CZ1106 Problem Solving and Computation II

Programming Lab 4

Compression for Numerical Data

15 March 2007

Part I

The program ac3-3.c can be downloaded from web. Execute the following steps and answer the questions.

- Compile and run the program. Enter `1 2 3 x` as input.
- Observe the screen output.
- Add watch for num[0], num[1], num[2], &num[0], &num[1], &num[2], j.
- Set a break point at `j++;`
- Compile and then execute the program 3 times.

How many bytes are used by an int in Turbo C++ ? Where and how do you derive the answer ?

- Use <ctl QA>, Origin=Entire scope, Change All, to replace all occurrences of int by long, and change the input format to %ld%*c. (Be careful, ***append a space*** before and after int.)
- Compile and then execute the program 3 times.

How many bytes are used by a long in Turbo C++ ? Where and how do you derive the answer ?

- Use <ctl QA>, Origin=Entire scope, Change All, to replace all occurrences of long by float, and change the input format to %f%*c.
- Compile and then execute the program 3 times.

How many bytes are used by a float in Turbo C++ ? Where and how do you derive the answer ?

- Use <ctl QA>, Origin=Entire scope, Change All, to replace all occurrences of float by double, and change the input format to %lf%*c.
- Compile and then execute the program 3 times.

How many bytes are used by a double in Turbo C++ ? Where and how do you derive the answer ?

Part II

The use of binary files allows for data compression across byte boundaries. This is useful in saving space, and is even more efficient in saving time. On a machine with 32-bit words, each instruction cycle processes 32 bits in parallel. If more data are packed in a word, the speed of data processing will be increased proportionately. In this practical we are going to write a program to compress a text file that contains only digits and reduce its storage requirement by half.

We first analyze the approach used in this exercise. Each digit in the text file is stored as a character and therefore it requires one byte (1 byte = 8 bits). Actually, half a byte (4 bits) is sufficient to keep the contents of the digit as you can observe from the following table:

Digit	Binary Representation
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

To compress the text file we shall proceed as follows. In every iteration we read eight digits from the file. To make life easier we assume that the number of digits in the text file is a multiple of 8. For each digit (character) we only keep the 4 least significant bits and pack them in a 32-bit unsigned long int. In this way the 32-bit word can keep the contents of up to eight digits as shown in the following example.

Suppose the contents of eight digits are `42315678`. The binary contents of the compressed data will be `0100 0010 0011 0001 0101 0110 0111 1000`.

8 digits

32 bits

The text file you are going to compress is named as **digit.dat**. You can unzip it from our course website.

Contents of digit.dat

```
77266667726666123456789012345678907726666772666 6
```

The compressed data will be stored in a binary file named as **compress.dat**. After you have run your program, the following file sizes should be displayed if you type the **DIR** command at the DOS prompt. You discover that the file size of **digit.dat** can be reduced from 48 bytes to 24 bytes.

The contents of **compress.dat** should be as follows if your program is correct:

```
gf&w↕ffr▯xV4xV4↕f&w▯ffrg
```

To show that our data compression is done correctly, we will read the binary file created, decompress the contents, and display the restored digits on the screen. All these compress and decompress tasks will be coded in the same program named as **compress.c**.

The following steps may be used in your program. Again, if you have better methods, please go ahead to code them your own way.

```
open digit.dat as indata, a text file with read mode;
open compress.dat as outdata,  a binary file with write mode;

initialize count to 0, long int packed to 0;
loop while indata is not empty and read a character this1 from indata
{
  convert the ASCII code of this1 to a numerical value;
  store the numerical value to temp;
  push the bit contents of temp to the left by a correct number of places if necessary;
  move only the relevant contents of temp to packed;
  increment count by 1;

  if count is equal to 8
  {
     write the contents of packed to outdata;
     reset count to 0, packed to 0;
  }
}

close the input and output files;

open compress.dat as indata,  a binary file with read mode;

print on the screen "Decompressed text contents:";

loop while indata is not empty and read a long int packed from indata
{
  set a correct mask;

  loop 8 times
  {
    use mask to extract 4 relevant bits from packed, store the results in temp;
    push the bit contents of temp to the right  by a correct number of places if necessary;
    restore the digit and print it on the screen;
    adjust the contents of mask;
  }

  close input file;
}
```

Use debugger !!