

Re: Binary to ascii encoder

Source: <http://coding.derkeiler.com/Archive/C/ CPP/comp.lang.c/2004-12/2364.html>

From: Charlie Gordon (news_at_chqrlie.org)

Date: 12/21/04

Date: Tue, 21 Dec 2004 13:01:51 +0100

"anirudhvr@sypmac.com" <anirudhvr@gmail.com> wrote in message
news:1103625820.183766.101690@f14g2000cwb.googlegroups.com...

> *Hi,*

>

> *I needed a basic binary to ascii encoder, so I wrote this piece of*

> *code:*

>

> */**

> *Encoding algo: suppose 11111010 is the byte to be encoded.*

> *It is first broken up into two 4-bit parts (1111 and 1010)*

> *This 4-bit quantity is then padded between a 01 and a 10, ie,*

> *01xxxx10 will be written out to the output file.*

> *Order: MSBs first, LSBs next*

> *Final output: 01111110, 01101010*

> **/*

This silly problem was dicussed not very long ago... Why encode this way when encoding in Hex is the obvious choice ? Looks like homework to me ;-) but since you are posting code, we will give you a helping hand.

As a general note, you should indent your code with spaces, not tabs, because these are interpreted wildly differently in different environments : such as here where they are ignored altogether.

> *#include <stdio.h>*

> *#include <stdlib.h>*

> *#include <string.h>*

>

> *int main(int argc, char *argv[])*

> *{*

> *FILE *in, *out;*

> *char *outfile;*

>

> *if(argc <= 1)*

> *return EXIT_FAILURE;*

You should issue a usage line here.

```
>  
> if( (in = fopen(argv[1], "r") ) == NULL )
```

Here is your bug : use "rb" to open the binary file in binary mode, not text mode.

Both modes are identical in unix (as they should be ;-)) but MS/Dos, MS/Windows and some other weird OS's need the 'b' to read raw binary from the file system instead of translating from their specific convention for text files such as \r\n line terminators.

```
> {  
> printf("File not found\n");
```

such error messages should be produced on stderr

```
> return EXIT_FAILURE;  
> }  
> outfile = (char*) malloc( sizeof(char) * ( strlen(argv[1]) + 5 ) );
```

the cast is unnecessary in C and sometimes error prone.
sizeof(char) is 1 by definition
5 is a magic constant here, strlen(".enc") + 1 would be more explicit.

```
> outfile = strcpy(outfile, argv[1]);  
> strcat(outfile, ".enc");  
>  
> if( (out = fopen(outfile, "w") ) == NULL )  
> {  
> printf("File could not be opened for writing\n");
```

stderr

```
> return EXIT_FAILURE;  
> }  
>  
> while(!feof(in))
```

feof() doesn't do what you want. Never use this function. Instead you can test for end of file by comparing the int result of getc(in) with EOF or testing fread() return value as <= 0 (0 for end of file, <0 for error)

```
> {  
> unsigned char inchar, outchar[2] = {0};  
> fread(&inchar, 1, 1, in);
```

Using fread is quite heavy to fetch just one byte from the stream, and you must test its return value : in case of a I/O error, feof() wont be set and an endless loop will ensue.

```
> outchar[0] = 66 | ( (inchar >> 4) << 2 );
```

comp.lang.c: Re: Binary to ascii encoder

The magical constant 66 should be in hex : 0x42 is more explicit about which bits will be set (it also has magical powers as every geek knows ;-)
You should also mask the bits you want to extract. While not strictly necessary for 8 bit bytes, it is more readable.

```
> outchar[1] = 66 | ( (unsigned char)(inchar << 4) >> 2 );
```

You should definitely mask the bits you want to extract. The cast to unsigned char does the only for 8 bit bytes and is quite obscure for newbie programmers.

```
> fwrite(outchar, 1, 2, out);
```

This is a very heavy way to write 2 chars to the output.
You might want to check its return value.

```
> }  
>  
> fclose(in);  
> fclose(out);
```

In case you run out of space on the output device (disquette ?) you should check fclose() return value as it may fail at flushing the data to the output media.

```
>  
> return EXIT_SUCCESS;  
> }
```

Here is a much simpler version of the loop:

```
int c;  
while ((c = getc(in)) != EOF) {  
    putc(0x42 | ((c & 0xF0) >> 2), out);  
    putc(0x42 | ((c & 0x0F) << 2), out);  
}
```

```
> This, coupled with a similar decoder, works in unix: I'm able to get  
> the initial binary back after encoding/decoding. However, on a windows  
> platform, the encoded file is much smaller than the binary used. I  
> tried with both djgpp and MSVC 6. Is there a non-standard feature that  
> I've used that might have caused this problem?
```

If it is much smaller, I suspect your C library treats ^Z in the input as an end of file marker in addition to converting \r\n sequences to \n. Use "rb" in fopen() to fix these problems.

glad to help

```
--  
Chqrli.e.
```