Enpicting

The depicter program allows an infinite number of black and white images to be constructed from a string. The Depicter's method of encoding is such an efficient method of storing whole screens of data that it will turn up again in future programs. For the purpose of helping people contribute this program is presented.

The method of encoding used for the depicter program uses a knowledge of binary coupled with a table that is available on the depicting code listing. However, if you are encoding pictures using this method by hand it may be of benefit to have a more detailed table available. But why make the table by hand when you can have the computer do it:

EnpictHelp.C	You will need: a C/C++ complier .

```
#include <stdio.h>

main () {
  int c,d;

  for (c = 0; c < 15; c++) printf ("\n%d %d\t%c .x%d",c,c+33,(c+33),21-c);
  for (c = 0; c < 64; c++) {
    printf ("\n%d %d\t%c ",c+15,c+48,(c+48));
    for (d = 0; d < 6; d++) printf ("%c", (c & 1 << d) ? 'X' : '.');
  }
  for (c = 0; c < 15; c++) printf ("\n%d %d\t%c Xx%d",c+79,c+112,(c+112),c+7);
  getchar();
  return 0;
}
```

```
    XXXXXX
  XXX    XXX
 XX        XX
XX   X  X   XX
X    X  X    X
X    X  X    X
X  X      X  X
XX  XX  XX  XX
 XX  XXXX  XX
  XXX    XXX
    XXXXXX
```

This program, quite possibly the shortest one that will be featured in this space, builds for you a table explaining the values of each of the characters in the depicter's encoding scheme. Let's walk through one example using the happy little fellow here to the left

To start we need to know the width of the drawing. Now, it looks like it's 6 Xs across the top, but hold on a second, in the middle there it gets wider. And spaces count. Also, for the purpose of encoding the whole thing needs to be in a box, which means those 6 Xs on the top have 4 spaces to the left and 4 spaces to the right of them. So the width of our drawing is actually 14.

Once we have the width we may as well think of the whole thing as a single line. The program will handle line breaks for us once it knows the width. Consequently our smiley face turns into a line like this with periods replacing the spaces for clarity's sake:

```
....XXXXXX......XXX....XXX...XX........XX.XX...X..X...XXX....X..X....XX....X..X.
...XX..X......X..XXX..XX..XX..XX.XX..XXXX..XX...XXX....XXX......XXXXXX....
```

Run the program above and then start from the left. (You may need to use the scroll bar to go up to see the whole table. We have 4 spaces, followed by 6 Xs. The shortest run of spaces we could have is 7, which is the slash (/) character. However, we only have 4 spaces. So continuing down the list we see that the 'at' (@) character has 4 spaces, but it follows that up with one X and another space. What character would fit? Scroll down some more and we find the left single quotation mark (`), character number 63, number 96 on the ASCII table, is the best we're going to find. (Note, this is not the quotation mark found under the quote on most keyboards, but the one found under that little squiggly character called the tilde (~))It will draw the four spaces and two of the 6 Xs. So we make the left single quotation mark (`) our first character and we can cross off the first 6 spaces and Xs in our picture.

We still have 4 Xs followed by 6 spaces. Like before we search the table for the character that fits and find the question mark (?) will draw the 4 Xs and two of the spaces. Next we need 4 spaces and 3 Xs, but the left single quotation mark (`) is the best fit again. So that leaves us one X, 4 spaces, and three Xs. The letter 'Q' will let us cross off six of those. So far we have "`?`Q" as our string.

We left of with 2 Xs, 3 spaces, and 2 Xs to search for. I leave it to the reader to continue the process until it's completion and hope that you will not peek at the finished result as it's displayed below until you think you have a grasp of how the technique works. And remember, upper case or lower case does matter.

I'm serious. No peeking until you've tried it yourself. This is really not that difficult an example.

Because of the difficulties of strings in the C language there are certain characters you need to keep an eye out for. For instance, if you need to encode a run of 20 spaces, which we don't in this example, you will use the quotation mark (**"**), however since that would end the string you need to put a backslash (\) in front of it, so instead of just putting the quotation mark you put a backslash-quotation mark (**"**). This prevents C from thinking you wanted to end the string. However, what if you want to do a backslash, and you will in this example? Easy enough. Put a backslash in front of the backslash, or a double backslash.

I trust that now you've had a chance to prove your own abilities and being rather proud of yourself you should have a string of characters that you can go back to depicter and replace the lines at the beginning with your string and change the line width. You'll need to delete all the lines that define the picture and replace it with your code so you should have a few lines that look like this:

```
string code =
"`?`QS1HS47BH8Q9@>c\\iILh0o0";
int width = 14;
```

Recompile and run Depicter and you should see our smiley face back at you drawn with your choice of word. Me, I used the word 'MILES' because I wanted to see 'SMILES' across the top and bottom.

Now that the method has been explained to you and you have a tool to help you build pictures you can start making your own depicter programs. You may also be asking yourself, 'Isn't there an easier way? Couldn't a program be written that does it all for you?' All I have to say is, "Do you think I encoded those pictures by hand?" I leave it to the reader to write their own enpicting program, but I will provide a big hint and say to look at the depicter program and try to do what it does in reverse. And remember, SMILE!

```
   SMILES
  ILE    ESM
 SM        SM
LE   L  M   SM
I    I  S    S
M    M  E    E
S  L      S  L
ES  LE  IL  MI
 ES  LESM  ES
  ESM    MIL
   SMILES
```