CS 4780 Summer 2011

Project 2

For the second project, I want you to implement the RIPEMD-160 hash function. This takes an arbitrary length string [I will not use more that 120 characters] and produces a 160-bit [20 byte] value. I want the program to be called `ripemd` (or `ripemd.class`); it takes no command line arguments, read the string from standard input, computes the hash, and prints the result to standard output.

RIPEMD first extends the string by adding enough padding to get to 8 bytes less than an exact multiple of 64 bytes. The padding consists of a byte of `0x80` followed by as many bytes of `0x00` as are needed. It then appends the 64-bit length of the string in BITS (bits = 8 times number of bytes) in little-endian form. It initialized the array $CV[0]$ through $CV[4]$ to initial values (and I have put these into a file `ripemd-consts.c` for your use).

Then it processes a block of 64 bytes [16 32-bit words] at a time. It calls these 16 words X_0 through X_15. It copies $CV[0]$ into Al and Ar, $CV[1]$ into Bl and Br, $CV[2]$ into Cl and Cr, $CV[3]$ into Dl and Dr, and $CV[4]$ into El and Er. It does 2 independent 5-round transformations on the X's: a "left" set, which works with Al through El, and a "right" set, which works with Ar through Er.

Each round consists of 16 steps. The steps look very similar. Each step consits of the (simutaneous) actions $A \leftarrow E$, $B \leftarrow \mathrm{rol}_{\mathrm{lsh\_amt(ind,\ j)}}(A + f_j(B,C,D) + X_{ind} + K_j) + E$, $C \leftarrow B$, $D \leftarrow \mathrm{rol}_{10}(C)$, and $E \leftarrow D$. In this $F_j(B,C,D)$ is a function that depends on the round number; on the left side, j is the round number; on the right side, j is 4-the round number. [I.e., in all the steps in round 0 on the left side, use F_0; in all the steps in round 0 on the right side, use f_4; for round 1, the left side uses f_1 and the right side uses f_3; etc.] K_j is a constant that depends on the round number and which side (and not the step number). The quantity ind is a permutation of the step number; on the left side, in round 0 it is i, in round 1 it is $\rho(i)$, in round 2 it is $\rho(\rho(i))$, in round 3 it is $\rho(\rho(\rho(i)))$, and in round 4 it is $\rho(\rho(\rho(\rho(i))))$; on the right side, in round 0 it is $\pi(i)$, in round 1 it is $\rho(\pi(i))$, in round 2 it is $\rho(\rho(\pi(i)))$, in round 3 it is $\rho(\rho(\rho(\pi(i))))$, and in round 4 it is $\rho(\rho(\rho(\rho(\pi(i)))))$. (In the file `ripemd-consts.c` I have put arrays `K_l` and `K_r` for the constants K, arrays `rho` and `pi` for the permutations ρ and π, an array `lsh_amt` indexed by the round number and the index of X, an array `CV` with their initial value, and a (real) function `f (round_number, B, C, D)`.)

At the end of each block [2 sides of 5 rounds of 16 steps] update the `CV` array by (simultaneously) doing `CV[0] = CV[1] + Cl + Dr`, `CV[1] = CV[2] + Dl + Er`, `CV[2] = CV[3] + El + Ar`, `CV[3] = CV[4] + Al + Br`, and `CV[4] = CV[0] + Bl + Cr`.

After you have done this for all the block in the message extended with padding and length, print the value of the `CV` array in little-endian form.

I have created file `ripemd.h` and `ripemd-consts.c` in the dirctory `~schultemw/pub/cs4780/`

on admiral. Note that all conversion from string to integer and back to final output is done little-endian; last project was big-endian. At several points above I used the word "simultaneous"; you will need a temporary variable to avoid overwriting a value you need later. The only place you need to worry about bytes is the initial string handling and the output; everything else is done on 32-bit unsigned ints.

Pseudocode:

```
read line and add padding and length
for each block do
    create Al-El and Ar-Er
    for j = 0 to 4 each round
        for i = 0 to 15 each step
            Tl = expression for new Bl; Tr = same expr for right
            update Al-El and Ar-Er
        end for i
    end for j
    update CV's
end for each block
print answer
```