Hacker 1.0

by ***WadeSoft***, March 1994

Disclaimer

This program has been added to the World's collection of public domain utilites. It is a free utility, and therefore neglecting the cost of diskette(s), no price is to be entailed with its distribution.

Purpose

This program was created as a programming aid: to help *WadeSoft* decipher various file formats, such as .WAV and .TGA files. It can be used for this purpose as well as another - to help gamers "cheat". As well as a programmers' best friend, this utility can be also be the gamers' most valued program.

Program Outline

Hacker was designed with four distinct goals in mind. These four functions, which are performed on either textual or binary (.exe) files, can be categorized in the following manner:

a. Numeric/character search.
b. Numeric/character search and replace.
c. Numeric/character modification.
d. File comparison.

Numeric/Character Search Explained

This is probably the single most useful part of Hacker's functions in the eyes of a programmer. It allows the user to search an entire file (textual or binary) for a given byte, integer, word, long integer or sequence of characters. Upon finding an occurrence of the specified value, the program will display it's location inside the file. We used this function to understand the .TGA file format. We knew our image was 640 pixels wide and 480 long, so we searched for the "word" (see "Data Types Explained") "640" inside the file. It told us it's exact location. Upon acquiring this knowledge, our program was able to know the resolution of any image we wanted, not just this particular one. The point is that we can now take more complete advantage of the .TGA file format.

Numeric/Character Search and Replace

When selection what one wants to search for in a file, the program asks if the user wants to do any replacing after the values have been located. If so, then the program asks what to replace the values with. Then it begins the search, and when any identical values are located, the program asks if the user wants the value at this particular location replaced. If yes is answered, the program replaces the located value with the desired one. This is useful for cheating in games. If

one knew that he or she had $540 dollars in a game, they could search their save-game file for 540, and if it were found, it could be replaced with $10,000!

Numeric/Character Modification

When searching for a value, let us say $540 in a saved game, the program will display the location of its findings. Instead of doing a Search/Replace every time the user wanted to change his or her money in the game (and therefore knowing exactly how much money he/she had at the time), the user could just specify the location of the value to change, and change it to $20,000. The gamer would hopefully have written down the location *inside the file* where the money was stored.

File Comparison

Almost always, saved game files are of a constant length. This would indicate that game-related information (such as money) is always stored in the same location. File comparison is only useful for which are of the same length. The reason is that the program reports *every* byte which is different between the two specified files. If one were larger than the other, each addition byte would be reported, making for a huge list of meaningless differences. Let us say that in a role-playing game, we saved the game just prior to and just after having changed our sword for an axe. If a file comparison were performed between the two saved-game files, these changed would be revealed. The user would then know to change byte 45 to value 32 if he wants an axe or value 31 if he wants a sword. What happens when we change it to 56? A halberd perhaps? After having made a comparison, it is best to use the "File Modification" tactic to alter these values. Note that it is useful to create lists of weapons or spells along with their associated byte values and locations. Include your list and this program together in a package, and distribute it them!

Data Types Explained (simplified)

A small amount of "programming" knowledge is required if one wants to use this program. The reason is simple. Because this software functions with ALL types of files and games, no preset locations or file definitions are known to the utility. In other words, the user must specify the *type* of data he is searching for. Data is stored in several different manners, and below is posted a listing of the different data types this program supports (which is virtually all of them). If you are unfamiliar with programming, or more specifically with data storage, here is a brief explanation.

A file is composed of a certain number of bytes. Let us take a 64 kilobyte file as an example. 64k represents 64,000 bytes of information (approximately, since in computerese *kilo* stands for 1024, which yields 65,536 bytes). A number between the range of 0 and 255 (256 possible numbers) requires 1 byte of storage space. If in your saved game, the value you are searching for cannot exceed 255, then it is likely to have been stored as a *byte* value. If your numbers get larger than this (say money, of which you have $4,000), then it has **not** been saved as a byte value. It has probably been saved as something larger, say a *word*. A word is composed of two bytes placed side-by-side. These can hold numbers as large as 65,536. If you think about it this makes sense, since 256 (largest value for a byte-value) multiplied by 256 gives you 65,536. Though

bytes are usually considered to be the smallest units, they are not. No file can be smaller than 1 byte, but one byte is actually composed of 8 *bits*. Here is a description of how the number 341 is stored in memory. Note that this is larger than 255, so must be stored as a word.

0

1
2^15

2^14

2^13

2^12

2^11

2^10

2^9

2^8

2^7

2^6

2^5

2^4

2^3

2^2

2^1

2^0

Please note than "2^15" Indicates 2 to the power of 15. If one wants to change this base-2 number to base-10, one needs to find the "1's" in the upper row, and for each of these, add the number represented below it. 341 would be 2^8 + 2^6 + 2^4 + 2^2 + 2^0 = 341. Notice that there are 16 elements, or in other words, 16 bits. This means two bytes side-by-side as was described above. This was called a "word". We said that 65,536 was the largest number which could be represented by a word. Is this true? Assume a "1" for each of the cells above, and you should get 65,535. Why are we missing 1? Because "0" is a number, and thus, we can store 65,536 numbers, of which the largest is 65,535.

Data Type

Stored Bytes

Stored Bits

Smallest

Biggest
Byte

0

ASCII #0

ASCII #255
Word

0

65,535
Integer

-32767

32767
Long Integer

-2.15 Billion

2.15 Billion

Using the Program

Throughout this documentation there have been several explanations and examples, but none formalized. Here we will present to you one formal example of cheating. *Origin's Privateer* is quite a popular game now, and so we thought we'd show you how to "help" you along in the game. Load up a saved game and check to see how much money you have. Let us say you have $5200 credits *exactly*. Quit the game and load up Hacker. It is best to place hacker in your path statement so you can access it from anywhere, though this is not required. From the main menu, choose option 4, which is "long int". We know we can have a lot of money, so we assume the information is stored as a long integer (see table above). The program will then prompt you for the name of the file. Enter it. They it will ask you if you want to search the file. Say *yes* here. If you knew the location of the byte to modify, then you would say no here. The utility will then ask you what you want to search for. Enter $5200 here. The program will then as you if you want to replace any of the values you find. Enter yes if you want to actually *change* your money. It will then ask you what your desired value is. Enter 20,000. The program will then begin to scan the file. If any matches were found, the program will tell you where it found the occurrence. It will then ask you if you want to replace this value with the desired quantitiy of cash you entered previously. If yes, then the change will be made. Hit "y". You have now made some good money without much effort!

Generally, all information related to your character rather than to the map of the game is stored at the beginning of the file. If a second or third match is made for $5,200, it is probably fluke. It is generally best to accept *only* the first occurrence, and to ignore the following ones.

Contacting WadeSoft

WadeSoft is at the moment a small shareware-oriented company dedicated to releasing useful programs to the public domain. In other words, the little utilities we write to help us write our larger programs are released as public, whereas we ask for some money for the larger ones!

WadeSoft runs a public/private bulletin board called *The Spinning Disk* at the following number: 416-447-9233 (416-447-WADE). If private access is desired (access to files and the other telephone lines) then the nominal fee of $15.00 is asked. You do not need to register ($15.00) with the Spinning Disk to obtain *WadeSoft* technical help. To do so, please leave a comment to the System Operator detailing your problem(s). That is option "C" at the main menu.

Registration

Registration of *any* WadeSoft product is mearly $10.00 US or $12.50 Canadian. Registration simply means that you receive a diskette in the mail, along with nicely printed and formatted documentation. Please indicate your mailing address, telephone number and diskette type in the envelope. By sending $25.00 ($30.00 CND), you will receive a copy of ***ALL*** our programs, along with their respective documentations. In order to minimize our costs however, all programs will be contained on one or two diskettes, rather than having a each program contained

individually.

Mailing Address

Please mail cheques to the following address, and make cheques payable to "Gregory Wade".

Gregory Wade
26 Farrington Drive
North York, On, Canada
M2L 2B6