.h>
void far floodfill(int x, int y, int border);
```

Description

floodfill fills an enclosed area on bitmap devices. (x,y) is a "seed point" within the enclosed area to be filled. The area bounded by the color border is flooded with the current fill pattern and fill color. If the seed point is within an enclosed area, the inside will be filled. If the seed is outside the enclosed area, the exterior will be filled.

Use fillpoly instead of floodfill whenever possible so that you can maintain code compatibility with future versions.

floodfill does not work with the IBM-8514 driver.

Return Value

If an error occurs while flooding a region, graphresult returns a value of -7.

See Also

drawpoly

fillpoly

graphresult

setcolor

setfillstyle

getarccoords

See Also Example Portability

```
#include <graphics.h>
void far getarccoords(struct arccoordstype far *arccoords);
```

Description

getarccoords fills in the arccoordstype structure pointed to by arccoords with information about the last call to arc. The arccoordstype structure is defined in graphics.h as follows:

```
struct arccoordstype {
   int x, y;
   int xstart, ystart, xend, yend;
};
```

The members of this structure are used to specify the center point (x,y), the starting position (xstart, ystart), and the ending position (xend, yend) of the arc. These values are useful if you need to make a line meet at the end of an arc.

Return Value

None.

See Also

arc

fillellipse

sector

getaspectratio

See Also Example Portability

Syntax

```
#include <graphics.h>
void far getaspectratio(int far *xasp, int far *yasp);
```

Description

The y aspect factor, *yasp, is normalized to 10,000. On all graphics adapters except the VGA, *xasp (the x aspect factor) is less than *yasp because the pixels are taller than they are wide. On the VGA, which has "square" pixels, *xasp equals *yasp. In general, the relationship between *yasp and *xasp can be stated as

*yasp = 10,000

*xasp <= 10,000

getaspectratio gets the values in *xasp and *yasp.

Return Value

None.

See Also

arc

circle

ellipse

fillellipse

pieslice

sector

setaspectratio

getbkcolor

See Also Example Portability

Syntax

```
#include <graphics.h>
nt far getbkcolor(void);
```

Description

getbkcolor returns the current background color. (See the table in setbkcolor for details.)

Return Value

getbkcolor returns the current background color.

See Also

getcolor

getmaxcolor

getpalette

setbkcolor

getcolor

See Also Example Portability

Syntax

```
#include <graphics.h>
int far getcolor(void);
```

Description

getcolor returns the current drawing color.

The drawing color is the value to which pixels are set when lines and so on are drawn. For example, in CGAC0 mode, the palette contains four colors: the background color, light green, light red, and yellow. In this mode, if getcolor returns 1, the current drawing color is light green.

Return Value

getcolor returns the current drawing color.

See Also

getbkcolor

getmaxcolor

getpalette

setcolor

getimage

See Also Example Portability

Syntax

```
#include <graphics.h>
void far getimage(int left, int top, int right, int bottom, void far
  *bitmap);
```

Description

getimage copies an image from the screen to memory.

left, top, right, and bottom define the screen area to which the rectangle is copied. bitmap points to the area in memory where the bit image is stored. The first two words of this area are used for the width and height of the rectangle; the remainder holds the image itself.

Return Value

None.

See Also

imagesize

putimage

putpixel

getmaxx

See Also Example Portability

Syntax

```
#include <graphics.h>
int far getmaxx(void);
```

Description

getmaxx returns the maximum (screen-relative) x value for the current graphics driver and mode.

For example, on a CGA in 320*200 mode, getmaxx returns 319. getmaxx is invaluable for centering, determining the boundaries of a region onscreen, and so on.

Return Value

getmaxx returns the maximum x screen coordinate.

See Also
getmaxy
getx

getmaxy

See Also Example Portability

Syntax

```
#include <graphics.h>
int far getmaxy(void);
```

Description

getmaxy returns the maximum (screen-relative) y value for the current graphics driver and mode.

For example, on a CGA in 320*200 mode, getmaxy returns 199. getmaxy is invaluable for centering, determining the boundaries of a region onscreen, and so on.

Return Value

getmaxy returns the maximum y screen coordinate.

See Also

getmaxx

getx

gety

getmodename

See Also Example Portability

Syntax

```
#include <graphics.h>
char *far getmodename(int mode_number);
```

Description

getmodename accepts a graphics mode number as input and returns a string containing the name of the corresponding graphics mode. The mode names are embedded in each driver. The return values ("320*200 CGA P1," "640*200 CGA", and so on) are useful for building menus or displaying status.

Return Value

getmodename returns a pointer to a string with the name of the graphics mode.

See Also
getmaxmode
getmoderange

getmoderange

See Also Example Portability

Syntax

```
#include <graphics.h>
void far getmoderange(int graphdriver, int far *lomode, int far *himode);
```

Description

getmoderange gets the range of valid graphics modes for the given graphics driver, graphdriver. The lowest permissible mode value is returned in *lomode, and the highest permissible value is *himode. If graphdriver specifies an invalid graphics driver, both *lomode and *himode are set to -1. If the value of graphdriver is -1, the currently loaded driver modes are given.

Return Value

None.

See Also

getgraphmode

getmaxmode

getmodename

initgraph

setgraphmode

getpalette

See Also Example Portability

Syntax

```
#include <graphics.h>
void far getpalette(struct palettetype far *palette);
```

Description

getpalette fills the palettetype structure pointed to by palette with information about the current palette's size and colors.

The MAXCOLORS constant and the palettetype structure used by getpalette are defined in graphics.h as follows:

```
#define MAXCOLORS  15

struct palettetype {
   unsigned char size;
   signed char colors[MAXCOLORS + 1];
};
```

size gives the number of colors in the palette for the current graphics driver in the current mode.

colors is an array of size bytes containing the actual raw color numbers for each entry in the palette.

getpalette cannot be used with the IBM-8514 driver.

Return ValueNone.

See Also

getbkcolor

getcolor

getdefaultpalette

getmaxcolor

setallpalette

setpalette

getpalettesize

See Also Example Portability

Syntax

```
#include <graphics.h>
int far getpalettesize(void);
```

Description

getpalettesize is used to determine how many palette entries can be set for the current graphics mode. For example, the EGA in color mode returns 16.

Return Value

getpalettesize returns the number of palette entries in the current palette.

See Also
setpalette
setallpalette

getpixel

See Also Example Portability

Syntax

```
#include <graphics.h>
unsigned far getpixel(int x, int y);
```

Description

getpixel gets the color of the pixel located at (x,y).

Return Value

getpixel returns the color of the given pixel.

See Also
getimage
putpixel

getviewsettings

See Also Example Portability

Syntax

```
#include <graphics.h>
void far getviewsettings(struct viewporttype far *viewport);
```

Description

getviewsettings fills the viewporttype structure pointed to by viewport with information about the current viewport.

The viewporttype structure used by getviewport is defined in graphics.h as follows:

```
struct viewporttype {
   int left, top, right, bottom;
   int clip;
};
```

Return Value

None.

See Also

clearviewport

getx

gety

setviewport

getx

See Also Example Portability

Syntax

```
#include <graphics.h>
int far getx(void);
```

Description

getx finds the current graphics position's x-coordinate. The value is viewport-relative.

Return Value

getx returns the x-coordinate of the current position.

See Also

getmaxx

getmaxy

getviewsettings

gety

moveto

gety

See Also Example Portability

Syntax

```
#include <graphics.h>
nt far gety(void);
```

Description

gety returns the current graphics position's y-coordinate. The value is viewport-relative.

Return Value

gety returns the y-coordinate of the current position.

See Also

getmaxx

getmaxy

getviewsettings

getx

moveto

graphdefaults

See Also Example Portability

Syntax

```
#include <graphics.h>
void far graphdefaults(void);
```

Description

graphdefaults resets all graphics settings to their defaults:

- sets the viewport to the entire screen.
- moves the current position to (0,0).
- sets the default palette colors, background color, and drawing color.
- sets the default fill style and pattern.
- sets the default text font and justification.

Return ValueNone.

See Also
initgraph
setgraphmode

grapherrormsg

See Also Example Portability

Syntax

```
#include <graphics.h>
char * far grapherrormsg(int errorcode);
```

Description

grapherrormsg returns a pointer to the error message string associated with errorcode, the value returned by graphresult.

Refer to the entry for errno in the Library Reference, Chapter 4, for a list of error messages and mnemonics.

Return Value

grapherrormsg returns a pointer to an error message string.

See Also
graphresult

_graphfreemem

See Also Example Portability

Syntax

```
#include <graphics.h>
void far _graphfreemem(void far *ptr, unsigned size);
```

Description

The graphics library calls _graphfreemem to release memory previously allocated through _graphgetmem. You can choose to control the graphics library memory management by simply defining your own version of _graphfreemem (you must declare it exactly as shown in the declaration). The default version of this routine merely calls free.

Return Value

None.

See Also

_graphgetmem

setgraphbufsize

_graphgetmem

See Also Example Portability

Syntax

```
#include <graphics.h>
void far * far _graphgetmem(unsigned size);
```

Description

Routines in the graphics library (not the user program) normally call _graphgetmem to allocate memory for internal buffers, graphics drivers, and character sets. You can choose to control the memory management of the graphics library by defining your own version of _graphgetmem (you must declare it exactly as shown in the declaration). The default version of this routine merely calls malloc.

Return Value

None.

See Also

_graphfreemem

initgraph

setgraphbufsize

graphresult

See Also Example Portability

Syntax

```
#include <graphics.h>
int far graphresult(void);
```

Description

graphresult returns the error code for the last graphics operation that reported an error and resets the error level to grOk.

The following table lists the error codes returned by graphresult. The enumerated type graph_errors defines the errors in this table. graph_errors is declared in graphics.h.

code	constant	Corresponding error message string
0	grOk	No error
-1	grNoInitGraph	(BGI) graphics not installed (use initgraph)
-2	grNotDetected	Graphics hardware not detected
-3	grFileNotFound	Device driver file not found
-4	grInvalidDriver	Invalid device driver file
-5	grNoLoadMem	Not enough memory to load driver
-6	grNoScanMem	Out of memory in scan fill
-7	grNoFloodMem	Out of memory in flood fill
-8	grFontNotFound	Font file not found
-9	igrNoFontMem	Not enough memory to load font
-10	grInvalidMode	Invalid graphics mode for selected driver
-11	grError	Graphics error
-12	grIOerror	Graphics I/O error
-13	grInvalidFont	Invalid font file
-14	grInvalidFontNum	Invalid font number
-15	grInvalidDeviceNum	Invalid device number
-18	grInvalidVersion	Invalid version number

Note: The variable maintained by graphresult is reset to 0 after graphresult has been called. Therefore, you should store the value of graphresult into a temporary variable and then test it.

Return Value

graphresult returns the current graphics error number, an integer in the range -15 to 0; grapherrormsg returns a pointer to a string associated with the value returned by graphresult.

See Also
detectgraph
drawpoly
fillpoly
floodfill
grapherrormsg
initgraph
pieslice
registerbgidriver
registerbgifont
setallpalette
setcolor
setfillstyle
setgraphmode
setlinestyle
setpalette
settextjustify
settextstyle
setusercharsize
setviewport
setvisualpage

getdefaultpalette

See Also Example Portability

Syntax

```
#include <graphics.h>
struct palettetype *far getdefaultpalette(void);
```

Description

getdefaultpalette finds the palettetype structure that contains the palette initialized by the driver during initgraph.

Return Value

getdefaultpalette returns a pointer to the default palette set up by the current driver when that driver was initialized.

See Also
getpalette
initgraph

getdrivername

See Also Example Portability

Syntax

```
#include <graphics.h>
char *far getdrivername(void);
```

Description

After a call to initgraph, getdrivername returns the name of the driver that is currently loaded.

Return Value

getdrivername returns a pointer to a string with the name of the currently loaded graphics driver.

See Also
initgraph

gettextsettings

See Also Example Portability

Syntax

```
#include <graphics.h>
void far gettextsettings(struct textsettingstype far *texttypeinfo);
```

Description

gettextsettings fills the textsettingstype structure pointed to by textinfo with information about the current text font, direction, size, and justification.

The textsettingstype structure used by gettextsettings is defined in graphics.h as follows:

```
struct textsettingstype {
   int font;
   int direction;
   int charsize;
   int horiz;
   int vert;
};
```

See settextstyle for a description of these fields.

Return Value

None.

See Also

outtext
outtextxy
registerbgifont
settextjustify
settextstyle
setusercharsize
textheight
textwidth

getfillpattern

See Also Example Portability

Syntax

```
#include <graphics.h>
void far getfillpattern(char far *pattern);
```

Description

getfillpattern copies the user-defined fill pattern, as set by setfillpattern, into the 8-byte area pointed to by pattern.

pattern is a pointer to a sequence of 8 bytes, with each byte corresponding to 8 pixels in the pattern. Whenever a bit in a pattern byte is set to 1, the corresponding pixel will be plotted. For example, the following user-defined fill pattern represents a checkerboard:

```
char checkboard[8] = {
   0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55
};
```

Return Value

None.

See Also
getfillsettings
setfillpattern

getfillsettings

See Also Example Portability

Syntax

```
#include <graphics.h>
void far getfillsettings(struct fillsettingstype far *fillinfo);
```

Description

getfillsettings fills in the fillsettingstype structure pointed to by fillinfo with information about the current fill pattern and fill color. The fillsettingstype structure is defined in graphics.h as follows:

```
struct fillsettingstype {
   int pattern;            /* current fill pattern */
   int color;              /* current fill color */
};
```

The functions bar, bar3d, fillpoly, floodfill, and pieslice all fill an area with the current fill pattern in the current fill color. There are 11 predefined fill pattern styles (such as solid, crosshatch, dotted, and so on). Symbolic names for the predefined patterns are provided by the enumerated type fill_patterns in graphics.h (see the following table). In addition, you can define your own fill pattern.

If pattern equals 12 (USER_FILL), then a user-defined fill pattern is being used; otherwise, pattern gives the number of a predefined pattern. See Fill_Patterns.

See Also

getfillpattern

setfillpattern

setfillstyle

Fill Patterns

The enumerated type fill_patterns, defined in graphics.h, gives names for the predefined fill patterns, plus an indicator for a user-defined pattern.

Name	Value	Description
EMPTY_FILL	0	Fill with background color
SOLID_FILL	1	Solid fill
LINE_FILL	2	Fill with ---
LTSLASH_FILL	3	Fill with ///
SLASH_FILL	4	Fill with ///, thick lines
BKSLASH_FILL	5	Fill with \\\, thick lines
LTBKSLASH_FILL	6	Fill with \\\
HATCH_FILL	7	Light hatch fill
XHATCH_FILL	8	Heavy crosshatch fill
INTERLEAVE_FILL	9	Interleaving line fill
WIDE_DOT_FILL	10	Widely spaced dot fill
CLOSE_DOT_FILL	11	Closely spaced dot fill
USER_FILL	12	User-defined fill pattern

Note: All but EMPTY_FILL fill with the current fill color; EMPTY_FILL uses the current background color.

Return Value

None.

getgraphmode

See Also Example Portability

Syntax

```
#include <graphics.h>
int far getgraphmode(void);
```

Description

Your program must make a successful call to initgraph before calling getgraphmode.

The enumeration graphics_mode, defined in graphics.h, gives names for the predefined graphics modes. For a table listing these enumeration values, refer to the description for initgraph.

Return Value

getgraphmode returns the graphics mode set by initgraph or setgraphmode.

See Also

getmoderange

restorecrtmode

setgraphmode

getlinesettings

See Also Example Portability

Syntax

```
#include <graphics.h>
void far getlinesettings(struct linesettingstype far *lineinfo);
```

Description

getlinesettings fills a linesettingstype structure pointed to by lineinfo with information about the current line style, pattern, and thickness.

The linesettingstype structure is defined in graphics.h as follows:

```
struct linesettingstype {
   int linestyle;
   unsigned upattern;
   int thickness;
};
```

linestyle specifies in which style subsequent lines will be drawn (such as solid, dotted, centered, dashed). The enumeration line_styles, defined in graphics.h, gives names to these operators:

Name	Value	Description
SOLID_LINE	0	Solid line
DOTTED_LINE	1	Dotted line
CENTER_LINE	2	Centered line
DASHED_LINE	3	Dashed line
USERBIT_LINE	4	User-defined line style

thickness specifies whether the width of subsequent lines drawn will be normal or thick.

Name	Value	Description
NORM_WIDTH	1	1 pixel wide
THICK_WIDTH	3	3 pixels wide

upattern is a 16-bit pattern that applies only if linestyle is USERBIT_LINE (4). In that case, whenever a bit in the pattern word is 1, the corresponding pixel in the line is drawn in the current drawing color. For example, a solid line corresponds to a upattern of 0xFFFF (all pixels drawn), while a dashed line can correspond to a upattern of 0x3333 or 0x0F0F. If the linestyle parameter to setlinestyle is not USERBIT_LINE (!=4), the upattern parameter must still be supplied but is ignored.

Return Value

None.

See Also
setlinestyle

getmaxcolor

See Also Example Portability

Syntax

```
#include <graphics.h>
int far getmaxcolor(void);
```

Description

getmaxcolor returns the highest valid color value for the current graphics driver and mode that can be passed to setcolor.

For example, on a 256K EGA, getmaxcolor always returns 15, which means that any call to setcolor with a value from 0 to 15 is valid. On a CGA in high-resolution mode or on a Hercules monochrome adapter, getmaxcolor returns a value of 1.

Return Value

getmaxcolor returns the highest available color value.

See Also

getbkcolor

getcolor

getpalette

getpalettesize

setcolor

getmaxmode

See Also Example Portability

Syntax

```
#include <graphics.h>
int far getmaxmode(void);
```

Description

getmaxmode lets you find out the maximum mode number for the currently loaded driver, directly from the driver. This gives it an advantage over getmoderange, which works for Borland drivers only. The minimum mode is 0.

Return Value

getmaxmode returns the maximum mode number for the current driver.

See Also
getmodename
getmoderange

imagesize

See Also Example Portability

Syntax

```
#include <graphics.h>
unsigned far imagesize(int left, int top, int right, int bottom);
```

Description

imagesize determines the size of the memory area required to store a bit image. If the size required for the selected image is greater than or equal to 64K - 1 bytes, imagesize returns 0xFFFF (-1).

Return Value

imagesize returns the size of the required memory area in bytes.

See Also
getimage
putimage

initgraph

See Also Example Portability

Syntax

```
#include <graphics.h>
void far initgraph(int far *graphdriver, int far *graphmode, char far
  *pathtodriver);
```

Description

initgraph initializes the graphics system by loading a graphics driver from disk (or validating a registered driver), and putting the system into graphics mode.

To start the graphics system, first call the initgraph function. initgraph loads the graphics driver and puts the system into graphics mode. You can tell initgraph to use a particular graphics driver and mode, or to autodetect the attached video adapter at run time and pick the corresponding driver.

If you tell initgraph to autodetect, it calls detectgraph to select a graphics driver and mode. initgraph also resets all graphics settings to their defaults (current position, palette, color, viewport, and so on) and resets graphresult to 0.

Normally, initgraph loads a graphics driver by allocating memory for the driver (through _graphgetmem), then loading the appropriate .BGI file from disk. As an alternative to this dynamic loading scheme, you can link a graphics driver file (or several of them) directly into your executable program file.

pathtodriver specifies the directory path where initgraph looks for graphics drivers. initgraph first looks in the path specified in pathtodriver, then (if they are not there) in the current directory. Accordingly, if pathtodriver is null, the driver files (*.BGI) must be in the current directory. This is also the path settextstyle searches for the stroked character font files (*.CHR).

*graphdriver is an integer that specifies the graphics driver to be used. You can give it a value using a constant of the graphics_drivers enumeration type, which is defined in graphics.h and listed below.

graphics_drivers constant	Numeric value
DETECT	0 (requests autodetection)
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

*graphmode is an integer that specifies the initial graphics mode (unless *graphdriver equals DETECT, in which case *graphmode is set by initgraph to the highest resolution available for the detected driver). You can give *graphmode a value using a constant of the graphics_modes enumeration type, which is defined in graphics.h and listed below.

graphdriver and graphmode must be set to valid values from the following tables, or you will get unpredictable results. The exception is graphdriver = DETECT.

Palette listings C0, C1, C2, and C3 refer to the four predefined four-color palettes available on CGA (and compatible) systems. You can select the background color (entry #0) in each of these palettes, but the other colors are fixed.

Palette number			
0	LIGHTGREEN	LIGHTRED	YELLOW
1	LIGHTCYAN	LIGHTMAGENTA	WHITE
2	GREEN	RED	BROWN
3	CYAN	MAGENTA	LIGHTGRAY

Color assigned to pixel value:	1	2	3

After a call to initgraph, *graphdriver is set to the current graphics driver, and *graphmode is set to the current graphics mode.

Graphics driver	graphics_modes	Value	Column x Row	Palette	Pages
CGA	CGAC0	0	320 x 200	C0	1
	CGAC1	1	320 x 200	C1	1
	CGAC2	2	320 x 200	C2	1
	CGAC3	3	320 x 200	C3	1
	CGAHI	4	640 x 200	2 color	1
MCGA	MCGAC0	0	320 x 200	C0	1
	MCGAC1	1	320 x 200	C1	1
	MCGAC2	2	320 x 200	C2	1
	MCGAC3	3	320 x 200	C3	1
	MCGAMED	4	640 x 200	2 color	1
	MCGAHI	5	640 x 480	2 color	1
EGA	EGALO	0	640 x 200	16 color	4
	EGAHI	1	640 x 350	16 color	2
EGA64	EGA64LO	0	640 x 200	16 color	1
	EGA64HI	1	640 x 350	4 color	1
EGA-MONO	EGAMONOHI	3	640 x 350	2 color	1 *
	EGAMONOHI	3	640 x 350	2 color	2 **
HERC	HERCMONOHI	0	720 x 348	2 color	2
ATT400	ATT400C0	0	320 x 200	C0	1
	ATT400C1	1	320 x 200	C1	1
	ATT400C2	2	320 x 200	C2	1
	ATT400C3	3	320 x 200	C3	1
	ATT400MED	4	640 x 200	2 color	1
	ATT400HI	5	640 x 400	2 color	1
VGA	VGALO	0	640 x 200	16 color	2
	VGAMED	1	640 x 350	16 color	2
	VGAHI	2	640 x 480	16 color	1
PC3270	PC3270HI	0	720 x 350	2 color	1
IBM8514	IBM8514HI	1	1024 x 768	256 color	
	IBM8514LO	0	640 x 480	256 color	

*64K on EGAMONO card
**256K on EGAMONO card

Return Value

initgraph always sets the internal error code; on success, it sets the code to 0. If an error occurred, *graphdriver is set to -2, -3, -4, or -5, and graphresult returns the same value as listed below:

grNotDetected	-2	Cannot detect a graphics card
grFileNotFound	-3	Cannot find driver file
grInvalidDriver	-4	Invalid driver

grNoLoadMem	-5	Insufficient memory to load driver

See Also
closegraph
detectgraph
getdefaultpalette
getdrivername
getgraphmode
getmoderange
graphdefaults
_graphgetmem
graphresult
installuserdriver
registerbgidriver
registerbgifont
restorecrtmode
setgraphbufsize
setgraphmode

installuserdriver

See Also Example Portability

Syntax

```
#include <graphics.h>
int far installuserdriver(char far *name, int huge (*detect)(void));
```

Description

installuserdriver lets you add a vendor-added device driver to the BGI internal table. The name parameter is the name of the new device-driver file (.BGI), and the detect parameter is a pointer to an optional autodetect function that can accompany the new driver. This autodetect function takes no parameters and returns an integer value.

There are two ways to use this vendor-supplied driver. Suppose you have a new video card called the Spiffy Graphics Array (SGA) and that the SGA manufacturer provided you with a BGI device driver (SGA.BGI). The easiest way to use this driver is to install it by calling installuserdriver and then passing the return value (the assigned driver number) directly to initgraph.

The other, more general way to use this driver is to link in an autodetect function that will be called by initgraph as part of its hardware-detection logic (presumably, the manufacturer of the SGA gave you this autodetect function). When you install the driver (by calling installuserdriver), you pass the address of this function, along with the device driver's file name.

After you install the device-driver file name and the SGA autodetect function, call initgraph and let it go through its normal autodetection process. Before initgraph calls its built-in autodetection function (detectgraph), it first calls the SGA autodetect function. If the SGA autodetect function doesn't find the SGA hardware, it returns a value of -11 (grError), and initgraph proceeds with its normal hardware detection logic (which can include calling any other vendor-supplied autodetection functions in the order in which they were "installed"). If, however, the autodetect function determines that an SGA is present, it returns a nonnegative mode number; then initgraph locates and loads SGA.BGI, puts the hardware into the default graphics mode recommended by the autodetect function, and finally returns control to your program.

You can install up to ten drivers at one time.

Return Value

The value returned by installuserdriver is the driver number parameter you would pass to initgraph in order to select the newly installed driver manually.

See Also
initgraph
registerbgidriver

installuserfont

See Also Example Portability

Syntax

```
#include <graphics.h>
int far installuserfont(char far *name);
```

Description

name is a filename in the current directory (pathname is not supported) of a font file containing a stroked font. Up to twenty fonts can be installed at one time.

Return Value

installuserfont returns a font ID number that can then be passed to settextstyle to select the corresponding font. If the internal font table is full, a value of -11 (grError) is returned.

See Also
settextstyle

line

See Also Example Portability

Syntax

```
#include <graphics.h>
void far line(int x1, int y1, int x2, int y2);
```

Description

line draws a line in the current color, using the current line style and thickness between the two points specified, (x1,y1) and (x2,y2), without updating the current position (CP).

Return Value

None.

See Also

getlinesettings

linerel

lineto

setcolor

setlinestyle

setwritemode

linerel

See Also Example Portability

Syntax

```
#include <graphics.h>
void far linerel(int dx, int dy);
```

Description

linerel draws a line from the CP to a point that is a relative distance (dx,dy) from the CP. The CP is advanced by (dx,dy).

Return Value

None.

See Also

getlinesettings

line

lineto

setcolor

setlinestyle

setwritemode

lineto

See Also Example Portability

Syntax

```
#include <graphics.h>
void far lineto(int x, int y);
```

Description

lineto draws a line from the CP to (x,y), then moves the CP to (x,y).

Return Value

None.

See Also

getlinesettings
line
linerel
setcolor
setlinestyle
setvisualpage
setwritemode

moverel

See Also Example Portability

Syntax

```
#include <graphics.h>
void far moverel(int dx, int dy);
```

Description

moverel moves the current position (CP) dx pixels in the x direction and dy pixels in the y direction.

Return Value

None.

See Also
moveto

moveto

See Also Example Portability

Syntax

```
#include <graphics.h>
void far moveto(int x, int y);
```

Description

moveto moves the current position (CP) to viewport position (x,y).

Return Value

None.

See Also
moverel

outtext

See Also Example Portability

Syntax

```
#include <graphics.h>
void far outtext(char far *textstring);
```

Description

outtext displays a text string in the viewport, using the current font, direction, and size.

outtext outputs textstring at the current position (CP). If the horizontal text justification is LEFT_TEXT and the text direction is HORIZ_DIR, the CP's x-coordinate is advanced by textwidth(textstring). Otherwise, the CP remains unchanged.

To maintain code compatibility when using several fonts, use textwidth and textheight to determine the dimensions of the string.

If a string is printed with the default font using outtext, any part of the string that extends outside the current viewport is truncated.

outtext is for use in graphics mode; it will not work in text mode.

Return Value

None.

See Also

gettextsettings

outtextxy

settextjustify

textheight

textwidth

outtextxy

See Also Example Portability

Syntax

```
#include <graphics.h>
void far outtextxy(int x, int y, char far *textstring);
```

Description

outtextxy displays a text string in the viewport at the given position (x, y), using the current justification settings and the current font, direction, and size.

To maintain code compatibility when using several fonts, use textwidth and textheight to determine the dimensions of the string.

If a string is printed with the default font using outtext or outtextxy, any part of the string that extends outside the current viewport is truncated.

outtextxy is for use in graphics mode; it will not work in text mode.

Return Value

None.

See Also

gettextsettings

outtext

textheight

textwidth

pieslice

See Also Example Portability

Syntax

```
#include <graphics.h>
void far pieslice(int x, int y, int stangle, int endangle, int radius);
```

Description

pieslice draws and fills a pie slice centered at (x,y) with a radius given by radius. The slice travels from stangle to endangle. The slice is outlined in the current drawing color and then filled using the current fill pattern and fill color.

The angles for pieslice are given in degrees. They are measured counterclockwise, with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.

If you're using a CGA or monochrome adapter, the examples in online Help that show how to use graphics functions might not produce the expected results. If your system runs on a CGA or monochrome adapter, use the value 1 (one) instead of the symbolic color constant, and see the second example under arc whis shows how to use the pieslice function.

Return Value

None.

See Also

fillellipse

graphresult

sector

setfillstyle

putimage

See Also Example Portability

Syntax

```
#include <graphics.h>
void far putimage(int left, int top, void far *bitmap, int op);
```

Description

putimage puts the bit image previously saved with getimage back onto the screen, with the upper left corner of the image placed at (left,top). bitmap points to the area in memory where the source image is stored.

The op parameter to putimage specifies a combination operator that controls how the color for each destination pixel onscreen is computed, based on the pixel already onscreen and the corresponding source pixel in memory.

The enumeration putimage_ops, as defined in graphics.h, gives names to these operators.

Name	Value	Description
COPY_PUT	0	Copy
XOR_PUT	1	Exclusive or
OR_PUT	2	Inclusive or
AND_PUT	3	And
NOT_PUT	4	Copy the inverse of the source

In other words, COPY_PUT copies the source bitmap image onto the screen, XOR_PUT XORs the source image with the image already onscreen, OR_PUT ORs the source image with that onscreen, and so on.

Return Value

None.

See Also
getimage
imagesize
putpixel
setvisualpage

putpixel

See Also Example Portability

Syntax

```
#include <graphics.h>
void far putpixel(int x, int y, int color);
```

Description1

putpixel plots a point in the color defined by color at (x,y).

Return Value

None.

See Also
getpixel
putimage

rectangle

See Also Example Portability

Syntax

```
#include <graphics.h>
void far rectangle(int left, int top, int right, int bottom);
```

Description

rectangle draws a rectangle in the current line style, thickness, and drawing color.

(left,top) is the upper left corner of the rectangle, and (right,bottom) is its lower right corner.

Return Value

None.

See Also

bar

bar3d

setcolor

setlinestyle

registerbgifont

See Also Example Portability

Syntax

```
#include <graphics.h>
int registerbgifont(void (*font)(void));
```

Description

Calling registerbgifont informs the graphics system that the font pointed to by font was included at link time. This routine checks the linked-in code for the specified font; if the code is valid, it registers the code in internal tables. Linked-in fonts are discussed in detail under BGIOBJ in UTIL.DOC included with your distribution disks.

By using the name of a linked-in font in a call to registerbgifont, you also tell the compiler (and linker) to link in the object file with that public name.

If you register a user-supplied font, you must pass the result of registerbgifont to settextstyle as the font number to be used.

Return Value

registerbgifont returns a negative graphics error code if the specified font is invalid. Otherwise, registerbgifont returns the font number of the registered font.

See Also

graphresult

initgraph

installuserdriver

registerbgidriver

settextstyle

registerbgidriver

See Also Example Portability

Syntax

```
#include <graphics.h>
int registerbgidriver(void (*driver)(void));
```

Description

registerbgidriver enables a user to load a driver file and "register" the driver. Once its memory location has been passed to registerbgidriver, initgraph uses the registered driver. A user-registered driver can be loaded from disk onto the heap, or converted to an .OBJ file (using BGIOBJ.EXE) and linked into the .EXE.

Calling registerbgidriver informs the graphics system that the driver pointed to by driver was included at link time. This routine checks the linked-in code for the specified driver; if the code is valid, it registers the code in internal tables. Linked-in drivers are discussed in detail in UTIL.DOC, included with your distribution disks.

By using the name of a linked-in driver in a call to registerbgidriver, you also tell the compiler (and linker) to link in the object file with that public name.

Return Value

registerbgidriver returns a negative graphics error code if the specified driver or font is invalid. Otherwise, registerbgidriver returns the driver number.

If you register a user-supplied driver, you must pass the result of registerbgidriver to initgraph as the driver number to be used.

See Also

graphresult

initgraph

installuserdriver

registerbgifont

restorecrtmode

See Also Example Portability

Syntax

```
#include <graphics.h>
void far restorecrtmode(void);
```

Description

restorecrtmode restores the original video mode detected by initgraph.

This function can be used in conjunction with setgraphmode to switch back and forth between text and graphics modes. textmode should not be used for this purpose; use it only when the screen is in text mode, to change to a different text mode.

Return Value

None.

See Also

getgraphmode

initgraph

setgraphmode

sector

See Also Example Portability

Syntax

```
#include <graphics.h>
void far sector(int x, int y, int stangle, int endangle, int xradius, int
  yradius);
```

Description

Draws and fills an elliptical pie slice using (x,y) as the center point, xradius and yradius as the horizontal and vertical radii, respectively, and drawing from stangle to endangle. The pie slice is outlined using the current color, and filled using the pattern and color defined by setfillstyle or setfillpattern.

The angles for sector are given in degrees. They are measured counter-clockwise with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.

If an error occurs while the pie slice is filling, graphresult returns a value of -6 (grNoScanMem).

Return Value

None.

See Also

arc

circle

ellipse

getarccoords

getaspectratio

graphresult

pieslice

setfillpattern

setfillstyle

setgraphbufsize

setactivepage

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setactivepage(int page);
```

Description

setactivepage makes page the active graphics page. All subsequent graphics output will be directed to that graphics page.

The active graphics page might not be the one you see onscreen, depending on how many graphics pages are available on your system. Only the EGA, VGA, and Hercules graphics cards support multiple pages.

Return Value

None.

See Also
setvisualpage

setallpalette

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setallpalette(struct palettetype far *palette);
```

Description

setallpalette sets the current palette to the values given in the palettetype structure pointed to by palette.

You can partially (or completely) change the colors in the EGA/VGA palette with setallpalette.

The MAXCOLORS constant and the palettetype structure used by setallpalette are defined in graphics.h as follows:

```
#define MAXCOLORS  15

struct palettetype {
   unsigned char size;
   signed char colors[MAXCOLORS + 1];
};
```

size gives the number of colors in the palette for the current graphics driver in the current mode.

colors is an array of size bytes containing the actual raw color numbers for each entry in the palette. If an element of colors is -1, the palette color for that entry is not changed.

The elements in the colors array used by setallpalette can be represented by symbolic constants which are defined in graphics.h. See Actual Color Table.

setallpalette cannot be used with the IBM-8514 driver.

Return Value

If invalid input is passed to setallpalette, graphresult returns -11 (grError), and the current palette remains unchanged.

See Also
getpalette
getpalettesize
graphresult
setbkcolor
setcolor
setpalette

setaspectratio

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setaspectratio(int xasp, int yasp);
```

Description

setaspectratio changes the default aspect ratio of the graphics system. The graphics system uses the aspect ratio to make sure that circles are round onscreen. If circles appear elliptical, the monitor is not aligned properly. You could correct this in the hardware by realigning the monitor, but it's easier to change in the software by using setaspectratio to set the aspect ratio. To obtain the current aspect ratio from the system, call getaspectratio.

Return Value

None.

See Also
circle
getaspectratio

setbkcolor

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setbkcolor(int color);
```

Description

setbkcolor sets the background to the color specified by color. The argument color can be a name or a number as listed below: (These symbolic names defined in graphics.h.)

Number	Name	Number	Name
0	BLACK	8	DARKGRAY
1	BLUE	9	LIGHTBLUE
2	GREEN	10	LIGHTGREEN
3	CYAN	11	LIGHTCYAN
4	RED	12	LIGHTRED
5	MAGENTA	13	LIGHTMAGENTA
6	BROWN	14	YELLOW
7	LIGHTGRAY	15	WHITE

For example, if you want to set the background color to blue, you can call

```
setbkcolor(BLUE) /* or */ setbkcolor(1)
```

On CGA and EGA systems, setbkcolor changes the background color by changing the first entry in the palette.

If you use an EGA or a VGA, and you change the palette colors with setpalette or setallpalette, the defined symbolic constants might not give you the correct color. This is because the parameter to setbkcolor indicates the entry number in the current palette rather than a specific color (unless the parameter passed is 0, which always sets the background color to black).

Return Value

None.

See Also

getbkcolor

setallpalette

setcolor

setpalette

setcolor

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setcolor(int color);
```

Description

setcolor sets the current drawing color to color, which can range from 0 to getmaxcolor.

The current drawing color is the value to which pixels are set when lines, and so on are drawn. The drawing colors shown below are available for the CGA and EGA, respectively.

Palette number			
0	CGA_LIGHTGREEN	CGA_LIGHTRED	CGA_YELLOW
1	CGA_LIGHTCYAN	CGA_LIGHTMAGENTA	CGA_WHITE
2	CGA_GREEN	CGA_RED	CGA_BROWN
3	CGA_CYAN	CGA_MAGENTA	CGA_LIGHTGRAY
Constant assigned to color number (pixel value):	1	2	3

Numeric Value		Symbolic Name	
0	BLACK	8	DARKGRAY
1	BLUE	9	LIGHTBLUE
2	GREEN	10	LIGHTGREEN
3	CYAN	11	LIGHTCYAN
4	RED	12	LIGHTRED
5	MAGENTA	13	LIGHTMAGENTA
6	BROWN	14	YELLOW
7	LIGHTGRAY	15	WHITE

You select a drawing color by passing either the color number itself or the equivalent symbolic name to setcolor. For example, in CGAC0 mode, the palette contains four colors: the background color, light green, light red, and yellow. In this mode, either setcolor(3) or setcolor(CGA_YELLOW) selects a drawing color of yellow.

Return Value

None.

See Also
getcolor
getmaxcolor
graphresult
setallpalette
setbkcolor
setpalette

setfillpattern

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setfillpattern(char far *upattern, int color);
```

Description

setfillpattern is like setfillstyle, except that you use it to set a user-defined 8*8 pattern rather than a predefined pattern.

upattern is a pointer to a sequence of 8 bytes, with each byte corresponding to 8 pixels in the pattern. Whenever a bit in a pattern byte is set to 1, the corresponding pixel is plotted.

Return Value

None.

See Also

getfillpattern

getfillsettings

graphresult

sector

setfillstyle

setfillstyle

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setfillstyle(int pattern, int color);
```

Description

setfillstyle sets the current fill pattern and fill color. To set a user-defined fill pattern, do not give a pattern of 12 (USER_FILL) to setfillstyle; instead, call setfillpattern. See Fill_Patterns.

If invalid input is passed to setfillstyle, graphresult returns -1(grError), and the current fill pattern and fill color remain unchanged.

Return Value

None.

See Also

bar
bar3d
fillpoly
floodfill
getfillsettings
graphresult
pieslice
sector
setfillpattern

setlinestyle

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setlinestyle(int linestyle, unsigned upattern, int thickness);
```

Description

setlinestyle sets the style for all lines drawn by line, lineto, rectangle, drawpoly, and so on.

The linesettingstype structure is defined in graphics.h as follows:

```
struct linesettingstype {
   int linestyle;
   unsigned upattern;
   int thickness;
};
```

linestyle specifies in which of several styles subsequent lines will be drawn (such as solid, dotted, centered, dashed). The enumeration line_styles, which is defined in graphics.h, gives names to these operators:

Name	Value	Description
SOLID_LINE	0	Solid line
DOTTED_LINE	1	Dotted line
CENTER_LINE	2	Centered line
DASHED_LINE	3	Dashed line
USERBIT_LINE	4	User-defined line style

thickness specifies whether the width of subsequent lines drawn will be normal or thick.

Name	Value	Description
NORM_WIDTH	1	1 pixel wide
THICK_WIDTH	3	3 pixels wide

upattern is a 16-bit pattern that applies only if linestyle is USERBIT_LINE (4). In that case, whenever a bit in the pattern word is 1, the corresponding pixel in the line is drawn in the current drawing color. For example, a solid line corresponds to a upattern of 0xFFFF (all pixels drawn), and a dashed line can correspond to a upattern of 0x3333 or 0x0F0F. If the linestyle parameter to setlinestyle is not USERBIT_LINE (in other words, if it is not equal to 4), you must still provide the upattern parameter, but it will be ignored.

Note: The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.

Return Value

If invalid input is passed to setlinestyle, graphresult returns -11, and the current line style remains unchanged.

See Also

arc
bar3d
circle
drawpoly
ellipse
getlinesettings
graphresult
line
linerel
lineto
pieslice
rectangle

setpalette

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setpalette(int colornum, int color);
```

Description

setpalette changes the colornum entry in the palette to color. For example, setpalette(0,5) changes the first color in the current palette (the background color) to actual color number 5. If size is the number of entries in the current palette, colornum can range between 0 and (size - 1).

You can partially (or completely) change the colors in the EGA/VGA palette with setpalette. On a CGA, you can only change the first entry in the palette (colornum equals 0, the background color) with a call to setpalette.

The color parameter passed to setpalette can be represented by symbolic constants which are defined in graphics.h. See Actual Color Table.

setpalette cannot be used with the IBM-8514 driver; use setrgbpalette instead.

Return Value

If invalid input is passed to setpalette, graphresult returns -11, and the current palette remains unchanged.

Actual Color Table

Changes made to the palette are seen immediately onscreen. Each time a palette color is changed, all occurrences of that color onscreen change to the new color value.

Note: Valid colors depend on the current graphics driver and current graphics mode.

CGA		EGA/VGA	
Name	**Value**	**Name**	**Value**
BLACK	0	EGA_BLACK	0
BLUE	1	EGA_BLUE	1
GREEN	2	EGA_GREEN	2
CYAN	3	EGA_CYAN	3
RED	4	EGA_RED	4
MAGENTA	5	EGA_MAGENTA	5
BROWN	6	EGA_LIGHTGRAY	7
LIGHTGRAY	7	EGA_BROWN	20
DARKGRAY	8	EGA_DARKGRAY	56
LIGHTBLUE	9	EGA_LIGHTBLUE	57
LIGHTGREEN	10	EGA_LIGHTGREEN	58
LIGHTCYAN	11	EGA_LIGHTCYAN	59
LIGHTRED	12	EGA_LIGHTRED	60
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61
YELLOW	14	EGA_YELLOW	62
WHITE	15	EGA_WHITE	63

See Also
getpalette
graphresult
setallpalette
setbkcolor
setcolor
setrgbpalette

setrgbpalette

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setrgbpalette(int colornum, int red, int green, int blue);
```

Description

setrgbpalette can be used with the IBM 8514 and VGA drivers.

colornum defines the palette entry to be loaded, while red, green, and blue define the component colors of the palette entry.

For the IBM 8514 display (and the VGA in 256K color mode), colornum is in the range 0 to 255. For the remaining modes of the VGA, colornum is in the range 0 to 15. Only the lower byte of red, green, or blue is used, and out of each byte, only the 6 most significant bits are loaded in the palette.

For compatibility with other IBM graphics adapters, the BGI driver defines the first 16 palette entries of the IBM 8514 to the default colors of the EGA/VGA. These values can be used as is, or they can be changed using setrgbpalette.

Return Value

None.

See Also
setpalette

settextjustify

See Also Example Portability

Syntax

```
#include <graphics.h>
void far settextjustify(int horiz, int vert);
```

Description

Text output after a call to settextjustify is justified around the current position (CP) horizontally and vertically, as specified. The default justification settings are LEFT_TEXT (for horizontal) and TOP_TEXT (for vertical). The enumeration text_just in graphics.h provides names for the horiz and vert settings passed to settextjustify.

Description	Name	Value	Action
horiz	LEFT_TEXT	0	left-justify text
	CENTER_TEXT	1	center text
	RIGHT_TEXT	2	right-justify text
vert	BOTTOM_TEXT	0	justify from bottom
	CENTER_TEXT	1	center text
	TOP_TEXT	2	justify from top

If horiz is equal to LEFT_TEXT and direction equals HORIZ_DIR, the CP's x component is advanced after a call to outtext(string) by textwidth(string).

settextjustify affects text written with outtext and cannot be used with text mode and stream functions.

Return Value

If invalid input is passed to settextjustify, graphresult returns -11, and the current text justification remains unchanged.

See Also

gettextsettings

graphresult

outtext

settextstyle

settextstyle

See Also Example Portability

Syntax

```
#include <graphics.h>
void far settextstyle(int font, int direction, int charsize);
```

Description

settextstyle sets the text font, the direction in which text is displayed, and the size of the characters. A call to settextstyle affects all text output by outtext and outtextxy.

The parameters font, direction, and charsize passed to settextstyle are described in the following:

font: One 8*8 bit-mapped font and several "stroked" fonts are available. The 8*8 bit-mapped font is the default. The enumeration font_names, which is defined in graphics.h, provides names for these different font settings:

Name	Value	Description
DEFAULT_FONT	0	8x8 bit-mapped font
TRIPLEX_FONT	1	Stroked triplex font
SMALL_FONT	2	Stroked small font
SANS_SERIF_FONT	3	Stroked sans-serif font
GOTHIC_FONT	4	Stroked gothic font
SCRIPT_FONT	5	Stroked script font
SIMPLEX_FONT	6	Stroked triplex script font
TRIPLEX_SCR_FONT	7	Stroked triplex script font
COMPLEX_FONT	8	Stroked complex font
EUROPEAN_FONT	9	Stroked European font
BOLD_FONT	10	Stroked bold font

The default bit-mapped font is built into the graphics system. Stroked fonts are stored in *.CHR disk files, and only one at a time is kept in memory. Therefore, when you select a stroked font (different from the last selected stroked font), the corresponding *.CHR file must be loaded from disk.

To avoid this loading when several stroked fonts are used, you can link font files into your program. Do this by converting them into object files with the BGIOBJ utility, then registering them through registerbgifont, as described in UTIL.DOC, included with your distributions disks.

direction: Font directions supported are horizontal text (left to right) and vertical text (rotated 90 degrees counterclockwise). The default direction is HORIZ_DIR.

Name	Value	Description
HORIZ_DIR	0	Left to right
VERT_DIR	1	Bottom to top

Charcter Size (charsize)

The size of each character can be magnified using the charsize factor. If charsize is nonzero, it can affect bit-mapped or stroked characters. A charsize value of 0 can be used only with stroked fonts.

- If charsize equals 1, outtext and outtextxy displays characters from the 8*8 bit-mapped font in an 8*8 pixel rectangle onscreen.
- If charsize equals 2, these output functions display characters from the 8*8 bit-mapped font in a 16*16 pixel rectangle, and so on (up to a limit of ten times the normal size).
- When charsize equals 0, the output functions outtext and outtextxy magnify the stroked font text using either the default character magnification factor (4) or the user-defined character size given by

setusercharsize.

Always use textheight and textwidth to determine the actual dimensions of the text.

Return Value

None.

See Also

gettextsettings

graphresult

installuserfont

settextjustify

setusercharsize

textheight

textwidth

setusercharsize

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setusercharsize(int multx, int divx, int multy, int divy);
```

Description

setusercharsize gives you finer control over the size of text from stroked fonts used with graphics functions. The values set by setusercharsize are active only if charsize equals 0, as set by a previous call to settextstyle.

With setusercharsize, you specify factors by which the width and height are scaled. The default width is scaled by multx : divx, and the default height is scaled by multy : divy. For example, to make text twice as wide and 50% taller than the default, set

```
multx = 2;  divx = 1;
multy = 3;  divy = 2;
```

Return Value

None.

See Also

gettextsettings

graphresult

settextstyle

setviewport

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setviewport(int left, int top, int right, int bottom, int clip);
```

Description

setviewport establishes a new viewport for graphics output.

The viewport corners are given in absolute screen coordinates by (left,top) and (right,bottom). The current position (CP) is moved to (0,0) in the new window.

The parameter clip determines whether drawings are clipped (truncated) at the current viewport boundaries. If clip is nonzero, all drawings will be clipped to the current viewport.

Return Value

If invalid input is passed to setviewport, graphresult returns -11, and the current view settings remain unchanged.

See Also

clearviewport

getviewsettings

graphresult

setvisualpage

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setvisualpage(int page);
```

Description

setvisualpage makes page the visual graphics page.

Return Value

None.

See Also
graphresult
setactivepage

setwritemode

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setwritemode(int mode);
```

Description

The following constants are defined:

```
COPY_PUT = 0       /* MOV */
XOR_PUT  = 1       /* XOR */
```

Each constant corresponds to a binary operation between each byte in the line and the corresponding bytes onscreen. COPY_PUT uses the assembly language MOV instruction, overwriting with the line whatever is on the screen. XOR_PUT uses the XOR command to combine the line with the screen. Two successive XOR commands will erase the line and restore the screen to its original appearance.

setwritemode currently works only with line, linerel, lineto, rectangle, and drawpoly.

Return Value

None.

See Also

drawpoly

line

linerel

lineto

putimage

setgraphmode

See Also Example Portability

Syntax

```
#include <graphics.h>
void far setgraphmode(int mode);
```

Description

setgraphmode selects a graphics mode different than the default one set by initgraph. mode must be a valid mode for the current device driver. setgraphmode clears the screen and resets all graphics settings to their defaults (current position, palette, color, viewport, and so on).

You can use setgraphmode in conjunction with restorecrtmode to switch back and forth between text and graphics modes.

Return Value

If you give setgraphmode an invalid mode for the current device driver, graphresult returns a value of -10 (grInvalidMode).

See Also

getgraphmode

getmoderange

graphresult

initgraph

restorecrtmode

setgraphbufsize

See Also Example Portability

Syntax

```
#include <graphics.h>
unsigned far setgraphbufsize(unsigned bufsize);
```

Description

Some of the graphics routines (such as floodfill) use a memory buffer that is allocated when initgraph is called and released when closegraph is called. The default size of this buffer, allocated by _graphgetmem, is 4,096 bytes.

You might want to make this buffer smaller (to save memory space) or bigger (if, for example, a call to floodfill produces error -7: Out of flood memory).

setgraphbufsize tells initgraph how much memory to allocate for this internal graphics buffer when it calls _graphgetmem.

You must call setgraphbufsize before calling initgraph. Once initgraph has been called, all calls to setgraphbufsize are ignored until after the next call to closegraph.

Return Value

setgraphbufsize returns the previous size of the internal buffer.

See Also

closegraph

_graphfreemem

_graphgetmem

initgraph

sector

textheight

See Also Example Portability

Syntax

```
#include <graphics.h>
int far textheight(char far *textstring
```

Description

The graphics function textheight takes the current font size and multiplication factor, and determines the height of textstring in pixels. This function is useful for adjusting the spacing between lines, computing viewport heights, sizing a title to make it fit on a graph or in a box, and so on.

For example, with the 8*8 bit-mapped font and a multiplication factor of 1 (set by settextstyle), the string BorlandC++ is 8 pixels high.

Use textheight to compute the height of strings, instead of doing the computations manually. By using this function, no source code modifications have to be made when different fonts are selected.

Return Value

textheight returns the text height in pixels.

See Also

gettextsettings

outtext

outtextxy

settextstyle

textwidth

textwidth

See Also Example Portability

Syntax

```
#include <graphics.h>
int far textwidth(char far *textstring);
```

Description

The graphics function textwidth takes the string length, current font size, and multiplication factor, and determines the width of textstring in pixels.

This function is useful for computing viewport widths, sizing a title to make it fit on a graph or in a box, and so on.

Use textwidth to compute the width of strings, instead of doing the computations manually. When you use this function, no source code modifications have to be made when different fonts are selected.

Return Value

textwidth returns the text width in pixels.

See Also

gettextsettings

outtext

outtextxy

settextstyle

textheight

/* absread example */

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <ctype.h>

#define SECSIZE 512

int main(void)
{
   unsigned char buf[SECSIZE];
   int i, j, sector, drive;
   char str[10];

   printf("Enter drive letter: ");
   gets(str);
   drive = toupper(str[0]) - 'A';

   printf("Enter sector number to read: ");
   gets(str);
   sector = atoi(str);
   if (absread(drive, 1, sector, &buf) != 0) {
       perror("Disk error");
       exit(1);
   }
   printf("\nDrive: %c   Sector: %d\n", 'A' + drive, sector);
   for (i = 0; i < SECSIZE; i += 16) {
      if ((i / 16) == 20) {
         printf("Press any key to continue...");
         getch();
         printf("\n");
      }
      printf("%03d: ", i);
      for (j = 0; j < 16; j++)
         printf("%02X ", buf[i+j]);
      printf("\t");
      for (j = 0; j < 16; j++)
         if (isprint(buf[i+j]))
            printf("%c", buf[i+j]);
         else printf(".");
      printf("\n");
   }
   return 0;
}
```

/* allocmem example */

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

int main(void)
{
    unsigned int segp, maxb;
    unsigned int size = 64; /* (64*16) = 1024 bytes */
    int largest;

    /* Use _dos_allocmem, _dos_setblock, and _dos_freemem. */
    if (_dos_allocmem(size, &segp) == 0)
        printf("Allocated memory at segment: %x\n", segp);
    else {
        perror("Unable to allocate block.");
        printf("Maximum no. of paragraphs"
               " available is %u\n", segp);
        return 1;
    }
    if (_dos_setblock(size * 2, segp, &maxb) == 0)
        printf("Grew memory block at segment: %X\n", segp);
    else {
        perror("Unable to grow block.");
        printf("Maximum number of paragraphs"
               " available is %u\n", maxb);
    }
    _dos_freemem(segp);

    /* Use allocmem, setblock, and freemem. */
    if ((largest = allocmem(size, &segp)) == -1)
        printf("Allocated memory at segment: %x\n", segp);
    else {
        perror("Unable to allocate block.");
        printf("Maximum number of paragraphs"
               " available is %u\n", largest);
        return 1;
    }
    if ((largest = setblock(segp, size * 2)) == -1)
        printf("Grew memory block at segment: %X\n", segp);
    else {
        perror("Unable to grow block.");
        printf("Maximum number of paragraphs"
               " available is %u\n", largest);
    }
    freemem(segp);
    return 0;
}
```

/* arc example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int stangle = 45, endangle = 135;
   int radius = 100;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* draw arc */
   arc(midx, midy, stangle, endangle, radius);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* bar example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, i;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {   /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* loop through the fill patterns */
   for (i=SOLID_FILL; i<USER_FILL; i++) {
      /* set the fill style */
      setfillstyle(i, getmaxcolor());

      /* draw the bar */
      bar(midx-50, midy-50, midx+50, midy+50);
      getch();
   }
   /* clean up */
   closegraph();
   return 0;
}
```

/* bar3d example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, i;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {   /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* loop through the fill patterns */
   for (i=EMPTY_FILL; i<USER_FILL; i++) {
      /* set the fill style */
      setfillstyle(i, getmaxcolor());

      /* draw the 3-d bar */
      bar3d(midx-50, midy-50, midx+50, midy+50, 10, 1);
      getch();
   }
   /* clean up */
   closegraph();
   return 0;
}
```

/* _bios_printer example */

```
#include <stdio.h>
#include <conio.h>
#include <bios.h>

int main(void)
{
   unsigned PortNum = 0x1;  /* LPT1 -> 0, LPT2 -> 1 */
  unsigned status, abyte = 0;
  printf("Please turn off your printer.\n\
        Press any key to continue\n");
  getch();
  status = _bios_printer(_PRINTER_STATUS, PortNum, abyte);
   if (status & 0x01)
      printf("Device time out.\n");
   if (status & 0x08)
      printf("I/O error.\n");
   if (status & 0x10)
      printf("Selected.\n");
   if (status & 0x20)
      printf("Out of paper.\n");
   if (status & 0x40)
      printf("Acknowledge.\n");
   if (status & 0x80)
      printf("Not busy.\n");
   return 0;
}
```

/* bioscom example */

```
/*
    This example can be used to communicate between
    two PCs via a null modem cable.
    The example is specific to COM1.  The appropriate values
    for COM2 are included in this example for the convenience
    of switching the code to work with COM2.

*/
#include  <bios.h>
#include  <conio.h>
#include  <dos.h>

#define DTR         0x01    // Data Terminal Ready
#define RTS         0x02    // Ready To Send
#define COM1PORT    0x0000  // Pointer to Location of COM1 port
#define COM2PORT    0x0002  // Pointer to Location of COM2 port
#define COM1        0
#define COM2        1
#define DATA_READY  0x100
#define FALSE       0
#define TRUE        !FALSE

#define SETTINGS ( 0xE0 | 0x00 | 0x02 | 0x00)  // 9600,N,7,1

int main( void )
{
  int     in,
          out,
          status,
          DONE    = FALSE,
      far *RS232_Addr;

  /*  Determine port location of COM1.
      0x40:0x00 = COM1 I/O port address
      0x40:0x02 = COM2 I/O port address
  */
  RS232_Addr = MK_FP( 0x0040, COM1PORT );
  if( !*RS232_Addr )
    return -1;

  bioscom( 0, SETTINGS, COM1 );
  cprintf( "... BIOSCOM [ESC] to exit ...\n" );

  while( !DONE )
  {
    /*  Reset DTR and RTS to prepare for send/receive of
        next character.
    */
    outportb( *RS232_Addr + 4, DTR | RTS );

    /*  Get status of com port.
    */
```

```
    status = bioscom( 3, 0, COM1 );

    if( status & DATA_READY )

      /*  There's a character on the port.  Get it and echo.
      */
      if( (out = bioscom( 2, 0, COM1 ) & 0x7F) != 0 )
        putch( out );

    if( kbhit() )

      /*  Key has been struck.  Get it and send to port.
      */
      if( (in = getch()) == '\x1B' )

        /* User pressed ESCAPE.  Don't send to port.
        */
        DONE = TRUE;

      else

        /*  Send character to com port.
        */
        bioscom( 1, in, COM1 );
  }
  return 0;
}
```

/* biosdisk example */

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

int main(void)
{
   #define CMD    2    /* read sector command */
   #define DRIVE  0    /* drive number for A: */
   #define HEAD   0    /* disk head number */
   #define TRACK  1    /* track number */
   #define SECT   1    /* sector number */
   #define NSECT  1    /* sector count */

   int result;
   char buffer[512];
   printf("Attempting to read from drive A:\n");
   result = biosdisk(CMD, DRIVE, HEAD, TRACK, SECT, NSECT, buffer);
   if (result == 0)
      printf("Disk read from A: successful.\n");
   else
      printf("Attempt to read from drive A: failed.\n");
   return 0;
}
```

/* _bios_serialcom example (for COM1) */

```
/*
    This example can be used to communicate between
    two PCs via a null modem cable.
    This example is specific to COM1.
*/
#include  <bios.h>
#include  <conio.h>
#include  <dos.h>

#define DTR         0x01    // Data Terminal Ready
#define RTS         0x02    // Ready To Send
#define COM1PORT    0x0000  // Pointer to Location of COM1 port
#define COM2PORT    0x0002  // Pointer to Location of COM2 port
#define COM1        0
#define COM2        1
#define DATA_READY  0x100
#define FALSE       0
#define TRUE        !FALSE

#define SETTINGS ( 0xE0 | 0x00 | 0x02 | 0x00)  // 9600,N,7,1

int main( void )
{
  int in,
    out,
    status,
    DONE    = FALSE,
    far *RS232_Addr;

  /*  Determine port location of COM1.
    0x40:0x00 = COM1 I/O port address
    0x40:0x02 = COM2 I/O port address
  */
  RS232_Addr = MK_FP( 0x0040, COM1PORT );
  if( !*RS232_Addr )
   return -1;

  _bios_serialcom( 0, COM1, SETTINGS );
  cprintf( "... BIOSCOM [ESC] to exit ...\n" );

  while( !DONE )
  {
   /*  Reset DTR and RTS to prepare for send/receive of
  next character.
   */
   outportb( *RS232_Addr + 4, DTR | RTS );

   /*  Get status of com port.
   */
   status =  _bios_serialcom( 3, COM1, 0 );

   if( status & DATA_READY )
```

```
    /*  There's a character on the port.  Get it and echo.
     */
    if(( out =_bios_serialcom( 2, COM1, 0 ) & 0x7F) != 0)
       putch( out );

    if( kbhit() )

     /*  Key has been struck.  Get it and send to port.
     */
    if( (in = getch()) == '\x1B' )

       /* User pressed ESCAPE.  Don't send to port.
       */
       DONE = TRUE;

     else

       /*  Send character to com port.
       */
     _bios_serialcom( 1, COM1, in );
  }
  return 0;
}
```

/* _bios_serialcom example (for COM2) */

```
/*
    This example can be used to communicate between
    two PCs via a null modem cable.
    This example is specific to COM2.
*/
#include  <bios.h>
#include  <conio.h>
#include  <dos.h>

#define DTR         0x01    // Data Terminal Ready
#define RTS         0x02    // Ready To Send
#define COM1PORT    0x0000  // Pointer to Location of COM1 port
#define COM2PORT    0x0002  // Pointer to Location of COM2 port
#define COM1        0
#define COM2        1
#define DATA_READY  0x100
#define FALSE       0
#define TRUE        !FALSE

#define SETTINGS ( 0xE0 | 0x00 | 0x02 | 0x00)  // 9600,N,7,1

int main( void )
{
  int in,
    out,
    status,
    DONE    = FALSE,
    far *RS232_Addr;

  /*  Determine port location of COM1.
    0x40:0x00 = COM1 I/O port address
    0x40:0x02 = COM2 I/O port address
  */
  RS232_Addr = MK_FP( 0x0040, COM2PORT );
  if( !*RS232_Addr )
   return -1;

  _bios_serialcom( 0, COM2, SETTINGS );
  cprintf( "... BIOSCOM [ESC] to exit ...\n" );

  while( !DONE )
  {
   /*  Reset DTR and RTS to prepare for send/receive of
  next character.
   */
   outportb( *RS232_Addr + 4, DTR | RTS );

   /*  Get status of com port.
   */
   status =  _bios_serialcom( 3, COM2, 0 );

   if( status & DATA_READY )
```

```
    /*  There's a character on the port.  Get it and echo.
    */
   if(( out =_bios_serialcom( 2, COM2, 0 ) & 0x7F) != 0)
      putch( out );

   if( kbhit() )

    /*  Key has been struck.  Get it and send to port.
    */
   if( (in = getch()) == '\x1B' )

      /* User pressed ESCAPE.  Don't send to port.
      */
      DONE = TRUE;

    else

      /*  Send character to com port.
      */
    _bios_serialcom( 1, COM2, in );
  }
  return 0;
}
```

/* _dos_allocmem example */

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

int main(void)
{
  unsigned int size, segp, err, maxb;
  size = 64; /* (64 x 16) = 1024 bytes */
  err = _dos_allocmem(size, &segp);
  if (err == 0)
    printf("Allocated memory at segment: %x\n", segp);
  else {
    perror("Unable to allocate block");
    printf("Maximum no. of paragraphs available is %u\n", segp);
    return 1;
  }
  if (_dos_setblock(size * 2, segp, &maxb) == 0)
    printf("Expanded memory block at segment: %X\n", segp);
  else {
    perror("Unable to expand block");
    printf("Maximum no. of paragraphs available is %u\n", maxb);
  }
  _dos_freemem(segp);
  return 0;
}
```

/* _dos_keep example */

```
/* * * * * * * * * * * * * *

NOTE: This is an interrupt service routine. You can NOT compile this
  program with Test Stack Overflow turned on and get an executable file
  which will operate correctly.

Due to the nature of this function the formula used to compute the number
  of paragraphs may not necessarily work in all cases.  Use with care!

Terminate Stay Resident (TSR) programs are complex and no other support for
  them is provided.

Refer to the MS-DOS technical documentation for more information.

 * * * * * * * * * * * * * */

#include <dos.h>
/* The clock tick interrupt */
#define INTR 0x1C
/* Screen attribute (blue on grey) */
#define ATTR 0x7900

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

/* reduce heaplength and stacklength to make a smaller program in memory */
extern unsigned _heaplen = 1024;
extern unsigned _stklen  = 512;

void interrupt ( *oldhandler)(__CPPARGS);

typedef unsigned int (far *s_arrayptr);

void interrupt handler(__CPPARGS)
{
   s_arrayptr screen[80];
   static int count;

/* For a color screen the video memory is at B800:0000.
   For a monochrome system use B000:000 */
   screen[0] = (s_arrayptr) MK_FP(0xB800,0);

/* increase the counter and keep it within 0 to 9 */
   count++;
   count %= 10;

/* put the number on the screen */
   screen[0][79] = count + '0' + ATTR;

/* call the old interrupt handler */
```

```
   oldhandler();
}

int main(void)
{

/* get the address of the current clock
   tick interrupt */
oldhandler = _dos_getvect(INTR);

/* install the new interrupt handler */
_dos_setvect(INTR, handler);

/* * *
_psp is the starting address of the  program in memory.  The top of the
  stack is the end of the program.

Using _SS and _SP together we can get the end of the stack.  You may want
  to allow a bit of safety space to insure that enough room is being
  allocated ie:

   (_SS + ((_SP + safety space)/16) - _psp)
 * * */

_dos_keep(0, (_SS + (_SP/16) - _psp));
return 0;
}
```

/* bioskey example */

```
#include <stdio.h>
#include <bios.h>
#include <ctype.h>

#define RIGHT  0x01
#define LEFT   0x02
#define CTRL   0x04
#define ALT    0x08

int main(void)
{
   int key, modifiers;

   /* function 1 returns 0 until a key is pressed */
   while (bioskey(1) == 0);

   /* function 0 returns the key that is waiting */
   key = bioskey(0);

   /* use function 2 to determine if shift keys were used */
   modifiers = bioskey(2);
   if (modifiers)
   {
      printf("[");
      if (modifiers & RIGHT) printf("RIGHT");
      if (modifiers & LEFT)  printf("LEFT");
      if (modifiers & CTRL)  printf("CTRL");
      if (modifiers & ALT)   printf("ALT");
      printf("]");
   }
   /* print out the character read */
   if (isalnum(key & 0xFF))
      printf("'%c'\n", key);
   else
      printf("%#02x\n", key);
   return 0;
}
```

/* biosprint example */

```
#include <stdio.h>
#include <conio.h>
#include <bios.h>

int main(void)
{
   #define STATUS  2    /* printer status command */
   #define PORTNUM 0    /* port number for LPT1 */

   int status, abyte=0;

   printf("Please turn off your printer. Press any key to continue\n");
   getch();
   status = biosprint(STATUS, abyte, PORTNUM);
   if (status & 0x01)
      printf("Device time out.\n");
   if (status & 0x08)
      printf("I/O error.\n");

   if (status & 0x10)
      printf("Selected.\n");
   if (status & 0x20)
      printf("Out of paper.\n");

   if (status & 0x40)
      printf("Acknowledge.\n");
   if (status & 0x80)
      printf("Not busy.\n");

   return 0;
}
```

/* _dos_freemem example */

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

int main(void)
{
  unsigned int size, segp, err, maxb;
  size = 64; /* (64 x 16) = 1024 bytes */
  err = _dos_allocmem(size, &segp);
  if (err == 0)
    printf("Allocated memory at segment: %x\n", segp);
  else {
    perror("Unable to allocate block");
    printf("Maximum no. of paragraphs available is %u\n", segp);
    return 1;
  }
  if (_dos_setblock(size * 2, segp, &maxb) == 0)
    printf("Expanded memory block at segment: %X\n", segp);
  else {
    perror("Unable to expand block");
    printf("Maximum no. of paragraphs available is %u\n", maxb);
  }
  _dos_freemem(segp);
  return 0;
}
```

```
/* _bios_disk example */
#include <bios.h>
#include <stdio.h>

int main(void)
{
  struct diskinfo_t dinfo;
  int result;
  static char dbuf[512];

  dinfo.drive =  0;    /* drive number for A: */
  dinfo.head  =  0;    /* disk head number */
  dinfo.track =  0;    /* track number */
  dinfo.sector  =  1;  /* sector number */
  dinfo.nsectors =  1; /* sector count */
  dinfo.buffer = dbuf; /* data buffer */

  printf("Attempting to read from drive A:\n");
  result = _bios_disk(_DISK_READ, &dinfo);
  if ((result & 0xff00) == 0)
  {
    printf("Disk read from A: successful.\n");
    printf("First three bytes read are 0x%02x 0x%02x 0x%02x\n",
      dbuf[0] & 0xff, dbuf[1] & 0xff, dbuf[2] & 0xff);
  }
  else
    printf("Cannot read drive A, status = 0x%02x\n", result);
  return 0;
}
```

/* _bios_keybrd example */

```
#include <stdio.h>
#include <bios.h>
#include <ctype.h>

#define RIGHT  0x01
#define LEFT   0x02
#define CTRL   0x04
#define ALT    0x08

int main(void)
{
   int key, modifiers;

   /* Wait until a key is pressed */
   while (_bios_keybrd(_KEYBRD_READY) == 0);

   /* Fetch the key that is waiting */
   key = _bios_keybrd(_KEYBRD_READ);

   /* Determine if shift keys are used */
   modifiers = _bios_keybrd(_KEYBRD_SHIFTSTATUS);
   if (modifiers){
      printf("[");
      if (modifiers & RIGHT) printf("RIGHT");
      if (modifiers & LEFT)  printf("LEFT");
      if (modifiers & CTRL)  printf("CTRL");
      if (modifiers & ALT)   printf("ALT");
      printf("]");
   }

   /* print out the character read */
   if (isalnum(key & 0xFF))
      printf("'%c'\n", key);
   else
      printf("%#02x\n", key);
   return 0;
}
```

/* biosprin example */

```
#include <stdio.h>
#include <conio.h>
#include <bios.h>

int main(void)
{
   #define STATUS  2   /* printer status command */
   #define PORTNUM 0   /* port number for LPT1 */

   int status, abyte=0;
   printf("_Please turn off your printer. Press any key tocontinue\n");
   getch();
   status = biosprint(STATUS, abyte, PORTNUM);
   if (status & 0x01)
      printf("Device time out.\n");
   if (status & 0x08)
      printf("I/O error.\n");
   if (status & 0x10)
      printf("Selected.\n");
   if (status & 0x20)
      printf("Out of paper.\n");
   if (status & 0x40)
      printf("Acknowledge.\n");
   if (status & 0x80)
      printf("Not busy.\n");
   return 0;
}
```

/* brk example */

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

int main(void)
{
   char *ptr;

   printf("Changing allocation with brk()\n");
   ptr = (char *) malloc(1);
   printf("Before brk() call: %lu bytes free\n", coreleft());
   brk(ptr+1000);
   printf(" After brk() call: %lu bytes free\n", coreleft());
   return 0;
}
```

/* circle example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, radius = 100;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {   /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* draw the circle */
   circle(midx, midy, radius);
   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* cleardevice example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {    /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);                 /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* for centering screen messages */
   settextjustify(CENTER_TEXT, CENTER_TEXT);

   /* output a message to the screen */
   outtextxy(midx, midy, "Press any key to clear the screen:");

   getch();   /* wait for a key */
   cleardevice();   /* clear the screen */
   /* output another message */
   outtextxy(midx, midy, "Press any key to quit:");
   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* closegraph example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode, x, y;

   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   if (errorcode != grOk) {   /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
      }

   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* output a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "Press a key to close the graphics system:");

   getch();   /* wait for a key */
   /* closes down the graphics system */
   closegraph();
   printf("We're now back in text mode.\n");
   printf("Press any key to halt:");
   getch();
   return 0;
}
```

```
/* clearviewport example */
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define CLIP_ON 1   /* activates clipping in viewport */

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode, ht;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {   /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   setcolor(getmaxcolor());
   ht = textheight("W");

   /* message in default full-screen viewport */
   outtextxy(0, 0, "* <-- (0, 0) in default viewport");

   /* create a smaller viewport */
   setviewport(50, 50, getmaxx()-50, getmaxy()-50, CLIP_ON);

   /* display some messages */
   outtextxy(0, 0, "* <-- (0, 0) in smaller viewport");
   outtextxy(0, 2*ht, "Press any key to clear viewport:");

   getch();   /* wait for a key */
   clearviewport();   /* clear the viewport */
   /* output another message */
   outtextxy(0, 0, "Press any key to quit:");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* coreleft example */

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

int main(void)
{
   printf("The difference between the highest allocated block and\n");
   printf("the top of the heap is:"
          " %lu bytes\n", (unsigned long) coreleft());
   return 0;
}
```

/* delay example */

```
/* emits a 440-Hz tone for 500 milliseconds */
#include <dos.h>

int main(void)
{
   sound(440);
   delay(500);
   nosound();
   return 0;
}
```

/* detectgraph example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the various cards supported */
char *dname[] = { "requests detection",
                  "a CGA",
                  "an MCGA",
                  "an EGA",
                  "a 64K EGA",
                  "a monochrome EGA",
                  "an IBM 8514",
                  "a Hercules monochrome",
                  "an AT&T 6300 PC",
                  "a VGA",
                  "an IBM 3270 PC"
                };

int main(void)
{
   /* used to return detected hardware info. */
   int gdriver, gmode, errorcode;

   /* detect the graphics hardware available */
   detectgraph(&gdriver, &gmode);

   /* read result of detectgraph call */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* display the information detected */
   clrscr();
   printf("You have %s video display card.\n", dname[gdriver]);
   printf("Press any key to halt:");
   getch();
   return 0;
}
```

/* drawpoly example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;

   int poly[10];   /* our polygon array */

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk){ /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);             /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();
   poly[0] = 20;           /* first vertex */
   poly[1] = maxy / 2;
   poly[2] = maxx - 20;    /* second vertex */
   poly[3] = 20;
   poly[4] = maxx - 50;    /* third vertex */
   poly[5] = maxy - 20;
   poly[6] = maxx / 2;     /* fourth vertex */
   poly[7] = maxy / 2;
   poly[8] = poly[0];      /* drawpoly doesn't automatically close */
   poly[9] = poly[1];      /* the polygon, so we close it */

   drawpoly(5, poly);   /* draw the polygon */

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* ellipse example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int stangle = 0, endangle = 360;
   int xradius = 100, yradius = 50;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* draw ellipse */
   ellipse(midx, midy, stangle, endangle, xradius, yradius);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* farcoreleft example */

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

int main(void)
{
  printf("The difference between the highest allocated block in the far\
  n");
  printf("heap and the top of the far heap is: %lu bytes\n",
  farcoreleft());

  return 0;
}
```

/* farheapcheck example */

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

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
char far *array[ NUM_PTRS ];
int i;

    for( i = 0; i < NUM_PTRS; i++ )
  array[ i ] = (char far *) farmalloc( NUM_BYTES );

    for( i = 0; i < NUM_PTRS; i += 2 )
  farfree( array[ i ] );

    if( farheapcheck() == _HEAPCORRUPT )
  printf( "Heap is corrupted.\n" );
    else
  printf( "Heap is OK.\n" );

    return 0;
}
```

/* farheapcheckfree example */

```
#include <mem.h>
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
  char far *array[NUM_PTRS];
  int i;
  int j;
  int res;

  for (i = 0; i < NUM_PTRS; i++)
    if ((array[i] = (char far *) farmalloc(NUM_BYTES)) == NULL)
    {
      printf("No memory for allocation\n");
      return 1;
    }

  for (i = 0; i < NUM_PTRS; i += 2)
    farfree(array[i]);

  if(farheapfillfree(1) < 0)
  {
    printf("Heap corrupted.\n");
    return 1;
  }

  for (i = 1; i < NUM_PTRS; i += 2)
    for (j = 0; j < NUM_BYTES; j++)
      array[i][j] = 0;

  res = farheapcheckfree(1);
  if (res < 0)
    switch(res)
    {
      case _HEAPCORRUPT:
        printf("Heap corrupted.\n");
        return 1;
      case _BADVALUE:
        printf("Bad value in free space.\n");
        return 1;
      default:
        printf("Unknown error.\n");
        return 1;
    }

  printf("Test successful.\n");
  return 0;
}
```

/* farheapchecknode example */

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

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char far *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char far *) farmalloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      farfree( array[ i ] );

   for( i = 0; i < NUM_PTRS; i++ )
   {
      printf( "Node %2d ", i );
      switch( farheapchecknode( array[ i ] ))
      {
         case _HEAPEMPTY:
            printf("No heap.\n" );
            break;
         case _HEAPCORRUPT:
            printf("Heap corrupt.\n" );
            break;
         case _BADNODE:
            printf("Bad node.\n" );
            break;
         case _FREEENTRY:
            printf("Free entry.\n" );
            break;
         case _USEDENTRY:
            printf("Used entry.\n" );
            break;
         default:
            printf("Unknown return code.\n");
            break;
         }
      }

   return 0;
}
```

/* farheapfillfree example */

```
#include <mem.h>
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char far *array[NUM_PTRS];
   int i;
   int j;
   int res;

   for (i = 0; i < NUM_PTRS; i++)
      if ((array[i] = (char far *) farmalloc(NUM_BYTES)) == NULL)
      {
         printf("No memory for allocation\n");
         return 1;
      }

   for (i = 0; i < NUM_PTRS; i += 2)
      farfree(array[i]);

   if(farheapfillfree(1) < 0)
   {
      printf("Heap corrupted.\n");
      return 1;
   }

   for (i = 1; i < NUM_PTRS; i += 2)
      for (j = 0; j < NUM_BYTES; j++)
         array[i][j] = 0;

   res = farheapcheckfree(1);
   if (res < 0)
      switch(res)
      {
         case _HEAPCORRUPT:
            printf("Heap corrupted.\n");
            return 1;
         case _BADVALUE:
            printf("Bad value in free space.\n");
            return 1;
         default:
            printf("Unknown error.\n");
            return 1;
      }

   printf("Test successful.\n");
   return 0;
}
```

/* farheapwalk example */

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

#define NUM_PTRS  10
#define NUM_BYTES 16

int main( void )
{
   struct farheapinfo hi;
   char far *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char far *) farmalloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      farfree( array[ i ] );

   hi.ptr = NULL;
   printf( "   Size   Status\n" );
   printf( "   ----   ------\n" );
   while( farheapwalk( &hi ) == _HEAPOK )
     printf( "%7lu    %s\n", hi.size, hi.in_use ? "used" : "free" );
   return 0;
}
```

/* fillellipse example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, i;
   int xradius = 100, yradius = 50;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* loop through the fill patterns */
   for (i = EMPTY_FILL; i < USER_FILL; i++) {
      /* set fill pattern */
      setfillstyle(i, getmaxcolor());

      /* draw a filled ellipse */
      fillellipse(midx, midy, xradius, yradius);
      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

/* fillpoly example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int i, maxx, maxy;

   /* our polygon array */
   int poly[8];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();

   poly[0] = 20;             /* first vertex */
   poly[1] = maxy / 2;
   poly[2] = maxx - 20;      /* second vertex */
   poly[3] = 20;
   poly[4] = maxx - 50;      /* third vertex */
   poly[5] = maxy - 20;
   poly[6] = maxx / 2;       /* fourth, fillpoly automatically */
   poly[7] = maxy / 2;       /* closes the polygon */

   /* loop through the fill patterns */
   for (i=EMPTY_FILL; i<USER_FILL; i++) {
      /* set fill pattern */
      setfillstyle(i, getmaxcolor());

      /* draw a filled polygon */
      fillpoly(4, poly);
      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

/* floodfill example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();

   /* select drawing color */
   setcolor(getmaxcolor());


   /* select fill color */
   setfillstyle(SOLID_FILL, getmaxcolor());

   /* draw a border around the screen */
   rectangle(0, 0, maxx, maxy);

   /* draw some circles */
   circle(maxx / 3, maxy /2, 50);
   circle(maxx / 2, 20, 100);
   circle(maxx-20, maxy-50, 75);
   circle(20, maxy-20, 25);

   /* wait for a key */
   getch();

   /* fill in bounded region */
   floodfill(2, 2, getmaxcolor());

   /* clean up */
   getch();
   closegraph();
   return 0;
```

```
}
```

/* getarccoords example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   struct arccoordstype arcinfo;
   int midx, midy;
   int stangle = 45, endangle = 270;
   char sstr[80], estr[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* draw arc and get coordinates */
   setcolor(getmaxcolor());
   arc(midx, midy, stangle, endangle, 100);
   getarccoords(&arcinfo);

   /* convert arc information into strings */
   sprintf(sstr, "*- (%d, %d)", arcinfo.xstart, arcinfo.ystart);
   sprintf(estr, "*- (%d, %d)", arcinfo.xend, arcinfo.yend);

   /* output the arc information */
   outtextxy(arcinfo.xstart, arcinfo.ystart, sstr);
   outtextxy(arcinfo.xend, arcinfo.yend, estr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getaspectratio example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

main()
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int xasp, yasp, midx, midy;


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* get current aspect ratio settings */
   getaspectratio(&xasp, &yasp);

   /* draw normal circle */
   circle(midx, midy, 100);
   getch();

   /* draw wide circle */
   cleardevice();
   setaspectratio(xasp/2, yasp);
   circle(midx, midy, 100);
   getch();

   /* draw narrow circle */
   cleardevice();
   setaspectratio(xasp, yasp/2);
   circle(midx, midy, 100);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getbkcolor example */

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int bkcolor, midx, midy;
   char bkname[35];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* for centering text on the display */
   settextjustify(CENTER_TEXT, CENTER_TEXT);

   /* get the current background color */
   bkcolor = getbkcolor();

   /* convert color value into a string */
   itoa(bkcolor, bkname, 10);
   strcat(bkname, " is the current background color.");


   /* display a message */
   outtextxy(midx, midy, bkname);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getcolor example */

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int color, midx, midy;
   char colname[35];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* for centering text on the display */
   settextjustify(CENTER_TEXT, CENTER_TEXT);

   /* get the current drawing color */
   color = getcolor();

   /* convert color value into a string */
   itoa(color, colname, 10);
   strcat(colname, " is the current drawing color.");

   /* display a message */
   outtextxy(midx, midy, colname);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getimage example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

void save_screen(void far *buf[4]);
void restore_screen(void far *buf[4]);

int maxx, maxy;
int main(void)
{
   int gdriver=DETECT, gmode, errorcode;
   void far *ptr[4];

   /* autodetect the graphics driver and mode */
   initgraph(&gdriver, &gmode, "");
   errorcode = graphresult();  /* check for any errors */
   if (errorcode != grOk) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }

   maxx = getmaxx();
   maxy = getmaxy();

   /* draw an image on the screen */
   rectangle(0, 0, maxx, maxy);
   line(0, 0, maxx, maxy);
   line(0, maxy, maxx, 0);
   save_screen(ptr);           /* save the current screen */
   getch();                    /* pause screen */
   cleardevice();              /* clear screen */
   restore_screen(ptr);        /* restore the screen */
   getch();                    /* pause screen */
   closegraph();
   return 0;
}

void save_screen(void far *buf[4])
{
   unsigned size;
   int ystart=0, yend, yincr, block;
   yincr = (maxy+1) / 4;
   yend = yincr;

   /* get byte size of image */
   size = imagesize(0, ystart, maxx, yend);
   for (block=0; block<=3; block++) {
      if ((buf[block] = farmalloc(size)) == NULL) {
         closegraph();
```

```
         printf("Error: not enough heap space in save_screen().\n");
         exit(1);
       }
     getimage(0, ystart, maxx, yend, buf[block]);
     ystart = yend + 1;
     yend += yincr + 1;
   }
}

void restore_screen(void far *buf[4])
{
   int ystart=0, yend, yincr, block;
   yincr = (maxy+1) / 4;
   yend = yincr;
   for (block=0; block<=3; block++) {
      putimage(0, ystart, buf[block], COPY_PUT);
      farfree(buf[block]);
      ystart = yend + 1;

      yend += yincr + 1;
   }
}
```

/* getmaxx example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   char xrange[80], yrange[80];


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* convert max resolution values to strings */
   sprintf(xrange, "X values range from 0..%d", getmaxx());
   sprintf(yrange, "Y values range from 0..%d", getmaxy());

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, xrange);
   outtextxy(midx, midy + textheight("W"), yrange);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getmaxy example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int midx, midy;
  char xrange[80], yrange[80];

  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) {    /* an error occurred */
     printf("Graphics error: %s\n", grapherrormsg(errorcode));
     printf("Press any key to halt:");
     getch();
     exit(1);                 /* terminate with an error code */
  }

  midx = getmaxx() / 2;
  midy = getmaxy() / 2;

  /* convert max resolution values into strings */
  sprintf(xrange, "X values range from 0..%d", getmaxx());
  sprintf(yrange, "Y values range from 0..%d", getmaxy());

  /* display the information */
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  outtextxy(midx, midy, xrange);
  outtextxy(midx, midy+textheight("W"), yrange);

  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

/* getmodename example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, mode;
   char numname[80], modename[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }
   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get mode number and name strings */
   mode = getgraphmode();
   sprintf(numname, "%d is the current mode number.", mode);
   sprintf(modename, "%s is the current graphics mode.",getmodename(mode));

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, numname);
   outtextxy(midx, midy+2*textheight("W"), modename);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getmoderange example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int low, high;
   char mrange[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
         printf("Graphics error: %s\n", grapherrormsg(errorcode));
         printf("Press any key to halt:");
         getch();
         exit(1);             /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get the mode range for this driver */
   getmoderange(gdriver, &low, &high);

   /* convert mode range info. into strings */
   sprintf(mrange, "This driver supports modes %d..%d", low, high);

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, mrange);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getpalette example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{

   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   struct palettetype pal;
   char psize[80], pval[20];
   int i, ht;
   int y = 10;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* grab a copy of the palette */
   getpalette(&pal);

   /* convert palette info into strings */
   sprintf(psize, "The palette has %d modifiable entries.", pal.size);

   /* display the information */
   outtextxy(0, y, psize);
   if (pal.size != 0) {
      ht = textheight("W");
      y += 2*ht;
      outtextxy(0, y, "Here are the current values:");
      y += 2*ht;
      for (i=0; i<pal.size; i++, y+=ht) {
         sprintf(pval, "palette[%02d]: 0x%02X", i, pal.colors[i]);
         outtextxy(0, y, pval);
      }
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getpalettesize example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   char psize[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* convert palette size info into string */
   sprintf(psize, "The palette has %d modifiable
  entries.",getpalettesize());

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, psize);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getpixel example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define PIXEL_COUNT 1000
#define DELAY_TIME  100  /* in milliseconds */

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int i, x, y, color, maxx, maxy, maxcolor, seed;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxx = getmaxx() + 1;
   maxy = getmaxy() + 1;
   maxcolor = getmaxcolor() + 1;
   while (!kbhit()) {
      seed = random(32767);  /* seed the random number generator */
      srand(seed);
      for (i=0; i<PIXEL_COUNT; i++) {
         x = random(maxx);
         y = random(maxy);
         color = random(maxcolor);
         putpixel(x, y, color);
      }
      delay(DELAY_TIME);
      srand(seed);
      for (i=0; i<PIXEL_COUNT; i++) {
         x = random(maxx);
         y = random(maxy);
         color = random(maxcolor);
         if (color == getpixel(x, y))
            putpixel(x, y, 0);
      }
   }

   /* clean up */
   getch();
   closegraph();
```

```
    return 0;
}
```

```
/* getviewsettings example */
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

char *clip[] = { "OFF", "ON" };

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   struct viewporttype viewinfo;
   int midx, midy, ht;
   char topstr[80], botstr[80], clipstr[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get information about current viewport */
   getviewsettings(&viewinfo);

   /* convert text information into strings */
  sprintf(topstr, "(%d, %d) is the upper left viewport
  corner.",viewinfo.left, viewinfo.top);
  sprintf(botstr, "(%d, %d) is the lower right viewport
  corner.",viewinfo.right, viewinfo.bottom);
   sprintf(clipstr, "Clipping is turned %s.", clip[viewinfo.clip]);

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   ht = textheight("W");
   outtextxy(midx, midy, topstr);
   outtextxy(midx, midy+2*ht, botstr);
   outtextxy(midx, midy+4*ht, clipstr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getx example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* move to the screen center point */
   moveto(getmaxx() / 2, getmaxy() / 2);

   /* create a message string */
   sprintf(msg, "<-(%d, %d) is the here.", getx(), gety());

   /* display the message */
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* gety example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* move to the screen center point */
   moveto(getmaxx() / 2, getmaxy() / 2);

   /* create a message string */
   sprintf(msg, "<-(%d, %d) is the here.", getx(), gety());

   /* display the message */
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* graphdefaults example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();

   /* output line with nondefault settings */
   setlinestyle(DOTTED_LINE, 0, 3);
   line(0, 0, maxx, maxy);
   outtextxy(maxx/2, maxy/3, "Before default values are restored.");
   getch();

   /* restore default values for everything */
   graphdefaults();

   /* clear the screen */
   cleardevice();


   /* output line with default settings */
   line(0, 0, maxx, maxy);
   outtextxy(maxx/2, maxy/3, "After restoring default values.");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* grapherrormsg example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define NONSENSE -50

int main(void)
{
   /* force an error to occur */
   int gdriver = NONSENSE, gmode, errorcode;

   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   /* if an error occurred, then output descriptive error message*/
   if (errorcode != grOk) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }


   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* _graphfreemem example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

int main(void)
{

   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode, midx, midy;

   /* clear the text screen */
   clrscr();
   printf("Press any key to initialize graphics mode:");
   getch();
   clrscr();

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* display a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, "Press any key to exit graphics mode:");

   /* clean up */
   getch();
   closegraph();
   return 0;
}

/* called by the graphics kernel to allocate memory */
void far * far _graphgetmem(unsigned size) {
   printf("_graphgetmem called to allocate %d bytes.\n", size);
   printf("hit any key:");
   getch();
   printf("\n");

   /* allocate memory from far heap */
   return farmalloc(size);
}
```

```
/* called by the graphics kernel to free memory */
void far _graphfreemem(void far *ptr, unsigned size) {
   printf("_graphfreemem called to free %d bytes.\n", size);
   printf("hit any key:");
   getch();
   printf("\n");

   /* free ptr from far heap */
   farfree(ptr);
}
```

/* _graphgetmem example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode, midx, midy;

   /* clear the text screen */
   clrscr();
   printf("Press any key to initialize graphics mode:");
   getch();
   clrscr();

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* display a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, "Press any key to exit graphics mode:");

   /* clean up */
   getch();
   closegraph();
   return 0;
}

/* called by the graphics kernel to allocate memory */
void far * far _graphgetmem(unsigned size) {
   printf("_graphgetmem called to allocate %d bytes.\n", size);
   printf("hit any key:");
   getch();
   printf("\n");

   /* allocate memory from far heap */
   return farmalloc(size);
}
```

```
/* called by the graphics kernel to free memory */
void far _graphfreemem(void far *ptr, unsigned size) {
   printf("_graphfreemem called to free %d bytes.\n", size);
   printf("hit any key:");
   getch();
   printf("\n");

   /* free ptr from far heap */
   farfree(ptr);
}
```

/* graphresult example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getdefaultpalette example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;


   /* far pointer to palette structure */
   struct palettetype far *pal = NULL;
   int i;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");


   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* return a pointer to the default palette */
   pal = getdefaultpalette();
   for (i=0; i<pal->size; i++) {
      printf("colors[%d] = %d\n", i, pal->colors[i]);
      getch();
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getdrivername example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;

   /* stores the device driver name */
   char *drivername;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) { /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }
   setcolor(getmaxcolor());

   /* get the name of the device driver in use */
   drivername = getdrivername();

   /* for centering text onscreen */
   settextjustify(CENTER_TEXT, CENTER_TEXT);

   /* output the name of the driver */
   outtextxy(getmaxx() / 2, getmaxy() / 2, drivername);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* gettextsettings example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>


/* the names of the supported fonts */
char *font[] = { "DEFAULT_FONT", "TRIPLEX_FONT", "SMALL_FONT",
  "SANS_SERIF_FONT", "GOTHIC_FONT" };

/* the names of the text directions supported */
char *dir[] = { "HORIZ_DIR", "VERT_DIR" };

/* horizontal text justifications supported */
char *hjust[] = { "LEFT_TEXT", "CENTER_TEXT", "RIGHT_TEXT" };

/* vertical text justifications supported */
char *vjust[] = { "BOTTOM_TEXT", "CENTER_TEXT", "TOP_TEXT" };

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   struct textsettingstype textinfo;
   int midx, midy, ht;
   char fontstr[80], dirstr[80], sizestr[80];
   char hjuststr[80], vjuststr[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get information about current text settings */
   gettextsettings(&textinfo);

   /* convert text information into strings */
   sprintf(fontstr, "%s is the text style.", font[textinfo.font]);
  sprintf(dirstr, "%s is the text direction.",dir[textinfo.direction]);
  sprintf(sizestr, "%d is the text size.", textinfo.charsize);
  sprintf(hjuststr, "%s is the horizontal justification.",
  hjust[textinfo.horiz]);
```

```
  sprintf(vjuststr, "%s is the vertical justification.",
  vjust[textinfo.vert]);

   /* display the information */
   ht = textheight("W");
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, fontstr);
   outtextxy(midx, midy+2*ht, dirstr);
   outtextxy(midx, midy+4*ht, sizestr);
   outtextxy(midx, midy+6*ht, hjuststr);
   outtextxy(midx, midy+8*ht, vjuststr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getfillpattern example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;
   char pattern[8] = {0x00, 0x70, 0x20, 0x27, 0x25, 0x27, 0x04, 0x04};

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();
   setcolor(getmaxcolor());

   /* select a user-defined fill pattern */
   setfillpattern(pattern, getmaxcolor());

   /* fill the screen with the pattern */
   bar(0, 0, maxx, maxy);
   getch();

   /* get the current user-defined fill pattern */
   getfillpattern(pattern);


   /* alter the pattern we grabbed */
   pattern[4] -= 1;
   pattern[5] -= 3;
   pattern[6] += 3;
   pattern[7] -= 4;

   /* select our new pattern */
   setfillpattern(pattern, getmaxcolor());

   /* fill the screen with the new pattern */
   bar(0, 0, maxx, maxy);

   /* clean up */
   getch();
```

```
    closegraph();
    return 0;
}
```

/* getfillsettings example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the fill styles supported */
char *fname[] = { "EMPTY_FILL", "SOLID_FILL", "LINE_FILL", "LTSLASH_FILL",
  "SLASH_FILL", "BKSLASH_FILL", "LTBKSLASH_FILL", "HATCH_FILL",
  "XHATCH_FILL", "INTERLEAVE_FILL", "WIDE_DOT_FILL", "CLOSE_DOT_FILL",
  "USER_FILL" };

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   struct fillsettingstype fillinfo;
   int midx, midy;
   char patstr[40], colstr[40];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get info about current fill pattern and color */
   getfillsettings(&fillinfo);

   /* convert fill information into strings */
   sprintf(patstr, "%s is the fill style.", fname[fillinfo.pattern]);
   sprintf(colstr, "%d is the fill color.", fillinfo.color);

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, patstr);
   outtextxy(midx, midy+2*textheight("W"), colstr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getgraphmode example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, mode;
   char numname[80], modename[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get mode number and name strings */
   mode = getgraphmode();
   sprintf(numname, "%d is the current mode number.", mode);
   sprintf(modename, "%s is the current graphics mode.",
  getmodename(mode));

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, numname);
   outtextxy(midx, midy+2*textheight("W"), modename);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

```
/* getlinesettings example */
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>


/* the names of the line styles supported */
char *lname[] = { "SOLID_LINE", "DOTTED_LINE", "CENTER_LINE",
  "DASHED_LINE", "USERBIT_LINE" };

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   struct linesettingstype lineinfo;
   int midx, midy;
   char lstyle[80], lpattern[80], lwidth[80];


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* get information about current line settings */
   getlinesettings(&lineinfo);

   /* convert line information into strings */
  sprintf(lstyle, "%s is the line style.", lname[lineinfo.linestyle]);
  sprintf(lpattern, "0x%X is the user-defined line pattern.",
  lineinfo.upattern);
  sprintf(lwidth, "%d is the line thickness.", lineinfo.thickness);

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, lstyle);
   outtextxy(midx, midy+2*textheight("W"), lpattern);
   outtextxy(midx, midy+4*textheight("W"), lwidth);

   /* clean up */
   getch();
   closegraph();
   return 0;
```

```
}
```

/* getmaxcolor example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   char colstr[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* grab the color info. and convert it to a string */
   sprintf(colstr, "This mode supports colors 0..%d", getmaxcolor());

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, colstr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* getmaxmode example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   char modestr[80];


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* grab the mode info. and convert it to a string */
   sprintf(modestr, "This driver supports modes 0..%d", getmaxmode());

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, modestr);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* harderr, hardresume, and hardretn example */

```
/*
This program will trap disk errors and
prompt the user for action. Try running it
with no disk in drive A: to invoke its
functions.
*/

#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define IGNORE  0
#define RETRY   1
#define ABORT   2

int buf[500];

/*
define the error messages for trapping disk problems
*/
 static char *err_msg[] = {
     "write protect",
     "unknown unit",
     "drive not ready",
     "unknown command",
     "data error (CRC)",
     "bad request",
     "seek error",
     "unknown media type",
     "sector not found",
     "printer out of paper",
     "write fault",
     "read fault",
     "general failure",
     "reserved",
     "reserved",
     "invalid disk change"
};

error_win(char *msg)
{
    int retval;

    cputs(msg);


/*
prompt for user to press a key to abort, retry, ignore
*/
    while(1)
    {
  retval= getch();
  if (retval == 'a' || retval == 'A')
```

```
  {
      retval = ABORT;
      break;
  }
  if (retval == 'r' || retval == 'R')
  {
      retval = RETRY;
      break;
  }
  if (retval == 'i' || retval == 'I')
  {
      retval = IGNORE;
      break;
  }
    }

    return(retval);
}

/*
pragma warn -par reduces warnings which occur
due to the non use of the parameters
not_used1 and not_used2 to the handler.
*/
 #pragma warn -par
 void handler(unsigned int ax, unsigned int not_used1, unsigned int
  *not_used2)

{
    static char msg[80];
    unsigned di;
    int drive;
    int errorno;

    di= _DI;
 /*
 if this is not a disk error then it was
 another device having trouble
 */

    if (ax & 0X80)
    {
       /* report the error */
       error_win("Device error");
       /* and return to the program directly requesting abort */
       _hardretn(ABORT);
    }
 /* otherwise it was a disk error */
    drive = ax & 0x00FF;
    errorno = di & 0x00FF;
 /* report which error it was */
    sprintf(msg, "Error: %s on drive %c\r\nA)bort, R)etry, I)gnore: ",
      err_msg[errorno], 'A' + drive);
 /*
 return to the program via dos interrupt 0x23 with abort, retry,
```

```
 or ignore as input by the user.
 */
    _hardresume(error_win(msg));
//    return ABORT;
 }
 #pragma warn +par

 int main(void)
 {
 /*
 install our handler on the hardware problem interrupt

 */
  _harderr(handler);
  clrscr();
  printf("Make sure there is no disk in drive A:\n");
  printf("Press any key ....\n");
  getch();
  printf("Trying to access drive A:\n");
  printf("fopen returned %p\n",fopen("A:temp.dat", "w"));
  return 0;
}
```

/* imagesize example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define ARROW_SIZE 10

void draw_arrow(int x, int y);

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   void *arrow;
   int x, y, maxx;
   unsigned int size;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxx = getmaxx();
   x = 0;
   y = getmaxy() / 2;

   /* draw the image to be grabbed */
   draw_arrow(x, y);

   /* calculate the size of the image */
   size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE);

   /* allocate memory to hold the image */
   arrow = malloc(size);

   /* grab the image */
   getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE, arrow);

   /* repeat until a key is pressed */
   while (!kbhit()) {
      /* erase old image */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
      x += ARROW_SIZE;
      if (x >= maxx)
          x = 0;
```

```
      /* plot new image */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
   }

   /* clean up */
   free(arrow);
   closegraph();
   return 0;
}

void draw_arrow(int x, int y)
{
   /* draw an arrow on the screen */
   moveto(x, y);
   linerel(4*ARROW_SIZE, 0);
   linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
   linerel(0, 2*ARROW_SIZE);
   linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}
```

/* initgraph example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* return with error code */
   }

   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* installuserdriver example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* function prototypes */
int huge detectEGA(void);
void checkerrors(void);
int main(void)
{
   int gdriver, gmode;

   /* install a user written device driver */
   gdriver = installuserdriver("EGA", detectEGA);

   /* must force use of detection routine */
   gdriver = DETECT;

   /* check for any installation errors */
   checkerrors();

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* check for any initialization errors */
   checkerrors();

   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}

/* detects EGA or VGA cards */
int huge detectEGA(void)
{
   int driver, mode, sugmode = 0;
   detectgraph(&driver, &mode);
   if ((driver == EGA) || (driver == VGA))
      return sugmode;      /* return suggested video mode number */
   else
      return grError;      /* return an error code */
}

/* check for and report any graphics errors */
void checkerrors(void)
{
   int errorcode;

   /* read result of last graphics operation */
```

```
   errorcode = graphresult();
   if (errorcode != grOk) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }
}
```

/* installuserfont example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* function prototype */
void checkerrors(void);
int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode;
   int userfont;
   int midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* check for any initialization errors */
   checkerrors();

   /* install a user-defined font file */
   userfont = installuserfont("USER.CHR");

   /* check for any installation errors */
   checkerrors();

   /* select the user font */
   settextstyle(userfont, HORIZ_DIR, 4);

   /* output some text */
   outtextxy(midx, midy, "Testing!");

   /* clean up */
   getch();
   closegraph();
   return 0;
}

/* check for and report any graphics errors */
void checkerrors(void)
{
   int errorcode;

   /* read result of last graphics operation */
   errorcode = graphresult();
   if (errorcode != grOk) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
```

```
        exit(1);
    }
}
```

/* line example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{

   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int xmax, ymax;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   if (errorcode != grOk) { /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }

   setcolor(getmaxcolor());
   xmax = getmaxx();
   ymax = getmaxy();

   /* draw a diagonal line */
   line(0, 0, xmax, ymax);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* linerel example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }

   /* move the CP to location (20,30) */
   moveto(20,30);

   /* create and output a message at (20,30) */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtextxy(20,30, msg);

   /* draw line to a point a relative distance away from current CP */
   linerel(100, 100);

   /* create and output a message at CP */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* lineto example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }

   /* move the CP to location (20,30) */
   moveto(20, 30);

   /* create and output a message at (20,30) */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtextxy(20,30, msg);

   /* draw a line to (100,100) */
   lineto(100, 100);

   /* create and output a message at CP */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* moverel example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* move the CP to location (20,30) */
   moveto(20,30);

   /* plot a pixel at the CP */
   putpixel(getx(), gety(), getmaxcolor());

   /* create and output a message at (20,30) */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtextxy(20,30, msg);

   /* move to a point a relative distance away from the current CP */
   moverel(100, 100);

   /* plot a pixel at the CP */
   putpixel(getx(), gety(), getmaxcolor());

   /* create and output a message at CP */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* moveto example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* move the CP to location (20,30) */
   moveto(20,30);

   /* plot a pixel at the CP */
   putpixel(getx(), gety(), getmaxcolor());

   /* create and output a message at (20,30) */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtextxy(20,30, msg);

   /* move to (100,100) */
   moveto(100,100);

   /* plot a pixel at the CP */
   putpixel(getx(), gety(), getmaxcolor());

   /* create and output a message at CP */
   sprintf(msg, " (%d, %d)", getx(), gety());
   outtext(msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* outtext example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* move the CP to the center of the screen */
   moveto(midx, midy);

   /* output text starting at the CP */
   outtext("This ");
   outtext("is ");
   outtext("a ");
   outtext("test.");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* outtextxy example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* output text at center of the screen; CP doesn't get changed */
   outtextxy(midx, midy, "This is a test.");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* pieslice example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int stangle = 45, endangle = 135, radius = 100;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* set fill style and draw a pie slice */
   setfillstyle(EMPTY_FILL, getmaxcolor());
   pieslice(midx, midy, stangle, endangle, radius);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* putimage example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define ARROW_SIZE 10

void draw_arrow(int x, int y);

int main()
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   void *arrow;
   int x, y, maxx;
   unsigned int size;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxx = getmaxx();
   x = 0;
   y = getmaxy() / 2;
   draw_arrow(x, y);

   /* calculate the size of the image and allocate space for it */
   size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE);
   arrow = malloc(size);

   /* grab the image */
   getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE, arrow);

   /* repeat until a key is pressed */
   while (!kbhit()) {
      /* erase old image */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
      x += ARROW_SIZE;
      if (x >= maxx)
         x = 0;

      /* plot new image */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
   }
```

```
   free(arrow);
   closegraph();
   return 0;
}
void draw_arrow(int x, int y) {
   moveto(x, y);
   linerel(4*ARROW_SIZE, 0);
   linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
   linerel(0, 2*ARROW_SIZE);
   linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}
```

/* putpixel example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define PIXEL_COUNT 1000
#define DELAY_TIME  100  /* in milliseconds */

int main()
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int i, x, y, color, maxx, maxy, maxcolor, seed;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxx = getmaxx() + 1;
   maxy = getmaxy() + 1;
   maxcolor = getmaxcolor() + 1;

   while (!kbhit())
   {
      /* seed the random number generator */
      seed = random(32767);
      srand(seed);
      for (i=0; i<PIXEL_COUNT; i++) {
         x = random(maxx);
         y = random(maxy);
         color = random(maxcolor);
         putpixel(x, y, color);
      }
      delay(DELAY_TIME);
      srand(seed);
      for (i=0; i<PIXEL_COUNT; i++) {
         x = random(maxx);
         y = random(maxy);
         color = random(maxcolor);
         if (color == getpixel(x, y))
            putpixel(x, y, 0);
      }
   }
```

```
   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* randbrd example */

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

int main(void)
{
   char far *save_dta;
   char line[80], buffer[256];
   struct fcb blk;
   int i, result;

   /* get user input file name for dta */
   printf("Enter drive and file name (no path - i.e. a:file.dat)\n");
   gets(line);

   /* put file name in fcb */
   if (!parsfnm(line, &blk, 1))
   {
      printf("Error in call to parsfnm\n");
      exit(1);
   }
   printf("Drive #%d  File: %s\n\n", blk.fcb_drive, blk.fcb_name);

   /* open file with DOS FCB open file */
   bdosptr(0x0F, &blk, 0);

   /* save old dta, and set new one */
   save_dta = getdta();
   setdta(buffer);

   /* set up info for the new dta */
   blk.fcb_recsize = 128;
   blk.fcb_random = 0L;
   result = randbrd(&blk, 1);

   /* check results from randbrd */
   if (!result)
      printf("Read OK\n\n");
   else
   {
      perror("Error during read");
      exit(1);
   }

   /* read in data from the new dta */
   printf("The first 128 characters are:\n");
   for (i=0; i<128; i++)
      putchar(buffer[i]);

   /* restore previous dta */
   setdta(save_dta);
```

```
    return 0;
}
```

/* randbwr example */

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

int main(void)
{
   char far *save_dta;
   char line[80];
   char buffer[256] = "RANDBWR test!";
   struct fcb blk;
   int result;

   /* get new file name from user */
   printf("Enter a file name to create (no path - ie. a:file.dat\n");
   gets(line);

   /* parse the new file name to the dta */
   parsfnm(line,&blk,1);
   printf("Drive #%d  File: %s\n", blk.fcb_drive, blk.fcb_name);

   /* request DOS services to create file */
   if (bdosptr(0x16, &blk, 0) == -1)
   {
      perror("Error creating file");
      exit(1);
   }

   /* save old dta and set new dta */
   save_dta = getdta();
   setdta(buffer);

   /* write new records */
   blk.fcb_recsize = 256;
   blk.fcb_random = 0L;
   result = randbwr(&blk, 1);

   if (!result)
      printf("Write OK\n");
   else
   {
      perror("Disk error");
      exit(1);
   }

   /* request DOS services to close the file */
   if (bdosptr(0x10, &blk, 0) == -1)
   {
      perror("Error closing file");
      exit(1);
   }

   /* reset the old dta */
```

```
    setdta(save_dta);

    return 0;
}
```

/* rectangle example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int left, top, right, bottom;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   left = getmaxx() / 2 - 50;
   top = getmaxy() / 2 - 50;
   right = getmaxx() / 2 + 50;
   bottom = getmaxy() / 2 + 50;

   /* draw a rectangle */
   rectangle(left,top,right,bottom);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* registerbgifont example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;

   /* register a font file that was added into GRAPHICS.LIB */
   errorcode = registerbgifont(triplex_font);

   /* report any registration errors */
   if (errorcode < 0) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* select the registered font */
   settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);

   /* output some text */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, "The TRIPLEX FONT");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* registerbgidriver example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;

   /* register a driver that was added into GRAPHICS.LIB */
   errorcode = registerbgidriver(EGAVGA_driver);

   /* report any registration errors */
   if (errorcode < 0) {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* draw a line */
   line(0, 0, getmaxx(), getmaxy());

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* restorecrtmode example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int x, y;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* output a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "Press any key to exit graphics:");
   getch();

   /* restore system to text mode */
   restorecrtmode();
   printf("We're now in text mode.\n");
   printf("Press any key to return to graphics mode:");
   getch();

   /* return to graphics mode */
   setgraphmode(getgraphmode());

   /* output a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "We're back in graphics mode.");
   outtextxy(x, y+textheight("W"), "Press any key to halt:");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* sbrk example */

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

int main(void)
{
   printf("Changing allocation with sbrk()\n");
   printf("Before sbrk() call: %lu bytes free\n",
          (unsigned long) coreleft());
   sbrk(1000);
   printf(" After sbrk() call: %lu bytes free\n",
           (unsigned long) coreleft());
   return 0;
}
```

/* sector example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, i;
   int stangle = 45, endangle = 135;
   int xrad = 100, yrad = 50;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* loop through the fill patterns */
   for (i=EMPTY_FILL; i<USER_FILL; i++) {

      /* set the fill style */
      setfillstyle(i, getmaxcolor());

      /* draw the sector slice */
      sector(midx, midy, stangle, endangle, xrad, yrad);
      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

/* setactivepage example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* select driver and mode that supports multiple pages */
   int gdriver = EGA, gmode = EGAHI, errorcode;
   int x, y, ht;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   x = getmaxx() / 2;
   y = getmaxy() / 2;
   ht = textheight("W");

   /*  select the off screen page for drawing */
   setactivepage(1);

   /* draw a line on page #1 */
   line(0, 0, getmaxx(), getmaxy());

   /* output a message on page #1 */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "This is page #1:");
   outtextxy(x, y+ht, "Press any key to halt:");

   /* select drawing to page #0 */
   setactivepage(0);

   /* output a message  on page #0 */
   outtextxy(x, y, "This is page #0.");
   outtextxy(x, y+ht, "Press any key to view page #1:");
   getch();

   /* select page #1 as the visible page */
   setvisualpage(1);

   /* clean up */
   getch();
   closegraph();
```

```
    return 0;
}
```

/* setallpalette example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   struct palettetype pal;
   int color, maxcolor, ht;
   int y = 10;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxcolor = getmaxcolor();
   ht = 2 * textheight("W");

   /* grab a copy of the palette */
   getpalette(&pal);

   /* display the default palette colors */
   for (color=1; color<=maxcolor; color++) {
      setcolor(color);
      sprintf(msg, "Color: %d", color);
      outtextxy(1, y, msg);
      y += ht;
   }

   /* wait for a key */
   getch();

   /* black out the colors one by one */
   for (color=1; color<=maxcolor; color++) {
      setpalette(color, BLACK);
      getch();
   }

   /* restore the palette colors */
   setallpalette(&pal);
```

```
   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* setaspectratio example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int xasp, yasp, midx, midy;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");


   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setcolor(getmaxcolor());

   /* get current aspect ratio settings */
   getaspectratio(&xasp, &yasp);

   /* draw normal circle */
   circle(midx, midy, 100);
   getch();

   /* clear the screen */
   cleardevice();

   /* adjust the aspect for a wide circle */
   setaspectratio(xasp/2, yasp);
   circle(midx, midy, 100);
   getch();

   /* adjust the aspect for a narrow circle */
   cleardevice();
   setaspectratio(xasp, yasp/2);
   circle(midx, midy, 100);

   /* clean up */
   getch();
   closegraph();
   return 0;
```

```
}
```

/* setbkcolor example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* _select driver and mode that supports multiple background colors*/
   int gdriver = EGA, gmode = EGAHI, errorcode;
   int bkcol, maxcolor, x, y;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {   /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* maximum color index supported */
   maxcolor = getmaxcolor();

   /* for centering text messages */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* loop through the available colors */
   for (bkcol=0; bkcol<=maxcolor; bkcol++) {

      /* clear the screen */
      cleardevice();

      /* select a new background color */
      setbkcolor(bkcol);

      /* output a messsage */
      if (bkcol == WHITE)
         setcolor(EGA_BLUE);
      sprintf(msg, "Background color: %d", bkcol);
      outtextxy(x, y, msg);
      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

/* setcolor example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* select driver and mode that supports multiple drawing colors */
   int gdriver = EGA, gmode = EGAHI, errorcode;
   int color, maxcolor, x, y;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* maximum color index supported */
   maxcolor = getmaxcolor();

   /* for centering text messages */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* loop through the available colors */
   for (color=1; color<=maxcolor; color++) {
      cleardevice();         /* clear the screen */
      setcolor(color);       /* select new background color */

      /* output a messsage */
      sprintf(msg, "Color: %d", color);
      outtextxy(x, y, msg);
      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

/* setfillpattern example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int maxx, maxy;

   /* a user-defined fill pattern */
   char pattern[8] = {0x00, 0x70, 0x20, 0x27, 0x24, 0x24, 0x07, 0x00};

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxx = getmaxx();
   maxy = getmaxy();
   setcolor(getmaxcolor());

   /* select a user-defined fill pattern */
   setfillpattern(pattern, getmaxcolor());

   /* fill the screen with the pattern */
   bar(0, 0, maxx, maxy);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* setfillstyle example */

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

/* the names of the fill styles supported */
char *fname[] = { "EMPTY_FILL", "SOLID_FILL", "LINE_FILL", "LTSLASH_FILL",
  "SLASH_FILL", "BKSLASH_FILL", "LTBKSLASH_FILL", "HATCH_FILL",
  "XHATCH_FILL", "INTERLEAVE_FILL", "WIDE_DOT_FILL", "CLOSE_DOT_FILL",
  "USER_FILL" };

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int style, midx, midy;
   char stylestr[40];


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   for (style = EMPTY_FILL; style < USER_FILL; style++) {
      /* select the fill style */
      setfillstyle(style, getmaxcolor());

      /* convert style into a string */
      strcpy(stylestr, fname[style]);

      /* fill a bar */
      bar3d(0, 0, midx-10, midy, 0, 0);

      /* output a message */
      outtextxy(midx, midy, stylestr);

      /* wait for a key */
      getch();
      cleardevice();
   }

   /* clean up */
```

```
    getch();
    closegraph();
    return 0;
}
```

/* setlinestyle example */

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

/* the names of the line styles supported */
char *lname[] = { "SOLID_LINE", "DOTTED_LINE", "CENTER_LINE",
  "DASHED_LINE", "USERBIT_LINE" };

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int style, midx, midy, userpat;
   char stylestr[40];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* a user-defined line pattern */
   /* binary: "0000000000000001" */
   userpat = 1;
   for (style=SOLID_LINE; style<=USERBIT_LINE; style++)
   {
      /* select the line style */
      setlinestyle(style, userpat, 1);

      /* convert style into a string */
      strcpy(stylestr, lname[style]);

      /* draw a line */
      line(0, 0, midx-10, midy);


      /* draw a rectangle */
      rectangle(0, 0, getmaxx(), getmaxy());

      /* output a message */
      outtextxy(midx, midy, stylestr);
```

```
      /* wait for a key */
      getch();
      cleardevice();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

/* setpalette example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int color, maxcolor, ht;
   int y = 10;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   maxcolor = getmaxcolor();
   ht = 2 * textheight("W");

   /* display the default colors */
   for (color=1; color<=maxcolor; color++) {
      setcolor(color);
      sprintf(msg, "Color: %d", color);
      outtextxy(1, y, msg);
      y += ht;
   }

   /* wait for a key */
   getch();

   /* black out the colors one by one */
   for (color=1; color<=maxcolor; color++) {
      setpalette(color, BLACK);
      getch();
   }

   /* clean up */
   closegraph();
   return 0;
}
```

/* setrgbpalette example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* select driver and mode that supports use of setrgbpalette */
   int gdriver = VGA, gmode = VGAHI, errorcode;
   struct palettetype pal;
   int i, ht, y, xmax;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* grab a copy of the palette */
   getpalette(&pal);


   /* create gray scale */
   for (i=0; i<pal.size; i++)
      setrgbpalette(pal.colors[i], i*4, i*4, i*4);

   /* display the gray scale */
   ht = getmaxy() / 16;
   xmax = getmaxx();
   y = 0;
   for (i=0; i<pal.size; i++) {
      setfillstyle(SOLID_FILL, i);
      bar(0, y, xmax, y+ht);
      y += ht;
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* settextjustify example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* function prototype */
void xat(int x, int y);

/* horizontal text justification settings */
char *hjust[] = { "LEFT_TEXT", "CENTER_TEXT", "RIGHT_TEXT" };

/* vertical text justification settings */
char *vjust[] = { "LEFT_TEXT", "CENTER_TEXT", "RIGHT_TEXT" };

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy, hj, vj;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* loop through text justifications */
   for (hj=LEFT_TEXT; hj<=RIGHT_TEXT; hj++)
      for (vj=LEFT_TEXT; vj<=RIGHT_TEXT; vj++) {
         cleardevice();

         /* set the text justification */
         settextjustify(hj, vj);

         /* create a message string */
         sprintf(msg, "%s  %s", hjust[hj], vjust[vj]);

         /* create crosshairs on the screen */
         xat(midx, midy);

         /* output the message */
         outtextxy(midx, midy, msg);
         getch();
```

```
      }

   /* clean up */
   closegraph();
   return 0;
}

void xat(int x, int y)       /* draw an x at (x,y) */
{
  line(x-4, y, x+4, y);
  line(x, y-4, x, y+4);
}
```

/* settextstyle example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the text styles supported */
char *fname[] = { "DEFAULT font",  "TRIPLEX font",
                  "SMALL font",    "SANS SERIF_font",
                  "GOTHIC_font",   "SCRIPT font",
                  "SIMPLEX font",  "TRIPLEX SCRIPT font",
                  "COMPLEX font",  "EUROPEAN font",
                  "BOLD font"};

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int style, midx, midy;
   int size = 1;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   settextjustify(CENTER_TEXT, CENTER_TEXT);

   /* loop through the available text styles */
   for (style=DEFAULT_FONT; style<=BOLD_FONT; style++) {
      cleardevice();
      if (style == TRIPLEX_FONT)
         size = 4;
      /* select the text style */
      settextstyle(style, HORIZ_DIR, size);

      /* output a message */
      outtextxy(midx, midy, fname[style]);
      getch();
   }
   /* clean up */
   closegraph();
   return 0;
}
```

/* setusercharsize example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* select a text style */
   settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);

   /* move to the text starting position */
   moveto(0, getmaxy() / 2);

   /* output some normal text */
   outtext("Norm ");

   /* make the text 1/3 the normal width */
   setusercharsize(1, 3, 1, 1);
   outtext("Short ");

   /* make the text 3 times normal width */
   setusercharsize(3, 1, 1, 1);
   outtext("Wide");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* setviewport example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define CLIP_ON 1           /* activates clipping in viewport */

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   setcolor(getmaxcolor());

   /* message in default full-screen viewport */
   outtextxy(0, 0, "* <-- (0, 0) in default viewport");

   /* create a smaller viewport */
   setviewport(50, 50, getmaxx()-50, getmaxy()-50, CLIP_ON);

   /* display some text */
   outtextxy(0, 0, "* <-- (0, 0) in smaller viewport");

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* setvisualpage example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* select driver and mode that supports multiple pages */
   int gdriver = EGA, gmode = EGAHI, errorcode;
   int x, y, ht;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   x = getmaxx() / 2;
   y = getmaxy() / 2;
   ht = textheight("W");

   /*  select the off screen page for drawing */
   setactivepage(1);

   /* draw a line on page #1 */
   line(0, 0, getmaxx(), getmaxy());

   /* output a message on page #1 */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "This is page #1:");
   outtextxy(x, y+ht, "Press any key to halt:");

   /* select drawing to page #0 */
   setactivepage(0);

   /* output a message  on page #0 */
   outtextxy(x, y, "This is page #0.");
   outtextxy(x, y+ht, "Press any key to view page #1:");
   getch();

   /* select page #1 as the visible page */
   setvisualpage(1);

   /* clean up */
   getch();
   closegraph();
```

```
    return 0;
}
```

/* setwritemode example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int xmax, ymax;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk)    /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   xmax = getmaxx();
   ymax = getmaxy();

   /* select XOR drawing mode */
   setwritemode(XOR_PUT);

   /* draw a line */
   line(0, 0, xmax, ymax);
   getch();

   /* erase the line by drawing over it */
   line(0, 0, xmax, ymax);
   getch();

   /* select overwrite drawing mode */
   setwritemode(COPY_PUT);

   /* draw a line */
   line(0, 0, xmax, ymax);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* setgraphmode example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int x, y;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* output a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "Press any key to exit graphics:");
   getch();

   /* restore system to text mode */
   restorecrtmode();
   printf("We're now in text mode.\n");
   printf("Press any key to return to graphics mode:");
   getch();

   /* return to graphics mode */
   setgraphmode(getgraphmode());

   /* output a message */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, "We're back in graphics mode.");
   outtextxy(x, y+textheight("W"), "Press any key to halt:");


   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* setgraphbufsize example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define BUFSIZE 1000 /* internal graphics buffer size */

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int x, y, oldsize;
   char msg[80];

   /* _set size of internal graphics buffer before calling initgraph */
   oldsize = setgraphbufsize(BUFSIZE);


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   x = getmaxx() / 2;
   y = getmaxy() / 2;

   /* output some messages */
   sprintf(msg, "Graphics buffer size: %d", BUFSIZE);
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(x, y, msg);
   sprintf(msg, "Old graphics buffer size: %d", oldsize);
   outtextxy(x, y+textheight("W"), msg);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* sound and nosound example */

```
/* Emits a 7-Hz tone for 10 seconds.

     True story: 7 Hz is the resonant
     frequency of a chicken's skull cavity.
     This was determined empirically in
     Australia, where a new factory
     generating 7-Hz tones was located too
     close to a chicken ranch: When the
     factory started up, all the chickens
     died.

     Your PC may not be able to emit a 7-Hz tone. */

#include <dos.h>

int main(void)
{
   sound(7);
   delay(10000);
   nosound();
   return 0;
}
```

/* textheight example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int y = 0;
   int i;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   /* draw some text on the screen */
   for (i=1; i<11; i++) {
      /* select the text style, direction, and size */
      settextstyle(TRIPLEX_FONT, HORIZ_DIR, i);

      /* create a message string */
      sprintf(msg, "Size: %d", i);

      /* output the message */
      outtextxy(1, y, msg);

      /* advance to the next text line */
      y += textheight(msg);
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/* textwidth example */

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int x = 0, y = 0;
   int i;
   char msg[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) {  /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);               /* terminate with an error code */
   }

   y = getmaxy() / 2;
   settextjustify(LEFT_TEXT, CENTER_TEXT);
   for (i = 1; i < 11; i++) {
      /* select the text style, direction, and size */
      settextstyle(TRIPLEX_FONT, HORIZ_DIR, i);

      /* create a message string */
      sprintf(msg, "Size: %d", i);

      /* output the message */
      outtextxy(x, y, msg);

      /* advance to the end of the text */
      x += textwidth(msg);
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

/*_OvrInitEms example */

```
#include <dos.h>
/* This function is in overlay.lib  */

int main(void)
{
   /* ask overlay manager to check for expanded
      memory and allow it to use 16 pages (256K) available
      only in medium, large, and huge memory models
   */

 /* In the IDE, set Options|Applications to Dos Overlay */


  _OvrInitEms (0, 0, 16);

   return 0;
}
```

/* _OvrInitExt example */

```
#include <dos.h>

int main(void)
{
/* Available only in medium, large and huge memory models  */

/* Use the extended memory from the linear address 0x200000L (2MB),
   as much as necessary */

 /* In the IDE, set Options|Applications to Dos Overlay */

  _OvrInitExt (0x200000L, 0);

  return 0;
}
```

/* setblock example */

```
#include <dos.h>
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)  /* Example for setblock. */
{
   unsigned int size, segp;
   int stat;
   size = 64;               /* (64 x 16) = 1024 bytes */
   stat = allocmem(size, &segp);
   if (stat == -1)
      printf("Allocated memory at segment: %X\n", segp);
   else {
      printf("Failed: maximum number of paragraphs available is %d\n",
  stat);
      exit(1);
   }

   stat = setblock(segp, size * 2);
   if (stat == -1)
      printf("Expanded memory block at segment: %X\n", segp);
   else
      printf("Failed: maximum number of paragraphs available is %d\n",
  stat);
   freemem(segp);
   return 0;
}
```

/* _dos_setblock example */

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

int main(void)  /* Example for _dos_setblock. */
{
   unsigned int size, segp, err, maxb;
   size = 64; /* (64 x 16) = 1024 bytes */
   err = _dos_allocmem(size, &segp);
   if (err == 0)
      printf("Allocated memory at segment: %x\n", segp);
   else {
      perror("Unable to allocate block");
      printf("Maximum no. of paragraphs available is %u\n", segp);
      return 1;
   }
   if (_dos_setblock(size * 2, segp, &maxb) == 0)
      printf("Expanded memory block at segment: %X\n", segp);
   else {
      perror("Unable to expand block");
      printf("Maximum no. of paragraphs available is %u\n", maxb);
   }
   _dos_freemem(segp);
   return 0;
}
```

/* _harderr, _hardresume, and _hardretn example */

```
/* This program traps disk errors and prompts the user for action. */
/* Try running it with no disk in drive A to invoke its functions. */

#include <stdio.h>
#include <ctype.h>
#include <dos.h>
#include <fcntl.h>

int buf[500];

/* Define the error messages for trapping disk problems. */
static char *err_msg[] =
{
  "write protect",    "unknown unit",
  "drive not ready", "unknown command",
  "data error (CRC)","bad request",
  "seek error",       "unknown media type",
  "sector not found","printer out of paper",
  "write fault",      "read fault",
  "general failure", "reserved",
  "reserved",         "invalid disk change"
};

static void mesg(char *s)
{
   while (*s)
      bdos(2,*s++,0);
}

static int getkey(void)
{
  return (bdos(7, 0, 0) & 0xff);
}

error_win(char *msg)
{
  int c;

  /* Prompt user to press a key to abort, retry, ignore, fail.*/
  while(1) {
      mesg(msg);
      c = tolower(getkey());
      mesg("\r\n");
      switch (c) {
         case 'a':
        return (_HARDERR_ABORT);
         case 'r':
            return (_HARDERR_RETRY);
         case 'i':
            return (_HARDERR_IGNORE);
         case 'f':
            return (_HARDERR_FAIL);
      }
```

```
   }
}

/* Pragma warn -par reduces warnings which occur due to the nonuse of the
  parameter devhdr */
#pragma warn -par

void far handler(unsigned deverr, unsigned errval,
            unsigned far *devhdr)
{
  static char msg[80];
  int drive, errorno;

  /* _If this not disk error then another device having trouble._*/
  if (deverr & 0x8000) {
    error_win("Device error"); /* report the error */
    /* return to the program directly requesting abort */
    _hardretn(5);              /* 5 = DOS "access denied" error */
  }
  drive = deverr & 0x00FF;      /* otherwise it was disk error */
  errorno = errval & 0x00FF;

  /* report which error it was */
  sprintf(msg, "Error: %s on drive %c\r\nA)bort, R)etry,\
        I)gnore, F)ail: ",
        err_msg[errorno], 'A' + drive);

  /* Return to program via dos interrupt 0x23 with abort, retry or ignore
  as input by the user */
   _hardresume(error_win(msg));
}

#pragma warn +par

int main(void)
{
   int handle;

   /* Install our handler on the hardware problem interrupt.*/
   _harderr(handler);
   printf("Make sure there is no disk in drive A:\n");
   printf("Press any key ....\n");
   getkey();
   printf("Trying to access drive A:\n");
   printf("_dos_open returned 0x%x\n",
           _dos_open("A:temp.dat", O_RDONLY, &handle));
   return 0;
}
```

DPMI - Borland PowerPack for DOS

You have requested Help about a DPMI feature in the Borland PowerPack for DOS. The Borland PowerPack for DOS is not installed on your system.

Turbo Vision Library - Borland PowerPack for DOS

You have requested Help about the Turbo Vision Library, a feature in the Borland PowerPack for DOS. The Borland PowerPack for DOS is not installed on your system.