

IEEEExtreme Global Programming Challenge



IEEE

DECEMBER 2006

<http://www.ieee.org/ieccextreme>

Instructions

- Read all the problems **carefully**.
- Each of the problems has a problem number (shown on top), a title, an approximate "value rank" which gives an idea on how much points it can give, a descriptive text and some examples of inputs/outputs.
- You **don't have to do all** the problems and you **don't have to follow any specific order** for submissions. This allows you to do some strategy play deciding whether to go for the more valuable but more difficult problems first or solve the easier ones, which may give lower point counts.
- Should any of the problems have incomplete information or require more data (e.g: if you are required to code a dot multiplication for vectors and you didn't know what that is) please **feel free to use resources** like the Internet, books or extrasensorial perception to learn about ways of solving your problem or for extra examples. The only thing the team **can't use is the help of other human beings** (this includes, but is not limited to, your proctor, members of other teams and your friends and family)
- Please make sure your program receives input and shows output in **EXACTLY the format** that is requested in the text and shown in the examples. This is necessary for the automated judging system to work properly. Also, make sure your solution complies with the procedures described in the "Submissions" section (see below).
- Updates and information will be sent to the teams **via the proctor's email** during the competition. Also, important **updates** will be published in the contest blog page <http://ieeextreme.org>, and a **backup** page will be hosted in <http://ieeextreme.googlepages.com>
- Should you have any issues or questions about general contest procedures please feel free to contact our **helpdesk team** during the competition writing email to ieeextreme@ieee.org. In case of issues, our backup mail address is ieeextreme@gmail.com. Please note the helpdesk team will **NOT** answer questions related to the programming problems nor provide extra examples nor any other information.
- And most important... **Have fun!** :)

Submission procedure

- You must keep **all files** for the solution of each problem **in a separate directory**, identified by the problem code. This means all files of the solution from problem 1 should be inside a directory called "problem1"
- The programming languages allowed in the competition are C, C++ (using gcc), Java (using Sun's JDK) and C# (using the Mono platform). For more details about versions, check the section "Target machine setup" below.
- Inside every problem directory there should be at least one special file called a **buildfile**. This file will be used to build your program in the automated system. After being executed, the file should create an executable file, named "**program**", in the problem directory.
- For each of the problems, your buildfile can be **one of these 3 types**:
 - a **bash shell script**, in which case it should be named "build.sh" and will be executed by the system
 - a **GNU make makefile**, in which case it should be named "Makefile"
 - an **Apache Ant build file**, in which case it should be named "build.xml"
- Apart from the buildfile and the source code files (and libraries or other external resources you want to use) we would advise **NOT** to include any other files (like temporary files, object files or executables from previous compilations). The total size of all the files in each of the solutions directories should be **less than 2 megabytes**.
- You are allowed to use external libraries or other **freely available** third-party code in your solution as long as it can be used for your purpose under an **open source license**, you **include them** in the solution directory and the total size of your directory files **do not exceed 2 megabytes**.
- Once you have all your source ready and your buildfile working properly to build your executable (remember, named "program"), you should **package the directory in a .ZIP, .RAR, .TAR or .TAR.GZ** file using any of the standard utilities available (like 7-Zip for Windows or any of the command-line utilities in Unix/Linux and MacOS X)
- Name that package file **using your team id and the problem number**, separated by dashes (-) and keeping the proper extension. For example, team 42 submission for problem 11, compressed with tar should be named 42-11.tar.

- Send the package using the **submission web form**. This should be available online at: [http://ieeextreme.org/contest/\(your-team-id\)](http://ieeextreme.org/contest/(your-team-id)), where "(your team id)" should be replaced by a unique id that should have been sent to your team at the beginning of the competition. If we change the submission form URL, the new URL will be published in our web page and the backup page, and sent via mail to the proctors.
- In the event of a **problem with the submission** web form, you can request to send your submissions via email. In this case, you can send the submission package file to ieeextreme@ieee.org as an attachment, and use your team id and the problem number, separated by dashes (-), as the mail subject. In case there are problems with that mail, send it to our backup mail address: ieeextreme@gmail.com.

Target machine setup

- All your programs will be **automatically tested** in a target system, therefore it is very important that you follow the submission instructions above.
- Specific versions of programs are available in the target system. Please **DO NOT** use any extensions or features not available in these versions.
- All these programs are **freely available for download** on the Internet for diverse platforms (including Linux, MacOS X and Windows)
- For the **buildfiles**, our automated testing system uses **ant 1.6.5**, **make 3.8** and **bash 3.1-2**
- For **C/C++ submissions**, the system uses **gcc/g++ compilers version 4.0.3-1** (available from <http://gcc.gnu.org/>)
- For **Java submissions**, the system uses **Sun's JDK for the Java 2 Platform Standard edition 5.0** (available from <http://java.sun.com/j2se/1.5.0/system-configurations.html>)
- For **C# submissions**, the system uses **Mono SDK version 1.2.1** (available from <http://www.mono-project.com/Downloads>)
- All executables are **included in the system path**, and the usual **environment variables** pointing to the installation directories will also be set (e.g: ANT_HOME, JAVA_HOME or MONO_HOME)
- The target machine system is based on a Linux distribution and will be tested for portability, therefore be careful **not to include any platform-specific code**.

PROBLEM 1

Cosine similarities

Value rank: 40

Simple matching, Jaccard, Tanimoto and cosine similarities are statistics used for comparing the similarity and diversity of sample sets. They are used in data mining for tasks ranging from classifying diverse chemical compounds to text processing and searching in large databases.

The sample sets for comparison are represented with vectors of the attributes we want to compare. Given 2 vectors of attributes, A and B, the cosine similarity, θ , can be calculated using the dot product and the magnitudes as:

$$\theta = \arccos \frac{A \cdot B}{\|A\| \|B\|} .$$

Since the angle, θ , is in the range of $[0, \pi]$, the resulting similarity will yield the value of π as meaning exactly opposite, $\pi / 2$ meaning independent, 0 meaning exactly the same, with in-between values indicating intermediate similarities or dissimilarities.

Your task is to write a program that receives A and B, attribute vectors, and outputs the cosine similarity of those vectors

A and B will be entered as parameters in the command line, as square-brackets-delimited, comma-separated lists of integer values, separated by a space.

The output should be a single line containing the cosine similarity value with 4 decimals of precision, or the word ERROR if any of the entries is not valid or the operation cannot be performed

examples

```
> program [3,2,0,5,0,0,0,2,0,0] [1,0,0,0,0,0,0,1,0,2]
1.2503
```

```
> program [6,5,5] [1,1]
ERROR
```

```
> program [,] []
ERROR
```

PROBLEM 2

Carmichael numbers

Value rank: 40

Prime numbers are quite special, as you probably know. They're very important in several Engineering fields and in Computer Science, in disciplines like cryptography, because of their various mathematical properties. Big prime numbers are particularly interesting here.

However, prime numbers have the problem of being costly to find in terms of computing power. The oldest known approach, the Erathostenes' Sieve, has exponential complexity.

Luckily, there are some probabilistic primality tests with a lower computing cost. A common one is the Fermat test, which basically states that for a given number n , being x a random number between 2 and $n-1$, our number n is probably prime if this equation holds:

$$a^n \bmod n = a$$

A way of testing primality this way is trying several values of a , and if the test passes, then we probably have a prime number.

Unfortunately, there are a special series of numbers who can pass the Fermat test for every number smaller than themselves, yet they are not prime numbers. Those are known as Carmichael numbers.

Your task is to write a program that receives a series of numbers and for each of them returns 1 if they are Carmichael numbers, 0 if they are not.

Numbers will be passed as parameters, separated by spaces

examples

```
> program 1105 7 561  
1 0 1
```

```
> program 2465  
1
```

```
> program 12 21 1 0  
0 0 0 0
```

Random excerpt #1

note: this is not a contest problem!

Hello there! Thanks from all the organization team for participating and being here reading this! We thought that as now you have a lot of tasks to do you may appreciate some company, so in this booklet you will find some pages of Random Excerpts that basically contain no useful information at all, but which may be useful for cheering you up a bit :)

For example, we thought that as the competition has just started, it would be nice if we all sing along. As the nerdy side is strong on us, we have decided to intone this adaptation of "Let it be", originally by The Beatles So, ready? Stand up (maybe even in your chair), click your fingers to the rhythm, tap on the table... One, two... one, two, three, four...

*When I find my code in tons of trouble,
Friends and colleagues come to me,
Speaking words of wisdom:
"Write in C."*

*As the deadline fast approaches,
And bugs are all that I can see,
Somewhere, someone whispers"
"Write in C."*

*Write in C, write in C,
Write in C, write in C.
LISP is dead and buried,
Write in C.*

*I used to write a lot of FORTRAN,
for science it worked flawlessly.
Try using it for graphics!
Write in C.*

*If you've just spent nearly 30 hours
Debugging some assembly,
Soon you will be glad to
Write in C.*

*Write in C, write in C,
Write In C, yeah, write in C.
Only wimps use BASIC.
Write in C.*

*Write in C, write in C,
Write in C, oh, write in C.
Pascal won't quite cut it.
Write in C.*

[insert air guitar solo here]

*Write in C, write in C,
Write in C, yeah, write in C.
Don't even mention COBOL.
Write in C.*

*And when the screen is fuzzy,
And the editor is bugging me.
I'm sick of ones and zeroes.
Write in C.*

*A thousand people swear
that T.P. Seven is the one for me.
I hate the word PROCEDURE,
Write in C.*

*Write in C, write in C,
Write in C, yeah, write in C.
PL1 is 80's,
Write in C.*

*Write in C, write in C,
Write in C, yeah, write in C.
The government loves ADA,
Write in C.*

PROBLEM 3

Incomplete communications

Value rank: 120

When transmitting messages over unreliable networks, sometimes pieces of the message get lost. Some streaming protocols (eg gaming or voice protocols used over UDP) prefer to deal with incomplete information instead of losing performance resending messages to maintain coherency. Sometimes these protocols reassemble information on destination based on the context or some other data.

For this example, we have emulated a faulty protocol that transmits a series of ASCII characters, being possible for some of them to be lost. In that case, we replace the lost characters with asterisks (*). The original message didn't contain asterisks (so no confusion possible). We also have a dictionary file with the words that may appear in the message, one per line. A word is defined as a sequence of characters such that the character before and after the word is not a letter. The dictionary file only contains the lower case version of the words (using only the letters but in the message any (or all) character of a word can be in upper case (for example, if the word "ieee" is present in the dictionary, then the message may contain words like "ieee", "Ieee", "IEEE" or "iEEe"))

Your task is to write a program that receives two files: one with the text (with suffix .txt) and other with the dictionary file (with suffix .dict), and outputs all the original text, replacing the asterisks with the proper characters, or a single line with the word ERROR in case of finding any issue with the inputs and/or their format

For simplicity, always use lower case letters when replacing an asterisk (leave the rest of the letters as they are, and in particular, end of line characters should appear in the output exactly as they were in the input file). You can also assume the dictionary-file correspondence is not ambiguous: for every word in the message there can be a maximum of one correct possibility in the dictionary file

examples

message.txt contents:

```
one,O*E,f**r (th*ee t*o) *ix
```

dictionary.dict contents:

```
one  
two  
three  
four  
five  
six
```

```
> program message.txt dictionary.dict
```

```
one,OnE,four (three two) six
```

```
> program message.txt
```

```
ERROR
```

PROBLEM 4

Amateur musicians

Value rank: 150

Did you know that a lot of people on Engineering like a whole range of music styles? It may be because spending so much time on the Internet exposes you to more bands and styles than the average person. It may be because music is somehow related to mathematics and physics (think all those frequencies and sets and operations). It may just be that your nerdy inside is always wanting to experience new stuff.

So, a friend of mine (who happens to be an engineer) likes music so much that apart from his work in the lab, decided to start being a manager for some amateur local bands on his free time. However, he has found this problem: for every night he has a number of bands that he manages and a number of clubs that are allowing performances that night. He uses a mapping tool to locate the places where each of the bands rehearses and the locations of the clubs. He knows that amateur musicians have problems with transport, so he has calculated that each band should travel a maximum of 50km away from their original location. As a good engineer, he thinks he would need a program that shows him, given this data, the maximum number of performances he can expect setting for that day. At most one show can be performed in every club, and each band cannot play more than one show on a given day.

Your task is to write a program that receives a text file with the following format:

- the first line has 2 space-separated numbers, one is the number of bands we have available that night (we call it B and varies from 0 to 100) and the other is the number of clubs accepting performances (we call it C and also varies from 0 to 100)
- then we have B lines, also with 2 space-separated numbers, containing the X and Y coordinates, in kilometers, that map the location of each of the bands
- and finally, we have C more lines with 2 space-separated numbers, containing the X and Y coordinates, in kilometers, that map the location of each of the clubs

Your program should output the maximum number of performances we can set up for that night, or ERROR in case there are problems.

Please note the numbers for the coordinates do not have to always be integers.

examples

input1.txt contents:

```
4 4
10 0
70 0
150 0
220 0
60 0
110 0
190 0
0 0
```

```
> program input1.txt
4
```

input2.txt contents:

```
3 2
3 3
34 3
0 1
5 0
```

```
> program input2.txt
ERROR
```

Random excerpt #2

note: this is not a contest problem!

So, how is it going? Doing better now? Yes, we know it's hard and long and sometimes one gets stuck... but then again, think that all that only makes winning this thing even cooler!

To cheer you up, here you have some funny computing rhymes. Feel free to recite them to your team partners, proctor, family, plush toys and/or friends (imaginary or real):

*If a packet hits a pocket on a socket on a port,
And the bus is interrupted as a very last resort,
And the address of the memory makes your floppy disk abort,
Then the socket packet pocket has an error to report.*

*If your cursor finds a menu item followed by a dash,
And the double-clicking icon puts your window in the trash,
And your data is corrupted 'cause the index doesn't hash,
Then your situation's hopeless and your system's gonna crash!*

*If the label on the cable on the table at your house,
Says the network is connected to the button on your mouse,
But your packets want to tunnel on another protocol
That's repeatedly rejected by the printer down the hall...*

*And your screen is all distorted by the side effects of gauss,
So your icons in the window are as wavy as a souse,
Then you may as well reboot and go out with a bang,
'Cause as sure as I'm a poet, the machine's gonna hang!*

*When the copy of your floppy's getting sloppy on the disk,
And the microcode instructions cause unnecessary risk,
Then you have to flash your memory and you'll want to RAM your ROM,
And quickly turn off the computer and be sure to tell your mom!*

PROBLEM 5

Telephone keyboard input recognition

Value rank: 60

On a standard telephone, the numbers 1-9 can be used to correspond to a set of letters:

1: space	2: ABC	3: DEF
4: GHI	5: JKL	6: MNO
7: PQRS	8: TUV	9: WXYZ

Using the keypad, you can 'spell' words by entering the digits that correspond to each letter of the word. For example, 'words' is spelled 96737.

For this problem, we are given a dictionary file called with no more than 100,000 words, one per line, sorted in alphabetical order. Each word is comprised of no more than 18 characters, all lowercase letters from the phone keypad. Here is a (very short!) example of a dictionary file we will use in the examples:

Your program should read a string of digits (from 2 to 9, not using 1 as space) from the console and find the words in the dictionary whose spellings contain that series of consecutive digits anywhere within the word.

- If there are no matches, print the string 'No matches'
- If there is one match, print the matching word.
- If there are $n > 1$ matches, print the string 'n matches:' followed by the matching words, one per line.

NOTE: To make it easier to read the examples below, these are the 'spellings' of the words in words.txt, in digits:

cappuccino: 2277822466

chocolate: 246265283

cinnamon: 24662666

coffee: 263333

latte: 52883
vanilla: 8264554

examples

```
contents of words.txt:
```

```
cappuccino  
chocolate  
cinnamon  
coffee  
latte  
vanilla
```

```
> program words.txt 22222  
No matches
```

```
> program words.txt 3333  
coffee
```

```
> program words.txt 626  
2 matches:  
chocolate  
cinnamon
```

Random excerpt #3

note: this is not a contest problem!

(this page intentionally left blank)

(and yes, the intention was to have you wondering why :P)

PROBLEM 6

Machine load

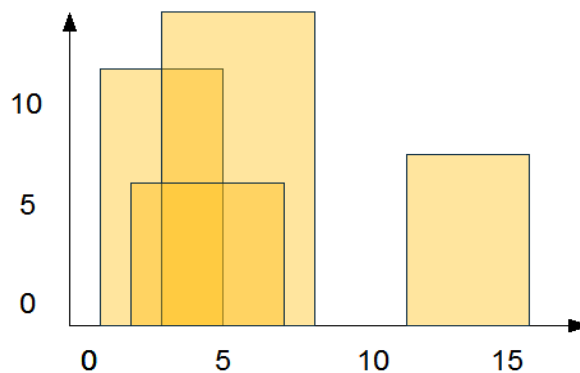
Value rank: 100

For this problem, we will be writing an output generator for a machine load display program. The idea is that we have a series of machines which return their load over time as rectangle coordinates, indicating the load (in the X axis) during the period of time (in the Y axis).

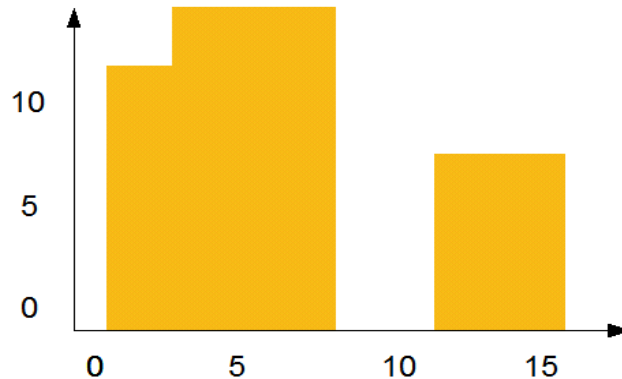
Input for each machine are 3 numbers indicating the time when we start measuring, the load level during that time and the time when we finish measuring. The idea is that an input like:

(1 11 5) (2 6 7) (3 13 9) (12 7 16)

Should represent something like:



However, we are only interested in getting the maximum level at each time, so would prefer to see something like this:



We have a drawing program that takes a list of number pairs, each one representing an X coordinate and a value, and draws a surface with height equal to the given value. Basically, for drawing the graphic shown above, it requires this input:

`(1,11,3,13,9,0,12,7,16,0)`

Which should read something like: "from 1 on, value is 11, until we reach 3, and then value is 13, until we reach 9, when value goes 0, but then on 12 value goes to 7 and then on 16 it gets back to zero" (more or less)

Your task is to write a program that receives the inputs for each of the machines (there can be up to 100 machines) as a list of number triplets in parantheses (as shown in the examples), as a parameter, and prints a string of comma-separated values between parantheses representing the appropriately formatted entry to our drawing system (also as shown) or the word ERROR if the input is not valid.

Please note the times and load values do not have to always be integers.

examples

```
> program (1 11 5) (2 6 7) (3 13 9) (12 7 16)
(1,11,3,13,9,0,12,7,16,0)
```

```
> program (1 5 3) (3 5)
ERROR
```

```
> program (12 7 16) (14 3 25) (19 18 22) (23 13 29) (24 4 28)
(12,7,16,3,19,18,22,3,23,13,29,0)
```

PROBLEM 7

Arithmetic coding

Value rank: 90

Arithmetic coding is a way of efficiently encoding a stream of symbols of varying probabilities using a minimal number of bits. Unlike Huffman coding, symbols can be encoded in a fractional number of bits, and arithmetic encoding is therefore particularly useful when probabilities are very skewed or the alphabet size is small (for example, when compressing monochrome images such as faxes).

Wikipedia provides a good introduction here:

http://en.wikipedia.org/wiki/Arithmetic_coding

For this problem, you will need to write an arbitrary precision arithmetic encoder to compress the binary alphabet consisting of the symbols 0 and 1. Because the encoder uses arbitrary precision (hint: use a library), renormalization and bit stuffing are not required.

The probabilities for your encoder should be provided by counting the number of time the symbol has been seen before, with both counts set to 1 initially. For example, before processing the symbols 0010, $P(0) = P(1) = 1/2$ and afterwards $P(0) = 4/6$ and $P(1) = 2/6$.

Input to your program will consist of a string of up to one hundred 0s and 1s as the first argument. The output of your program should be a decimal fraction lying in the final range, or ERROR if the input is missing or malformed. The lower portion of the range corresponds to 0, and the higher to 1.

examples

```
> program 0010  
0.3
```

```
> program 1010101  
0.604
```

```
> program 1100011000110001100011  
0.6801221
```

```
> program apples  
ERROR
```

PROBLEM 8

Sudoku

Value rank: 60

Probably all of you know, but for those who have been living in another planet for the last years, a Sudoku is a puzzle played on a 9 by 9 grid. The grid is subdivided into 9 blocks of 3 by 3 cells. When you start the puzzle, only a few numbers have been added to the grid. The objective is to fill the grid so that every column, every row and every 3×3 box contains the digits 1 to 9. Sudokus can get quite complex, depending on the amount of cells initially filled, and can require backtracking or other exploration algorithm designs. A subset of all possible Sudoku puzzles, however, are simpler and can be solved by directly addressing the three conditions listed above. We will assume all the possible sudoku puzzles we will use are of this later kind.

For this problem, you will need to write a sudoku problem solver. Your program will receive a text file in which each line can contain 3 integer values, which are the X, Y and value of the cells in the grid (empty cells are either not indicated in the file or marked with a -1 as value) X and Y coordinates are 0-based (therefore ranging from 0 to 8) counting from the top-left corner. For example, the sudoku.txt file in the examples represents the following grid:

9		4		8	7			1
	5					6	8	
2			3					9
	3	6	1		5	2		4
					9		7	
5		7	4				3	8
								5
4		9	6				1	
	1	5			2		6	3

The output of your program should be an ASCII formatted version of the completed sudoku grid (as you can see in the example) if it can be solved and has a unique solution, or ERROR in any other case. Note there are 2 whitespaces between each number, but just one space between numbers and the vertical bars "|", and between the later and the next number.

examples

contents of sudoku.txt:

```
0 0 9
2 0 4
4 0 8
5 0 7
6 0 -1
8 0 1
1 1 5
6 1 6
7 1 8
0 2 2
3 2 3
8 2 9
1 3 3
2 3 6
3 3 1
5 3 5
6 3 2
8 3 4
5 4 9
7 4 7
0 5 5
2 5 7
3 5 4
7 5 3
8 5 8
8 6 5
0 7 4
2 7 9
3 7 6
7 7 1
0 8 -1
1 8 1
2 8 5
5 8 2
7 8 6
8 8 3
```

```
> program sudoku.txt
```

```
9 6 4 | 5 8 7 | 3 2 1
3 5 1 | 2 9 4 | 6 8 7
2 7 8 | 3 6 1 | 4 5 9
```

```
-----
8 3 6 | 1 7 5 | 2 9 4
1 4 2 | 8 3 9 | 5 7 6
5 9 7 | 4 2 6 | 1 3 8
```

```
-----
6 2 3 | 7 1 8 | 9 4 5
4 8 9 | 6 5 3 | 7 1 2
7 1 5 | 9 4 2 | 8 6 3
```

```
contents of sudoku2.txt:
```

```
0 0 9
2 0 4
4 0 8
5 0 7
6 0 -1
8 0 1
1 1 5
6 1 6
7 1 8
0 2 2
3 -2 3
8 2 9
1 3 3
2 3 6
3 3 1
5 3 5
6 3 2
8 3 4
5 4 9
7 7 1
0 8 -1
1 8 1
```

```
> program sudoku2.txt
```

```
ERROR
```

Random excerpt #4

note: this is not a contest problem!

Ok, so this is the end... the end of the booklet, that is...

There are no more problems after this page, just some info we had to put somewhere. You can go back to the rest of pages before. They are way cooler.

Oh! Or it can also be that you're reading this because you have finished ALL the problems and still had time to browse through these last pages to check if we left something else around here.

Well, I'm afraid we have not, but we still would like to tell you that it was great having you participating and that we hope you have enjoyed every single of the 86400 seconds of our competition, we definitely did! It took some work to prepare this contest, find a team, prepare questions, check how to submit your solutions, find a proper combination of automated and manual judging... but it was definitely worth it if you think it was worth it!

So, feel free to let us know what do you think about the contest. What did you like? What you think we should do better? All your feedback can help us prepare a better competition for the next year!

Oh! And of course, we expect to see you there too! :)

Thanks a lot for your help in making IEEEExtreme a reality!

- the IEEE Orga team (ieeextreme@ieee.org)

Table of Contents

Instructions.....	2
Submission procedure.....	3
Target machine setup.....	4
PROBLEM 1	
Cosine similarities.....	5
examples.....	6
PROBLEM 2	
Carmichael numbers.....	7
examples.....	8
Random excerpt #1.....	9
PROBLEM 3	
Incomplete communications.....	10
examples.....	11
PROBLEM 4	
Amateur musicians.....	12
examples.....	13
Random excerpt #2.....	14
PROBLEM 5	
Telephone keyboard input recognition.....	15
examples.....	16
Random excerpt #3.....	17
PROBLEM 6	
Machine load.....	18
examples.....	20
PROBLEM 7	
Arithmetic coding.....	21
examples.....	22
PROBLEM 8	
Sudoku.....	23
examples.....	25
Random excerpt #4.....	27
Disclaimer.....	29

Disclaimer

All the brands, names and registered trademarks that may appear on this document are marks (trademarks, service marks, registered trademarks, or registered service marks) of their respective owners in the USA and/or other territories.

Some of the materials contained in this document are included from articles in the Wikipedia project (<http://wikipedia.org>) or other sources covered by open licenses (like Creative Commons) and are used here for non-profit purposes.

Other legal terms may apply to this document. Before using the content in the booklet for any other purpose than participating in the 2006 edition of IEEEExtreme contest, please contact the team at ieeextreme@ieee.org to request permission.

No computers were harmed during the creation of this booklet.