

Q. 1) What is File Mode? Types of file mode? What is Pointer? Which one is most useful in Structure or Union?

Answer: The meaning of file mode means in which mode we want to open the file. Types of modes are read mode, write mode. pointer is the variable which holds the address of another variable. union is useful because of memory no wastage but structures are frequently used than unions.

Q. 2) How can I open a file so that other programs can update it at the same time?

Answer: Your C compiler library contains a low-level file function called `sopen()` that can be used to open a file in shared mode. Beginning with DOS 3.0, files could be opened in shared mode by loading a special program named SHARE.EXE. Shared mode, as the name implies, allows a file to be shared with other programs as well as your own.

Using this function, you can allow other programs that are running to update the same file you are updating.

The `sopen()` function takes four parameters: a pointer to the filename you want to open, the operational mode you want to open the file in, the file sharing mode to use, and, if you are creating a file, the mode to create the file in. The second parameter of the `sopen()` function, usually referred to as the "operation flag" parameter, can have the following values assigned to it:

Constant	Description
O_APPEND	Appends all writes to the end of the file
O_BINARY	Opens the file in binary (untranslated) mode
O_CREAT	If the file does not exist, it is created
O_EXCL	If the O_CREAT flag is used and the file exists, returns an error
O_RDONLY	Opens the file in read-only mode
O_RDWR	Opens the file for reading and writing
O_TEXT	Opens the file in text (translated) mode
O_TRUNC	Opens an existing file and writes over its contents
O_WRONLY	Opens the file in write-only mode

Q. 3) How can I make sure that my program is the only one accessing a file?

Answer: By using the `sopen()` function you can open a file in shared mode and explicitly deny reading and writing permissions to any other program but yours. This task is accomplished by using the SH_DENYWR shared flag to denote that your program is going to deny any writing or reading attempts by other programs.

For example, the following snippet of code shows a file being opened in shared mode, denying access to all other files:

```
/* Note that the sopen() function is not ANSI compliant... */  
fileHandle = sopen("C:DATASETUP.DAT", O_RDWR, SH_DENYWR);
```

By issuing this statement, all other programs are denied access to the SETUP.DAT file. If another program were to try to open SETUP.DAT for reading or writing, it would receive an EACCESS error code, denoting that access is denied to the file.

Q. 4) If we develop a project in C, then how can we create an exe file of it?

Answer: Assume Your Project Contains 3 different files + 1 main file containing `main()` function Required Steps 2 Compile are:

1) Add/Link The File To The File Containing `main()` function by using preprocessor statements i.e

```
#include"<filename>"  
#include"<filename>"  
#include"<filename>"  
#include"<filename>"
```

2) Use Compile Option or Use Compile Project

Q. 5) How can we read/write structures from/to data files?

Answer: By using `fread()` and `fwrite()` functions one can read or write data to data files.

Q. 6) How do you redirect a standard stream?

Answer: Most operating systems, including DOS, provide a means to redirect program input and output to and from different devices. This means that rather than your program output (`stdout`) going to the screen; it can be redirected to a file or printer port. Similarly, your program's input (`stdin`) can come from a file rather than the keyboard. In DOS, this task is accomplished using the redirection characters, `<` and `>`. For example, if you wanted a program named `PRINTIT.EXE` to receive its input (`stdin`) from a file named `STRINGS.TXT`, you would enter the following command at the DOS prompt:

```
C:>PRINTIT < STRINGS.TXT
```

Notice that the name of the executable file always comes first. The less-than sign (<) tells DOS to take the strings contained in `STRINGS.TXT` and use them as input for the `PRINTIT` program. The following example would redirect the program's output to the print device, usually the printer attached on `LPT1`:

```
C :> REDIR > PRN
```

Alternatively, you might want to redirect the program's output to a file, as the following example shows:

```
C :> REDIR > REDIR.OUT
```

In this example, all output that would have normally appeared on-screen will be written to the file `REDIR.OUT`.

Redirection of standard streams does not always have to occur at the operating system. You can redirect a standard stream from within your program by using the standard C library function named `freopen()`. For example, if you wanted to redirect the `stdout` standard stream within your program to a file named `OUTPUT.TXT`, you would implement the `freopen()` function as shown here:

```
freopen("output.txt", "w", stdout);
```

Now, every output statement (`printf()`, `puts()`, `putch()`, and so on) in your program will appear in the file `OUTPUT.TXT`.

Q. 7) Write a program for creating your own header file and library function?

Answer:

```
int fact(int n)
{
    int f=1;
    if(n>7)
    {
        printf("SORRY We can't find a factorial for a no>7");
        return -1;
    }
    while(n)
    f=f*n00;
    return f;
}
```

now this function save as `myfuncs.h` if u want to reuse this function in any another program then You first write this header file name with codes like `# include "myfuncs.h"` .after that u can write your program and u can cal and use fact fun in this entire program..

Q. 10) How can you restore a redirected standard stream?

Answer: The preceding example showed how you can redirect a standard stream from within your program. But what if later in your program you wanted to restore the standard stream to its original state. By using the standard C library functions named `dup()` and `fdopen()`, you can restore a standard stream such as `stdout` to its original state.

The `dup()` function duplicates a file handle. You can use the `dup()` function to save the file handle corresponding to the `stdout` standard stream. The `fdopen()` function opens a stream that has been duplicated with the `dup()` function.

Q. 11) read a A file aaaaabbbbb.. and read a B file 11112222... in c file to write aaaaa11111bbbbbb22222 like this?

Answer: Read the contents of file a in a buffer ... fileA. Read the contents of file B in a buffer fileB Read the first character of the buffer fileA in a variable sample. Now start going through the buffer. The moment you hit a char which is not same as sample, change the value of sample to this new character and now start reading from fileB.

Q. 12) What is the difference between text and binary modes?

Answer: Streams can be classified into two types: text streams and binary streams. Text streams are interpreted, with a maximum length of 255 characters. With text streams, carriage return/line feed combinations are translated to the newline `n` character and vice versa. Binary streams are uninterpreted and are treated one byte at a time with no translation of characters. Typically, a text stream would be used for reading and writing standard text files, printing output to the screen or printer, or receiving input from the keyboard.

A binary text stream would typically be used for reading and writing binary files such as graphics or word processing documents, reading mouse input, or reading and writing to the modem.

Q. 13) How can I create the batch files? What is the purpose batch file and use of it?

Answer: Batch files is to make your work easy. instead of executing the command one by one, just write the commands to be executed in a notepad and store as a .bat Now type the name of bat file in the execute path , it starts to execute the commands you have inside of the batch file

Q. 14) What is the benefit of using an `enum` rather than a `#define` constant?

Answer: The use of an enumeration constant (`enum`) has many advantages over using the traditional symbolic constant style of `#define`. These advantages include a lower maintenance requirement, improved program readability, and better debugging capability.

The first advantage is that enumerated constants are generated automatically by the compiler. Conversely, symbolic constants must be manually assigned values by the programmer. For instance, if you had an enumerated constant type for error codes that could occur in your program.

Q. 15) Can include files be nested?

Answer: Yes. Include files can be nested any number of times. As long as you use precautionary measures , you can avoid including the same file twice. In the past, nesting header files was seen as bad programming practice, because it complicates the dependency tracking function of the MAKE program and thus slows down compilation. Many of today's popular compilers make up for this difficulty by implementing a concept called precompiled headers, in which all headers and associated dependencies are stored in a precompiled state.

Many programmers like to create a custom header file that has `#include` statements for every header needed for each module. This is perfectly acceptable and can help avoid potential problems relating to `#include` files, such as accidentally omitting an `#include` file in a module.

Q. 16) how to merge to file in c?

Answer:

Merging to a file is simply appending to a file.