PROGRAMMING CONTEST RESULT MANAGER

PCRM

Version 1.8

User's Manual

CREATED BY LENARD GUNDA

Contents

1 LICENSE

By installing and using this product, you agree that you have read, you understand and you accept the terms of this license agreement.

COPYRIGHT

The PCRM (Programming Contest Result Manager) and PCRMPOP3 programs, the included documentation and optionally – for registered users – the source code (referred to as “PCRM” from now on) is copyright (c) 2003 by Lenard Gunda

NO WARRANTY

PCRM IS PROVIDED ON AN "AS IS" BASIS: THE AUTHOR WILL NOT BE LIABLE FOR ANY KIND OF DATA LOSS, DAMAGE, LOSS OF PROFITS OR ANY OTHER KIND OF LOSS WHILE USING OR MISUSING THIS SOFTWARE.

USAGE

The program is provided to you as *postcardware*.

You can use and evaluate it for free, without any restrictions for 21 days (three weeks). After that, you must either register or discontinue using it. During the evaluation period, the program can be used for personal use only (mainly for testing). Please do not use the program for programming contests during the evaluation phase (that is not considered to be personal use).

Registering the program is very simple: what *postcardware* means is that you simply have to send me a postcard! Preferably a card with your city/town/country on it. Thanks!

By registering (with sending a postcard), you are entitled to use the program in any way for any non-commercial use. Any commercial use is prohibited!! (That is, you cannot make money with this program). You are, however, free to host programming contests with the help of the program, or to use it in education.

Registered users, if they want to, can also receive the source code.

SOURCE CODE

The source code can be used, modified, recreated in any way, given that:

1. The original copyrights remain in place. (That is, all copies of the source code must also contain the original copyrights, as do the program itself. You cannot claim any rights on the original code, except of course that you modified them, but this must also be clearly stated in the modified code.)
2. You cannot distribute the source code in any way!
3. Any product you base on the source code cannot be used for commercial purposes! You cannot sell any product based on PCRM source code.

It would also be nice if your program (if it uses PCRM) would contain a copyright message, something like this:

PCRM System copyright (C) 2003 by Lenard Gunda or if you use it as part of something else, replace *PCRM System* with *Portions*, if desired.

If you would still require a commercial license, please contact me via email. Contact information can be found in chapter 1.

DISTRIBUTION

The evaluation software package can be distributed in it’s original, unmodified form without any restrictions. You may not add or remove files from the software package, and you may not modify or alter the content in any way.

You are free to put it to your web page, BBS, include it on a CD (magazine cover, for example), but including a link to my homepage would be appreciated.

No money may be charged for the distribution of the program package, except the fee of the medium the software is distributed on.

2 INTRODUCTION

This chapter will introduce you to the Programming Contest Result Manager program and to some terms that are used in this manual. We will also discuss, what the program can be used for, even if the name may seem obvious.

2.1 What is it?

Programming Contest Result Manager or PCRM for short is a program that can be used to help judge a programming contest, where the contestants must solve problems on a computer, and then send the solution to the online jury. You could also use to judge an offline competition as well.

The PCRM program can automatically check the sent solutions – it compiles them, runs them, and checks the output they give. It can both work with programs that will produce an output file and also with programs that read and write from/to the standard input (*stdin*), standard output (*stdout*).

PCRM also includes a POP3 client program that can accept incoming solutions from the contestants via email, and this fully automating the entire judging process.

2.2 Programming contest?

What is a programming contest anyway? What kind of contests can PCRM help judge.

In a programming contest we talk about, *contestants* (which can also be teams) compete against each other. During the competition, the contestants will solve *problems*. There are several problems in a competition. For each problem, the contestants create a solution, which is a source file written in a programming language.

This source file, when compiled, generates a program, which then can read input data from either the standard input or from a file. It writes the generated output either to standard output or to a file.

When the jury receives the source file from a contestant for a problem, it compiles it with the appropriate compiler and runs the program with *test cases*. For each problem, there is one or maybe more (but at least one) test case file(s), which the program must process, and generate a correct output. An output is correct, if it is the same as a pre-generated one, or if a program that verifies outputs find that the output is correct.

When using PCRM, there are test case files for each problem. However, each physical file can of course contain more real test cases – this is the case in ACM mode, where all test cases must actually be in one file. This usually means, that the input is built in a way, that it can contain several test cases following one another.

PCRM helps you automate the judging process, by receiving the solutions (automatically), compiling them, running them, evaluating the result, and keeping a track of events and generating (or showing) the complete result.

2.2.1 Problem based

In this mode, the only measure is how many problems were solved. The individual test cases do not count, but a problem is only solved if all test cases were solved correctly.

You can have as many problems as you wish and each problem can have as many test cases as needed.

2.2.2 Score based

In this scoring mode ,the winner is determined by who solved more problems correctly. Every test case that is evaluated is equal to a score of 1 (or more, if a solution checking program is used. In this case, the solution checking program determines the score for a test case).

That is, if we have 5 problems and 3 test cases each, the maximum score is 15 and the minimum score is 0.

You can have as many problems as you wish and each problem can have as many test cases as needed.

2.2.3 ACM based

PCRM was originally created for ACM style competitions. Here, the contestant with the most problems wins. If there are two or more contestants with the same number of solutions, a score is calculated, and the contestant with the **less** score wins.

Score is calculated as follows. For each problem, the contestant gets as many scores as many minutes passed since the beginning of the competition. For every unsuccessful entry (compile error, run time error, wrong answer, etc) he also gets penalty time. But penalty is only given, if the problem is accepted in the end.

So, if I sent in problem A after 80 minutes, got a wrong answer, sent it in after 84 minutes again (I am fast at finding the error), then again at 92 minutes, which is finally accepted, I will get 92 + 2*penalty scores (assuming penalty is 20, that sums up to score of 132). If someone else sent in a correct solution after me (say at 118 minutes in the competition), that contestant still beats me.

In ACM mode, there can only be **one** test case file for each problem. This does not mean one test case, because these problems usually have a test case format that many test cases can actually be put into a single file. Because of how the score is calculated and maintained, it is necessary to have this restriction. Even if you specify mode test cases, PCRM will only work with 1.

2.3 Jury responses

In ACM contests, when the result is sent in to the jury / judges, a result is returned after some time to the contestants. Based on this results, the contestant has to decide what and how to do next.

In practical terms, this usually means that the contestant after sending in the source file, can expect a result on some webpage, which will contain the time of entry, which program he sent in and what result he got.

Here is a list and an explanation of the results that PCRM can return:

- **ACCEPTED** – of course this is what all contestants want. It means the program passed all tests and all restrictions and was accepted.
- **COMPILE ERROR –** if the compiler returns an error, this message is sent back to the contestant.
- **RUNTIME ERROR –** this means that the program has not terminated successfully. This probably means a segmentation fault, access violation, exception or something like that. Please make sure the programs return 0 at the end, because otherwise it might be treated as a runtime error. See section 2.5 for more information.
- **TIME LIMIT EXCEEDED –** for every problem, a time limit can be set. This is the maximum amount of time the program is allowed to run. When exceeded, the program will get terminated, and this error is returned.
- **MEMORY LIMIT EXCEEDED –** for every problem, a memory limit can be set. The PCRM program checks if this has been exceeded, and if it has been, then it will return this error, possibly terminating the program. Note: under Windows a check is only made at the end of a successful run (when none of the above errors occurred), because Windows provides a facility for safely determining the peak memory usage at that time. Under Linux, a more manual and error prone process is used – Windows wins out in process information requests. ☺
- **WRONG ANSWER –** the program finished and produced output, but the output is bad.

2.4 How it works?

Let me share some notes on the internal workings of how PCRM works. This will only be a short introduction.

Let’s start with PCRMPOP3. This is a simple shell, that uses POP3 protocol to get messages from a server, store them, and then uses a MIME message parser to parse the message and store the source code received in it. Then it can optionally invoke PCRM to do the checking.

PCRM uses the settings in an .ini file to determine how it works. It compiles the programs and runs them with the test files. Both the compilation and running command lines can be set for each of the languages, this way you can customize the way programs are handled, and it is also possible to use other programs to run the compiled program (as in the case of Java).

If the program read input from stdin, PCRM redirects the stdin for the program so that it receives the test case needed. Stdout can be redirected the same way, if that is where output is written.

After the program terminates successfully in the given time limit, not exceeding during the run the given memory limit, the output is compared to the pre determined outputs, and if they match, the test case is accepted. PCRM is also able to invoke a third party program, that has to do the checking of the output and inform PCRM with its exit code, whether the solution is correct or not.

2.5 Solution requirements

First of, solutions must be written in a programming language that is supported by PCRM. You can find a list of supported languages in other parts of this manual.

Second, return value. All solutions the contestants submit should be returning a 0 value at the end, including a successful run of the program. Failing to do so can make the verification code think the program has encountered a run-time error, and will display this as a result, even if the output would be correct. This limitation is because for example, under Linux, a segmentation fault like runtime error can be detected by looking at the return value, and for this reason, the return value of 0 was chosen to indicate a success.

For C and C++ languages this is very easy, because you simply return 0; at the end of the *main* function. For other languages, please consult those languages, how program return values are handled and created.

3 REQUIREMENTS

There are two versions available from PCRM

The Windows version requires Microsoft® Windows™ operating system. PCRM will run on any of the following systems: Windows 95/98/Me/NT4/2000/XP/2003Server. The program runs in console mode.

Actually, there are two version for Windows. The general version cannot measure process memory and execution time information, because of limitations in the 9x/Me series Windows operating systems. For the version that has the memory and timing measurements working, you will need NT4/2000/XP/2003Server.

Another version is available for i386 based Linux systems (libc6). Under Linux, only execution time information can be requested, detailed timing information (kernel time, user time) is not available.

Source code is available to registered users (see License, in Chapter 1). PCRM (and PCRMPOP3) was coded entirely in C++, no other libraries or 3rd party source code was used to create it.

3.1 Windows requirements

To use memory and time information and/or memory limit functionality under Windows, you will need Windows NT4/2000/XP/2003Server. For NT4 you will also need to have a redistributable component installed. This component is called PSAPI (Process Status Helper), and can be found in the redistributable components of the Platform SDK.

More information is available in the MSDN library under Process Status Helper or look at the following address:

http://www.microsoft.com/msdownload/platformsdk/sdkupdate/psdkredist.htm

3.2 Further Requirements

If you want to use PCRMPOP3, you will need access to a POP3 server. PCRMPOP3 uses the POP3 protocol to receive emails from the POP3 server. If you also wish to read emails there, please make sure your email client does not delete or move emails on the server, because in that case, PCRMPOP3 might not be able to find the emails.

If you are using a firewall, then you will have to allow PCRMPOP3 program access to the internet. It will connect to a POP3 server using TCP port 110.

4 COMPILING

If you have the source code for PCRM, you can compile it. PCRM can be compiled under Windows and under Linux as well. Linux might be slightly better for judging, because of how PCRM is able to detect segmentation faults (runtime errors) under Linux (Windows makes this a little bit difficult). See section 8.1 for details about this problem.

4.1 Windows

For Windows, the makefile is "makefile.vs". It has been tested with Microsoft Visual C++ 6.0 and also with .NET and .NET 2003 (that is, VC++ 7.0 and 7.1).

To execute, type:

```
nmake -f makefile.vs
```

This will build *pcrm.exe* and *pcrmpop3.exe*.

4.1.1 Workspace and Solution

For Microsoft Visual Studio .NET 2003 (that is, VC++ 7.1), there is also a solution file with two projects that you can use.

The solution file is: `pcrm.sln`

The project files are: `pcrm.vcproj` and `pcrmpop3.vcproj`

4.2 Linux

For Linux, the makefile is "makefile.lnx". It works with GCC.

To execute, type:

```
make -f makefile.lnx
```

This will build *pcrm* and *pcrmpop3* ELF binaries.

4.3 Other options

You can also add commands to the make. These are:

• `all`	build everything (default)
• `clean`	clean .obj/.o files, executables
• `rebuild`	clean and then rebuild

5 THE PCRM.INI

The PCRM.INI file is the most important part for PCRM. In contains all the information needed to run a competition. We will now look at all the sections and all parameters of the ini file.

5.1 global section

The *[global]* section contains generic settings for the program and the environment, that it executes in.

- *numcont* – The number of contestants in the competition. This number must be greater than 0, and does not have currently an upper limit. If you do not have individual contestants, rather teams, then this should be treated as the number of teams.
- *numprob* – The number of problems in the competition. This number must be greater than 0, and currently you cannot have more than 64 problems.
- *mode* – Program operating mode. This basically determines, how PCRM will calculate the score after evaluating the results. Currently, there are three operating modes that are available. You can read more about them – especially the third one – in section 2.2.
  - 1= good/bad
    This one simply keeps track of which problem was solved and which was not.
  - 2= score based
    Score based counts scores for all test cases, sums them up and more score is better.
  - 3= acm style
    ACM style is the standard ACM style race, with ACM style score counting. Contestant with more problems solved is better. If two contestants solve equal number of problems, then whoever has a lower score, wins. See section 2.2.3 for details on this scoring method.
- *compout* – When PCRM compiles the solutions, you can choose to see the output or you can choose to hide it (send it to limbo☺). This settings governs, what happens with the output.
  - 0= no compiler output (default)
  - 1= display compiler output
- *runout* – Display runtime output for certain programs (should be 0). This includes the program that is run, and stdout is not redirected (because the program writes output to a file). Should not be used, only for testing and debugging.
  - 0= no output (default)
  - 1= display output
- *result* – Result calculation method. For ACM style races, I recommend setting 2.
  - 0= Every time a test is requested, test the program with all of the test cases, and make a history entry for every test that was made.
  - 1= Every time a test is requested, test the program with all of the test cases, but only make 1 history entry.
  - 2= test all test cases, but stop testing on first error, and only make 1 history entry.

- *retest* – Retesting solution setting. This setting tells PCRM what to do, when a second or third or more test is requested.
  - 0= If a test case was tested and accepted, do not retest.
  - 1= Retest always.
  - 2= Retest all test cases, unless all test cases already accepted
- *pop3head* – Do we check for pop3 header. PCRMPOP3 adds a small header to all files, which contain information about the file, like contestant, problem and a time stamp. When enabling this option, PCRM will use this information, which of course makes the judging more accurate. It also verifies the integrity of files this way.
  - 0= don't check
  - 1= check
- *acmstart* – Competition start time. HHMM format
- *acmstop* – Competition end time. HHMM format
- *acmpenal* – Penalty for wrong solution. Default is 20 (ACM standard).

5.2 html section

PCRM can also generate the output in HTML format. In this case, you can fine tune the settings used to generate with the *[html]* section. Here are the options you can use:

- *frame* – In case the Jury needs to show additional messages, it is possible to insert an <iframe> between the results and the contest history. You can also set the file from which the frame content is included with other settings (see below). To turn on the frame generation, set this value to 1. If you set it to 0, then no frame will be showed between the results and the history.
- *framesrc* – Filename to show. This will be written into the <iframe>'s *src* property.
- *framewidth* – The width of the frame in pixels.
- *frameheight* – The height of the frame in pixels.
- *showlang* – If the value of this settings is 1, then the programming language a solution was submitted in will also be included in the results table.
- *destfile* – file where to generate the html result. Normally, by default, the result is written to a file named *pcrm_res.html* in the same directory, where PCRM is run. You can also specify a path here, like *../public_html/index.html* or something similar.
- *autorefresh* – if this value is set, the written HTML file will also contain the setting, so the browser will refresh the HTML automatically over a period of time. The value is given in seconds
- *gfxproblem* – if this value is 1, then instead of the problem name the little circle with letter is displayed. Only used when using *resgfx* mode in the program.
- *historynum* – adds numbers to the history list, beginning with 1. Can be used to see how many history rows there are. Set to 1 to enable or 0 to disable (default=0)
- *noresultafter* – specifies a time in HHMM format (similar to *acmstart*/*acmstop* in *global*). After this time, the global results will not be generated (this is the result file given in *destfile*).
- *judgeresults* – sets mode for another type of output generation. Set to 1 to enable and 0 to disable (default=0). If enabled, will generate more result files (see *judgefiles* option). These files contain the complete result (_judge file), as if it was generated with normal generation, and also individual files for the contestants. This option can be used with *noresultafter* so after a certain time, only judges can view the complete result and contestants can view only their own results.

- *judgefiles* – specify path and name of files for *judgeresults*. Add extension as well. When generating, _judge and _# (where # is the contestant number) will be added to filename. For example, if you specify *../public_html/judgeonly/result.htm* then the actual files created will go into that directory but will be named: *result_judge.htm, result_1.htm, result_2.htm* … Default value is: *judge_res.htm*
- *addtime* – If set to 1 (default=0), adds time information to accepted programs (how long it ran)
- *addtime2* – If set to 1 (default=0), adds detailed time information to accepted programs (how long it ran). *addtime* must also be enabled. Currently, this option will not work on Linux systems.
- *addmemory* – If set to 1 (default=0), adds memory usage information to accepted programs (how much memory it consumed).

5.3 pop3 section

The *pop3* section contains settings that are used by the PCRMPOP3 program. These are mostly email settings and options how to launch the PCRM program itself.

- *server* – POP3 Server domain name or IP address
- *user* – Username to send to server
- *pass* – Password for user name
- *checkwait* – If listen or full auto-judge mode, wait time between checks.
- *subject* – mail subject type and verification
  0=nothing special about subject line
  1=required subject line, heavy format (see Section 7.2.3)
  11=required subject line, **with** passphrase (see Section 7.2.37.2.4)
  12=required subject line, **only** passphrase (see Section 7.2.4)
- *savemail* – save all messages to pop3/ folder
  0=no saving
  1=save
- *genresult* – generate results after pop3 operation? If set to 1 or 2, then after an *auto1* or *autoall* command is executed, a result generation will also be requested from the PCRM program.
  default value: 2 (will generate GFX output)
  0=no generation
  1=execute the *reshtm* command (simple HTML file)
  2=execute the *resgfx* command (HTML file with graphics)
- *genalways* – if set to true (other than 0) then results will be generated in *autoall* mode after every run, not just ones that update the source files and run PCRM. This can be used to display the time left “constantly”.
  0=only generate results when needed
  1=generate result in every run of *autoall*
- *pcrmpath* – set where the PCRM.EXE (or PCRM for linux) executable can be found. This defaults to the current directory, but you can set it to something else, like `../debug/windows/pcrm.exe` or something like that.
- *autoend* – automatically end competition? This too can be used with *autoall* and is used to end the checking cycle as soon as the end of the competition is reached. This option also checks the *acmstop* time in the *[global]* section to see when it needs to stop. Will always generate a result according to the

genresult parameter before exiting. The option can be used to terminate the competition automatically, when time is up, and no more results will be generated. (**note:** this also exits from *listen* mode)

5.4 compilerxxx sections

The *compilerxxx* section contains compile lines, which say to the program, how to compile the solution programs. There are two compiler sections, [compilerwin] and [compilerlnx], under Windows systems, the first one is used and under Linux systems, the second one is used.

The section has entries in the form:

```
lang=compileline
```

Where *lang* can be: **[c|cpp|asm|java|pas|bas|cs]**
These are for languages (in same order):

C
C++
Assembly
Java
Pascal
Basic
C#

`compileline` contains the command line to invoke. Here, every % mark will be replaced with the source file, without extension, but with full path.

So, for example, if we have the following line:

```
cpp=g++ -wAll -o % %.cpp
```

And we want to compile *problem3/p4c19.cpp*, then % will be *problem3/p4c19* and the actual command line will be:

```
g++ -wAll -o problem3/p4c19 problem/p4c19.cpp
```

Same goes for Windows.

5.5 runxxx sections

The *runxxx* section contains run lines, which say to the program, how the compiled solution programs will be run. There are two run sections, [runwin] and [runlnx], under Windows systems, the first one is used and under Linux systems, the second one is used.

Works basically the same way as the compile lines, except this time you specify the run command. Normally, this can be a simply "%" or "./%", but some languages might require something else, like "java %".

5.6 problem? sections

Every problem has it's own [problem?] section, where the ? mark is replaced with the problem number, counting from 1. Every problem can have some test cases (at least 1).

- *numtest* – number of test cases
- *maxtime* – time limit for each test case, in milliseconds (1/1000 s)
- *maxmemory* – memory limit for the program, in Kbytes. Default value is 0, which disables the memory checking. Only available under Linux and Windows NT4/2000/XP.
- *test?type* – test case type for testcase ?
  - 1= One type of input, one type of output. Input is read from stdin, output is written to stdout
  - 2= One type of input, one type of output. Input is read from a file, output is written to a file The program gets the input filename as 1st argument when run, it can use it or not.
  - 11= One type of input, multiple types of output. Input is read from stdin, output is written to stdout
  - 12= One type of input, multiple types of output. Input is read from a file, output is written to a file. The program gets the input filename as 1st argument when run, it can use it or not.
- *test?parm1* – input filename (for types 2 and 12) for test case ?
- *test?parm2* – output filename (for types 2 and 12) for test case ?
- *test?chk* – test program name (for types 11 and 12) for test case ?

The parm1 and parm2 are the names the program will expect/get the input and has to write the output. They do *NOT* affect the files in the *test/* folder of the problem, which are always named in the same way (see Chapter 1 on setup for details).

The *chk* parameter gives the name of the check program. This has to be an executable in the root of the pcrm environment. Read section 6.2.1 for details about how to write a solution verifier program.

5.7 contestant section

This is where contestant information is stored. There are two types of entries. The first one lists the name of the contestant or team, and the second one gives a passphrase, which is used with the PCRMPOP3 program when handling email messages.

```
?=name
P?=passphrase
```

? is replaced with the contestant number, starting at 1. The first is a name entry, that stores the name for the contestant. The second stores the passphrase that is used with PCRMPOP3. The passphrase cannot contain space characters!

6 SETUP

Before usage, you need to set up the program environment. PCRM.INI is used to set up general things. Read Chapter 1 for information about settings in PCRM.INI.

6.1 Directory structure

For each problem you will have to create a separate directory, called problem?, where substitute a number to the ? mark. This is a 1 based index of the problem. The problem? directory is where the solution source code is put in (copied) and there it is automatically compiled by PCRM, and run also.

Each directory will contain a sub directory called 'test', which will contain test?.in and test?.out files, for each of the test cases. Here, ? is replaced with the 1 based index of the test case. If you use a solution verifier program, the output solution file is not used.

You will also need one global directory if you intend to use the pop3 client. This is called 'contestant' and contains numbered directories for each of the contestants. These numbers are also 1 based. As PCRMPOP3 receives solutions, they are stored for each contestant in the corresponding folder.

If you wish to save the incoming emails (they are never deleted from the server either), you will also need a pop3/ folder. All emails with timestamp and count number are stored here, in their original form.

So for example, you might create the following structure for a 6 problem, 12 contestant competition:

```
pcrm/
 + contestant/
 |  + 1/
 |  + 2/
 |  + 3/
 |  ...
 |  + 12/
 + pop3/
 + problem1/
 |  + test/
 + problem2/
 |  + test/
 ...
 + problem6/
   + test/
```

If you have set up all parameters in PCRM.INI you can run the "pcrm dirctr" command. This will create all the directories needed, and you will only need to put in the proper files into the test directories after this.

6.2 Problem setup

Each problem has it's own folder. Each of these folders contain a test/ folder. This is where the test case inputs and outputs go. In PCRM.INI you have already specified how many test cases a problem has. In the test/ folder, you must place

```
  test?.in
  test?.out
```

files. These are the sample input and output files for the test case. They are fed to the contestant problems (.in) and the programs must produce the corresponding output (.out).

Before running a program, PCRM will copy these files, so they will not get lost during testing.

If you use a result verifier program, the output is not used.

6.2.1 Result verifier program

For some problems, there might be multiple solutions. In this case, you will also have to provide a solution verifier program, because the built in verification cannot handle multiple output verification.

The verification program must be an executable, that the PCRM program is able to execute (have enough rights to do so). It has to be placed in the PCRM root folder, and when PCRM runs it, it will get 3 parameters:

```
executable <testcasenumber> <testcaseinput> <programoutput>
```

The first argument is the test case number. This is simply a 1 based number of the test case, that needs verification.

The second argument is the relative path to the input file the user program was called with. This is the input file for the given test case.

The third argument is the relative path to the file, that contains the output generated by the user program, and the verification program must decide if the information in this file is correct for this given test case.

The verifier program should return a number between 0 and 127, indicating the score. 0 is a wrong answer. If you have score based rating, use any number. If you have ACM style or good/bad style rating, you should use 1 to indicate a success and 0 for failure.

6.3 Source code names

The problem directories must also contain the source code files that are created by the contestants. Each source code must be named as:

```
   p?c?.ext
```

Here, after *p* comes the 1 based index of the problem, after *c* comes the 1 based index of the contestant, and *ext* is replaced with the source code extension (c, cpp, cs, java, bas, pas, asm). A valid names looks like this for example:

```
p6c17.cs
```

This would the solution of contestant (or team) number 17 for problem 6, and is written in C#.

If you use PCRMPOP3, it will automatically copy the corresponding source file with the correct name into the correct problem folder, you don't need to worry about this. However, for some emails, it is important that the sources are named like this by the contestants themselves. See section 7.2 for details.

7 USAGE

This chapter details the usage of the program.

7.1 PCRM

Just type *pcrm.exe* or *./pcrm* to start the program. It will give you a brief look at how to use it, what options and commands are available. Before actually using the program, you must set up your environment by modifying the PCRM.INI file in the same folder. This manual contains all the configuration options and their description.

PCRM.INI must be modified for every competition you intent to run, because it contains such information, as number of contestants, number of problems, etc. It can be used to fine tune the workings of the program.

7.1.1 Program command line options

To run PCRM, you must type a command line:

```
    pcrm <options> <command> [<command arguments>]
```

Options are:

`-q`
Queiet mode. No output to screen

`-w`
No logo to screen.

`-l`
Log output to file (pcrm.log)

`-t`
Timestamp all entries printed to either screen or log

This means that "pcrm -q -l" will only log to a file.

7.1.2 Program commands

The following are the commands that PCRM accepts. You can also find the arguments needed by the commands here. If you start PCRM without arguments, you will get a listing of all commands and their syntax.

`ver`
Print version information

`ppl`
Print contestant list

`dirctr`
Create directory structure based on pcrm.ini

`chk <problem> <contestant>`
This command will check one problem for one contestant.

`chkcon <contestant>`
Verify all problems, one contestant

`chkprb <problem>`
Verify one problem, all contestnats

`chkall`
Verify all problems, all contestants

`clr`
Clear all resulsts

`clrtst`
Clear all testing results

`res`
Print results

`rescon <contestant>`
Single contestant results

`reshtm`
HTML output

`resgfx`
HTML output, with GFX support (needs the directory called `htm/`You can find more information about the HTML output in the PCRM.INI Chapter.)

7.1.3 Result generation

PCRM can generate different results. At the simplest, you can have a text only output to console (screen). You can also have HTML results, with various levels. You can have plain text only HTML and graphical HTML.

Please check the section 5.2 for details on the various options that are available to customize the output the program produces. There are many options that you can use to fine tune the settings.

The program is now also able to generate judge results. These results are intended only for judges to be seen, because now the main result generation can be halted after a time. On ACM contests, it is a custom not to show the complete results to contestants after some time in the competition (for example, in the last hour).

For this reason, the judging results also generates separate result files for all the contestants, which only contain data for that particular contestant.

7.2 PCRMPOP3

PCRMPOP3 can monitor incoming emails, and is able to fully automate the judging process. In can monitor incoming email, save problems as they arrive and launch PCRM to check these problems as they arrive, and also to automatically regenerate the output html file. This simply means you just start it, and you don't need to worry about anything else, just sit back and relax until the contest ends. ☺

For the letters received, there is no restriction on the email address that is used. The letter must always contain **one and only one** attachment, which is a source code file. If the letter is not like this, PCRMPOP3 simply ignores it. Including the source code in the body is not good, the file **has to be attached** to the email. Also, the source code file usually has to be named strictly as required by PCRM! See section 6.3 for details on source code names.

If you use heavy subject lines for emails (see section 5.3 for details how to set the subject line), then the attached file can have any name the contestants desire, but the extension has to be the one for the programming language the file is in. (so for example, if you code in C++, the extension has to be cxx or cpp). If you don’t use subject lines or use light subject lines, then the attached file has to be named the way section 6.3 requires it!

The source file must be attached! It must not be embedded in the letter body. If it would be, it would not be recognized by PCRMPOP3. PCRMPOP3 can also handle base64 encoded attachments.

7.2.1 Program command line options

Similar to the PCRM program, the POP3 program can also accept some command line options before the command itself.

To run PCRMPOP3, you must type a command line:

```
    pcrmpop3 <options> <command>
```

Options are:

`-w`
No logo to screen.

`-l`
Log output to file (pcrmpop3.log)

`-t`
Timestamp all entries printed to either screen or log

This means that "pcrmpop3 -l" will also log to a file.

Note, that these settings, with the exception of the *–t* argument **do not** affect the PCRM program, which might be invoked from the PCRMPOP3 program!

7.2.2 Program commands

You can specify the following commands in the command line. If you run loop commands (*listen, autoall*), the normal way to exit the program is by using Ctrl-C (under Windows, you can also use Ctrl-Break).

`check`
Run one email check.

`listen`
Run in a loop and check incoming emails periodically.

`auto1`
Simple auto judging. Checks mails and then runs a check on any of the solutions that were downloaded (if any arrived). Also generates results into HTML if requested.

`autoall`
Full auto judging. This will run in a loop a mail checking, and if any new solutions arrive, will also start the PCRM program to check those solutions. After checking, if not set otherwise, will also generate an output (with the *resgfx* command of PCRM – for output generation details see the *[html]* configuration section).

7.2.3 Heavy subject lines

If you choose to use heavy subject lines, all incoming emails must contain a subject line in the following format:

```
    PCRM:<contestantnumber>:<problemnumber>[:<passphrase>]
```

Heavy subject line is used for subject setting 1 and 11. In the case of 11, the passphrase must be specified, in case of 1, it is not needed. In both cases, the subject line must be in this format. There can still be only one attachment, however, the name does not matter, because the subject line is used to identify the contestant and problem. The language is, however, determined based on the extension for the attachment.

7.2.4 Light subject lines

A light subject line is used when subject is set to 12. In this case, the subject line should only contain the passphrase for the player.

Passphrase is given in PCRM.INI to every contestant. No spaces.

8 ADDITIONAL NOTES

Here are some notes you might find interesting, important.

8.1 *Windows version*

When the program terminated abnormally (exception), Windows pops up a stupid dialog. The person who checks the problems must manually close the dialog. Even before this is done, PCRM can detect a timeout for the program and if the dialog is not closed, it will prevent further proper functions. This is a limitation in Windows. If you can and want to, turn off the exception notification window.

The problem here is, that the PCRM program when it runs another process, cannot detect that there was an error until that notification window was closed. However, it does detect a timeout, while the window is up, and so instead of a program error, a timeout error is detected. Also, because by not properly closing the application (until that dialog is up), further testing might also suffer. Currently, I have found no workaround for this problem (preventing the dialog from ever appearing).

The Linux version is not affected by this, it runs just fine there (segmentation faults, run time errors are detected correctly).

8.2 *Hardcoded entries*

Compile time is limited to 30 seconds. No program can compile longer than 30 seconds.

If there are multiple solutions and a solution checker is used, the solution checker cannot run longer than 30 seconds.

These values are #define-d in the source code.

8.3 *Result verification*

After the program has run, PCRM will compare the output from the program with the provided output file. Comparing the files is done line by line. Every line, all trailing spaces are ignored. The two files must contain exactly the same information. The result file of the contestant is allowed to have **ONE** extra empty line at the end.

Note, that \r\n and \n as line termination is treated equal.

We speak of files, because the output always goes to a file, if the program has to write to stdout, PCRM handles the standard handle redirection itself.

8.4 Web interface

PCRM doesn’t provide a web interface – one where contestants log in, and then upload their results through HTTP in a browser. However, it would be fairly easy to build upon the current provided options and create a web interface.

One would only need to handle the file upload and then send the uploaded file to a mail account, that can be monitored with PCRMPOP3. For example, PHP provides easy to use interface for sending emails (in fact, PHP could be used to create a web application that provides a front-end to PCRMPOP3).

9 CONTACT, FEEDBACK, SUPPORT

You can contact me in many ways.

Postcards should be sent to the following address :-)

Lenard Gunda
Mikszath K. u. 61
H-4032 Debrecen
Hungary

For registrations, please include you email address. If you do not wish to write your email on a registration postcard, either send it in an envelope or simply include your name and send me an email about your registration. :-)

Contact by email: frenzy@frenzy.hu

You can find the latest version of the product on my home page: http://www.frenzy.hu/

You are free to report any ideas, bug reports or wishes you have in connection with this program.

9.1 Acknowledgement

I would like to thank *Márk Kósa* and *János Pánovics* from the University of Debrecen, who both provided an oppurtunity to test the program with real programming contests, and who also provided ideas, suggestions and feedback during the making of the program.

The program for graphical HTML output uses a set of images I do not know who has done originally, but nevertheless I think I needed to mention that those graphics are not by me.

10 HISTORY

This chapter lists the changes for the different versions. Only the first public version is listed here.

The following symbols are used with the history:

+ new feature
- bug fixed
! improved feature

10.1 History milestones

Some notes on how PCRM was created. The idea, that we should automate programming contest has been around for a long time at our University. In 2002, we held a programming contest with ACM style scoring, but for individuals. A friend of mine – also organizing - wrote a small shell script that was used to do some minor automation, but at that time, we still had to save messages from the emails by hand, copy them to the proper location, and do partially manual evaluation of the results.

One year later, we had the same contest organized, this time we already had a PCRM version available, but it was full of bugs (well, actually the PCRMPOP3 was very buggy, the PCRM itself worked fine), but despite the problems (PCRMPOP3 seg faulted at about every 5th email ☺), the result verification was slightly better.

In fall 2003, I myself could not watch the program perform the organization of the local ACM contest, because I myself also took part in the competition, but this time, apart from some very small bugs, the whole process was successfully automated. (The version used in this competition was 1.7b).

Version 1.8 (which is current as I write this) contains some improvements and bug fixes found in the last contest it was used it in.

10.2 Version 1.7b

First public version

10.3 Version 1.8

- Fixed bug: in the ini file reader a bug could cause an exception (segmentation fault)
- Fixed bug, where the *genresult* parameter was not defaulting to the value it was supposed to.
- Fixed bug: The *gfxproblem* option worked bad, and printed wrong images into the problem solved column.
- Fixed bug: result comparison could result in an exception (segmentation fault)

! Rewritten exception handling in most of the code.
! Graphical html output now has a new separator line that is used (included as htm/line.gif), which makes the format more nice.

+ Added new option: *genalways* (see section 5.3)
+ Added new option: *pcrmpath* (see section 5.3)
+ Added new option: *autoend* (see section 5.3)
+ Added new option: *historynum* (see section 5.2)
+ Added new option: *noresultafter* (see section 5.2)
+ Added new options: *judgeresults* and *judgefiles* (see section 5.2)
+ Added new options: *addtime*, *addtime2*, *addmemory* (see section 5.2)
+ Added new option: *maxmemory* (see section 5.6) – Memory limit testing added to program, it can now monitor the memory usage of the executed program. A new error type has been introduced: memory limit exceeded.