Introduction to Software Systems (308-206B)
Assignment 2: atobm/bmtoa bitmap conversion

Winter Term 2001

Practical information

- Due date: Wednesday 21 March, before 23:59. Submissions after the due date will be discarded.
- Submission medium: VisualCM. Submit *three* files (or more): `Makefile`, `atobm.c`, and `bmtoa.c`, not your input/output. Beware: your submission is for 206B, *not* for 206A !
- *Individual* solutions and submissions. I will (pseudo-)randomly check whether you can explain your "own" assignment. Also, the final exam *may* contain some assignment-related questions.
- You may develop your code on any platform. Your code will however be evaluated on a SOCS UNIX machine. Make certain it runs correctly on a SOCS UNIX machine. In your Makefile, use

  ```
  CC=gcc
  CFLAGS=-g -Wall -ansi -pedantic
  ```

- Your code will be evaluated by comparing (using the UNIX diff) its output with correct output. Be certain to generate the same output (including blank lines) as the UNIX atobm and bmtao. For information on the use of the UNIX `atobm` and `bmtoa`, look at the `man` pages. When you run your own atobm and bmtoa, either have ./ in your path before anything else or run them explicitly referring to the current path (`./atobm` and `./bmtoa`). Forgetting this may mean executing the UNIX versions rather than your own.

Objectives

- Thorough understanding of C *bit operations* and basic use of *Makefile*s.
- An exercise in *design*. A good design will be elegant and include all special cases (*e.g.*, no input, too much input, ...).
- Thorough *testing* of code: inspect different input/output cases including pathological ones.
- An exercise in *coding style*. Make the code re-usable by including requirements and design information in the comments.

Assignment

Two programs need to be built, `atobm` and `bmtoa`.

atobm

`atobm` will ask the user for

1. A string `<name>` to be used as the first part
   - of a filename `<name>.xbm` which will contain the bitmap in XBM format, and
   - of identifiers `<name>_width`, `<name>_height`, and `<name>_bits[]`.

2. The character denoting a 1 bit, default '#'.
3. The character denoting a 0 bit, default '-'.

In the above, default means that the user is allowed to press return rather than enter a single character. In that case, the default character is used.

To be valid, `<name>` may only contain characters in the range 'a'-'z', 'A'-'Z', and '0'-'9'.

After the user has supplied the above required information, the program accepts a number of lines consisting of *only* the characters denoting a 0 bit and a 1 bit. The only other valid input is an empty line, denoting the end of the input. All entered lines must be of equal length.

Invalid input results in an appropriate error message (depending on the type of error) to the standard output (not to file) and program termination with an exit status 1.

The number of lines entered is used as the bitmap `height`. The number of characters entered per line is used as the bitmap `width`.

Example 1

If the `name` is "example1" and the characters denoting a 0 and a 1 bit are the defaults, the input

```
#--#-###---#--#-##
#--#-###---#--#-##
#--#-###---#--#-##
```

should produce the file example1.xbm containing

```
#define example1_width 18
#define example1_height 3

static unsigned char example1_bits[] = {
 0xe9, 0x48, 0x03, 0xe9, 0x48, 0x03, 0xe9, 0x48, 0x03 };
```

The same output would be produced (on stdout) when using the UNIX `atobm -name example1 -chars -#` with the above input.

Example 2

If in the above, the characters '0' and '1' were specified to denote 0 and 1 bits, the input

```
100101110001001011
100101110001001011
100101110001001011
```

should produce exactly the same output as in Example 1.

The same output would be produced (on stdout) when using the UNIX `atobm -name example1 -chars 01` with the above input.

Example 3

If the `name` is "example3" and the characters denoting a 0 and a 1 bit are the defaults, the input

```
#--#-###---#--#-##
#--#-#-#--#------#
```

should produce the file example3.xbm containing

```
#define example3_width 18
#define example3_height 2

static unsigned char example3_bits[] = {
 0xe9, 0x48, 0x03, 0xa9, 0x04, 0x02 };
```

Example 4

Let's take a closer look at how the output is generated. If the `name` is "example4" and the characters denoting a 0 and a 1 bit are the defaults, the input

```
#--##---#--##---#
#-------------###
```

should produce the file example4.xbm containing

```
#define example4_width 17
#define example4_height 2

static unsigned char example4_bits[] = {
 0x19, 0x19, 0x01, 0x01, 0xc0, 0x01 };
```

This is the example presented in class.

All (both) input lines have length 17, so `example4_width` needs to be `# defined` as 17. As there are two lines, `example4_height` needs to be `# defined` as 2.

The content of the character vector is obtained as follows (we are using the first input line as an example):

1. write the bit representation:
   `10011000100110001`
2. re-write this, starting from the right and pad with 0-bits to obtain a multiple of 8 bits (`example4_bits[]` is a character vector):
   `000000010001100100011001`
   The padding to obtain a multiple of 8 bits is done per line.
3. writing each byte in hexadecimal notation gives:
   `0x01`, `0x19`, `0x19`
4. Writing this from right to left:
   `0x19`, `0x19`, `0x01`

Similarly, the second line will lead to `0x01`, `0xc0`, `0x01`.

In the program, processing can be done byte by byte ! Also, the reversing need not be done explicitly, but can be done "on the fly".

bmtoa

`bmtoa` will be constructed by compiling and linking `bmtoa.c` as well as `bitmap.c`. `bitmap.c` is produced by using "bitmap" as the name input in `atobm`, renaming or copying (manually) the file `bitmap.xbm` created by `atobm` to `bitmap.c` and removing the `static` in that file. The program in `bmtoa.c` must use the definitions in `bitmap.c` (via an `external` reference) to print the bitmap to standard output. To access bitmap height and width information, a file `bitmap.h` must be included which is constructed from the first two lines of `bitmap.xbm`. The program `bmtoa` must ask the user for

1. The character denoting a 1 bit, default '#'.

2. The character denoting a 0 bit, default '-'.

As before, default means that the user is allowed to press return rather than enter a single character. In that case, the default character is used.

Invalid input results in an appropriate error message (depending on the type of error) to the standard output (not to file) and program termination with an exit status 1.

Note

You may assume that the total number of bits (rows x columns) input will never be more than 10000.

Resources

- The template to use for your C code: template.c
- The template/example to use for your Makefile: Makefile
- Be consistent in your indentation. Using the GNU tool "indent" can help. Below are some reasonable command-line options (which you may wish to put in your .cshrc file if you use csh).

  ```
  #         indent (GNU C beautifier)
  #
  #          -ts1: tab stop width = 1
  #          -bli0: block indent ({}) 0
  #          -c28: try to start code-line comments in column 28
  #          -cd28: try to start declaration-line comments in column 28
  #          -npcs: no space after function call names
  #          -l72: maximum non-comment-line length
  #          -lc72: maximum comment-line length
  #
  alias indent  'indent -ts1 -bli0 -c28 -cd28 -npcs -l72 -lc72'
  ```

  Beware of TABs. TAB expansion (as a number of blank spaces) on one machine may be different from TAB expansion on another. This may result in code which looks properly indented on one platform and chaotic on another. To avoid this:

  - do never insert TABs willingly for indentation purposes,
  - make sure your editor does not substitute a number (8, for example) of consecutive blanks by a TAB. In vi for example, you may wish to `set tabstop=100` to avoid substitution.

- For a graphical look at bitmaps (not required), you can use the UNIX/X Windows utility `bitmap`.