13 Code Obfuscation

"Wisdom is better than weapons of war."

The word *obfuscate* means "to confuse". Code Obfuscation refers to confusing others with your code. In other words, *Code Obfuscation* is the technical term for crypting your code and preventing others from reading the code (Just opposite to Readability). Code Obfuscation is very interesting to most of the C programmers. Every year we have **The International Obfuscated C Code Contest**. Throughout the world most of the C programmers participate in this contest. As far as I know no Indian has yet received this prize. So in this chapter let's see the most interesting Code Obfuscation.

13.1 Where to contest?

To contest in **The International Obfuscated C Code Contest**, visit their official website **www.ioccc.org**. There you can find the rules and important dates.

13.2 Guidelines

```
char a[ ] = "ABCD";              /* string representation */
char b[ ] = "\x41\x42\x43\x44";  /* hexadecimal representation */
char c[ ] = "\101\102\103\104";  /* octal representation */
char d[ ] = "A" "B" "C" "D";     /* using string properties */
char e[ ] = {'A', 'B', 'C', 'D', '\0'}; /* using char propery */
```

In C all the above strings a, b, c, d and e represent "ABCD". This is one of the simple tricks used in code obfuscation.

13.3 Real Code

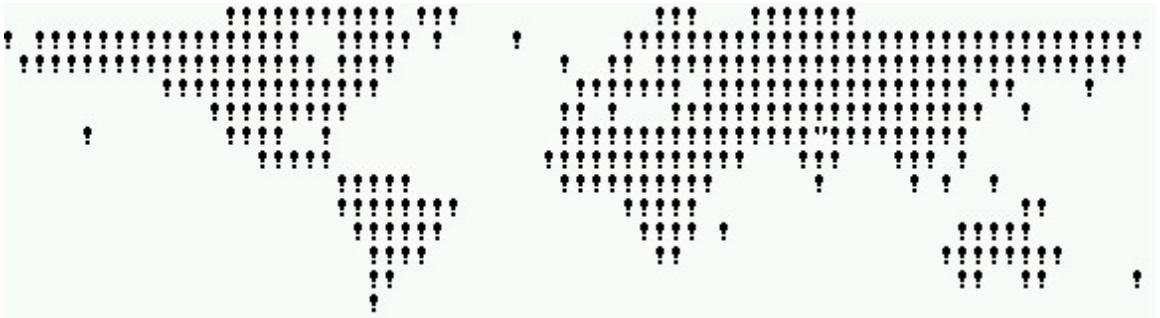
13.3.1 Wherami

The following program `Whereami.c` won "Best Small Program" prize in **The International Obfuscated C Code Contest** held in 1992. This program was by **Brian Westley (aka Merlyn LeRoy)**.

```
            main(l
         ,a,n,d)char**a;{
   for(d=atoi(a[1])/10*80-
  atoi(a[2])/5-596;n="@NKA\
CLCCGZAAQBEAADAFaISADJABBA^\
SNLGAQABDAXIMBAACTBATAHDBAN\
ZcEMMCCCCAAhEIJFAEAAABAfHJE\
TBdFLDAANEfDNBPHdBcBBBEA_AL\
 H E L L O,    W O R L D! "
    [l++-3];)for(;n-->64;)
       putchar(!d+++33^
            l&1);}
```

Any idea about the above code? It prints the world map! Quite amazing isn't it?

Output: World Map - New Delhi marked with " (obtained by executing `whereami 29 77`)



13.3.2 Note

Following is a part of note added by Westley.

Run the program as `whereami <lat> <long>`

Where `lat` and `long` correspond to your latitude and longitude.

To find the approximate place where this entry (**The International Obfuscated C Code Contest**) was judged, type:

`whereami 37 -122` (`-` means west of meridian)

Run the program with your latitude & longitude as integer arguments; it will produce a map made up of '!' with the given position marked with either a "" (if the position is over a '!') or a '#' (if the position is over a space). Southern latitudes and western longitudes are entered as negative numbers. For example, to find San Francisco, run with "whreami 38 -122". The resolution of the map is five degrees horizontally, ten degrees vertically. The map is a Mercator projection with equal spacing of the latitudes, so the areas near the poles are very distorted. Latitudes near the poles and Antarctica are not shown.

The program requires the ASCII character set, `putchar( )`, `atoi( )`, and a display that auto-wraps at 80 characters(!). If your display does not work this way, you will have to massage the output; for example, you can pipe it to a file and edit it with an editor, which will do autowrap for you.

If you run it with fewer than 2 arguments, it will likely give you an exception, as it will access arguments that don't exist and characters before a string constant.

Logic

The map is printed as one long string of ' ' and '!' characters, with the autowrap used to stack up slices of 80. The map data is a string; the first character is how many '!'s are printed ('A'=1, 'B'=2, etc), the second character is how many ' 's, the third is how many '!'s, etc. ASCII characters less than 'A' print no characters but still change the polarity, so any map of ' 's and '!'s is possible. This is done in the `putchar( )` argument as "33^l&1", where l is the character position+4; if l is odd, ' ' is printed, if l is even, '!' is printed.

The position of latitude & longitude is changed into a single character position within the one long string via the first expression "d = latitude/10*80 - longitude/5 - offset" The latitude is divided by ten because the vertical resolution is ten degrees, then multiplied by 80 because of the 80 character wrap (i.e. each ten degrees moves the position up or down one entire row). The longitude is divided by five and added, because five degrees of change moves the location one character. The signs are opposite because latitude is decreasing and longitude is increasing as you go from upper left to lower right. The offset is where the origin (latitude=0, longitude=0) is found.

The position counting down to zero changes the `putchar( )` from printing ('!' or ' ') to printing ('"' or '#').

The "H E L L O, W O R L D!" string inside the data string prints the line of blanks past Tierra del Fuego and the last blank line. It's just for show, really.

Since the resolution is coarse, a few costal cities are shown to be just off the map; this is an unavoidable artifact. The map is reasonably accurate. Here are some cities you might like to try:

City	Lattitude	Longitude
New York	41	-74
London	52	0
Moscow	56	38
New Delhi	29	77
Sydney	- 34	151
Los Angeles	34	-118
Paris	45	2
Beijing	40	116
Rio de Janeiro	-23	-43
Tokyo	36	140