An address often has the same number of bits as an integer (or long integer). Creating a pointer that represents a specific address is easy: we just cast an integer into a pointer. For example, here's how we might store the address 1000 (hex) in a pointer variable:

```
BYTE *p;

p = (BYTE *) 0x1000;    /* p contains address 0x1000 */
```

PROGRAM

Viewing Memory Locations

Our next program allows the user to view segments of computer memory; it relies on C's willingness to allow an integer to be used as a pointer. Most CPUs execute programs in "protected mode," however, which means that a program can access only those portions of memory that belong to the program. This prevents a program from accessing (or changing) memory that belongs to another application or to the operating system itself. As a result, we'll only be able to use our program to view areas of memory that have been allocated for use by the program itself. Going outside these regions will cause the program to crash.

The `viewmemory.c` program begins by displaying the address of its own `main` function as well as the address of one of its variables. This will give the user a clue as to which areas of memory can be probed. The program next prompts the user to enter an address (in the form of a hexadecimal integer) plus the number of bytes to view. The program then displays a block of bytes of the chosen length, starting at the specified address.

Bytes are displayed in groups of 10 (except for the last group, which may have fewer than 10 bytes). The address of a group of bytes is displayed at the beginning of a line, followed by the bytes in the group (displayed as hexadecimal numbers); followed by the same bytes displayed as characters (just in case the bytes happen to represent characters, as some of them may). Only printing characters (as determined by the `isprint` function) will be displayed; other characters will be shown as periods.

We'll assume that `int` values are stored using 32 bits and that addresses are also 32 bits long. Addresses are displayed in hexadecimal, as is customary.

viewmemory.c

```
/* Allows the user to view regions of computer memory */

#include <ctype.h>
#include <stdio.h>

typedef unsigned char BYTE;

int main(void)
{
  unsigned int addr;
  int i, n;
  BYTE *ptr;

  printf("Address of main function: %x\n", (unsigned int) main);
  printf("Address of addr variable: %x\n", (unsigned int) &addr);
```