developer.com
codeguru

CyberAtlas
And point your business in the right direction!

Search

CodeGuru Sites

Visual C++/MFC
C# and.NET
Visual Basic

Discussion Boards
Books on.NET
Book Reviews
Newsletters (subscribe)
Newsletters (archived)

Article Sections

C++
algorithms & formulas
c++ & mfc
date & time
string

COM-based Technologies
atl & wtl
com & activex
com+
shell programming

Algorithms & Formulas

Forums | Latest updates | Recommend it

Section Manager : CodeGuru

Checksum Algorithms

- RSA MD5 Message Digest - Nick Stone *(2000/10/03)*
  C++ implementation of the RSA MD5 message digest algorithm
- NEW CRC32: Generating a checksum for a file - Brian Friesen *(2001/08/30)*
  This article describes what a CRC is, how to generate them, what they can be used for, and lastly source code showing how it's done.

Compression/Decompression

- Optimizing Tip on Adaptive Arithmetic Coding - Alexey V. Shanin *(2001/01/23)*
  Describes an optimization point found in the most commonly used adaptive Witten-Neal-Cleary implementation

Developer's Guide to the Euro

- Duncan Jones - Shows everything a developer needs to know regarding the Euro, including conversion rules, post-conversion rounding rules and other development issues. *(2001/03/06)*

Factorials

- Method of Handling Factorials of Any Size - Hai Yi *(2000/09/24)*
  Method uses linked list to capture each digit of factorial result thereby not limiting you to C/C++ types

String Algorithms

- Algorithm for Detecting Duplicate/Previously Encountered Strings - Jeremy Cattone *(2001/06/11)*
  Enables you to quickly check for duplicates when traversing large amounts of text

Controls
button control
combobox
edit control
imagelist control
listbox control
listview control
menu
other controls
property sheet
rich edit control
static control
status bar
toolbar
treeview control

Data
database
miscellaneous

Frameworks
ui & printing frameworks

Graphics & Multimedia
bitmaps & palettes
directx
gdi
multimedia
opengl

Internet & Networking
ie programming
internet protocols
isapi
network protocols

Miscellaneous
miscellaneous

Visual Studio
debugging
add-ins & macros
editor tips

Windows Programming
ce
clipboard
dll
file & folder
help systems
printing
win32
system

Windows & Dialogs
console
dialog
docking window
doc/view
splitter

Interact

Newsletters
Guestbook
Recommend Us!
About Us

Nanotech™ PLANET Fall 2001
Conference & Expo
The Center of Nanotechnology Business
Hosted by: Nanotech PLANET.COM™
Click Here Now!

developer.com

codeguru

Dec. 6-7, 2001 ORLANDO, FL Premier Co-Sponsor: Microsoft Services for the Enterprise

Search

CodeGuru Sites

Visual C++/MFC
C# and.NET
Visual Basic

Discussion Boards
Books on.NET
Book Reviews
Newsletters (subscribe)
Newsletters (archived)

Article Sections

C++
algorithms & formulas
c++ & mfc
date & time
string

COM-based Technologies
atl & wtl
com & activex
com+
shell programming

Algorithms & Formulas

Forums | Latest updates | Recommend it

Section Manager : CodeGuru

Checksum Algorithms

- RSA MD5 Message Digest - Nick Stone *(2000/10/03)*
  C++ implementation of the RSA MD5 message digest algorithm
- NEW CRC32: Generating a checksum for a file - Brian Friesen *(2001/08/30)*
  This article describes what a CRC is, how to generate them, what they can be used for, and lastly source code showing how it's done.

Compression/Decompression

- Optimizing Tip on Adaptive Arithmetic Coding - Alexey V. Shanin *(2001/01/23)*
  Describes an optimization point found in the most commonly used adaptive Witten-Neal-Cleary implementation

Developer's Guide to the Euro

- Duncan Jones - Shows everything a developer needs to know regarding the Euro, including conversion rules, post-conversion rounding rules and other development issues. *(2001/03/06)*

Factorials

- Method of Handling Factorials of Any Size - Hai Yi *(2000/09/24)*
  Method uses linked list to capture each digit of factorial result thereby not limiting you to C/C++ types

Controls
button control
combobox
edit control
imagelist control
listbox control
listview control
menu
other controls
property sheet
rich edit control
static control
status bar
toolbar
treeview control

Data
database
miscellaneous

Frameworks
ui & printing frameworks

Graphics & Multimedia
bitmaps & palettes
directx
gdi
multimedia
opengl

Internet & Networking
ie programming
internet protocols
isapi
network protocols

Miscellaneous
miscellaneous

Visual Studio
debugging
add-ins & macros
editor tips

Windows Programming
ce
clipboard
dll
file & folder
help systems
printing
win32
system

Windows & Dialogs
console
dialog
docking window
doc/view
splitter

String Algorithms

- Algorithm for Detecting Duplicate/Previously Encountered Strings - Jeremy Cattone *(2001/06/11)*
  Enables you to quickly check for duplicates when traversing large amounts of text

DECEMBER 6-7, 2001
ORLANDO, FL
PREMIER CO-SPONSOR:
Microsoft Services
for the Enterprise

developer.com
codeguru

Dec. 6-7, 2001 Orlando, FL Premier Co-Sponsor: Microsoft Services for the Enterprise

Search

CodeGuru Sites

Visual C++/MFC
C# and.NET
Visual Basic

Discussion Boards
Books on.NET
Book Reviews
Newsletters (subscribe)
Newsletters (archived)

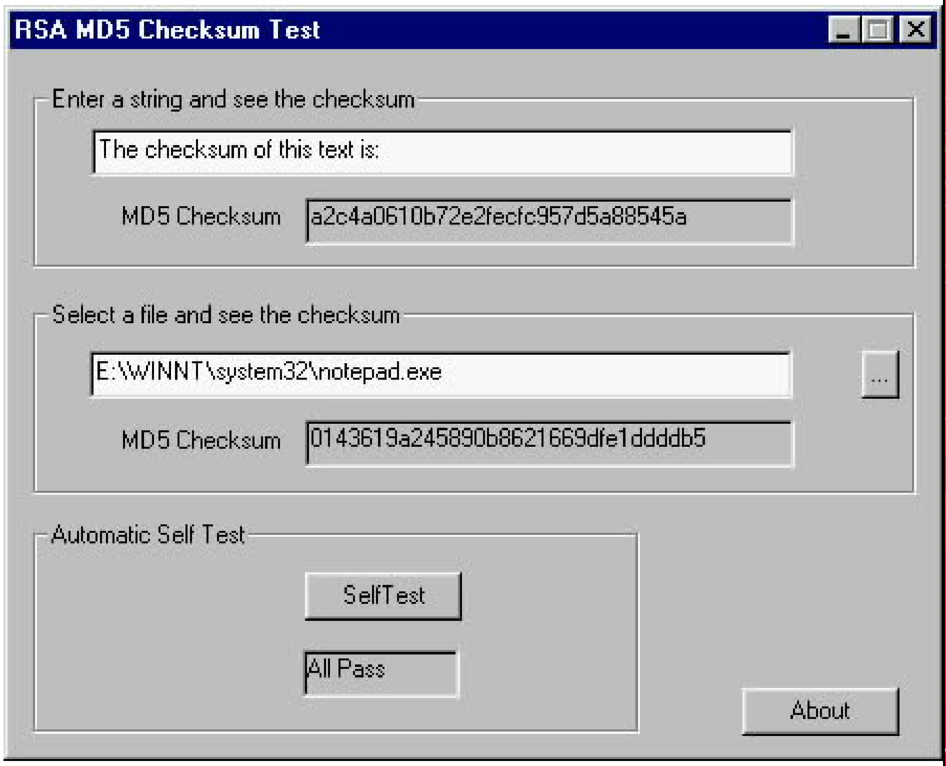
Article Sections

C++
algorithms & formulas
c++ & mfc
date & time
string

COM-based Technologies
atl & wtl
com & activex
com+
shell programming

RSA MD5 Message Digest

This article was contributed by Nick Stone of Langfine Ltd.



RSA MD5 Checksum Test

Enter a string and see the checksum

The checksum of this text is:

MD5 Checksum: a2c4a0610b72e2fecfc957d5a88545a

Select a file and see the checksum

E:\WINNT\system32\notepad.exe

MD5 Checksum: 0143619a245890b8621669dfe1ddddb5

Automatic Self Test

SelfTest

All Pass

About

Overview

This is a C++ implementation of the RSA MD5 message digest algorithm. The algorithm calculates a 32 byte checksum for any data sequence (e.g., array of bytes, a string or a file). Full details of the MD5 algorithm are provided within the code.

Two projects are provided:

- MD5.dsw - builds a library that provides the MD5 calculation routines.
- MD5ChecksumTest.dsw - builds a modeless dialog test environment that demonstrates use of the MD5 library and provides a simple verification tool.

Important Notes

Wherever this implementation of the RSA MD5 algorithm is used, the RSA copyright notices etc. must be adhered to, as described within the code and the test application's "About" box.

All use of this algorithm is at the users risk - no liability of any kind whatsoever is accepted by Nick Stone nor Langfine Ltd.

Further Details

The MD5 message digest algorithm is wrapped in a C++ class named `CMD5Checksum`.

Three public functions are exported:

Controls
button control
combobox
edit control
imagelist control
listbox control
listview control
menu
other controls
property sheet
rich edit control
static control
status bar
toolbar
treeview control

Data
database
miscellaneous

Frameworks
ui & printing frameworks

Graphics & Multimedia
bitmaps & palettes
directx
gdi
multimedia
opengl

Internet & Networking
ie programming
internet protocols
isapi
network protocols

Miscellaneous
miscellaneous

Visual Studio
debugging
add-ins & macros
editor tips

Windows Programming
ce
clipboard
dll
file & folder
help systems
printing
win32
system

Windows & Dialogs
console
dialog
docking window
doc/view
splitter

Interact

Newsletters
Guestbook
Recommend Us!
About Us

```
CString GetMD5(BYTE* pBuf, UINT nLength);
CString GetMD5(CFile& File);
CString GetMD5(const CString& strFilePath);
```

All are implemented as static functions. The checksum calculated is returned as a 32 character hexadecimal number held in a CString. **GetMD5** is overloaded to take data to be checksummed as either an array of bytes or a file.

The class `CMD5Checksum` is held within the MD5.dsw project. Building this project creates MD5.lib.

A test environment is provided in the project MD5ChecksumTest.dsw. This allows the user to type a string and see its checksum calculated in real time and also to select a file for checksum. Data files are provided containing simple test data; get the checksum for these and then try typing the file's contents into the first edit box and compare the two checksums thus obtained (they should be the same!) The test environment also includes a "Self Test" button. This checksums each of the supplied test data files in turn and checks the calculated checksum with the known correct value.

Comments are welcome at md5pds@langfine.com.

Downloads

Download source - 18 Kb
Download demo project - 46 Kb

History

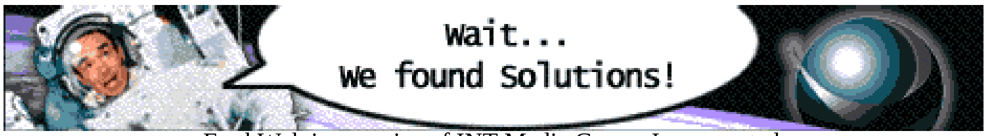
Date Posted: October 3, 2000

Comments:

- Modified for speed - Adam Strasel (2001/09/25)
- Great Job - Denny Depok (2001/06/27)
- Thanks! - Michael G. Spohn (2001/06/10)
- about MD5 - edzy (2001/05/30)

Add Comment

internet jobs



http://www.internet.com

developer.com
codeguru

Dec. 6-7, 2001 ORLANDO, FL
Premier Co-Sponsor: Microsoft Services for the Enterprise

Search

CodeGuru Sites

Visual C++/MFC
C# and.NET
Visual Basic

Discussion Boards
Books on.NET
Book Reviews
Newsletters (subscribe)
Newsletters (archived)

Article Sections

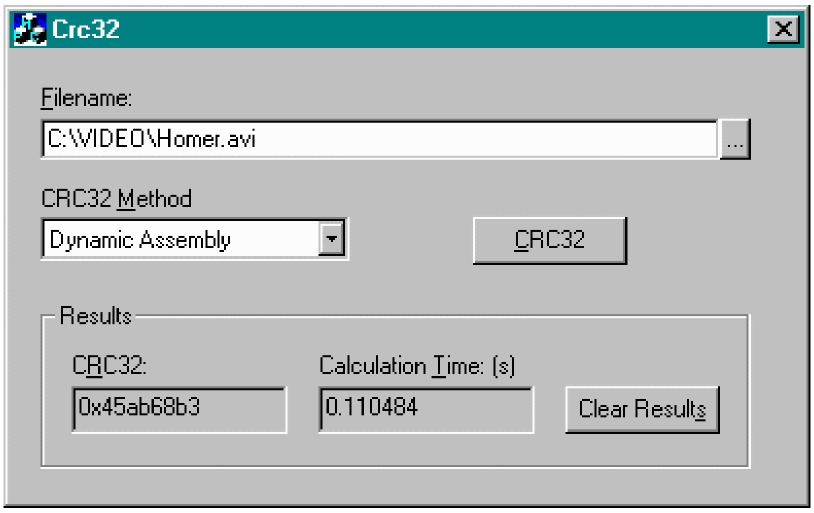
C++
algorithms & formulas
c++ & mfc
date & time
string

COM-based Technologies
atl & wtl
com & activex
com+
shell programming

Controls
button control
combobox
edit control
imagelist control
listbox control
listview control
menu
other controls
property sheet
rich edit control
static control
status bar

CRC32: Generating a checksum for a file

This article was contributed by Brian Friesen.



Environment: VC5, VC6

Introduction

Recently I wrote a program in which I wanted to generate a CRC for a given file. I did some checking on the web for sample CRC code, but found very few algorithms to help me. So I decided to learn more about CRCs and write my own code. This article describes what a CRC is, how to generate them, what they can be used for, and lastly source code showing how it's done.

What is a CRC

CRC is an acronym for Cyclic Redundancy Checksum or Cyclic Redundancy Check (depending on who you ask). A CRC is a "digital signature" representing data. The most common CRC is CRC32, in which the "digital signature" is a 32-bit number. The "data" that is being CRC'ed can be any data of any length; from a file, to a string, or even a block of memory. As long as the data can be represented as a series of bytes, it can be CRC'ed. There is no single CRC algorithm, there can be as many algorithms as there are programmers. The ideal CRC algorithm has several characteristics about it. First, if you CRC the same data more than once, you must get the same CRC every time. Secondly, if you CRC two different pieces of data you should get two very different CRC values. If you CRC the same data twice, you get the same digital signature. But if you CRC data that differs (even by a single byte) then you should get two very different digital signatures. With a 32-bit CRC there are over 4 billion possible CRC values. To be exact that's 2^{32} or 4,294,967,296. With that many CRC values it's not difficult for every piece of data being CRC'ed to get a unique CRC value. However, it is possible for spurious hits to happen. In other words two completely different pieces of data can have the same CRC. This is rare, but not so rare that it won't happen.

Why use CRCs

Most of the time CRCs are used to compare data as an integrity check. Suppose there are two files that need to be compared to determine if they are identical. The first file is on *Machine A* and the other file is on *Machine B*. Each file is a rather large file (say 500 MB), and there is no network connection between the two machines. How do you compare the two files? The answer is CRC. You CRC each of the two files, which gives you two 32-bit numbers. You then compare those 32-bit numbers to see if they are identical. If the CRC values are different, then you can be 100% guaranteed that the files are not the same. If the CRC values are the same, then you can be 99% sure that the files are the same. Remember, because spurious hits can happen you cannot be positive that the two files are identical. The only way to be positive they are the same is to break down and do a comparison one byte at a time. But CRCs offer a quick way to be reasonably certain that two files are identical.

How to generate CRCs

Generating CRCs is a lot like cryptography in that involves a lot of mathematical theories. Since I don't fully understand it myself, I won't go into a lot of those details here. Instead I'll focus on how to program a CRC algorithm. Once you know how the algorithm works you should be able to write a CRC algorithm in any language on any platform. The first part of generating CRCs is the CRC lookup table. In CRC32 this is a table of 256 specific CRC numbers. These numbers are generated by a polynomial (the computation of these numbers and what polynomial to use are part of that math stuff I'm avoiding). The next part is a CRC lookup function. This function takes two things, a single byte of data to be CRC'ed and the current CRC value. It does a lookup in the CRC table according to the byte provided, and then does some math to apply that lookup value to the given CRC value resulting in a new CRC value. The last piece needed is the actual data that is to be CRC'ed. The CRC algorithm reads the first byte of data and calls the CRC lookup function which returns the CRC value for that single byte. It then calls the CRC lookup function with the next byte of data and passes the previous CRC value. After the second call, the CRC value represents the CRC of the first two bytes. You continuously call the CRC lookup function until all the bytes of the data have been processed. The resulting value is the CRC for the whole data.

Code Details

In this sample program I wanted to show that there are many different ways of generating CRCs. There are over 8 different CRC functions, all based on the above steps for generating CRCs. Each function differs slightly in it's intended use or optimization. There are four main CRC functions, each described below. There are also two separate CRC classes, but more on that later. And lastly there are a few helper functions that CRC strings.

C++ Streams: The first function represents the simplest CRC function. The file is opened using the C++ stream classes (ifstream). This function uses nothing but standard C++ calls, so this function should compile and run using any C++ compiler on any OS.

Win32 I/O: This function is more optimized in that it uses the Win32 API for file I/O; CreateFile, and ReadFile. This will speed up the processing, but by using the Win32 API the code is no longer platform independent.

Filemaps: This function uses memory mapped files to process the file. Filemaps can be used to greatly increase the speed with which files are accessed. They allow the contents of a file to be accessed as if it were in memory. No longer does the programmer need to call ReadFile and WriteFile.

Assembly: The final CRC function is one that is optimized using Intel Assembly. By hand writing the assembly code the algorithm can be optimized for speed, although at the sacrifice of being easy to read and understand.

Those are the four main CRC functions. But there are actually two versions of each function. There are two classes, CCrc32Dynamic and CCrc32Static, each of which have the above four functions for a total of eight. The only difference between the static and dynamic classes is the CRC table. With the static class the CRC table and all the functions in the class are static. The trade off is simple. The static class is simpler to use, but the dynamic class uses memory more efficiently because the CRC table (1K in size) is only allocated when needed.

```
// Using the static class is as easy as one
//  line of code
dwErrorCode =
     CCrc32Static::FileCrc32Assembly(m_strFilename,
                                     dwCrc32);

// Whereas there is more involved when using the
//   dynamic class
CCrc32Dynamic *pobCrc32Dynamic = new CCrc32Dynamic;
pobCrc32Dynamic->Init();
dwErrorCode =
     pobCrc32Dynamic->FileCrc32Assembly(m_strFilename,
                                        dwCrc32);
pobCrc32Dynamic->Free();
delete pobCrc32Dynamic;
```

Whenever you calculate a CRC you need to take into account the speed of the algorithm. Generating CRCs for files is both a CPU and a disk intensive task. Here is a table showing the time it took to CRC three different files. The columns are the different file sizes, the rows are the different CRC functions, and the table entries are in seconds. The system these numbers were captured on is a dual Pentium III at 1 GHz with a 10,000 RPM SCSI Ultra160 hard drive.

	44 KB	**34 MB**	**5 GB**
C++ Streams	0.0013	0.80	125
Win32 I/O	0.0009	0.60	85
Filemaps	0.0010	0.60	87
Assembly	0.0006	0.35	49

As expected the C++ streams is the slowest function followed by the Win32 I/O. However, I was very surprised to see the filemaps was not were not faster than the Win32 I/O, in fact they are slower. After I thought about it some, I realized memory mapped files are designed to provide fast random access to files. But when you CRC you access the file sequentially. Thus filemaps are not faster, and the extra overhead of creating the "views" of the file are why it's slower. Filemaps do have one advantage that none of the other functions have. Memory mapped files are guaranteed to be able to access files up to the maximum file size in NT which is 2^{64} or 18 exabytes. Although the Win32 I/O may handle files of this size, none of the documentation confirms this. [Note: The largest file I have CRC'ed is 40 GB, which all eight functions successfully CRC'ed, but took over 10 minutes each.]

If anyone who reads this article knows a way to improve the speed even more, please post the code or email me. Especially if you know of a speed improvement for the assembly code. I will bet there are further optimizations that can be made to the assembly code. After all I don't know Intel Assembly very well, therefore I'm sure there are optimizations I don't know about.

Downloads

Download demo project - 8 Kb
Download source - 17 Kb

History

Date Posted: August 30, 2001

Comments:

- anagramitic crcs query - D.Deere (2001/09/29)
- Another good source of CRC32 code. - Patrick Hoonhout (2001/09/17)
- Good Stuff Man! - Joel Watson (2001/09/12)
- Thanks - Eic Sanchez (2001/09/10)

- I needed such a timerfunction ;-) - The Blackbird (2001/09/07)
- Thanks looking for this... - Sam C (2001/08/31)
- CRC Library - Pete McEvoy (2001/08/31)
- MD5 vs CRC - Mark Jackson (2001/08/31)
- MapFileAndCheckSum problem - Dani Nuestro (2001/08/31)
- Win32 API: MapFileAndCheckSum - Brian Friesen (2001/08/30)
- MapFileAndChecksum seems to be faster! - Munish (2001/08/30)
- Reflected CRCs - Tim Smith (2001/08/30)

Add Comment

developer.com

codeguru

Dec. 6-7, 2001 ORLANDO, FL Premier Co-Sponsor: Microsoft Services for the Enterprise

Search

CodeGuru Sites

Visual C++/MFC
C# and.NET
Visual Basic

Discussion Boards
Books on.NET
Book Reviews
Newsletters (subscribe)
Newsletters (archived)

Article Sections

C++
algorithms & formulas
c++ & mfc
date & time
string

COM-based Technologies
atl & wtl
com & activex
com+
shell programming

Controls
button control
combobox
edit control
imagelist control
listbox control
listview control
menu
other controls
property sheet
rich edit control
static control
status bar
toolbar
treeview control

Data
database
miscellaneous

Optimizing Tip on Adaptive Arithmetic Coding

This article was contributed by Alexey V. Shanin, Russia.

The **arithmetic coding** data compression method is very well-known. It has a number of implementations used in of popular complex, multistaged compression techniques. This article, therefore, assumes that the reader will be readers with the basic concepts of arithmetic coding.

This article describes an optimization point I found in the most commonly used adaptive Witten-Neal-Cleary implementation (also known as "the finite-precision algorithm"). For details (including source code samples), you can refer to the Mark Nelson's article on arithmetic coding. Below I'm using the common terminology that article also follows.

In the practical implementations I've seen, the cumulative symbol frequency table (a component of the algorithm) was represented by an array of integer numbers containing, for each given symbol, the sum of frequencies of all the symbols having indices less than given. The array was sorted in arbitrary fashion (for instance, by "values" of symbols) or by (non-cumulative) frequencies of symbols. Encoding a regular symbol needs corresponding frequency interval to be determined; decoding needs determining a frequency interval containing a given point. Both actions are followed by updating the adaptive model, what's commonly done by increasing the non-cumulative symbol's frequency (= increasing the cumulative frequencies of all the symbols having indices not less than the one of the symbol encoded/decoded). The first thing is fast, the second can be done fast enough by binary search in the array (which is monotone; I've noted it's not done in the sample I've referenced above), but the third requires a considerable number of increments to perform. Sorting the array by frequencies gives effect only for small arrays (= small total number of symbols) and EXTREMELY non-uniform distribution of symbols.

So I've suggested to use a tree-like representation of the frequency information (stored in an array). Assume we totally have 8 symbols : 0, 1, ..., 7 and their cumulative frequencies Q[0] = 0, Q[1], ... Q[7], Q[8] = (the sum of all non-cumulative frequencies). We can break the whole frequency interval (0, Q[8]) with Q[4] into 2 subintervals (of lengths Q[4] and Q[8] - Q[4]), then do the same with the intervals obtained (using Q[2] and Q[6]), and so on. Thus we obtain a binary tree of "breaking points", each node of which represents breaking of an interval into the two. Let the node contain the breaking location counting from the lower bound of an interval. If we now store the resulting tree in an array using the indices of breaking points (as Q[] elements) in the Q[] array, we will obtain the following array A[]:
Q[0] = 0, Q[1], Q[2], Q[3] - Q[1], Q[4], Q[5] - Q[4], Q[6] - Q[4], Q[7] - Q[6] (, Q[8]).
Now, obtaining interval bounds and updating the model while encoding/decoding can be done parallelly. To pass a given symbol while encoding, binary-search it's index in the "0, 1, ..., 7, 8" array (starting to compare with "8", then "4" ...) with increasing A[k] when going left the node "k". To pass a given point while decoding, binary-search it in A[] with subtracting A[k] from the point when going right the node "k" (and increasing A[k] if going to the left). Initial state of A[] is not hard to determine (in our case, it is "0, 1, 2, 1, 4, 1, 2, 1, 8"), and the overflow preventing (halving the frequencies) is also trivial. Q[8] can be stored in A[0] instead of A[8]. I'm attaching a code shortcut containing 'CFreqModIncr' class implementing all this functionality (sorry, but I didn't have time to make up a complete project demonstrating it :) (maybe someone will do it "for me and others" ? ;).

```
// *** A short sample.

// Construct the model.
CFreqModIncr model;

// Initialize to keep 11 symbols and
// set up the increment.
model.Initialize(11);
model.dwIncrement = 1;
```

```
// To pass a symbol with index 'dwIndex'
// while encoding, use :
DWORD dwLeft, dwRight, dwTotal = model.GetTotalFrequency();
model.PassByIndex(dwIndex, dwLeft, dwRight);
... // ... and encode the (dwLeft/dwTotal,
... // dwRight/dwTotal) interval

// To decode a symbol, obtain a point
// 'dwPoint' in frequency units, then use :
DWORD dwLeft, dwRight, dwTotal = model.GetTotalFrequency();
DWORD dwIndex = model.PassByPoint(dwPoint, dwLeft, dwRight);
... // dwIndex = index of the symbol
... // decoded; other 'dw's are to be
// used in the rest of decoding process ...

model.Uninitialize();
```

Downloads

Download source - 2 Kb

History

Date Posted: January 23, 2001

Comments:

- Lexics... - Ivan V. Borsevici (2001/03/20)

Add Comment

A Developer's Guide to the Euro

This article was contributed by Duncan Jones.

What is the EMU/Euro

On January 1st 1999, 11 currencies fixed their exchange rates against a new currency unit - the Euro.

The fixed rates are:

1	Euro is
40.3399	Belgian franc
1.95583	German mark
166.386	Spanish peseta
6.55957	French franc
0.787564	Irish pounds
1936.27	Italian lira
40.3399	Luxembourg franc
2.20371	Dutch guilder
13.7603	Austrian schilling
200.482	Portuguese escudo
5.94573	Finnish markka

This was phase 1 of the European Monetary Union (EMU) and the birth of the currency called the Euro.

On January 1st 2002 the currencies listed above will cease to exist. This means that all transactions, prices and values expressed in these currencies must be converted such that they are expressed in Euros.

Conversion rules

Single rate:

The rule is that the above rate must be used regardless of whether you are converting to or from Euros - the inverse rate may not be used. Thus, to convert from Irish pounds to Euro, divide by 0.787564. To convert from Euro to Irish pounds, multiply by .787564

Precision

Six digit precision must be used in all Euro conversions.

Controls
button control
combobox
edit control
imagelist control
listbox control
listview control
menu
other controls
property sheet
rich edit control
static control
status bar
toolbar
treeview control

Data
database
miscellaneous

Frameworks
ui & printing frameworks

Graphics & Multimedia
bitmaps & palettes
directx
gdi
multimedia
opengl

Internet & Networking
ie programming
internet protocols
isapi
network protocols

Miscellaneous
miscellaneous

Visual Studio
debugging
add-ins & macros
editor tips

Windows Programming
ce
clipboard
dll
file & folder
help systems
printing
win32
system

Windows & Dialogs
console
dialog
docking window
doc/view
splitter

Interact

Rounding

After conversion to Euros, the result must be rounded to the nearest cent. A cent is one hundredth of a Euro - thus the post conversion precision is 2 digits.

Triangulation

All currency conversions to or from a member currency must be converted through the euro - this process is known as triangulation. Thus, to convert from Deutchmarks to US dollars, an amount must first be converted to Euros (according to rules 2.1 to 2.3) and then this resulting amount converted to US dollars.

Practical considerations for developers

These rules lead to the following practical considerations:

Storing the conversion rates

The rates above are fixed and immutable. This means that you can hard code them in a program if this suits your development environment (although it is probably better to store them in a database table to allow them to be accessed elsewhere). However you store the rate, however, the data type of the variable must allow 6-digit precision.

Performing the conversion

As above, any interim store, variable or function used to perform a Euro conversion must be capable of at least six digits of precision. In Visual Basic (and by extension, Visual basic for Applications) this means that you must not store any calculations as a Currency, Integer or Long Integer variable types and that conversions to/from these data types must not be part of the conversion routine. In SQL, the data types MONEY and SMALLMONEY must be avoided.

Post conversion rounding

Rounding is to two digits, where anything with a third significant digit of less than 5 is rounded down and more than 5 is rounded up. Exactly 5 is rounded to the nearest EVEN number. For the most part any system defined conversion (e.g. `CONVERT(MONEY,@value)` in SQL or `Format$(vValue,"Fixed")` in Visual Basic) will already do this - but you should check any rounding functions (implicit or explicit) in your programs.

Triangulation

The currencies within the euro zone (noted above) should be known to your systems and any conversion to or from them should be done in two steps. 1st convert to Euros, 2nd convert to your target currency.

Timetable

Like Y2K, this is another IT problem with a hard deadline. If your systems are not Euro enabled by Jan 1st 2002 you will not be able use them to trade in the Euro zone. If you aren't working on this right now, get started.

Other sources of information

A European Commission paper on IT and the Euro

Downloads

Download demo executable and setup program - 1400 Kb
Download source code - 14 Kb

History

Date Posted: March 06, 2001

Comments:

- euro - Leon Schell (2001/07/09)
- http://www.codeguru.com/algorithms/CodeGuruEuro.html - Leon Schell (2001/07/09)
- Rounding Conventions - Chris Meech (2001/03/15)
- Wonderful article! - Tom Archer (2001/03/13)

Add Comment

Nanotech PLANET Fall 2001
Conference & Expo
The Center of Nanotechnology Business
Hosted by: Nanotech PLANET.COM
Click Here Now!

Method of Handling Factorials of Any Size

This article was contributed by Hai Yi.

Overview

We know that the data type,whatever it is,has minimum and maximum value.But sometime we need to process a very large number such as the factorial of 1000.It's hard to find any data type to store it.

However, one method to get around this is to dynamically create a linked list such that each element in the list represents a single digit of the result. That way, theorically speaking, the result's length is only limited by the memory!

Steps to Use

1. Set up a dual_direction linked list,and allocate the first element which is 1 (I'm assuming straight C++. However, if you're using MFC, you can also use an MFC collection class such as the CPtrList class. I personally prefer to create my own class for this purpose.
2. From the first element to the nth element (last element), simply create an element in the list of each digit until you don't have any more digits in the return result.

Code

```
//
//
```

Discussion Boards
Books on.NET
Book Reviews
Newsletters (subscribe)
Newsletters (archived)

Article Sections

C++
algorithms & formulas
c++ & mfc
date & time
string

COM-based Technologies
atl & wtl
com & activex
com+
shell programming

Controls
button control
combobox
edit control
imagelist control
listbox control
listview control
menu
other controls
property sheet
rich edit control
static control
status bar
toolbar
treeview control

Data
database
miscellaneous

```
// Nnn.cpp : Defines the entry point for the console application.
// author  : Hai Yi
// date    : Sept,11,2000
//
//

#include "stdafx.h"

#include "iostream.h"
#include "stdlib.h"


//here is a dual link list
class Node{

private:
 int data;
 Node *next;
 Node *prev;
 Node *head;
 Node *rear;


public:
 Node(const int& item)
 :data(item),prev(NULL),next(NULL),head(NULL),rear(NULL){};

 //get next node
 Node* GetNextNode(){return next;};
 Node* GetPrevNode(){return prev;};

 //insert after
 void InsertAfterMe(Node* p);

 //Delete the appointed
 void DeleteMe(void);

 int GetData(void){return data;};
 void SetData(int item){data = item;};

 //reset
 Node* GoBacktoHead();

 //go to the rear
 Node* GoForwardtoRear();
 //clear the whole
 void ClearAll(void);

 //get the counts of the link
 int GetElementNum();
};


int Node::GetElementNum()
{
 int count = 0;
 Node* p =GoBacktoHead();

 while(p->GetNextNode()!=NULL){
  count++;
  p = p->GetNextNode();
 }

 count++;
 return count;
}

void Node::InsertAfterMe(Node* p)
{
 //     Node* p;
 if(prev == NULL) { head = this;}
 p->next = next;
 p->prev = this;
 next = p;
 if(p->next == NULL){rear = p;}
};
```

```
void Node::DeleteMe(void)
{
 if(prev == NULL) { // if this node is the first one
  next->prev = NULL;
  head = next;  // then the next one becomes the first one
  delete this;  //delete this node
  return;
 }

 if(next == NULL){  //if this node is the last one
  prev->next = NULL;
  rear = prev; // then the previous one becomes the last one
  return;
 }

 prev->next = next;
 delete this;
};

Node* Node::GoBacktoHead()
{
 if(head == this){ //this is the first node
  return this;
 }

 Node *p = this;
 while(p->prev != NULL){
  p = p->prev;
 }

 return p;
}

Node* Node::GoForwardtoRear()
{
 if(rear == this){
  return this;
 }

 Node *p = this;
 while(p->next != NULL){
  p = p->next;
 }

 return p;
}

void Node::ClearAll(void)
{
 Node* p = GoBacktoHead();
 Node* p2;
 while(p->GetNextNode() != NULL){
  p2 = p;
  p = p->GetNextNode();
  delete p2;
 }

 delete p;
};

int main(int argc, char* argv[])
{
 int remain;
 int carry;
 int result;
 int N;
 Node* p = new Node(1);

 cout<<"Pls input the number:";
 cin>>N;
 for(int n=1;n<=N;n++)
 {
  remain = carry = 0;
  p = p->GoBacktoHead();
  //while not the end of the list,process the element one by one
  while(p->GetNextNode() != NULL){
   result = p->GetData()*n+carry;
   if(result>=10){
    remain = result%10;
    carry = result/10;
    p->SetData(remain);
   }
   else{p->SetData(result);}

  p = p->GetNextNode();
  carry = result/10;
  }

  result = p->GetData()*n+carry;

  //if carry occurs,process the carry and
  //store into the newly allocated space.

  while(result >= 10){
   Node * newNode = new Node(0);
   p->SetData(result%10);//remainder
   result = result/10;
   p->InsertAfterMe(newNode);
   p = p->GetNextNode();
  }

  p->SetData(result);

 }//end of if

 p = p->GoForwardtoRear();

 while(p->GetPrevNode()!=NULL){
  cout<<p->GetData();
  p=p->GetPrevNode();
 }
```

```
  cout<<p->GetData()<<endl;
  int num = p->GetElementNum();
  if(num >=5){
   p = p->GoForwardtoRear();

   cout<<endl<<"Or"<<endl<<endl;

   cout<<p->GetData()<<".";
   p = p->GetPrevNode();

   for(int i=1;i<5;i++){
    cout<<p->GetData();
    p = p->GetPrevNode();
   }

   cout<<"E"<num-1<endl;
  }

  //clear the memory
  p->ClearAll();

  return 0;
}
```

History

Date Posted: September 24, 2000

Comments:

- Use Buffer instead of this.. That's better and faster - Rishab (2001/10/08)
- Nice approach - Francisco Almeida (2001/10/03)
- Nice approach - Francisco Almeida (2001/10/03)
- Method of Handling Factorials of Any Size (WITH ARRAYS) - D.Deere (2001/09/29)
- Factorials - Anil Kumar (2001/09/19)

- Good Work - devendra (2001/09/03)
- Efficiency concerns - andi payn (2001/08/30)
- Good - Manohar (2001/08/27)
- fact - maharipu (2001/08/24)
- Code is Very Simple... Boring... - Dong Back, Kim (2001/08/02)
- algorithms And formula - vipin gupta (2001/07/29)
- Code is good - VINIT AGRAWAL (2001/07/18)
- How about speed of the code? - Alok Govil (2001/06/25)
- response about missing stafx header - tj dombrowski (2001/06/02)
- Factorials in Lisp - Christian (2001/05/30)
- method of finding a factorial of any size - yogananda.T (2001/05/28)
- why 10?? - totoro (2001/05/17)
- Hurrah, a simple example of a linked list! - Yilmaz (2001/04/26)
- factorial - ali farid (2001/04/24)
- Good Work - majid_khattak (2001/04/23)
- Answer (For Anand) - Noam (2001/04/17)
- Factorials - Anand Mirasdar (2001/04/12)

Add Comment

developer.com
codeguru

Algorithm for Detecting Duplicate/Previously Encountered Strings

This article was contributed by Jeremy Cattone.

Environment: Straight C++

Overview

A challenge I've seen pop up often while writing search tools is the ability to verify that your code is not traversing the same source multiple times. A spider traversing the web, for example, would never wish to traverse the same URL twice in a session - to do so would merely allow circular loops and endless headaches.

To address this issue, I wrote an N-tree based search algorithm that breaks strings down by their leading characters. Identical portions are 'clipped' and stored in a single node, with the remainder of the string in similarly divided child nodes.

Example

```
Inserting the following text:
1234
12345
123456
1234567
www.another.site.org/index.html
www.site.org/index.html
www.test.com/index2.html
www.test.com/index.html
www.test.com

Results in this data structure
+-NULL
  +-1234
    +-NULL
    +-5
      +-NULL
      +-6
        +-NULL
        +-7
  +-www.
```

Search

CodeGuru Sites

Visual C++/MFC
C# and.NET
Visual Basic

Discussion Boards
Books on.NET
Book Reviews
Newsletters (subscribe)
Newsletters (archived)

Article Sections

C++
algorithms & formulas
c++ & mfc
date & time
string

COM-based Technologies
atl & wtl
com & activex
com+
shell programming

Nanotech PLANET Fall 2001
Conference & Expo
The Center of Nanotechnology Business
November 29-30, 2001
Marriott Copley Place, Boston MA
Hosted by Nanotech PLANET.COM
Click Here Now!
Nanotech PLANET Fall 2001
Conference & Expo

Controls
button control
combobox
edit control
imagelist control
listbox control
listview control
menu
other controls
property sheet
rich edit control
static control
status bar
toolbar
treeview control

Data
database
miscellaneous

Frameworks
ui & printing frameworks

Graphics & Multimedia
bitmaps & palettes
directx
gdi
multimedia
opengl

Internet & Networking
ie programming
internet protocols
isapi
network protocols

Miscellaneous
miscellaneous

Visual Studio
debugging
add-ins & macros
editor tips

Windows Programming
ce
clipboard
dll
file & folder
help systems
printing
win32
system

Windows & Dialogs
console
dialog
docking window
doc/view
splitter

```
    +-another.site.org/index.html
    +-site.org/index.html
    +-test.com
     +-NULL
     +-/index
        +-2.html
        +-.html
```

The advantage to this approach over typical linear searches is two-fold.

1. The memory overhead can be reduced as redundant text is partially eliminated
2. Since the tree is ordered, a search can be performed in O(Log(N)) time (someone please correct me if my estimate of this algorithm is incorrect!)

On a 600 MHz PIII system, the following benchmarks were obtained for inserting 3000 random strings of length 0-254 characters into an array that already contains 12,000 strings.

```
Linear search: 2.7 seconds
StringTree: 0.02 seconds
```

A reasonable improvement!

The attached code implements the algorithm as a CStringTree class, and provides a test / performance driver to demo the code.

Updates

Changes to current source (zip file below):

- Fixed several small memory leaks
- Added significant functionality for searching and File / CStringArray I/O
- Compiler defs for Borland compilers
- String counting and data association per node

Downloads

Download demo and source code - 12 Kb

History

Date Posted: February 22, 2001
Date Last Updated: June 11, 2001

Comments:

- Static is deprecated! - Chandru Aswani (2001/09/13)
- memory leak - Rene Rapp (2001/08/23)
- memory leak.. - jihoon kim (2001/06/13)
- Standard Data-Structure - Mark Watts (2001/06/12)

- Article Updated - WebMaster (2001/06/11)
- What an improvement! - Adam (2001/05/14)
- I couldn't debug it. - NETHORSE (2001/04/16)

Add Comment