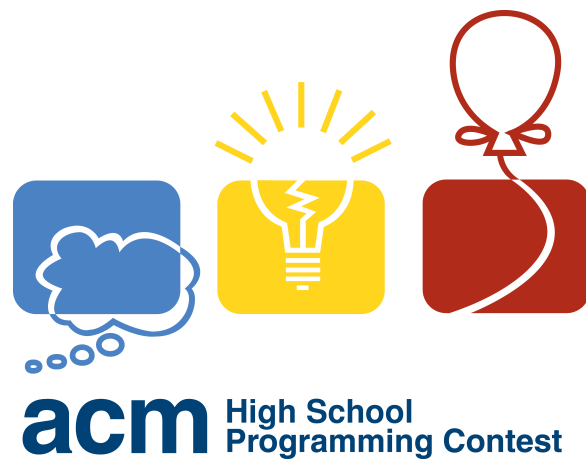


High School Programming Contest Guide



Compiled by ACM @ UVa

February 23, 2018

Contents

1	Introduction	7
1.1	Licensing	7
2	Preparation	9
2.1	Early Planning	9
2.2	Schedule	12
3	Problem Creation	15
3.1	Background	15
3.2	Problem Difficulty	15
3.3	Development	16
3.4	Input and output considerations	16
3.5	Theming	17
3.6	Validators	17
4	Configuration	19
4.1	System Configuration	19
4.2	Submission systems	19
5	Preparation Todo Lists	23
5.1	Previous semester	23
5.2	2 months out...	23
5.3	1 month out...	23
5.4	Week-of checklist	23
5.5	Day before checklist	24
5.6	Day of checklist	24

6 Contest Logistics	25
6.1 Other notes to file somewhere...	25
6.2 Roles	25
6.2.1 Head Judge	25
6.2.2 Contest Judge	26
6.2.3 Site Director	26
6.2.4 Balloon Manager	27
6.2.5 Food Tzar	27
6.2.6 System Administrator	28
6.2.7 Coach Liason	28
6.2.8 Volunteer Coordinator	28
6.2.9 Contest Proctor	29
6.2.10 Tshirt Tzar	29
6.2.11 Registrar	29
6.3 Day of the Contest	30
6.3.1 Registration	30
6.3.2 Opening ceremony	30
6.3.3 Practice contest	30
6.3.4 Lunch	31
6.3.5 Actual contest	31
6.3.6 Closing Ceremony	31
7 Finances	33
A VirtualBox image configuration	35
A.1 Introduction	35
A.2 Basic installation	36
A.3 Development installation	37
A.4 Programming Contest configuration sections	38
A.5 Image completion	40
B PC² configuration	43
B.0.1 Configuration files	43
B.0.2 Pre-configuring PC ²	43

B.0.3	pc2server	44
B.0.4	pc2admin	45
B.0.5	pc2team	46
B.0.6	pc2judge	46
B.0.7	pc2board	47
C	Sample Emails	49
C.1	Open Registration	49
C.2	Notification two weeks prior to the contest	49
C.3	Media release email request	50

Chapter 1

Introduction

Put intro here...

This guide is the result of efforts by a number of individuals at the University of Virginia. We have hosted high school programming contests since 2011, and started collecting our notes shortly afterward. This document evolved over the next few years, and was written in this for for the “Organizing a High School Programming Contest” workshop at SIGCSE 2018¹. The latest version is available online at <http://aaronbloomfield.github.io/hspc>, which is sourced from a Github repository at <https://github.com/aaronbloomfield/hspc>.

1.1 Licensing

This document, and the rest of the materials contained therein – except for the logo – are released under a Creative Commons Attribution-ShareAlike 4.0 International License² (CC BY-SA).

The following HSPC logo is used throughout this document:



This logo was derived, with permission, from the main ICPC logo, which can be found online at the ICPC website³. The logo is owned by the ICPC Foundation, and they have given permission for high school contests to use the system if...

¹<https://sigcse2018.sigcse.org/attendees/workshops.html>

²<https://creativecommons.org/licenses/by-sa/4.0/>

³<https://icpc.baylor.edu/>

Chapter 2

Preparation

2.1 Early Planning

All of these should be done around 6 months before the date of the contest itself.

Who is going to run the contest?

Determining who is going to run the contest is the very first step. Different institutions use different models. At UVa, it is primarily run by the students in our local ACM chapter, with assistance from a faculty advisor. The University of Maryland, which has run a very popular contest every year since at least 1998, has much more of the organization run by the department (faculty and/or staff). Obtaining administrative support from your department for various tasks will be quite helpful.

For the contest itself, there are two primary roles that need to be defined. One is the Contest Director, who is the person coordinating the majority of the contest. The Head Judge (or Chief Judge) is the individual responsible for coordinating the problem generation, and managing the judging at the contest itself.

What computers will be used?

Different contests will use different computer setups. Part of this step will be to decide the team size (see below).

We have used computer labs at UVa for our contests. This allowed students to have a full sized monitor, as well as a keyboard and a mouse. We used a VirtualBox environment (described later in this document) for their environment. The drawback to this model is that the capacity is dependent on the number of computers and computer labs – and if they are spread throughout your institution’s campus, then there is more travel time to the contest sites. At UVa, we used 5 labs that can hold 49 people each (as per the fire marshal’s limits) – however, one of them only has 7 usable computers due to the lab layout. This allowed a total of 55 teams (12 in the four larger labs, 7 in the smaller one), which was our registration cap. Many college-level ICPC contests use this method.

Many schools will use laptops for each team – the University of Maryland has used this very successfully. The challenge to this is where to securely store the laptops between contests. Purchase and/or

support of the laptops is another support issue. The laptop screens are typically smaller than a monitor screen, which makes it more difficult for a four person team to use it effectively. Advantages are that one can still load up a custom environment on the machines, which we found very necessary (see later in this document for details). Furthermore, there are many more lecture rooms that can accommodate students with laptops, as one does not require a lab with pre-installed computers. One can supply keyboards, monitors, and mice along with the laptops, although this will add to both the cost and storage issue.

One could require the students to bring their own computers. This is easiest to support, but it is difficult to enforce a custom environment. A very common aspect of the contests is to eliminate Internet access (except to the submission server) during the contest itself. We do this through the setup on our VirtualBox image. This is more difficult when the machines are not managed by the host institution. We realize that one can exit a VirtualBox image on any machine, but this has not yet been a problem. One could distribute a VirtualBox image, and require each participating team to have VirtualBox installed prior to the contest.

Lastly, one can adopt a hybrid approach. For UVa's 2018 HSPC, we have 70-80 teams that are interested, and only 55 spots in the computer labs. We are currently investigating using UVa supplied laptops in a lecture hall to allow more participants to participate. In future years (2019 and beyond), we are hoping to obtain funding to buy a set of laptops (along with monitors, keyboards, and mice) specifically for this purpose.

Understand the financial costs

The contest will cost some amount of money: food, prizes, t-shirts, etc., all will have some expense. Some of these can be lowered or eliminated (no t-shirts, for example), or paid for by other entities (the school or department). A 50 team contest at UVa typically costs around \$7,000 to run – the budget section is included later in this document.

Team registrations are one way to pay for this. However, having too high of a registration cost will deter schools from participating. For our contests, that would mean about \$150 per team to register. High schools often do not have the budget to spend \$450 to send three teams. Thus, we sent the cost at \$40 per team, as this seemed like an affordable expense. The registration cost from the teams only paid for the t-shirts. This meant, however, that we had to raise about \$5,000 to allow the contest to run.

Set reasonable expectations

Our first contest, in 2011, had a total of three teams. It was a huge amount of work, as we had never done this before. But the contest only grew in size, and we now have to turn teams away, as we typically reach our room capacity (ways to mitigate that are explained in this document). However, we all had a great time with the contest, and we had more teams in successive years (17 in 2012, 34 in 2013, and we hit our cap of 50 teams in 2014 and every year since).

Generate a list of all high schools to be invited

As the contest becomes more well known, this becomes easier, as the schools will find you. However, when starting the contest from scratch, one will need to generate a list of emails to be sent.

We defined the target area that we were expecting to attract students – since we are located in central

Virginia, we defined the entire state of Virginia as that target area. For a high school running such a contest, it may be that the high schools in the local area are the extent of the target area.

We looked up each high school therein, and found an email address for each school. We attempted to get the email address for a computer science teacher at the school – often this was not available online, so one could call the high school to ask who the computer science teacher was. If there was no obvious contact, we would ask for the best person to send the information to. The task of finding was split among multiple student volunteers, as there are over 300 high schools in Virginia.

Another option would be to leverage affordable services from Amazon Mechanical Turk, although we did not pursue this option.

We kept this information, as we used the list in future years. Creating an appropriate HSPC announcement mailing list for your institution is another possibility. Often we would receive a response that they are forwarding it to a more appropriate person, so we updated the list appropriately.

Decide if it is going to piggyback on another event

One option is to have it be the same date as an on-campus event – for many years, we did this with the Engineering Open House (a day where accepted or interested students can take a tour of our Engineering school and facilities). This has the benefit of bringing in many high school students interested in computing to the open house. They are typically coming from far away, and are often interested in attending both events. The disadvantage is that scheduling rooms is very difficult, as all the other participants in the other event will want the same rooms. As our contest grew in size, and we needed more rooms to accommodate the participants, we ran into too many scheduling problems when it was the same date, and we have since moved it to a different weekend. However, this worked very well when our contest was small.

Pick a date

If the date is not decided upon by the above step, then this is far more difficult than it sounds. It can't be too close to the AP exams. The date needs to avoid other contests, such as robotics tournaments (as many students are active in both robotics and programming contests). It will need to avoid high school spring breaks and prom season. In order to recruit a sufficient number of volunteers to help, we had to avoid UVa's spring break (both the start weekend and the end weekend as well).

Because UVa is active in the collegiate level ICPC, we would hold our HSPC in the spring (in March or April); other schools hold theirs throughout the year. Some years there were few options due to the constraints above. After our participation rate increased, we would email the coaches from last year a request to fill out a quick Google Forms survey as to what weekends worked the best. For us, most high schools seemed to be on a somewhat similar schedule, so this greatly aided us to find a date.

Reserve rooms

This is critical, as you can't have a programming contest if you don't have the computer labs, auditorium, etc. Make sure that, if it is on the date of another event, that those from the other event will not try to take your rooms away from you. We were always sure to reserve our rooms prior to opening up the

registration, for obvious reasons. On some years, other events would suddenly “need” those rooms on that date, but as they did not reserve them ahead of time, we were still able to use them.

Start generating problems

Good quality programming contest problems take many, many, MANY iterations to get right. Getting the difficulty level correct is the biggest challenge. people generate multiple problems, and then let the group pare down the selection from there. See the Problem Generation section, below.

Start advertising

Email all the coaches, post on social media, and on the departmental website. Emailing the coaches from previous years helped as well. We would generate a Twitter hash tag (we would use #uvahspc2017), and promote it there. Be sure to state the registration cost when advertising.

Open up registration

We discuss online registration systems elsewhere. A custom system works great, but can be a chore to maintain. One can also use various Google Forms to handle registration. The first one would be for a school to state how many teams. Other forms can follow (team members, food allergy information, etc.). Be sure to include payment information (amount and how to pay) along with the registration.

2.2 Schedule

It will be easiest on the participating schools if the schedule is known during registration, as the schools will have to arrange transportation. Many of our schools would drive from 2 hours away (the time taken to drive from northern Virginia to Charlottesville), so we could not start the contest too early.

Here is a typical schedule:

- 8:30 a.m. – 9:30 a.m.: Registration (with some type of breakfast)
- 9:30 a.m. – 10:30 a.m.: Opening talk and rules
- 10:30 a.m. – 11:30 p.m.: Practice contest (includes time to travel to and from)
- 11:30 p.m. – 12:30 p.m.: Lunch
- 12:30 p.m. – 4:30 p.m.: Actual contest
- 4:30 p.m. – 5:00 p.m.: Close-down time (travel to the closing talk, etc.)
- 5:00 p.m. – 5:30 p.m.: Closing talk

A number of things to keep in mind:

- We have often tweaked the schedule after registration, but we have never modified the starting time from what was first stated during registration, as that would have required a change to the schools’ transportation plans.

- The contest length itself is the most important aspect, as that will determine the overall schedule. We have found a four hour contest to be about right for high school students, but different contests will have different lengths.
- There will be time needed for the participants to move from one location to the other, so that should be considered when determining the schedule.

Chapter 3

Problem Creation

Problem generation has been one of our biggest challenges. From a collegiate perspective – both faculty and students – it is difficult to understand the level of knowledge of high school students. If the problems are too hard, then some teams will not solve any, which can be dispiriting. If they are too easy, then some teams will finish early, which is less than ideal.

3.1 Background

Include info here from Blythe’s presentation about what high school students know (based on their classes).

3.2 Problem Difficulty

Include info here from Blythe’s presentation about the beginner / experienced / advanced difficulty levels, and how to have a good “slope” of problems.

In 2012, there were probably too many easy problems. Consider having a few more medium problems and a few less easy ones. Also, the contestants found problems involving numbers much easier than anything that involved string parsing, so consider making the easiest few problems involve simple math operations.

In the 2011 contest, there were eight problems that spanned the easier half of ICPC regional problems. These problems were almost universally viewed as too difficult. The 2012 featured ten problems, containing 5 problems on par with the easiest problem at an ICPC regional, two medium-level problems, and three hard problems. The medium problems were still probably on the easier side by ICPC standards; the hard problems were much more typical ICPC problems. These problems were much better received, although there were a few complaints about the jump between the easy and medium problems and that there were not enough problems that required the teams to think.

3.3 Development

Our problem development would start early – ideally 6 months prior to the contest, although we did not always achieve this. The problems progressed through many different stages of development. The first step was general brainstorming; a one or two sentence description of a potential problem, categorized by difficulty and problem type. About four times the necessary problems should be created using this method, so that culling can occur very easily. It is very important not to get attached to a particular problem at this point; if the problem is confusing or overly difficult it should not end up in the final problem set.

From there, we selected roughly 15 problems to investigate further and write up full problem descriptions for them. One very useful thing that was done in the past was ensuring that for the first few iterations, only a few people had read each problem. This ensured that at the later stages, a fresh pair of eyes looked at the problems without bias and inherent knowledge of the problems. At this point, there should be a pretty clear indication of which problems are worth pursuing.

Next was to assemble the packet of problems and proofreading the packet. Again, try to get a new set of eyes to look over each problem. Try to get at least three distinct solutions to each problem; one would be surprised how many bugs appear when different people write solutions. Specifically, each problem should be solved at least once in each contest language. A few times when we did not do this, we ran into problems: in 2012, we could that C++ handles floats differently relative to the other languages (even though we had a validator for that problem!), and in 2014, we found that C++ handled longs differently relative to the other languages. Consider having your solution throw an Exception when illegal input is passed in (meaning, a negative number is passed when only positive numbers are allowed). This will help detect errors in the input generation code.

Problems should be 100% complete more than a week before the contest. There are far too many other things to do right before the contest to be worrying about the specifics of the problems.

3.4 Input and output considerations

Program input should be as easy to parse as possible, as this will allow the students to focus on the problem solving, and not input parsing. Similarly for the output format.

Any problem will be tested against multiple test cases. Some contest submission systems will have all the test cases in one file, and the program will run once, and then determine the out for all of the test cases. PC² traditionally uses this format. Other submission systems will have many different input files, one test case each, and the program will be run multiple times – once for each test case. Kattis uses this format, and PC² has started using it recently as well. Which system you use will determine some of the input and output specifications.

Older ICPC problems would read in test cases until the end of input – however, finding the end of the input stream always took a bit longer to code, and we have found that to be more difficult for some of the beginner high school students. Many recent problems will always start with a single integer on the first line of the input file which is the number of test cases, and then each test case will have a specific format. If there are multiple “things” that a test case has (such as multiple nodes in a graph path problem), then having a single integer preceding the list of nodes will help input parsing.

Whitespace in output can be a problem. If the output requires a list of integers on one line, space

separated, then is a space at the end of the line allowed? Having said space is easier to implement (in the `for` loop that prints the output, just print out the value followed by the space) rather than not allowing it (in the `for` loop that prints the output, one has to put a test to see if it is the last value to be printed on the line). Having very strict requirements for the output format will disadvantage less experienced teams, as they will not be as familiar with these details. Submission systems can ignore whitespace, although some of them will ignore *all* whitespace on a line, which may not be desired (as “1 2 3” will then be the same as “123”). Often the required output format can be changed to require one value per line, which is easier to both output and to verify.

Input that can be easily read in without dealing with whitespace makes it much easier to parse. Students will typically use an input means that ignores whitespace (`Scanner` in Java, `cin` in C++, etc.). An input file that is just a series of numbers will be easiest to handle, as the whitespace between them will be ignored by these routines. Reading in a string that can contain whitespace will be very difficult – but a string that has no whitespace is also easy for them to handle.

Problems with multiple correct answers will require a validator program; these are described below.

We would often create an input generator that would create a large input file. This should test most corner cases, but feel free to add in additional cases as well. We would add to this certain cases (corner cases, empty cases, etc.) to generate the full judging input. Ensure that the judging data takes no more than 20 seconds to run on any of the solutions, as this gives a reasonable timeout level to test against.

Having all of the problems use the same input and output specification (having “Case x:” before each solution, etc.) was exceptionally helpful. It allowed teams to figure out how to specify output in the practice contest and then use what they learned in the actual contest. It also allowed us to be more consistent when generating the sample and judging output.

3.5 Theming

Starting in 2014, we started theming our problems – we would rephrase the “plot” of the problems so that they all followed a theme. This certainly wasn’t necessary, but we felt it made the contest more enjoyable. The themes we have used:

- 2014: Lord of the Rings
- 2015: Avengers (the movie had recently come out)
- 2016: Breakfast
- 2017: Classic Nintendo Games
- 2018: Pixar movies

For more difficult problems, we found it was easier to generate the problem first, and then apply a theme to it. For the less difficult problems, we found it easier to pick a theme (Pixar movie, breakfast food, etc.) and then create a simple problem based on that idea.

3.6 Validators

A validator is necessary when a problem has more than one correct answer.

One example of this is a problem whose answer is a floating point number – rounding errors or differing precision can cause values to vary slightly in the less significant digits.

Some contests will require the answers be a fixed precision. For high school contests, we feel that this causes them to focus a lot of their time and effort on the exact formatting of the output, rather than the problem solving. There are also rounding differences between the programming languages, which will cause a language-dependent difference. As an aside, did you know that there are five different standards for how to round numbers¹?

Other examples include finding a path through a maze or other graph (as there may be multiple paths), or finding the order to do a set of tasks (some items can be done in either order).

If there is only one correct answer, then the correctness of the solution can be checked via file comparison with the correct solution. The submission systems, discussed later, have the ability to ignore whitespace when performing said comparison.

If a problem has multiple correct answers, then a *validator* – a program that reads in the solution and returns whether it is a correct solution or not – will have to be written for that problem.

We do not recommend using validators! At least not until one is familiar with the contest submission system. It is often the case that a problem can be rephrased so that there is only one correct solution. A validator takes a fair amount of time to write and test, especially during the busy time of contest preparation. If the validator does not work properly during the contest, then the contest judges will have to manually judge each answer.

Recent contests at the collegiate ICPC level, have generally avoided problems that require validators. UVA's HSPC has generally avoided them as well – we tried using a few in 2012, and didn't like the experience, and have avoided it ever since.

The format for custom validators will vary based on the contest submission system used.

The high-level overview of validators is that the program will read in the problem input and the submission solution that was generated. There are two methods for writing effective validators; one is having essentially a solution embedded within the validator which calculates the result and then compares it with the returned result. This is useful for floating point results, as one can compare within a certain accuracy (say, 10^{-5}). The other is that the validator will have to trace through the provided solution to see if it really does solve the problem. This is necessary for solutions that can have different paths.

¹https://en.wikipedia.org/wiki/IEEE_754#Rounding_rules

Chapter 4

Configuration

4.1 System Configuration

We have tried using multiple configurations for our contests. Initially our system support staff allowed us to customize the environment in the labs. As the number of teams grew, and we were no longer able to fit in one lab, this became impractical. Next we tried using USB keys to boot to a customized version of Linux for a few years, but found the USB key reliability – as well as speed – to be difficult to make that work. Furthermore, replicating the keys took a lot of time.

As of the 2015 contest, we are now using a VirtualBox image. The instructions for setting up this image can now be found in Appendix A: VirtualBox image configuration. They are based on the directions that can be found online¹.

4.2 Submission systems

There are multiple submission systems available. We have used PC², as one can easily download it and install it on an institution’s own servers. We discuss three submission systems here.

PC²

PC² stands for Program Contest Control, and is a widely used system. It is written in Java, and is still being updated. It can be downloaded for free online². Its usage is described in detail in Appendix B: PC² configuration.

Pros:

- This system is free to use, and can be run on one’s own servers.
- It is widely used, and has a lot of documentation available.
- This usually uses the input format of all test cases in one input file (see 3.4). If a problem is online in the other format (one test case per input file), it can easily be converted to all be in one file (just put a integer number at the top with the number of test cases therein).

¹<http://aaronbloomfield.github.io/slp/docs/virtualbox-image-details.html>

²<https://pc2.ecs.csus.edu/>

Cons:

- The system is often buggy, and we had the entire judging system be so buggy that it was unusable because of this. Other features (such as problem copy) do not work, but this is not always evident until much later.
- It is not open source, which makes it very difficult to figure out what is going wrong, much less to fix it.

Warnings:

- There is a separate connector program available, which allows for the submission (and judging) to happen through a web-based interface. However, this connector has some **SERIOUS** issues – it forks off a thread for *each* connection (team, judge, etc.). However, most apache2 installations have a default limit of 50 threads, which can easily be reached by the connector. The mid-Atlantic ICPC regional contest in 2016 was a disaster because of this – with over 200 connections, nobody was able to connect for hours after the contest started.

KATTIS

KATTIS³ is an online system originally developed by KTH Royal Institute of Technology in Stockholm⁴. It is now its own entity. It is described in detail at ...

Pros:

- This is, by far, the most refined system available. It has been used in multiple ICPC world finals for this reason.

Cons:

- One cannot run this on one's own servers, which means that one will have to provide an open Internet connection to the kattis servers – which also means that contestants can view other problems and solutions.
- Because one cannot run this on one's own servers, one will have to get KATTIS to setup and configure the contest. It is unclear if they would be willing or able to do this for high school contests.
- This usually uses the input format of one test case per input file (see 3.4). If a problem is online in the other format (all test cases in one input file), it is more difficult to convert in this direction.

Warnings:

- As mentioned above, one has to get the assistance of the KATTIS staff to setup and configure a contest.

³<https://open.kattis.com/>

⁴<https://www.kth.se/en>

DOMjudge

DOMjudge⁵ is a free and open source web-based submission system. Its usage is described in detail in ...

Pros:

- This system is free to use, and can be run on one's own servers.
- It is open source, so modifications or bug fixes can be contributed by many individuals

Cons:

- ...

Warnings:

- ...

⁵<https://www.domjudge.org/>

Chapter 5

Preparation Todo Lists

5.1 Previous semester

- Reserve rooms, set date, open registration

5.2 2 months out...

- Get t-shirt artwork

5.3 1 month out...

- Photo release form for minors...
- Close registration
- Order UVa catering lunch
- Order t-shirts
- Order coach gifts (mugs)
- Order prizes, certificate paper, name badges, holders, and lanyards, balloons (if needed), balloon ribbon
- Arrange for the media to show up! Contacts:
 - Derek Quizon (dquizon@dailyprogress.com) at the Daily Progress
 - ...

5.4 Week-of checklist

- Acquire coach gifts and t-shirts
- Communicate with admissions, if they are going to show up
- PC² configuration; also plan for backup servers if the main PC² server dies.
- Final email(s) to coaches: parking information, finalize the team member names (and team names!), get t-shirt sizes for chaperones
- Poster (ideally in a holder) for outside the initial meeting building
- Reserve balloon helium

- Server account setup (the Linux login setup, not PC² configuration). Also plan for backup servers (backing up of accounts).
- Contestant web site setup: this is the website that contains the following types of links: cheat sheets, scoreboard link, Java API, sample input and output, problems PDF, etc. Ideally, this will be the browser's default home page.
- Email reminder to teams that have not paid
- Public website setup; this includes the scoreboard
- Reserve Bodo's
- Buy stuff from Costco: candy, bottled water, soda, coffee stuff, etc.
- Coordinate volunteers, along with roles (judge, etc.) (who, when, how); likely at a volunteer meeting
- Create floor plan of the competition

5.5 Day before checklist

- Name tags: print them, put them in the badge holders, attach lanyards
- Pick up helium tank
- Prepare contest presentations, for both the welcome ceremony and the conclusion ceremony
- Set up print system
- Pick up Bodo's
- Creation of the registration packets. Put it in a nice folder, and include various things: schedule, information about UVa and the CS program, ...
- Copying of problem packets
- Prep the explanation of all the questions, including slides; this is in the closing ceremony.
- Webcams if there is time: this allows the coaches to see their teams competing.
- Printing of the certificates
- **SYSTEM TEST!!!!**
- Create signs for the day of the contest (this way to the computer labs, that way to the auditorium, etc.)
- Detailed script for the intro presentation and the awards ceremony, including who is saying what

5.6 Day of checklist

- Printing of packets (all)

Chapter 6

Contest Logistics

6.1 Other notes to file somewhere...

This is a section for notes that need to go somewhere, but where that somewhere is has not yet been determined. This section is not expected to be in the final version of this HSPC guide.

- For the nametags, print out two of each nametag, and have them facing out in both directions. This way, when the name tag gets flipped around, you can still see who that person is.

6.2 Roles

After the 2013 HSPC, when we had 39 teams, and over 150 students, we had the realization that we would need different roles that volunteers could fill, and a set of responsibilities for those roles. The responsibilities are split into two parts: preparation for the contest, and during the contest.

Many of the tasks that are listed in these roles start with 'ensure', as it is not necessarily the person in that role who will be doing the task. But it is the person in that role who will be ensuring that the task is completed successfully.

Unless listed otherwise, most roles only require one person.

6.2.1 Head Judge

Before the contest day:

- Ensure that all the contest judges have a computer that can boot up Linux, and that they can connect with PC²
- Ensure that the judges are assigned to the problem(s) they will judge, and that they are very familiar with those problems
- Ensure that s/he is familiar with ALL the problems

Contest day:

- Prior to both the practice and actual contest, ensure that the judges are all in the judging room and logged into PC² when the practice or contest is about to begin

6.2.2 Contest Judge

There will be multiple contest judges, with each one judging a small number of problems (1-3).

Before the contest day:

- Ensure that s/he has a working Linux system, with PC² installed and configured
- Once the judging problem assignments are assigned, familiarize himself/herself with the problems
- Ensure that they know how to use pc2judge

Contest day:

- Arrive at the judging room well prior to the start of the practice / contest
- Judge the problems!

6.2.3 Site Director

There is one site director for each site that the contest is being held at. Neighboring classrooms are considered one site, but classrooms in different buildings are different sites. The site director is expected to stay on-site during the practice and contest.

Before the contest day:

- Ensure that the computer configuration is all ready to go. The site director is not in charge of the configuration itself, but s/he will need to know how to boot up the machines, solve problems, etc.
- Ensure that the team mappings are known (each team has a location in the room, a team number in PC², and a computer number; this needs to be able to be quickly mapped during the contest, and this will also be needed by the balloon manager)
- Ensure that lines of communication with the contest director (and others) are known: cell numbers, etc.

Contest day:

- Ensure all the machines are booted into the contest environment well before both the practice and contest starts.
- Ensure that the home directories are wiped (before both the practice and the contest)
- Ensure that printing works, and that the printer is all configured and ready to go
- Supervise the volunteers who are acting as monitors
- Communicate with the person managing the balloons to make sure that they have everything they need (management of the balloons is a different role; but that role and the site director will need to communicate)

6.2.4 Balloon Manager

There is one balloon manager, who will supervise the balloon distributors at the various sites. Note that balloons are typically not distributed during the practice contest.

Before the contest day:

- Ensure that there are enough balloons, of the appropriate colors, to distribute to all the teams
- Ensure that the ribbon is ordered, and that scissors are ready for each site
- Ensure that the helium tank(s) is/are ordered; one tank will be needed for each site; alternatively, if helium is not being used, then ensure that whatever balloon display method is ready to go
- Ensure that the PC² judge logins are ready for each site, as the people distributing the balloons will need to be logged into PC² to determine which balloons to distribute
- Ensure that each site has a Linux machine available for logging into pc2judge
- Know how to configure PC² for balloon distribution (see section B.0.6)
- Ensure that the balloon distributors are trained how to read pc2judge and distribute the balloons. This is essential.
- Ensure that the site mappings (done by each site director) are known and available during the contest

Contest day:

- Ensure that every site is ready to go for the balloon distribution for the contest
- Ensure that the balloon pc2judge accounts are logged in at each site
- Supervise the volunteers distributing the balloons

6.2.5 Food Tzar

There are three food meals that need to be ordered for the contest: a light breakfast, lunch, and snacks/drinks during the contest.

Before the contest day:

- Ensure that each of the three food meals have been ordered
- Check the rules at each site as to what food is allowed at which site; all sites must have the same types of food available
- Ensure that the various special food considerations (vegan, vegetarian, allergies, gluten-free) are handled
- In the event that we're dealing with UVA dining, make clear to them that you are the person they should contact when food arrives, and make this clear to the other volunteers as well.

Contest day:

- Manage the acquisition and/or delivery of that food; in particular, be there to receive the catering, if that is how lunch is arriving
- Ensure that clean-up is handled properly (grab volunteers that day for this)
- Ensure that the various special food considerations (vegan, vegetarian, allergies, gluten-free) are handled

6.2.6 System Administrator

The list below assumes that the computers in the contest are going to be booted off of a USB key.

Before the contest day:

- Ensure that enough USB keys are available
- Oversee the configuration of the Linux image used during the contest
- Oversee the backup system of the contestant's home directories to the appropriate backup sever
- And be sure to know how to restore from a backup image
- Ensure that all the USB keys are properly duplicated
- Ensure that the printing system is working and ready to go
- Coordinate with the faculty advisor to ensure that the computers being used can be booted up on the USB keys, and that printers are available (one per site, with extra paper and toner cartridges)
- Test the system prior to the contest

Contest day:

- Help the site directors with the booting up of the computers via the USB keys
- Ensure that the printing is working
- Be available to solve any system-related issues

6.2.7 Coach Liason

Before the contest day:

- Obtain guest wireless access codes for the coaches (from <https://network-setup.itc.virginia.edu/>, click on "generate wireless passcodes". In the right-hand box (labeled "need a lot of passcodes"), click "advanced guest wireless passcode generation", and fill out the form.

Contest day:

- Ensure that a volunteer is sitting in the coach room during the practice and the contest; this need not be the coach liason himself/herself
- Check in on the coaches from time to time during the practice and contest

6.2.8 Volunteer Coordinator

Often, this role coincides with that of Contest Director.

Before the contest day:

- Maintain a list of volunteers who sign up to help for the contest, and keep track of their shirt sizes
- Ensure, along with the contest coordinator, that each of the roles listed here are filled
- As necessary, assign volunteers to help out with the various roles listed here
- Hold the volunteer coordination meeting in the week prior to the contest

Contest day:

- Manage the volunteers during the contest, helping to assign them to whatever work needs being done

6.2.9 Contest Proctor

There will be multiple contest proctors, a few for each site.

Before the contest day:

- Little needs being done prior to the contest; the site directors and volunteer coordinator can assign this on the contest day

Contest day:

- Ensure that they are familiar with the rules of the contest
- Make sure that nothing inappropriate happens during the contest

6.2.10 Tshirt Tzar

The tasks listed here may be done by the contest coordinator and the faculty advisor...

Before the contest day:

- Coordinate with the artist (manage payment amount, etc.) to ensure that the artwork arrives at LEAST 1 month prior to the contest (preferably 6 weeks prior)
- Obtain, from the Registrar (see below), the tshirt sizes need; make educated guesses as to the rest of the sizes needed
- Coordinate with the faculty advisor or treasurer to ensure that the tshirt company is paid, and that the tshirts are printed
- Ensure that the tshirts are picked up
- Assist the Registrar in creating the contest packets, and the tshirts are a big part of these packets
- Ensure the artist gets paid, if that was the agreement

Contest day:

- Not much to do; take on another role

6.2.11 Registrar

The tasks listed here may be done by the contest coordinator and the faculty advisor...

Before the contest day:

- Ensure the registration system is working, potentially adding new features (or ensuring that they are added)
- Manage the opening, closing, etc., of registration
- Ensure that the nametags and badges are printed
- Create each team packet (along with the other volunteers): certificates, nametags, shirts, etc.
- Provide a list of the tshirt sizes to the tshirt czar
- Coordinate with the volunteer coordinator to ensure that there are a sufficient number of people to help on the contest day with registration

Contest day:

- Manage the registration of the teams
- Print out the placement certificates, after the contest itself, for distribution at the closing ceremony

6.3 Day of the Contest

6.3.1 Registration

Probably need 30 minutes for registration. . . This is a good time to get coaches and chaperones configured for wireless access.

6.3.2 Opening ceremony

What to talk about: welcome, preparation, excitement, sponsors, schedule, contacts, rules, . . .

6.3.3 Practice contest

The practice contest is essential. It is an opportunity to work out all the kinks, understand the environment being used, and allow them to start the actual contest on time without having to figure out anything then (other than the contest problems). In particular, it is a chance for the students to:

- Understand how to log in to the computer, as some may never have used Linux.
- Understand how to edit and compile their programs, as it may be a different one than they are used to.
- Introduce the students to PC² (some may never have used it before), in particular, pc2team: how to submit problems, how to submit clarifications, etc. And perhaps the 'Test' button.
- Practice printing a document.

The practice contest should allow the coaches to work with the teams to help them through everything; in particular, if students are not familiar with how to compile in that environment (i.e., they are Windows users), then the coach can help them with the editing and compilation.

One thing to tell the teams is that nobody is going to care about the results of the practice contest for more than 5 minutes after the conclusion of the practice contest. So it's a time to practice submitting wrong answers, understanding how to use the system, etc.

There are a number of things that students should be encouraged to do during the practice contest:

- Practice submitting incorrect answers (wrong answers, run time errors, compile errors, time limit exceeded, etc.) to see the type of response. Note that we have seen some contests (at the collegiate level) where they view submitting a run time error as 'hacking' the system. We don't understand why, however.
- Practice printing a document.
- Practice submitting a clarification.

Note that there will be a huge number of submissions during the practice contest, more so than during any time in the actual contest, as all the teams are going to try to submit the various wrong answers at once.

The practice problem(s) should be very easy, so that every team can solve it. A typical length of the practice contest is 45 minutes; we often only have one problem. And balloons are typically not handed out during the practice contest.

After the practice contest (typically at lunch), any last minute issues can be addressed via announcements.

6.3.4 Lunch

Ensure that there are options for those with dietary restrictions. Chipotle was the caterer we used in HSPC 2012, and the reviews were polarizing (many students did not like the choices and complained that it was too much to digest during the contest). A blander option may be more appealing.

6.3.5 Actual contest

- Balloon delivery to the teams is a full-time job; one person should be dedicated to this and this alone. If it's a big contest (more than 30-40 teams), consider putting two people on this. If more than one person is scheduled for balloon delivery, ensure that they have a very good system for determining what balloons have been delivered and what have not.

6.3.6 Closing Ceremony

A sample closing ceremony. Be sure to pick an MC closing ceremony: Bloomfield is MC'ing

- A "congratulations" speech by someone important (department chair, site director, etc.). This to talk about:
 - welcome on behalf of the department
 - congratulations to everybody who participated
 - computer science is an exciting field with lots of opportunities both in research and industry
 - acm @ uva is very active, and does X Y and Z
 - icpc @ uva is also very active, and heads to the world finals most years
 - thank you for participating
- admissions representative, if present, gives their speech

- presentation of coach gifts (typically by the faculty member)
- awarding of prizes
- display of the final scoreboard
- please fill out the surveys!
- thank-you to the sponsors, as this would not have been possible without them
- closing: thank you for coming, hope you come back next year, drive home safely
- some teams may want to work on the problems later, so go over them after people have the opportunity to leave

Chapter 7

Finances

content to come...

Appendix A

VirtualBox image configuration

This appendix lists the full installation directions for the VirtualBox image that we have used in our contests for the last few years. They are based on what can be found online¹. The version online has more sections than what is listed here, as the original image is used at UVa for various classes. Only the sections relevant to a programming contest are included here.

This system uses 64 bit Ubuntu 16.04.

A.1 Introduction

Software Versions

This installation document installs the following versions:

- Kubuntu 16.04, 64-bit²
- Clang++ version 3.8.0
- Python 3.5.2³

Newer versions of the above may have since come out, but at the time of the writing of this document (August 2017), they were either the versions installed via apt-get (Clang, Python).

Notes

- The guest hard drive reported 9.3 Gb of space available, prior to distribution of the image. The disk image itself was, after compaction, 7.6 Gb. When compressed via zip, it was 2.6 Gb in size.
- You will likely need to use a different unzip program to extract the image; the ones that come bundled with most operating systems can not handle archives of that size. We have used 7-zip⁴ with success.
- Video and sound (via youtube) worked fine with Chrome.

¹<http://aaronbloomfield.github.io/slp/docs/virtualbox-image-details.html>

²<http://kubuntu.org/getkubuntu/>

³<http://packages.ubuntu.com/xenial/python3>

⁴<http://www.7-zip.org/>

- The VM may capture the mouse – to uncapture it, you press the “host key”, (which is the right Control key on Linux and Windows hosts, and the left Command key on Mac hosts). To have it warn you about what this is, you can reset all warnings via the VirtualBox help menu, and it will warn you about this at boot-up.
- In the image creation process, you may run into a problem with VirtualBox where it cannot register a new (or different) disk because it has the same UUID as a previous disk that you are replacing. If so, then the command `VBoxManage internalcommands sethduuid disk.vdi` (changing `disk.vdi` appropriately) will change the UUID, and allow you to proceed.

A.2 Basic installation

- Created a new VirtualBox image
 - I named it “UVa HSPC 2017” or similar; I manually selected “Ubuntu (64 bit)” as the version
 - I set the memory at 1536 Mb (instead of the default of 512 Mb), ensured that the disk size was “dynamically allocated” and was set to 20 Gb (instead of the default 8 Gb); everything else was set at the default. If your computers can support it, setting the memory at 2048 Mb would be better.
- I installed the latest Kubuntu 16.04.1 LTS (64 bit), desktop edition, from the DVD image online⁵.
 - When prompted, I clicked on ‘download updates’ and ‘install 3rd party software’ when the options were given
 - For hard drive, I used the default: “Guided – use entire disk”
 - The computer name is cassiopeia, the login name is ‘student’, full name is ‘L33t H4x0r’, and the password is ‘password’
 - This account can run root (system) commands via ‘sudo’ - if you don’t know what this means, you can safely ignore it
- Once it was finished, I rebooted, and logged in
- Via a Konsole, ran `sudo apt-get update` then `sudo apt-get dist-upgrade`
- Reboot!
- Ran `apt-get autoremove` (which didn’t have to remove anything)
- VirtualBox guest additions
 - These are the utilities so that VirtualBox will work correctly with the host computer (proper full screen, etc.)
 - From the VirtualBox Device menu, select “Insert Guest Additions CD Image”, and follow the prompts
 - Once done, run `autorun.sh` from `/media/student/VBOXADDITIONS_4.3.36_105129` (or similar), and follow the prompts. Alternatively, if that does not work, try running `sudo bash VBoxLinuxAdditions.run` from that same directory.
- Reboot!

⁵<http://www.kubuntu.org/getkubuntu>

A.3 Development installation

- Installed the other packages: `sudo apt-get install clang emacs24 vim nasm astyle tofrodos source-highlight gdb lldb doxygen doxygen-doc graphviz ddd git g++ evince g++-multilib libc6-dev-i386 libc6-dev:i386 flex`

- Ran the following two commands to change the default C/C++ compiler to clang:

```
sudo update-alternatives --set cc /usr/bin/clang
sudo update-alternatives --set c++ /usr/bin/clang++
```

- Downloaded Google Chrome from here⁶, and installed it via:

```
sudo dpkg -i google-chrome-stable_current_i386.deb
```

- That installation did not work perfectly (which was expected), and to fix an installation such as this you run `sudo apt-get -f install`
 - Then the .deb file was removed
- Added konsole, emacs, and chrome icons to favorites (from the K (start) menu, right-click and select 'add to favorites'), and the task bar (from the favorites menu, right-click and select 'add to panel'; this may require right-clicking on the panel and selecting panel options -> panel settings prior to moving the icons)
- Browser customization
 - Set Chrome's home page the in-contest status page
 - Chrome is set as the default browser
- I loaded up emacs from the command line, and then told it to disable showing the startup messages (this could also be accomplished by following guidelines available online⁷).
- Added a few aliases were added (the last 4 lines of .bashrc) to help prevent people from accidentally removing files (adding -i for rm, mv, and cp; and aliasing xemacs to emacs). This was done both in /home/student/.bashrc and /etc/skel/.bashrc.

```
alias mv='mv -i'
alias rm='rm -i'
alias cp='cp -i'
alias xemacs='emacs'
```

- Removed all the empty default directories in ~/student other than Desktop and Downloads
- Changed the auto-lock feature: K-menu -> Computer -> System Settings -> Desktop Behavior -> Screen Locking, uncheck the "lock screen automatically..." button, and click Apply.
- Plasmashell, which is part of the graphical window system, was crashing repeatedly. The reason seems to be tooltip previews of application windows, so these should be disabled: right click taskbar -> Task Manager Settings -> General -> Uncheck "show tooltips"

⁶<https://www.google.com/chrome/browser/desktop/index.html>

⁷<http://xenon.stanford.edu/~manku/dotemacs.html>

A.4 Programming Contest configuration sections

One of the sections below will detail how to turn off the Internet for use in the actual contest, and that should only be completed for the final contest image.

Programming Contest configuration

- **Install the packages:** `sudo apt-get install emacs24 vim eclipse g++ git gdb gedit openjdk-8-jdk`
 - If the Basic Installation section, above, was installed, then some of these packages have already been installed
 - The `openjdk-8-doc` package is not installed here to keep the image size down, but the packages above install it anyway
- **Install Eclipse**
 - We have found that the version of Eclipse that can be installed via `apt-get` is often out of date. Thus, we installed it from the packages online...
 - Fill this in... **TODO**
- If the submission system that you are using requires installation, then download that. We used PC², so the directions for that are included here.
 - Download the latest PC² software and unzip it in `/usr/local/`
 - At the time of this writing, the latest version is 9.3.2-3449, and the direct download link is here; thus, the directory it is unzipped into would be `/usr/local/pc2-9.3.2`
 - After unzipping, create a symlink: from `/usr/local/` run (changing the version as necessary):
`sudo ln -s pc2-9.3.2 pc2`
 - Edit `/usr/local/pc2/pc2v9.ini` and replace `localhost` on the `server` line (likely line 12) with the full server name of the primary submission server
- Create a script to restore the Internet access after the contest (this does not hurt anything to have it ready, as it won't do anything if the Internet is not turned off). This can be `/usr/local/bin/restore-internet`. As the commands therein need `sudo`, only the `student` user can run them. This is typically executed at the end of the contest if the contestants want to save their work (usually by emailing it to themselves). All that has to be done is to modify the default policy for the chains. Be sure to also run `chmod 755 /usr/local/bin/restore-internet`. The contents of the `restore-internet` script are the following four lines:

```
#!/bin/bash
sudo iptables -P INPUT ACCEPT
sudo iptables -P OUTPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
```

- Create a `/usr/local/bin/pc2team` shell script, and allow it to be executed via `chmod 755 /usr/local/bin/pc2team`. The contents of the `pc2team` script file are the following four lines:

```
#!/bin/bash
cd
/bin/cp -f /usr/local/pc2/pc2v9.ini .
/usr/local/pc2/bin/pc2team &
```

User account configuration

The student account can execute commands via `sudo`, so we have to create another account for the students to use.

- Change the password for `student` to something that they will not be able to guess.
- Create a `hspcteam` account (or similar), and set its password to `password` (or similar).
- If there are more users in the image, then run `chmod 700 /home/*` to prevent the second user from accessing any other user accounts. If there is only one user account on the system, then this is not necessary. There isn't anything of use in the `student` account, however.
- Log in as `hspcteam`; the rest of the commands in this list are from that account.
- Add in the four aliases to `.bashrc`, as described above in the "Basic installation" section
- Put the often used icons in the tool bar: Firefox, Emacs, Konsole, and Eclipse
- Load up Chrome, and set the contest site as the default home page
- Load up emacs, click on "never show this again", and then "dismiss startup screen"; exit emacs
- Remove all directories in the account other than Downloads and Desktop
- Create an icon in the task bar for `pc2team`. It may be easiest to use an icon adapted from another application. You can find an appropriate logo online⁸ to use for that

Turning off Internet access (and other things)

- Prior to the contest itself, a bunch of things need to occur to the image for it to work: Internet needs to be turned off, etc. These modifications are **NOT** made to the image as released to the contestants prior to the contest, and should be made only right before the final image is prepared for the contest.
- Edit `/etc/hosts` to allow that to resolve domain names to IP addresses. Specifically, the submission server(s) and the server(s) that have the contest materials should be included in that file, as follows:

```
1.2.3.4 server1.university.edu server1
5.6.7.8 server2.university.edu server2
9.10.11.12 server3.university.edu server3
```

- Configure iptables: this details how to turn *off* network access. **It is ONLY to be used on the final image for a programming contest!!!**
 - These directions based on the ones here⁹ and ¹⁰. We want to allow access to the submission servers, but deny everything else. We set the policy to all three chains to DROP, and add

⁸<https://www.google.com/search?q=icpc+logo>

⁹<https://help.ubuntu.com/community/IptablesHowTo>

¹⁰<http://stackoverflow.com/questions/21870386/iptables-block-access-to-all-ports-except-from-a-partial-ip-address>

ACCEPT lines for localhost and the course servers for the INPUT and OUTPUT chains (we don't do anything, other than changing the default policy, to the FORWARD chain). For example, the following commands will work (assuming the servers are server1.cs.univeristy.edu through server3.cs.university.edu):

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A INPUT -j ACCEPT -s server1.cs.university.edu
iptables -A INPUT -j ACCEPT -s server2.cs.university.edu
iptables -A INPUT -j ACCEPT -s server3.cs.university.edu
iptables -A OUTPUT -j ACCEPT -d server1.cs.university.edu
iptables -A OUTPUT -j ACCEPT -d server2.cs.university.edu
iptables -A OUTPUT -j ACCEPT -d server3.cs.university.edu
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

- Once those commands are entered via the command line, we save the rules: `iptables-save > /etc/iptables.rules` (note the greater-than sign in that command)
- After saving the rules, we configure the system to apply those rules on boot (specifically, when bringing the network interface up). Edit `/etc/network/interfaces`. There will only be a clause for `lo` (the loopback interface) present. Under the `iface lo inet loopback` line, put in the following line (indented): `pre-up iptables-restore < /etc/iptables.rules`
- You may want to be able to reset the image between the practice contest and the full contest. To do this, log in as `student` (as you will need to use `sudo`, and run the following command: `sudo /bin/cp -a /home/hspcteam .;` then enter the following script (and make it executable) as `/usr/local/bin/reset-hspcteam`:

```
#!/bin/bash
cd
sudo rsync -a --del hspcteam/ /home/hspcteam/
```

A.5 Image completion

Image finalization

- Reboot (rebooting also clears out `/tmp`)
- Run `apt-get autoremove` and `apt-get clean`
- Cleared browser history
- To reduce the size of the hard disk (since I have to host it for people to download), I ran `sudo cat /dev/zero > zero; sudo /bin/rm -f zero`. This creates a large file of all 0's until it runs out of space, then removes that file.
- Ran `history -c` to remove the history of the commands entered on the command line
- Logged out then shut down the guest, and ran: `VBoxManage modifyhd disk.vdi --compact` (using the real image file name instead of "disk.vdi"). A better way would be to load up into recovery mode and run `zerofill`, but the grub menu does not seem to be easily available to load into recovery mode, so I didn't pursue it any further.

Increasing the disk size

This should not be necessary, as the 20 Gb hard drive in the image has about half of that space available. Should you want to do this, you can find the directions online ¹¹.

¹¹<http://aaronbloomfield.github.io/slp/docs/virtualbox-image-details.html>

Appendix B

PC² configuration

As of the 2012 contest (i.e., March 17, 2012), the latest version of PC² was 9.2.1. There are other contest systems, including KATTIS (which is used at many world finals). However, PC² is used at the regional contests – and typically used for local practices – so that is what we opted to use.

This section assumes that the reader is familiar with PC².

An ideal way to configure a contest is to launch a server through an ssh shell, and have all the other PC² clients connect to that server.

B.0.1 Configuration files

There will be two different versions of the configuration files: one on the remote server, and one that all the clients use. Here is the client configuration file:

```
[client]
server=hostname.domain.edu:50002
minSecsBetweenUpdate=0
[server]
```

There are many other options in the configuration file – such as having multiple sites – but we did not need them. The `server` line specifies the server and port to connect to. For the configuration file on the remote server, just change the `hostname` from `hostname.domain.edu` to `localhost`. The `minSecsBetweenUpdate` line forces the various clients (such as the scoreboard) to immediately write the updated scoreboard files to the disk, rather than buffering them for a period of time.

B.0.2 Pre-configuring PC²

PC² can be pre-configured, and the configuration files moved over to the server that is actually running the contest. This makes it easy to configure it locally, and then upload the files to the remote server.

When you run `pc2server` (see next section), there are two directories created (`logs/` and `profiles/`) and one file (`profiles.properties`). The `logs` directory need not be preserved, unless you want to maintain a log of your PC² configuration.

In the `profiles/` directory, there will be a directory with a random name (such as `'P3b9dcd2a-1d67-4106-8e2b-6b407198e16b/'`). And in there are a number of files and directories:

- `db.1/`
- `logs/`
- `packets/`
- `pc2.startup-0.log`
- `pc2.startup-0.log.lck`

Once the PC² configuration is complete (and `pc2server` is shut down!), only the files in the `db.1/` directory actually need to be preserved. Thus, a fully configured contest would have these files/directories:

- `./profiles/`
- `./profiles/P3b9dcd2a-1d67-4106-8e2b-6b407198e16b/`
- `./profiles/P3b9dcd2a-1d67-4106-8e2b-6b407198e16b/db.1/`
- `./profiles/P3b9dcd2a-1d67-4106-8e2b-6b407198e16b/db.1/settings.dat`
- `./profiles/P3b9dcd2a-1d67-4106-8e2b-6b407198e16b/db.1/recovery.key`
- `./profiles/P3b9dcd2a-1d67-4106-8e2b-6b407198e16b/db.1/contest.key`
- `./pc2v9.ini`
- `./profiles.properties`

The `pc2v9.ini` file should have `localhost` as the hostname, as that is the directory that you will be running `pc2server` out of. Note that you may want to include a `.htaccess` file to prevent anybody from viewing the contents of this directory, if it is going to be on a web server (it only needs two lines: `'Order deny,allow'`, and `'deny from all'`).

PC² version 9.3 promises to include more detailed profiles, which would allow for easy switching between the practice contest configuration and the actual contest configuration. While PC² version 9.3 was released, it was so buggy that it was retracted, and 9.2.1 is the current latest version (as of March 17, 2012). Thus, you will probably want two different directories with the above setup: one called `contest/` and one called `practice/`. This makes it easy to switch between the two contests: just kill the server, `cd` into the other directory, and re-launch the server.

While most of the files are fairly small, the size of `settings.dat` will be proportional to the size of the files that you configure PC² with: judging input, judging output, validators, etc. Thus, if you have a 10 Mb output file for the judging data, the `settings.dat` will be slightly larger than 10 Mb.

Note that `settings.dat` is a binary file (it's Java Serialized data), so it will not work well with revision control software, such as SVN. And if you make a small change (such as one account name), SVN will copy the entire updated file into the repository; too many of these minor updates will dramatically increase the size of the repository. This may not be a problem, but it may also be, depending on if you are using revision control and where you are hosting it.

B.0.3 `pc2server`

Run `pc2server`. The initial login username is `site1`, and the password is also `site1`, and you can pick anything for the contest password (but remember it!). There is very little configuration needed to be done from the `pc2server` GUI – the only one is in the Sites tab, where you can change the Site Title (to `'UVa'`

instead of 'Site 1') and the password (that replaces 'site1', above). All the other configuration can be done from pc2admin.

When running it remotely, here is a command to kick off the server (this is all on one line):

```
nohup pc2-9.2.4/bin/pc2server --nogui --login site1 --password site1
    --contestpassword mypassword | tee server.out &
```

The `nohup` (for 'no hang-up') prevents terminating the server upon a logout, and the 'tee' causes output to be displayed both to standard output and to the file specified. Note that the `--password` password starts as 'site1', but can be changed via the GUI. The contest password is what you set as the contest password on the first login (i.e., the running of this command). The `--nogui` flag prevents loading of the GUI interface.

Note that closing the server (through the 'Exit' button in the GUI (admin GUI?) or killing the process) will terminate all connected clients.

B.0.4 pc2admin

Configuring accounts

One can easily create accounts in pc2admin via the 'generate' button. And while one can set all the options through the pc2admin interface, it is much easier to load all of them into a CSV file (tab separated), and load them into PC² directly.

The `accounts.csv` file needs to contain the following fields:

- `site` is the site number that the account is going to be used on; if a single-site contest (as most high school contests are), then they are all '1'.
- `account` is the account name: 'administrator1', 'team1', etc. Numbering should start at 1 for all accounts to enable easy loading of the file into PC². This is the login name.
- `displayname` is the name that should be displayed. For contest staff, this is their full name. For teams, this is their school name and team name.
- `password` is the login password; see below for how to easily generate random passwords.
- `group` is if they are in a group (i.e., all teams from one high school are in a given group). We have not needed this feature.
- `permdisplay` is whether they have permission to be displayed on the scoreboard. This should be 'true' for all teams that are competing, and 'false' for any other team accounts (extra team accounts, a team account for staff testing, etc.). If another team shows up, one can always toggle this permission in the pc2admin GUI.
- `permlogin` is whether they can log in. Should be 'false' for any extra accounts that are not being used, and 'true' otherwise; this can also be toggled through the pc2admin GUI.
- `externalid` is not something we used, and was left blank.
- `alias` is not something we used, and was left blank.
- `permpassword` is whether they have permission to change their password. Set to 'true' or 'false' as you would prefer.

In LibreOffice (and, presumably, in Microsoft Excel), one can generate a random 10-digit password via the following command:

```
=concatenate(char(randbetween(97,122)),char(randbetween(97,122)),  
char(randbetween(97,122)),char(randbetween(97,122)),char(randbetween(97,122)),  
char(randbetween(97,122)),char(randbetween(97,122)),char(randbetween(97,122)),  
char(randbetween(97,122)),char(randbetween(97,122)))
```

Note that when the file is saved as a CSV, the actual value of the password is saved, not the formula.

To load the accounts.csv file, you first have to generate the same number of accounts via the pc2admin GUI (Configure Contest -> Accounts -> Generate). If not all the accounts have been generated, then pc2admin will not load the accounts.csv file. If you are changing properties that are not account, displayname, or password (i.e., if you are changing any of the permissions), you have to click the 'Include unchanged accounts' check box for those changes to take effect.

As the accounts are likely to change often, and it's a text file, it works much better with revision control to edit this file, and always load the accounts when launching the practices or system tests; this prevents the settings.dat file from having any modifications.

Configuring problems

(including setting up the auto-judge)

Configuring a custom validator is described in the end of section 3.6.

Other configuration details

- setting the time for the contest
- languages that they can use
- ...

B.0.5 pc2team

There isn't much to talk about this – once the accounts are set up, and the passwords are distributed, teams can login. Note that if they can access pc2team before the contest clock starts, they will be able to see the names of the problems.

Talk about the 'Test' button... (Ben?)

B.0.6 pc2judge

There will need to be many judges. Each problem will probably need an auto-judge, which is a separate login. And the human judges will each need an account. It is desirable to have the auto-judges running on a separate computer, as every time a submission comes in, it pops up many dialog boxes (and if you happen to hit the space bar, because you were typing something, it may click the 'cancel' button on those dialog boxes).

We found that running 10 auto-judges on a single computer caused too many problems: memory and CPU usage. We would have 2 problems auto-judged on a single computer, and typically tried to balance the problems (so the hardest problem was auto-judged on the same computer as the easiest problem, etc.).

With the configuration set above, the auto-judge will determine the right answer, and then a human judge to review and release.

Judging problems

PC² cannot distinguish between a wrong answer (WA) and a run-time error (RTE). In the case of the built-in validator, they will all report as WA. If a custom validator is being used (see Validators, below), then they will all report as “No – Undetermined”. In both cases, when the human pulls up the output, a RTE will show the non-zero exit code in big, bold, red letters at the bottom; a WA will not show this. So while it’s easy for a human to determine which is which, the auto-judges cannot.

Using pc2judge

Depending upon the number of judges, it may be most useful to take advantage of the filter features. Each judge is responsible for dealing with their own subset of problems, which helps with consistency and keeps things organized during a rush of problem submissions and/or clarifications. In practice, all of the judges will be collaborating, but this can help things running smoothly. Many problems involving a validator will come in as “No – Undetermined”, and verifying the input will be the only way to get an actual response is to check the output manually. Runtime errors are easy to identify, as the error code will be displayed at the bottom of the window. “Wrong Answer” will be the most likely response needed, although “Output Format” will also arise. “Output Format” is rather difficult to judge, unless you can convince yourself that everything is correct but in the wrong place. It is generally advised to bias in favor of a “Wrong Answer.” Further, the judging system does not allow “Time Limit Exceeded” to be simply accepted as a correct judging, and so this will need to be handled manually.

Balloon distribution

Balloon people need to prepare beforehand with pc2 so that they know how to set it up and deal with the system in order to get balloons delivered on time.

B.0.7 pc2board

Appendix C

Sample Emails

The e-mails in this section were the ones that were sent out for the 2012 UVa HSPC contest.

C.1 Open Registration

Dear John Doe,

I have opened registration to the High School Programming Competition at UVa. The competition will be held on Saturday, March 17th in Charlottesville, VA. You can register up to three, four person, teams per school for the competition here, and find out more information about the competition here. If you have additional questions, either reply to this message or email me at pmc8p@virginia.edu. Registration is \$40 per team and will occur on a first-come, first-serve basis with an upper limit of 30 teams. Registration will close on March 1st, but I'd appreciate it if you registered earlier.

Regards,

Jane Doe

HSPC Chair

C.2 Notification two weeks prior to the contest

Email subject: "UVa HSPC: Important Information"; no attachments.

Hello Teams,

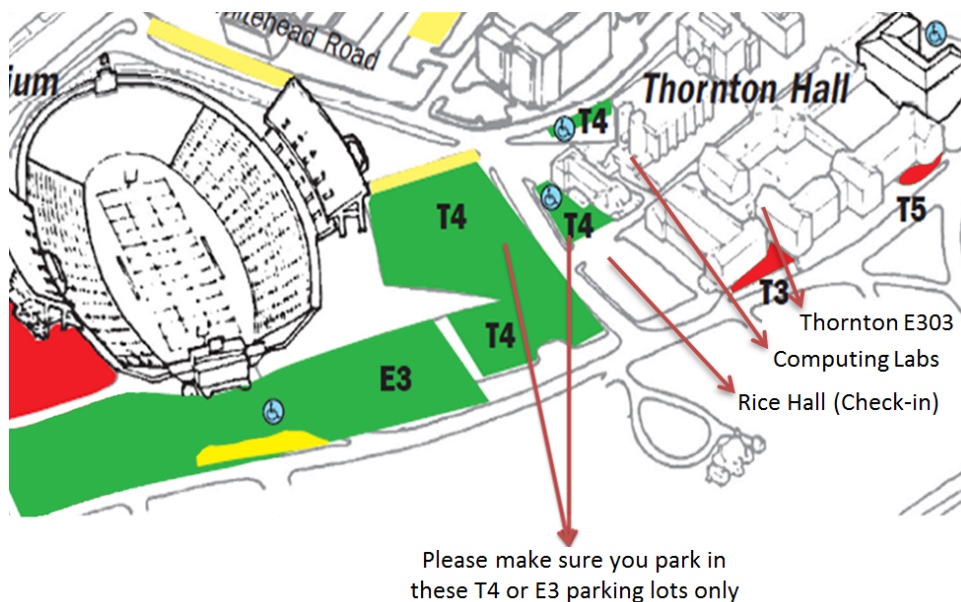
With just under two weeks to go, the contest is right around the corner and we have some important information for you. Next **Monday (3.12.12)** will be the last day to update us on: the **name of contestants / coach** (we would like **first and last** names), the **team name**, and any **dietary restrictions** or preferences. Just reply to this email with any updates you may have.

We are also happy to announce that the School of Engineering and Applied Sciences, here at UVa, will be holding their annual open house on the same day as our contest. You can read more information about the open house here. We want you to have time explore the open house, so we've revised the contest schedule as follows:

Teams must register by 9am in Rice 130.

- 8:00 - 9:00a: Registration in Rice 130
- 9:00 - 9:50a: Teams Explore SEAS Open House
- 10:00 - 11:00a: Introduction, Rules, and Questions in Rice 130
- 11:00 - 11:45a: Practice Contest in Mechanical Engineering Computer Labs (with coaches)
- 12:00 - 12:45p: Lunch in Thornton E303
- 1:00 - 4:00p: Contest in Mechanical Engineering Computing Labs, coaches stay in Thornton E303
- 4:00 - 5:00p: Awards Ceremony, Wrap-up in Rice 130

For your convenience, I've embedded the site map and directions also available on our website. An interactive site map and directions are also available [here](#).



We have all been working hard to get ready, and are really excited; hopefully you're looking forward to the contest as much as we are.

Regards,
Jane Doe
HSPC Chair

C.3 Media release email request

Email subject: "UVa HSPC: Media Release Form"; attachment was the Photo-Permission-Form.pdf.

Hello again teams,

We have a last minute request for you; we are trying to get the local media to cover this weekend's event. In order for them to include a photo and for us to publish a gallery of the event online we need a release from everyone pictured. This is completely **optional** and will have no effect on the team on the day of the

contest, but we would appreciate it if you can get the signatures. For contestants under the age of 18, we need a parent or legal guardian's signature. If you are able to get the forms signed, you may turn them in to us at registration.

Thanks,
Jane Doe
HSPC Chair