ECE 543 Project 2a: Implement the Message Digest Algorithm MD5

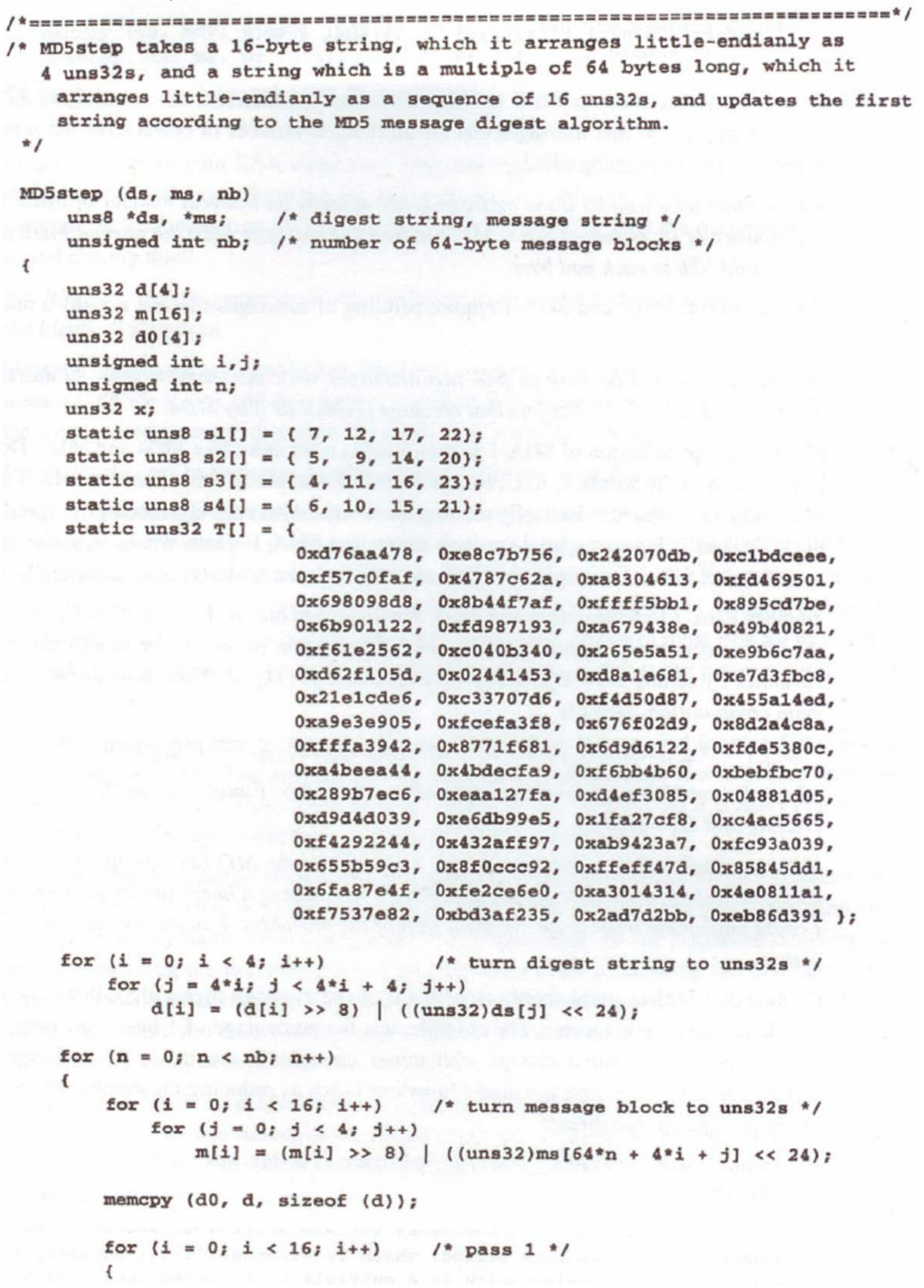
Implement the message digest algorithm MD5 and test how “random” the output appears. For example, test the percentage of 1 bits in the output, or test how many bits of output change with minor changes in the input. Also, design various simplifications of the message digest functions (such as reducing the number of rounds) and see how these change things.

For your convenience, some codes are provided (on pages 2-4) for you to do the project. Of course, you can develop your own codes.

You need to write a technical report for this project. The report should describe your observations of the randomness in the output. You should have executable codes to support your observations. TA will arrange a timeslot to check your codes. Using your codes, we should be able to generate the results you present in your report.

- Your report is **due at 23:59 pm, April 30, 2017, Chicago time.** Submit your technical report through the blackboard system with document named as **“ID_Lastname_prjt2a_report.pdf”.** Please attach your codes to your report.
- Late submissions will not be graded and lead to a “0” grade on this project.
- TA will assign a timeslot for every one in the **week April 24 – April 28 to check your codes.**
- You need to independently work on the project. **Projects identified with plagiarism problem will get a “0” grade.**

MD5 codes for your reference:



```
/*=====================================================================*/
/* MD5step takes a 16-byte string, which it arranges little-endianly as
   4 uns32s, and a string which is a multiple of 64 bytes long, which it
   arranges little-endianly as a sequence of 16 uns32s, and updates the first
   string according to the MD5 message digest algorithm.
 */

MD5step (ds, ms, nb)
     uns8 *ds, *ms;      /* digest string, message string */
     unsigned int nb;    /* number of 64-byte message blocks */
{
     uns32 d[4];
     uns32 m[16];
     uns32 d0[4];
     unsigned int i,j;
     unsigned int n;
     uns32 x;
     static uns8 s1[] = { 7, 12, 17, 22};
     static uns8 s2[] = { 5,  9, 14, 20};
     static uns8 s3[] = { 4, 11, 16, 23};
     static uns8 s4[] = { 6, 10, 15, 21};
     static uns32 T[] = {
                    0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
                    0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
                    0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
                    0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
                    0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
                    0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
                    0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
                    0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
                    0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
                    0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
                    0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
                    0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
                    0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
                    0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
                    0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
                    0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 };

    for (i = 0; i < 4; i++)          /* turn digest string to uns32s */
        for (j = 4*i; j < 4*i + 4; j++)
            d[i] = (d[i] >> 8) | ((uns32)ds[j] << 24);

    for (n = 0; n < nb; n++)
    {
        for (i = 0; i < 16; i++)     /* turn message block to uns32s */
            for (j = 0; j < 4; j++)
                m[i] = (m[i] >> 8) | ((uns32)ms[64*n + 4*i + j] << 24);

        memcpy (d0, d, sizeof (d));

        for (i = 0; i < 16; i++)     /* pass 1 */
        {
          x = m[i] + T[i] + d[(-i)&3] +
               ((d[(1-i)&3] & d[(2-i)&3]) | (~d[(1-i)&3] & d[(3-i)&3]));
          d[(-i)&3] = d[(1-i)&3] + (x << s1[i&3]) + (x >> (32 - s1[i&3]));
      }
      for (i = 0; i < 16; i++)      /* pass 2 */
      {
          x = m[(5*i+1)&15] + T[i+16] + d[(-i)&3] +
               ((d[(1-i)&3] & d[(3-i)&3]) | (d[(2-i)&3] & ~d[(3-i)&3]));
          d[(-i)&3] = d[(1-i)&3] + (x << s2[i&3]) + (x >> (32 - s2[i&3]));
      }
      for (i = 0; i < 16; i++)      /* pass 3 */
      {
          x = m[(3*i+5)&15] + T[i+32] + d[(-i)&3] +
               (d[(1-i)&3] ^ d[(2-i)&3] ^ d[(3-i)&3]);
          d[(-i)&3] = d[(1-i)&3] + (x << s3[i&3]) + (x >> (32 - s3[i&3]));
      }
      for (i = 0; i < 16; i++)      /* pass 4 */
      {
          x = m[(7*i)&15] + T[i+48] + d[(-i)&3] +
               (d[(2-i)&3] ^ (d[(1-i)&3] | ~d[(3-i)&3]));
          d[(-i)&3] = d[(1-i)&3] + (x << s4[i&3]) + (x >> (32 - s4[i&3]));
      }
      for (i = 4; i--; )                /* end of stage */
          d[i] += d0[i];
   }
   for (i = 0; i < 4; i++)              /* turn uns32s back to digest string */
       for (j = 4*i; j < 4*i + 4; j++)
           ds[j] = d[i], d[i] >>= 8;
}

/*===========================================================================*/
/* MD5 takes a 16-byte string where it will place the MD5 message digest,
   an arbitrary length message string (of bytes),
   and the byte count of the message string
*/

MD5 (ds, ms, bc)
    uns8 *ds;                  /* digest string */
    uns8 *ms;                  /* message string */
    unsigned int bc;           /* message byte count */
{
    static uns8 md0[16] = {0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
                           0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10};
    uns8 ms0[128];             /* for padding */

    unsigned int b;

    memcpy (ds, md0, sizeof (md0)); /* initialize the message digest */
    MD5step (ds, ms, bc / 64);      /* process full blocks */
    memcpy (ms0, ms + bc / 64 * 64, b = bc % 64);

    ms0[b++] = 0x80;                    /* pad the message */
    while ( (b + 8) % 64 )
        ms0[b++] = 0;

    ms0[b++] = bc << 3;
    ms0[b++] = bc >> 5;
    ms0[b++] = bc >> 13;
    ms0[b++] = bc >> 21;
    ms0[b++] = bc >> 29;
    ms0[b++] = 0;
    ms0[b++] = 0;
    ms0[b++] = 0;

    MD5step (ds, ms0, b / 64);          /* process the padded block(s) */
}
```