decimal.cpp

Functions for conversion of integer vectors to and from comma-separated lists of decimal ASCII numbers

Extension to C++ vector class library

Agner Fog

Chapter 1

Introduction

The decimal ASCII extension to the Vector Class Library contains functions for conversion of integer vectors to and from comma-separated lists of numbers as human-readable decimal ASCII strings. This is useful for efficient reading and writing of comma-separated files.

These functions cannot read or write floating point numbers. If you have fractional numbers, then you may consider if the numbers can be converted to integers by appropriate scaling. For example, if your have dollars with two decimals, then you can multiply the numbers by 100 to get cents as integer numbers. This will make data handling faster.

These functions are highly efficient. Whether this efficiency actually shows in the overall program performance depends on whether string processing is a bottleneck. In many applications, the transfer of data files is the limiting bottleneck, rather than string processing.

Anyway, the code presented here can serve as an interesting show case. The code illustrates how strings can be processed or parsed in parallel using vector instructions in a highly efficient way. The ascii2bin function shows that it is possible to parse a string with variable-length fields without looping through the characters of the string.

1.1 Compiling

The decimal extension to the Vector Class Library consists of the files decimal.cpp and decimal.h. The decimal.cpp file is added to the project that needs it, and the decimal.h file is #included in C++ files that call these functions.

The decimal extension to the Vector Class Library is compiled in the same way as the Vector Class Library itself. All x86 and x86-64 platforms are supported, including Windows, Linux, and Mac OS. The following C++ compilers can be used: Gnu, Clang, Microsoft, and Intel. See the manual for the vector class library for further details.

Chapter 2

Binary to decimal ASCII conversion

The bin2ascii function has the following parameters:

Table 2.1: bin2ascii function

Parameter	Description
Vec4i a Vec8i a	A vector of four or eight signed or unsigned integers.
char * string	This char array will receive the ASCII string of decimal numbers. It is the responsibility of the programmer that the array is big enough to contain the resulting string, even in case of overflow.
int fieldlen	Desired length of each field in the output list.
int numdat	Number of data fields to write. The maximum value is 4 or 8 for Vec4i and Vec8i, respectively.
char ovfl	This ASCII character will indicate overflow if a number is too big to fit into a field of size `fieldlen`. The default value is '*'. The field will be extended to fit the number if `ovfl` is set to 0 (without quotes).
char separator	This ASCII character will be used as separater between the number fields. The default value is ','. No separator will be used if `separator` is set to 0 (without quotes).
bool signd	Set this to `true` (default) to write signed numbers. Set to `false` if the input vector should be interpreted as unsigned numbers.
bool term	Indicates whether the written ASCII string string should be zero-terminated. The default is true. A terminating zero will not be written if `term` is false.
Return value	The bin2ascii function returns the length of the written string.

This example shows how to use the bin2ascii function:

Example 2.1.

```
// Example for binary to decimal ASCII conversion
#include <stdio.h>
#include "vectorclass.h"
#include "decimal.h"
#include "decimal.cpp"

int main() {
    // make a vector of eight integers
    Vec8i a(1, 20, 300, 4000, -12345, 67890, 1234567890, 8);
    // make a char array big enough to hold the string
    char text[100];
```

```
    // convert to human-readable decimal ASCII numbers
    bin2ascii(a, text, 6, 8, '*', ',', true, true);
    // print text
    printf("List of numbers:\n%s\n", text);
    return 0;
}
/* The output will be:
List of numbers:
     1,    20,   300,  4000,-12345, 67890,******,     8
*/
```

Chapter 3

Decimal ASCII to binary conversion

The ascii2bin function has the following parameters:

Table 3.1: ascii2bin function

Parameter	**Description**
const char * string	An ASCII string containing integer numbers separated by comma or by some other separator character.
int * chars_read	This parameter will receive the number of characters that the function has read. In other words, the part of the string that has been used by the function.
int * error	This parameter will receive an indication of any errors. The error codes are listed below.
int max_stringlen	The maximum length of the string that the function is allowed to read. Any contents after max_stringlen will be ignored. The string may be shorter than max_stringlen if terminated by a zero or newline.
int numdat	The number of data fields to read. The maximum value is 8.
char separator	The character used as separator between numbers. The default is ','
Return value	The function returns a vector of type Vec8i, containing up to eight signed integers.

The input string must be an ASCII string using the following syntax. The string can contain up to eight fields, each containing a signed or unsigned integer. The fields are separated by the character indicated as `separator`. The default separator is a comma. Each field can contain an optional sign (+ or −) followed by any number of digits 0 - 9. Spaces are allowed before and after the number, and between the sign and the number. No other characters are allowed. Nothing is allowed between the digits.

A separator (comma) after the last field is not required, but it can be useful to prevent the function from reading any irrelevant text that comes after the relevant fields, which would cause syntax errors. A terminating separator will be included in `chars_read`.

The following error codes are returned in `*error` in case of syntax errors in the string:

Table 3.2: ascii2bin error codes

Error code	Description
1	Parameters out of range. This happens if numdat > 8 or max_stringlen > 10000.
2	Illegal character. This happens if a numeric field contains any other characters than +, -, 0-9, or space. The value will be interpreted as if the illegal character was a space, if possible.
4	Misplaced character. This happens if a + or - sign is placed after the number rather than before the number, or if there is anything between the digits. The resulting value will be zero.
8	Too few separators. The string has less than `numdat-1` separators. The remaining values will be zero.
16	Overflow. A value is too big to fit into a 32-bit signed integer. The resulting value will be INT_MAX or INT_MIN.
0	Missing value. An empty field will be interpreted as a zero. This is not indicated as an error.

The error codes can be combined if the string has multiple syntax errors.

Any control characters in the string, such as newline or tab, will be interpreted as end of string, unless the same character is indicated as separator. A Windows newline cannot be used as separator because it consists of two control characters (\r\n).

This example shows how to use the ascii2bin function:

Example 3.1.

```
// Example for conversion from comma-separated decimal ASCII
// string to binary vector
#include <stdio.h>
#include "vectorclass.h"
#include "decimal.h"
#include "decimal.cpp"

int main() {
    // text string to interpret
    char text[] = " 1, -20, 30, , 555, -6, 7000, 88888  ";

    // length and error will be returned in these variables
    int length, error;

    // interpret the comma-separated string
    Vec8i dat = ascii2bin(text, &length, &error, 64, 8, ',');

    // check if syntax error
    if (error) {
        printf ("\nerror = 0x%X",error);
    }
    else {
        // write results
        for (int i = 0; i < 8; i++) {
            printf("%i ", dat[i]);
        }
    }
```

```
    return 0;
}
// Program output:
// 1 -20 30 0 555 -6 7000 88888
```

The chars_read variable tells where to begin the next read if the string or line contains more than eight numbers. This is illustrated in the next example:

Example 3.2.

```
// Example for converting a comma-separated decimal ASCII
// string containing more than eight numbers
#include <stdio.h>
#include "vectorclass.h"
#include "decimal.h"
#include "decimal.cpp"

int main() {
    // text string containing twelve numbers
    char text[] = " 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37";

    // length and error will be returned in these variables
    int length1, length2, error;

    // data vectors of eight integers each
    Vec8i dat1, dat2;

    // read first eight numbers
    dat1 = ascii2bin(text, &length1, &error, 64, 8, ',');

    // check if syntax error
    if (!error) {
        // read the next four numbers
        dat2 =
        ascii2bin(text + length1, &length2, &error, 64, 4, ',');
    }

    // check if syntax error
    if (error) {
        printf ("\nerror 0x%X",error);
    }
    else {
        // join the two data vectors
        Vec16i dat12(dat1, dat2);

        // write results
        for (int i = 0; i < 12; i++) {
            printf("%i ", dat12[i]);
        }
    }
    return 0;
}
```

3.1 Efficiency

The ascii2bin function can be highly efficient. The performance is highest if the following conditions are satisfied:

- The input string is no more than 64 characters long
- No number is more than 8 characters long, including sign
- The code is compiled for the highest instruction set supported by the CPU it is running on. The following instruction set extensions give particular advantage: AVX2, AVX512BW, and the future AVX512VBMI2