

# **PROGRAMMING CONTESTS: TWO INNOVATIVE MODELS FROM NEW ZEALAND**

Raewyn Boersen  
Faculty of Business and Information Technology  
Whitireia Community Polytechnic  
Auckland Campus  
New Zealand  
r.boersen@whitireia.ac.nz

M. Phillipps  
Department of Mathematics  
Lynfield College  
Auckland New Zealand  
mphilipps@lynfield.school.nz

## **ABSTRACT**

The New Zealand Programming Contest and the Young Women's Programming Contest are both believed to be unique. The first caters for programmers with a range of skills from aspiring to experienced. There are several categories of contestant including the relatively recent addition of one for high school students. Problems are weighted according to perceived difficulty. The winner of each category is the team with the most points. Languages used to solve the problems are those generally available and used by either teaching institutions or industry. Contestants compete in 3 person teams over a five hour period and they solve mainly algorithmic problems.

The Young Women's Programming Contest, by contrast, is only for female high school students. As the contest uses a specially written assembler-like language that may be mastered in a very short time, the contest event includes the teaching of the language as well as the competition. Young women compete in teams of two, the questions in the problem set are multi-part with a variety of points per part. These parts are incremental. Partial marks may also be allocated. The type of problems solved vary from common mathematical topics to simple business ones.

The infrastructural components of successful programming competitions are analysed. These are deemed to be having a 'champion', marketing the contest, administrative systems, site resources, rewards, problem sets, judging systems, training opportunities and contestants. Whilst all major components are present to some degree in both competitions, and are agreed necessary, the factors of having a 'champion' and marketing are paramount for a contest to grow and develop.

Future developments of both contests are suggested. The marketing of the New Zealand Programming Contest to high school students is a priority. The Young Women's Programming Contest, currently in recess, is still considered a valid enterprise and worthy of restoration.

## **INTRODUCTION**

New Zealand is a small country with its nearest neighbour of any size (Australia) being at least 3.5 hours away by plane. This isolation has meant that as a race, New Zealanders are used to creating their own solutions uninfluenced by others. This has been the case for 2 computer programming contests which have been 'born' in New Zealand and which are available to New Zealand high school students. They are the New Zealand Programming Contest (NZPC) and the Young Women's Programming Contest (YWPC). New Zealand also has the ACM Collegiate Programming Contest but high school students are not eligible to compete. Another high school competition, the Informatics Olympiad International (IOI), is not hosted in NZ.

The NZPC and the YWPC are two models of contests which the authors' believe are unique and have strengths that other contest models and in particular, the ACM and IOI models do not have.

This paper firstly explores all aspects of the NZPC and the YWPC. For each contest, the rationale, team composition, contest particulars, language, problem set, contest dynamics, prizes are described. The development of each model and the links to the curriculum are identified and then each model is compared to the ACM-like model. The components of a successful programming contest are then identified and used as the basis to critique the 2 models. The paper concludes with a plan of how each contest may be developed further.

## **NEW ZEALAND PROGRAMMING CONTEST**

The first New Zealand-born contest started in 1986 and was for tertiary students. It was called the New Zealand Programming Contest and is still operating successfully today. It was replicated very successfully in Australia for a number of years but was discontinued as there wasn't a 'champion'. In 2004, (New Zealand ) high school students competed officially for the first time. In the 2005 contest, it was pleasing to note that high school teams beat several tertiary teams in the overall rankings.

### ***Rationale***

The educational environment in New Zealand at the time was growing, with more tertiary educational institutions being established. The Auckland Institute of Technology was in a strong position with an established reputation for developing industry-ready, competent programmers. It was thought that if other institutions provided competitors for a contest, then the AUT ones would win/dominate, showing that the newcomers were not providing graduates of the same calibre. The contest would effectively reinforce the power of the AUT brand in respect to programming graduates.

### ***Team Composition***

Initially students competed in pairs as the industry required graduates to have team-work experience, and so this was included as a facet of the contest from its inception. The contest spanned a 3-hour period and had a graded set of problems to solve.

Teams entered specific sections, e.g. first year student section, second year student section, 3+ years section and business section, with prizes being given for each section. At the first contest, a tertiary team soundly beat one of the business teams and this resulted in the immediate demise of the Business section. The contest was designed primarily for undergraduate students and so a limit was placed on each team – there could be only one student with over 4 year's study in any team.

Then as now, students in a single team must belong to the same school/tertiary institution. Generally teams select themselves however a teacher/lecturer may provide the forum by which like-minded students come into contact with each other.

### ***Contest Particulars***

There are 5 sites throughout New Zealand. Each site assumes responsibility for its own administration, organisation of the contest day, and marketing. They may provide additional contest languages if they wish and may choose whatever judging system they want. The contest is held on one day in July or August each year, prior to the ACM South Pacific Programming Contest. Contests are 5 hours duration.

Some sites have a small practice session of 1 - 1.5 hours before the contest proper. This is to ensure that teams are comfortable making submissions with a different text editor and operating system. It also allows the judging system to be tested.

High School students are charged \$30, tertiary teams \$75 and Open section programmers (only 1 person per team) \$150. This entry fee is often paid for by the institution.

Lunch is not provided between the practice session and the contest itself . However, a light meal may be provided at the conclusion of the contest. This serves not only to provide sustenance but allows time for the final collation of results and winners to be announced. It is paid for out of the entry fees and any local sponsorship that the site has sought.

## ***Contest Languages***

The languages universally available are Pascal, C/C++ and Java, although individual sites may include others such as Visual Basic and C#.

## ***Problem Set***

The problem set is often constructed by a team of people although sometimes a single person has assumed responsibility for producing it. The focus on the problem set is therefore closely associated with the interests and experiences of the judge(s). These are typically academics from the university and polytechnic sectors however with the addition of high school students, secondary school teachers are now involved/consulted. The 'Head Judge', in collaboration with the judging team, takes responsibility for producing a clearly specified and well-balanced problem set, with solutions.

Problems within the set are graded and are allocated a certain number of points. For example, the point range was initially 5, 10, 50 or 100 point questions with 4 questions at each point level. In theory a 10-point question should take twice as long as a 5-point question and a 50-point question 5 times longer than a 10-point question. This changed over time to a logarithmic rather than linear scale. Currently, the points allocated are 3, 10, 30, 100 with 4 questions at each point level. The 3-point problems were added for high school students and it is interesting to note that in the 2005 contest, high school teams solved 3, 10, 30 and 100 point problems. Graded problem sets add an element of strategy to the contest. The winning team in each section is the one with the most points.

Contestants, no matter which section they enter, choose their own degree of competitiveness by choosing the level of problems to solve. For example a Tertiary Open team would need to weigh up the time and effort required to solve one 100-point problem versus three 30-point problems. This adds further challenge and encourages teamwork.

To ensure that all contestants in all sections have appropriate level problems to solve, each section is given a set of criteria. For example, the High School section is for high school students who will have had 1 semester of programming and the Tertiary Junior section is for the students who have had 1 - 2 semesters programming. The problems for these sections of teams, therefore, are what one would encounter when first learning programming. Tertiary Open by contrast is for students who have done a considerable

amount of programming and who are very proficient in a language. The higher point problems for them are more algorithmic.

By designing problems to meet the criteria for each section, teams achieve success at their own level.

A sample problem set may be found in Appendix 1.

## ***Contest Dynamics***

Submissions at some sites are made and marked using a piece of judging software called PC<sup>2</sup>. This is supplied by the ACM for use in programming competitions to automate submissions, judging and result recording. Penalties are not applied if a solution is incorrect nor are partial points allocated. Some sites use PC<sup>2</sup> but others use systems that they have developed themselves. PC<sup>2</sup> is more than adequate for making submissions and judging correctness but it does not adequately handle scoring because problems have different weightings. Nor does it provide a collated scoreboard for the combined sites. Feedback to contestants about their submissions is given using a set of standard messages. The winning team in each category is the one which has the most points. In the case of a tie, time is used as a tie-breaker.

## ***Prizes***

Each competitor receives a certificate of participation. The two top performing tertiary teams get free entry into the ACM South Pacific Programming Contest. Site specific prizes may be given to the top performers in each section if there is local sponsorship. The top tertiary team in the overall rankings receives a trophy.

While there is no trophy for the top high school team, if the competitors are under 17 years of age, they are sponsored by the New Zealand Computer Society to compete in the International Schools Software Competition which is administered by the South East Asian Regional Computer Conference (SEARCC). This is a confederation of national information technology professional societies. The participating countries are Australia, Canada, Hong Kong, India, Japan, Malaysia, New Zealand, Pakistan, Philippines, Sri Lanka, Taiwan and Thailand. Teams must be endorsed by the national computer society of their home country. The host country provides food and accommodation for the teams and their chaperones during the competition period.

Teams consist of up to 3 secondary (high) school students and each country is allowed to send a maximum of 2 teams, i.e. 24 teams in total. The competition is 2 hours duration and teams have 4 problems to solve during this period. Solutions are judged correct when the output generated matches

the judges data. The programming languages available to the students are Pascal and C/C++.

The justification for inclusion of high school teams in the NZPC came in 2005 with the second placed team winning the SEARCC contest in Sydney. (The winning high school team in the NZPC had team members who didn't meet the age requirements).

## ***Contest Developments***

The contest has evolved over the years. Initially the model was refined (team size and languages). After contact was made with the ACM Programming Contest in 1988 and the decision was made to have an ACM contest as well, it was determined that the NZ Programming Contest would continue to serve as a contest in its own right. There were many benefits of this. The New Zealand Programming Contest model met the needs of a wider range of contestants by means of the structure of the problem set, the classification of problem difficulty using the points system removed some of the stress for the beginning student/competitor, and without the need to operate within the constraints of the ACM Finals, business-oriented as well as computer science-oriented problems could be included in the problem set. Additionally, it would also provide a forum for practice for those wishing to compete in the ACM Contest.

Other changes over time were the adding of a new competitor category (high schools), the changing of the section names (School, Tertiary Junior, Tertiary Intermediate, Tertiary Open and Open), adding and deleting languages, increasing the team size to 4 then reducing it again to 3 and lastly, expanding the duration to 5 hours.

It should be noted that the high school section was introduced after the demise of a local contest designed to select students to compete in the SEARCC contest. This section commenced in 2004. Initially, teams competed for fewer hours than the other sections of contestant but in 2005, they competed for the whole 5 hours.

Some aspects of the ACM model were incorporated. The process of run submissions, problem clarifications and judging feedback has also evolved over time and now, with the ACM judging system PC<sup>2</sup> being widely available, this forms the basis of judging component of the contest management system today. The recording and ranking of results is done manually.

As the judging software has become more stable over time and students are more familiar with text editors and operating systems, some sites have chosen not to have a practice session.

Prior to the High School section being introduced, a dispute arose over an unofficial entrant who was allowed to compete even though they were not a tertiary student. This is part of being a New Zealander - we would prefer to be flexible and allow opportunities to be taken as and when they can. After this

incident however, category definitions were tightened and were strictly adhered to.

## ***Curriculum Links***

There is no high school curriculum which relates to the format of this competition. Successful teams are those that have an extra-curricular interest in programming and who are highly motivated, or that have a teacher who has provided extension classes.

With respect to the other categories of contestants, New Zealand has two main types of post-secondary educational institutions - polytechnics and universities. The former has traditionally had a strong focus in its IT courses on producing business-orientated graduates. Universities also met this demand through IT/MIS courses in business faculties with Computer Science faculties tending to focus on algorithms and problem solving. The degree of complexity of each category of question e.g. 10-pointer, 30-pointer, will be based on a university's computer science curriculum. As there is not a standard university curriculum across all universities in New Zealand, this may lead to some localisation of the problem set. The 'flavour' of the problem sets is also driven by the experiences and background of those who construct the problems. Typically these are people with a strong computer science background. This impacts on high school students who are being exposed to the computer science programming model as opposed to the business focused model.

## ***Comparison with ACM-like Contests***

The major point of difference between this model of competition and the ACM competition is that of allocating a value to each problem. Whereas the ACM contest has a problem set with uncategorised problems, the NZPC problem set very explicitly indicates the level of complexity of the problem, removing one aspect of difficulty and therefore point of stress for the teams.

Teams have examples of questions for each point level and so they will be aware of how their skills match each level. A high school team for example will know that they can definitely solve 3-point problems, that they will probably be able to solve some of the 10-point problems and that they will possibly be able to solve a 30-point problem but that a 100-point problem will stretch them considerably.

In the ACM contest, teams need to read and understand all the problems in the entire problem set. In the NZPC contest, this is not necessary. Teams may choose to read all the problems of a particular level of complexity (i.e. points) but they do not need to read the entire set. This has the added advantage of not raising feelings of inadequacy when they read a problem

and have no idea of how to approach solving it. This is of particular advantage with novice programmers.

It is also reassuring to high school teams to know that the 3-point problems are designed for their level of knowledge and skill. They know that they will be more likely solve a problem successfully and repeated successes means that they will be more willing and motivated to develop their problem solving and programming skills over time. In the short time this might mean that they will compete in the contest several years in a row and in the long term, they might go on to have a career in programming.

The cost of competing in the NZPC is lower than the ACM contest and this means that institutions can afford to enter almost twice as many teams for the same amount of money. In theory, this should mean that the number of teams in the NZPC should be growing but this is not the case.

The NZPC has no over-arching body and rules to comply with. This has the advantage that each site has the freedom to conduct the contest as they wish (within date/time/problem set constraints). A disadvantage however is that there is no infrastructure in place to help with marketing and growth of the contest. The effect of the lack of sponsorship (and subsequent funding) is a noticeable difference between the NZPC and the ACM contests. NZPC competitors do not receive any promotional materials and prizes, and at some sites, there might not be a prize-giving ceremony or a celebration of any sort.

## **YOUNG WOMAN'S PROGRAMMING CONTEST**

The second contest unique to New Zealand was called the 'Young Woman's Programming Contest' (YWPC). The first contest was held in 1988. It was organized through Auckland Institute of Technology (AIT – now known as Auckland University of Technology, AUT). It went into recess in 1998. Some small-scale follow up research was conducted in 1999 and it was found that, of the 1988 contestants, 38% entered the IT industry. (Costain, 1999).

### ***Rationale***

The YWPC grew out of an initiative by the Computing Department of AIT to run a New Zealand Programming Competition along the lines of the ACM contest in the United States. The first of these attracted only tertiary age males.

The YWPC was intended to redress the gender imbalance not only in such competitions, but also in IT study. It aimed to attract female high school students into considering IT study (the first prize was a scholarship for a year's full-time study at AIT) and to give the young women an enjoyable but challenging experience in the absence of males – who may unintentionally have dissuaded young women from considering such an option.

## ***Team Composition***

Teams consisted of 2 girls from Form 6 and Form 7 (i.e. Year 12 and Year 13 or 16-18 years of age) from the same school. These were the last 2 years of their high school education and so by definition, students could only enter a maximum of twice. This meant that there was a steady number of students being exposed to programming as well as denying any one team the opportunity to continually dominate. More than one team from a school could enter as the emphasis was on participation. The team size of 2 was chosen so that girls had support in sharing the problem-solving with a friend. Two was logistically more feasible than a larger team size. The need for team work was not an imperative.

## ***Contest Particulars***

The competition was conducted over a weekend towards the end of the academic year (September), with dates being arranged so as not to impose a burden on students studying for external exams. On the Saturday, tuition in a text editor and an artificial assembler-like language was given and on the Sunday, the actual competition was held. The tuition was approximately 2 ½ hours with a further 3 ½ hours practice. The competition was 3 hours in duration, with between 9 and 15 problems to solve. Each team had a single workstation. On the Sunday, the competition was held from 9.00am to 12.00pm.

## ***Contest Language***

This artificial assembler-like language was called SPLAT – Simple Programming Language Auckland Tech. It was developed from an idea attributed to Auckland University lecturers and developed by Neil Binney, a mathematics lecturer at AIT. His initial development was for use on a programmable calculator. Ms Phillipps developed the concept further by creating an assembler simulator which took simply written instructions to do sequence, decision and iteration commands. The language had 11 commands and could be taught and applied in less than 4 hours. These covered Input/Output, Arithmetic operators, and Jumps allowing decisions and loops. The students wrote in Splat, and the simulator, written in BASIC, performed a syntax check and executed the commands. A sample programme may be found in Appendix 2.

A non-commercial language was chosen because different schools taught a number of different programming languages such as BASIC and LOGO and without a nationally accepted syllabus for high-school programming, it was considered fairer to expose everyone to a new programming language. In addition, its use enabled students who did not have any previous

programming language knowledge, to enter and experience programming first hand and then perhaps consider an IT vocation.

## ***Problem set***

The problem set was initially constructed by high school-experienced staff at AUT, typically from a mathematics background. It was usually a team effort with several people contributing problems. The flavour of the problems remained consistent over the life of the contest. The contest organiser took the 'head judge' role and selected the final problem set and ensured solutions were available.

Different problems were allocated different points. The points ranged from 10 to 20 and initially a few problems had extensions for extra points. This grew over time into a concept of "basic" marks for a question with "bonus" marks added for meeting increasingly sophisticated requirements.

This concept of developing an existing solution further means that the judges and students need consider fewer different scenarios. For example, the beginner's section may have had 4 different scenarios and each scenario had up to 5 different parts. Each part could be allocated up to 12 points, whatever was appropriate for the degree of difficulty and the time it would take to code/solve.

Part marks could be awarded and bonus marks were given for 'efficient' code. The mark of efficiency was having solved the problem in fewer steps than the Judge's model solution. The winning team was the one with the most points.

The type of problems teams were asked to solve were more of an 'everyday situation' like scoring for a game, calculating profits or producing invoices. A sample problem set may be found in Appendix 3.

## ***Contest Dynamics***

The code solutions were printed centrally and "runners" delivered code to the teams. When a solution was deemed by the team fit to be marked, it was delivered to judges by the runners. The judges compared the code to a model answer and also executed the programme with pre-determined test data. Teams were not given any feedback from the judges but a team could resubmit if it wished. Teams were not informed of the marks they gained while the contest was in progress. This meant that girls could not compare their performance with other teams and therefore were not discouraged. They remained positive throughout the contest in anticipation of their results. They did not need incremental progress reports to remain focussed and competitive.

## ***Prizes***

After the contest was over, a lunch was provided and after the lunch, a prize-giving ceremony took place. Some of the costs were covered by an entrance fee of \$20 per team and sponsorship was sought for the balance. All participants were given certificates of participation. Each girl in the winning team could choose from \$400 worth of either sporting goods, books or clothes. They were also awarded a study scholarship worth approximately \$1000 which would have paid their first year's IT study fees at AUT. This was paid for by the Institute's Affirmative Action grant. The option to take this up could be deferred for up to 3 years. This was taken up only once in the 11 years. Often a guest speaker, who was a successful role model from the IT industry, spoke about her experience. The school of the winning girls team also benefited as it received book prizes for their library. The contest was funded by a small cost to the teams and prizes were donated by educational institutions, women's organisations and small businesses in exchange for publicity.

## ***Contest Developments***

With students wanting to return and to differentiate these from those who had competed before, it was decided to have 2 tiers – novice and experienced. A further change was made when, after a dispute, electronic submissions were introduced, to act as a simple timestamp. Submission was still physical but this allowed the recording of the time for each submission and verification of each submission. It also allowed the determination of the earlier submission and therefore the winner in the case of a tie.

In the first year, 36 teams from 19 schools entered from the upper North Island. New Zealand's geography (2 main islands) and the cost of internal travel made it prohibitive for schools from further afield to compete. The competition ran successfully for 11 years in spite of the difficulty of finding sponsorship. In 1999 the contest was cancelled due to street closures for the APEC leader's conference and although it was intended to hold it in 2000, the champion (Gay Costain) left AUT. As there was no successor and on-going sponsorship had not been arranged, the contest was not run again.

## ***Curriculum Links***

New Zealand high schools have never had a coherent IT curriculum. Programming is not taught under a national syllabus but rather in an ad-hoc fashion by teachers with a strong interest in IT. The extensiveness and quality are dependent on an individual's knowledge and experience. While the area has not been researched, there is anecdotal evidence that most teachers in high schools teaching programming are self taught, without a formal computer science education or IT industry experience.

## ***Comparison with ACM-like Competitions***

There are many points of difference between this model of competition and ACM-like competitions. The contest had a political agenda in that its rationale was to introduce young women to programming in a supportive non-masculine environment. It was therefore a single-sex competition. There was a deliberate emphasis on participation rather than competitiveness, although winning was celebrated.

The contest was spread over 2 days. Because a specific 2GL teaching language (SPLAT) was used which was not available outside of the competition, there was a tuition day when it was taught and practised and another day when the competition was held. The fact that the basic tenets of programming were taught meant that it was possible to enter and compete as a complete novice in programming. The limit to the number of times a student could compete was twice.

Teams were comprised of 2 contestants and this was to allow them to share their experience with a friend, not so much as to encourage teamwork as in the ACM-type contests.

The problems were multi part with the basic problem being developed further for additional marks. Part marks could be awarded for partial correctness and as the code produced was manually judged, this allowed for efficient code to be rewarded with bonus marks. Scoring was invisible during the contest which aided the participatory rather than the competitive aspect of the contest. Lastly, the YWPC was a single site contest without a national or international framework.

The points of similarity with ACM-like contests were minimal.

## **SUCCESSFUL COMPETITIONS**

When analysing the various contest models, we believe that there are 9 major components that are common to all contests and which must be the basis of any contest development. This section discusses these components and uses them to critique the NZPC and the YWPC models.

### **1. Having a Champion**

This is a person who is committed to the programming contest. They will be philanthropic and committed to the greater social good in that they believe that they are providing a unique experience for students to develop their IT skills. Where there is an international component to the contest, they are committed to showcasing the skills of their country. They provide an opportunity for contestants to develop skills in a forum that transcends individual educational institutions and the limits of geographic barriers.

A champion must have the ability to communicate the mission and vision to others so that there is a cohort of like-minded volunteers who will participate at local site levels. For multiple site contests, the champion assumes the role

of 'general manager' seeking and verifying new sites and 'serving' the local site co-ordinators. The co-ordinators in turn need to act as local champions and invite other academics and coaches to be involved in the contest organisation. This organic form of succession planning ensures the longevity of a contest.

A champion needs commitment to the philosophy, to be entrepreneurial and to be able to adapt and develop the contest with changing circumstances. They also need the ability to build a team of supporting volunteers who are willing to accept responsibility for a specific function.

The NZPC has a champion who is able to manage the operational aspects of the contest. There is a very small team of volunteers over the 5 sites who provide support.

The YWPC currently does not have a champion. However, one of the author's, with institutional support, is willing to revive the contest. Currently, a tertiary institution appears keen to provide this support. This institution now has access to expert marketing resources.

## **2. Marketing the Contest**

There are 5 components to this. The first is 3<sup>rd</sup> party marketing to schools that are the conduits to the teachers who in turn will manage and coach teams of contestants. Not only do this group need to be informed about the contest, its purpose and organisation, but they also need to extend the invitation to the teams to participate in the contest itself.

Secondly, the seeking of a sponsor who will help fund the contest costs and prizes. As part of this relationship it is appropriate to market the sponsor's involvement to the contestants as well as to the wider community.

The third component of marketing is to publicise the institutions and teams who have performed well during the contest. This might take the form of providing marketing information that can be published in institutional newsletters etc or in local newspapers.

The fourth component is marketing to potential volunteers, typically academic and technical staff. By building up the profile of the contest and making it an attractive and prestigious activity, the volunteer base is continually being expanded and succession is assured.

The final component is the marketing of the contest to the actual participants, making it an exciting event with attractive prizes. Post-event follow-up might take the form of certificates acknowledging participation, certificates of achievement and sponsors' gifts.

The NZPC does not do any marketing. Communication is done via the website and institutions and contestants need to know where to find this. There is no sponsor and they have a narrow volunteer base. Some sites have prizes if local sponsorship has been sought, others do not have any type of post-contest celebration. Coaches may be given a soft copy of a participation certificate that they may print if they choose.

The YWPC will need to focus on this area when it restarts. It does have a small base to work with, in that it can invite those teachers and schools who have participated previously to participate again.

When it was operating, the numbers had dropped from 36 to 19 teams and from 19 to 12 schools and it is believed that the lack of new schools was due to marketing issues. In 1998, the sponsor for the contest was a software company and so the prizes for contestants became copies of software. We believe that these prizes were less attractive to the young women and that this may also have been a factor in the drop in team numbers.

### **3. Administrative Systems**

These need to include the issuing of invitations to compete and acknowledging entries. Contestant, coach and institutional details need to be recorded so that they can be used for further marketing and to provide contestant/team lists to site co-ordinator(s). Entry fees need to be billed, banked and if necessary refunded. Site and contest information for contestants and coaches and judges need to be developed and distributed.

If information is communicated via a website, then the development and maintenance of the website would also be included.

The NZPC has a website and uses this to communicate with competitors and coaches. All other administrative tasks are local to a specific site and each sets up their own systems and collects and uses entry fees how they choose.

The YWPC does not have any administrative systems. When it was operating, all systems were manual and were managed by one person.

### **4. Site Resources**

Each site requires a person who assumes responsibility for organising and co-ordinating activities at a site, before, during and after the contest.

Before the contest, the site co-ordinator needs to source and train volunteers and local judges who will help during the contest proper as well as co-ordinate and manage technicians. They organise refreshments for during the contest as well as at the prize-giving ceremony, arrange for local support and prizes. Often they need to reorganise/manage hardware/computer labs and provide software for loading. Signs must also be prepared.

On the contest day, signage must be erected with directions to labs, refreshments and other facilities. The site co-ordinators will introduce the contest to contestants, manage the practice sessions and announce results and present prizes should a sponsor not be available. It is very important that they acknowledge the role of the volunteers at the site and also acknowledge the contribution of local and national sponsors and the judging team.

Post contest the site co-ordinator will publicise local winners, perhaps by updating the local website and/or giving copy to institutional newsletters. All labs and hardware then need to be restored. If certificates of participation

have not been already distributed on the contest day, then these must be processed.

All NZPC sites act independently and there are no quality standards that must be met.

The YWPC site would need to develop these resources.

## **5. Rewards**

There are tangible and intangible rewards to all stakeholders in the contest.

Tangible rewards for contestants take the form of certificates, trophies, spot prizes, scholarships, sponsors' gifts, financial grants and free entry to international competitions etc. A tangible reward for coaches, is that they may make valuable contacts with industry, via the sponsors. This contact is valuable for potentially providing sabbaticals, guest speakers, industry projects for students and research projects for themselves.

As well as rewarding contestants, their coaches and their schools, it is also appropriate to reward and acknowledge the long/distinguished service of volunteers.

The sponsor's role must be acknowledged to contestants as well as in any publicity as this helps create an on-going relationship.

Intangible rewards also result. For contestants, a sense of achievement, learning and technical skill development and the experience of being part of a wider programming community. The facilitators may experience collegiality with like-minded people, networking with fellow enthusiasts and recognition of the fact that, without their contribution, the teams and students would not have had this experience. In addition to experiencing a 'feel good' factor, they are also able to network with peers and informally market their courses and institutions. Lastly, broader philosophical aims may also have been met, for example, the belief in empowering young women in a technical subject.

The only tangible reward for NZPC contestants is a certificate which they may receive if their coach chooses to print it and a trophy for the overall winning team.

The YWPC had monetary prizes for contestants, plus scholarships. Intangible rewards were considerable as volunteers shared the philosophical belief that they were making a difference in advancing young women in the field of IT.

## **6. Problem sets**

Having a rigorous (i.e. accurate, unambiguous, succinct, complete and solvable) problem set is the key to a successful contest. If the base philosophy is that all teams must be able to achieve at least one successful solution then this sets the lower end of the problem range in respect to difficulty. The upper end is set so that the better performing teams do not finish all problems before the contest has ended.

It is also important to make sure that there are appropriate numbers of problems for each section of contestant so that they experience some

success. The step between the categories must not be so large as to demotivate teams from attempting the next level of difficulty. Multiple part questions benefit the contestants and the judges as it reduces the ratio of time spent understanding the problem to solving the problem. Where a group of problems build on each other there is the issue that precursors must be correct. Where partially correct answers are awarded points, consistent judging requires a well formed marking scheme. For some this proves difficult as there is some ambiguity possible. Distribution to sites of the problem set and judging test data needs to be secure. Post contest, the problem set should be made available for practice purposes.

The NZPC currently has a judging team with no head judge as he has recently retired. The head judge takes responsibility for the problem set being written and distributed and for collating and posting results on the website.

The YWPC initially drew on colleagues in the institution to provide the problem set and model answers but the champion took the head judge role.

## **7. Judging System**

Judging systems may be manual or automated. Tasks include accepting submissions, date and time stamping them, judging (whether manual or automated), recording of results, collating site results, (and, if multiple sites, collating all sites' results) and determining overall rankings.

Post contest analysis of the number of submissions per problem, the number of attempts per problem and other statistics, provide helpful feedback to the judging team.

An automated system has the benefit of being able to be replicated over multiple sites. With automated judging, consistency between sites is ensured.

Some sites of the NZPC use the ACM produced PC<sup>2</sup> software for judging, others do not. Results are collated manually.

All YWPC submissions were accepted and judged manually.

## **8. Training Opportunities**

An archive of previous contest problems must be available for team practice prior to the contest.

A more sophisticated system allows teams to submit solutions to a web-based problem pool, where the solution is judged electronically and results returned.

A variation of this is a simulation based on an actual programming contest. For example, when a team in training makes a submission to the automated judge, they see the results of the original contest and how their submission compares.

The NZPC makes the problem set and the judges' test data available on the website after the contest. An archive of past year's problems are also made available.

The YWPC provided a day of teaching and practice prior to the contest. Schools which entered teams several years in a row were able to build up their own archive.

## **9. Contestants**

A successful contest requires a steady stream of new and repeat contestants. There is more effort and cost expended in extending the contestant and institution base. This is necessary if the goal is growth of the contest. Capacity may become an issue and so a limit on the number of times a contestant may enter and/or a limit on the number of teams per institution becomes necessary.

Some contests require a team coach or a manager as the point of communication with a team.

The number of NZPC tertiary contestants is declining. No analysis has been done on whether contestants are new or have competed previously. Awareness of the contest in high schools must be assumed to be minimal.

In respect to the YWPC, by the last competition in 1998, many of the schools which competed had also competed in 1988. This indicates that the contest was meeting a need and that the model was durable.

## **Possible developments for the New Zealand Programming Contest**

The NZPC is not growing. It has the capacity to grow its sites and with more central support, this would be a more attractive proposition for institutions. In particular they can grow their high school section.

Marketing is central to growth. IT and Computer Science lecturers/teachers may need additional skilled support to manage marketing effectively, particularly in seeking and publicising sponsors. Sponsors supply additional funding which can be used for expansion - for prizes, celebrations, publicity and building the contest profile generally. This in turn will then attract more volunteers so that it can broaden its resource base.

Details of the contest could be more widely advertised in the appropriate teacher/school publications. This could take the form of a 'human interest' story where a group of contestants could relate their experiences or it could be an advertisement calling for contestants or alternatively it could be a call for expressions of interest from those who might be prepared to administer a site or to contribute to the problem set. This would grow the contest from the 'bottom up'.

The idea of having a separate section for high school girl's teams in the New Zealand Programming Contest was mooted with the top team also winning a trip to the SEARRC contest – but it was felt that this affirmative action was, in fact, disparaging to young women – they might be perceived as not being

competitive in their own right. Currently the school section does not require all contestants in a team to be from the same school. Consideration could be given to creating a same-school section and a mixed gender section.

Some elements of infrastructure would need to be developed/upgraded. The website could be developed to show site specific details and resources, to accept entries to a specific site and to bill teams/schools. An extranet which only site co-coordinators would have access to would mean that bulk emails, certificate printing, artwork etc would be readily available. To ease the load on site co-ordinators and to ensure consistency, a purpose built judging and results system would simplify contest operations on the day.

## **Possible developments for the Young Women's Programming Contest**

Recent research (Sanders 2003) indicates that boys are nine times more likely to enter tertiary computer science courses than girls so providing an opportunity to encourage young women into the area is still necessary.

Developments could take many forms but the first step must be to secure sponsorship and a source of funding to develop the contest. There is a tertiary institution which has expressed interest as they feel that it would provide them with access to a source of students. They would function effectively as local sponsors but if the vision is to expand the contest nationally and internationally, then several sponsors would be required.

To restart the contest in its current form would be relatively simple. Contact could be made with past competing schools and with other local (Auckland) schools who are teaching programming to investigate their level of interest. If it was substantial and a team of volunteers could be found who were willing to participate in the contest organisation, then a site would need to be found, a contest date set and then advertised.

If the focus however was to be in establishing a national/international contest, then the base would need to be widened to include institutions in other population centres. It would require the establishment of a national body to determine the expansion model e.g. simultaneous contests on the same date or several regional contests possibly culminating in a national contest.

It would be beneficial to liaise with the curriculum development project (Fluency in IT) that is currently being considered, to determine whether it is appropriate to link the contest to the proposed curriculum. The contest may qualify towards a recognised qualification-based academic endeavour.

## **Other potential developments and contests**

1. Design a business oriented programming/systems contest. This would be based on a simplified Software Development Life Cycle, including

Planning/Problem Analysis, Design, Implementation and Testing/Changes with a presentation to a 'user'. This model was developed by Sherrell and McCauley (2004). This could be extended to include debugging / extending / documenting existing code, designing / critiquing test data.

2. Link the curriculum with a poster/science fair type competition where students choose and research an aspect of IT. This idea has been developed by Fitzgerald and Hines (1996).
3. Explore the concept of after school / school holiday / summer camp type courses. These may be single sex and focus on a web solution. This could be similar to the 'POWER PROGRAM' (Programming of the **Web Rocks**) developed by Pollock, McCoy et al (2004).

## **Conclusion**

The two models of programming contests presented in this paper have the capacity to meet the needs of high school programmers. The Young Women's Programming Contest has the additional advantage of allowing girls to have a 'taste' of programming without the possible distraction of more dominating males. To ensure the longevity of any programming contest however, there are several components that must be in place, the most important being a champion who volunteers to provide this opportunity to students on an on-going basis. Currently the New Zealand Programming Contest is functioning adequately although there are opportunities for it to be expanded and developed further. The concept of a Young Women's Programming Contest is still considered valid and worthy of being reinstated. There are also opportunities to explore further types of contests targeting high school students.

## Bibliography

Conlon, M.,(2005). Rocktest: A programming contest management system, Journal of Computing Sciences in Colleges, Volume 20 Issue 5, Retrieved 10th September 2005 from:

<http://portal.acm.org/citation.cfm?id=1059892&coll=ACM&dl=ACM&CFID=56840308&CFTOKEN=27906109>

Costain, G., (1999). "Benefits of holding Young Women's Programming Contests?", Living Science Conference 1999.

Fitzgerald, S. & Hines, M.L., (1996). The computer science fair: an alternative to the computer programming contest. ACM SIGCSE Bulletin, Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education SIGCSE '96, Volume 28 Issue 1, Retrieved 10th September 2005 from:

<http://portal.acm.org/citation.cfm?id=236581&coll=ACM&dl=ACM&CFID=56840308&CFTOKEN=27906109>

Pollock, L. McCoy, K., Carberry, S., Hundigopal, N., You, X., (2004). Increasing high school girls' self confidence and awareness of CS through a positive summer experience. Proceedings of the 35th SIGCSE technical symposium on Computer science education,36, 185 - 189 Retrieved 10<sup>th</sup> September 2005 from:

<http://portal.acm.org/citation.cfm?id=971369&coll=ACM&dl=ACM&CFID=56840308&CFTOKEN=27906109>

Sherrell, L., & McCauley, L., (2004). A programming competition for high school students emphasizing process Proceedings of the 2nd annual conference on Mid-south college computing MSCCC '04. Retrieved 10th September 2005 from:

<http://portal.acm.org/results.cfm?coll=Portal&dl=Portal&CFID=58641904&CFTOKEN=27992559>

Sanders, J., (2003). Teaching Gender Equity in Teacher Education, Education Digest, Vol. 68: 25-29

## Appendix 1: Sample NZPC problem set

### New Zealand Programming Contest 2005

#### Preamble

Please note the following very important details relating to input and output:

1) Read all input from the keyboard, i.e. use `stdin`, `System.in`, `cin` or equivalent. Input will be

redirected from a file to form the input to your submission.

2) Write all output to the screen, i.e. use `stdout`, `System.out`, `cout` or equivalent. Do not write

to `stderr`. Do NOT use, or even include, any module that allows direct manipulation of the screen,

such as `conio`, `Crt` or anything similar. Output from your program is redirected to a file for later

checking. Use of direct I/O means that such output is not redirected and hence cannot be checked.

This could mean that a correct program is rejected!

3) Unless otherwise stated, all *integers* will fit into a standard 32-bit computer word. If more than one integer

appears on a line, they will be separated by white space, i.e. spaces or tabs.

4) Unless otherwise stated, a *word* is a continuous sequence of lower case letters without any punctuation

or other characters and, in particular, without intervening white space. As with integers, successive

words will be separated by white space.

6) Unless otherwise stated, a *name* is a continuous sequence of letters without any punctuation or other

characters and, in particular, without intervening white space. As with words, successive names will be

separated by white space.

7) Unless otherwise stated, a *string* is a continuous sequence of characters without any intervening white

space. As with words, successive strings will be separated by white space.

8) If it is stated that 'a line contains no more than  $n$  characters', this does not include the character(s)

specifying the end of line.

9) All input files are terminated by a 'sentinel' line, followed by an end of file marker. This line should

not be processed.

#### Program Names

These will be site specific.

## Problem A - Who's First?

3 points

In this problem you are given a number of sets of words. In each set, you are to determine which word comes first alphabetically. You should ignore case, so "apPle" will come before "Bat" but after "AnT", for example.

Each set starts with a single integer,  $n$  ( $2 \leq n \leq 1000$ ), which gives the number of words in the set.

The next  $n$  lines contain one word(20) per line. A set will never contain two words that differ only in case. The last line of input will contain 0 on its own – this line should not be processed.

For each set of words, output just the word that comes first alphabetically, on a line by itself, as it appears in the input.

### Sample input

```
3
Cat
fat
bAt
4
call
ball
All
Hall
0
```

### Sample output

```
bAt
All
```

## Problem B - Rugby Club

3 points

The All Golds Rugby Club categorises members as Senior or Junior. Anyone over 17 is a Senior, as is anyone weighing 80 kg or more. Everyone else is a Junior. Your task is to correctly classify club members.

Each line of input contains a name followed by two positive integers representing age and weight, in that order. The last line of input will be # 0 0. This line should not be processed.

For each line of input you must output a single line containing the name, followed by a space and the appropriate category. Note that the category must begin with an upper case letter.

### **Sample input**

```
Joe 16 34  
Bill 18 65  
Billy 17 65  
Sam 17 85  
# 0 0
```

### **Sample output**

```
Joe Junior  
Bill Senior  
Billy Junior  
Sam Senior
```

## Problem C - Your Days Are Numbered 3 points

There are many situations where we need to calculate the number of days between two dates, for instance, how many days leave have you taken between the start and end of your holiday. The easiest way to do those calculations is to assign an integer number to each day by choosing a suitable starting date and counting the number of days since then. For this problem we will use the first of January of the relevant year as the starting date.

This means that the day number of today (13th August 2005) is 225. Last year, the day number of 13th August was 226 because 2004 was a leap year. A leap year occurs when the year is divisible by 4 with the exception that years divisible by 100 are not and with the further exception that years divisible by 400 are, thus 2000 and 1976 were leap years, but 1900 and 1977 weren't.

Hopefully you also know that most months have 31 days, except for April, June, September and November which have 30, and February which has 28 (29 in a leap year).

Input will consist of a number of dates each on a separate line. Each date will consist of three positive integers representing the day, month and year respectively. The day and month will always be valid and the year will always lie between 1800 and 2200 (both dates inclusive). Input will be terminated by a line containing 0 0 0 — this line should not be processed.

Output will be one line for each line of input, the line containing just the day number.

### Sample input

```
14 8 2004
1 1 2004
31 1 1976
1 3 1974
1 3 1976
0 0 0
```

### Sample output

```
227
1
31
60
61
```

## Problem D - Digit Sums

3 points

A simple operation that you can perform on a number is to add up its digits. If the result has more than 1 digit, the process may be repeated until a single digit answer is given. For example, applying the operation to 673 gives 7;  $6 + 7 + 3 = 16$  and  $1 + 6 = 7$ .

Input will consist of a number of lines, each containing a single positive number less than 100,000. The last line of input will contain 0 – this line should not be processed.

Output will consist of a line containing the digit sum (a single digit number) for each line of input.

### Sample input

```
673
51
1000
99
0
```

### Sample output

```
7
6
1
9
```

## Problem F - Super 12

10 Points

In a rugby-mad country such as New Zealand, fans eagerly await the ranking of the teams in league tables such as the Super 12 (soon to be the Super 14). These rankings are really only meaningful at the end of a round, when all teams have played the same number of games.

A team is awarded 4 points for each win, 2 points for each draw and 0 for each loss. Any team scoring 4 or more tries in a game is awarded a bonus point, as is any team that loses a game by less than 8 points. At the end of a round we can produce a table showing the relative standings of each team, i.e. a table sorted in descending order of points. If two or more teams have the same number of points, then apply the following rules there are no tied teams:

Sort in descending order of spread, i.e. the cumulative difference between total points scored and total points against.

Sort in descending order of the total number of tries they scored.

Sort in ascending alphabetic sequence.

Input will consist of the results for a single league and will consist of the names of the competing teams and details of each game. The list of competing teams will be a series of no more than 20 lines each containing a single name(20) and terminated by a line consisting of a single '#'. This will be followed by a series of lines giving the results of each game, also terminated by a line consisting of a single '#'. Each game will start with the names of the two teams (home side first) followed by 4 integers representing, in order: the score of the home team, the score of the away team, the number of tries scored by the home team, and the number of tries scored by the away team (see the sample input). Note that there will not be any indication of the end of a round, it is determined by the number of competing teams (which will always be even), i.e. a round will end when every team has played exactly one game.

Output will consist of a league table for each round. The first line of the league table will consist of the word 'Round' followed by a space and the number of the round (a running number starting at 1). This will be followed by the teams listed in order of their standing, according to the rules outlined above. Each line consists of the name of the team, followed by, starting in column 22 and right justified in fields of widths as specified in parentheses, their points (2), their cumulative score since the beginning of the league (4), the cumulative score of the teams they have played (4), the total number of tries that they have scored (3) and the total number of tries scored against them (3). Leave a blank line between rounds. See the sample output below.

### Sample Input

```
Highlanders  
Crusaders  
Warriors
```

```
ThisIsTheLongestName
#
Highlanders Crusaders 25 24 3 2
Warriors ThisIsTheLongestName 32 16 5 1
ThisIsTheLongestName Highlanders 10 20 1 2
Crusaders Warriors 17 16 3 3
```

```
#
```

### **Sample Output**

```
Round 1
```

```
Warriors 5 32 16 5 1
Highlanders 4 25 24 3 2
Crusaders 1 24 25 2 3
ThisIsTheLongestName 0 16 32 1 5
```

```
Round 2
```

```
Highlanders 8 45 34 5 3
Warriors 6 48 33 8 4
Crusaders 5 41 41 5 6
ThisIsTheLongestName 0 26 52 2 7
```

## Problem G - GPS Encoding

10 Points

Traditionally, simple codes have taken a permutation of the alphabet, numbered each character in the range 01 to 26 (or 00 to 25) and then encrypted the message as a string of digits. It is then relatively easy to conceal the structure of the text by breaking the string into groups of a fixed length.

This problem turns that encoding around and assumes that one has a sequence of digits that one wishes to encrypt (possibly a GPS location that you want to transmit to friend, telling her where a treasure is located). We could do this the simple way and use only 10 characters to encrypt the 10 digits, or we could use all 26 letters and use the additional letters to encrypt suitable pairs of digits.

Thus the sequence 941177 could be encoded as

9 4 1 1 7 7 or as

9 4 11 7 7 or as

9 4 1 17 7.

Given a permutation of the upper case letters (which implicitly defines an encoding of the numbers (0)0, (0)1, ..., (0)25) and a sequence of digits, determine the shortest encryption of the sequence as a sequence of letters. Note that decryption of such a sequence may not be unique. Input will consist of a series of encryption problems. Each problem will begin with a permutation of the uppercase letters, followed by a series of lines each containing a string of between 3 and 20 digits. The string of digits will be terminated by a single zero ('0') on a line by itself. The series of problems will be terminated by a line consisting of a '#' on a line by itself. Neither of these lines should be processed.

Output for each problem will consist of a line starting with 'Problem ' followed by the problem number, a running number starting at 1. This will be followed by a series of lines, one for each digit sequence in the input for that problem, giving the shortest encoding for that sequence. If there are several such encodings, then choose the lexicographically greatest (i.e. the one that would appear nearest the end of a dictionary if they were to be considered as words). Leave a blank line between successive problems.

### Sample Input

```
ABCDEFGHIJKLMNPOQRSTUVWXYZ
111
234
0
CUGEBRHKNXDMIQSLTFVWOYPJZA
111213
11121
212223
0
#
```

## Sample Output

Problem 1

LB //Could be BBB, BL or LB. Choose LB as shortest and lexicographically greatest

XE //CDE is longer

Problem 2

MIQ

UMY

YPJ

## Problem H - Custom table sorter

10 Points

Web site users are often presented with data in tables. Different users may want table rows displayed in different orders. For a web site listing available hotels, possible orders include hotel name, hotel locality, hotel star rating and room rate. For this problem you are to write a piece of support software for possible inclusion in such a site.

Input will consist of a number of data sets. Each data set consists of a heading line, a table section and a sorter section. The heading line contains the title of the data set. The sequence of data sets is terminated by a (heading) line consisting of a single '#'. A table section consists of at least 1 and no more than 20 lines, terminated by a line consisting of a single '#'. Each line contains between 1 and 10 fields, separated by commas; each field contains a string(20). All lines have the same number of fields and there are no empty fields.

A sorter section consists of several sorter lines. Each sorter line contains one or more field sorters separated by commas, each consisting of a field number (a distinct number in the range 1 to the number of fields), and a direction ('A' or 'D'). A sorter section is terminated by a line consisting of '0#'.

The output starts with the title of the data set, followed by several groups of lines, indented two spaces and separated by a blank line between groups. Each group consists of the contents of the table section, sorted according to the corresponding sort specification. Sorting is primarily done based on the first field sorter, and the second and subsequent field sorters are only used for those

rows with the same value(s) in the field(s) used by previous field sorter(s). If there are still ties (equal elements), the tied elements should appear in the order of the original table. Leave a blank line between the output for successive data sets.

### Sample input

```
Hotel rooms and locations
Lowton_Hotel,Airport,**
Hotel_foobar,CBD,*
Dug_Inn,Airport,*
#
3A,1D
2A
0#
General enquiry
One_value,here
#
1A
0#
#
```

### Sample output

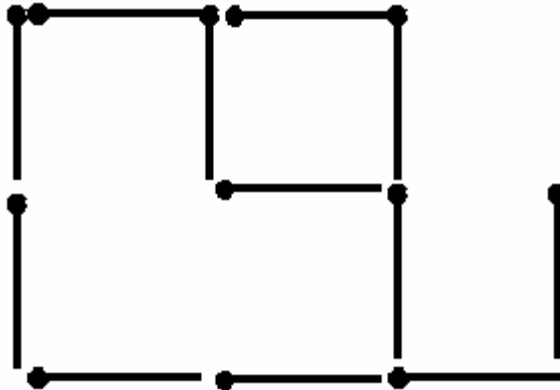
```
Hotel rooms and locations
```

Hotel\_foobar,CBD,\*  
Dug\_Inn,Airport,\*  
Lowton\_Hotel,Airport,\*\*  
Lowton\_Hotel,Airport,\*\*  
Dug\_Inn,Airport,\*  
Hotel\_foobar,CBD,\*  
General enquiry  
One\_value,here

## Problem I - A game with matches

30 Points

Consider a grid of matches on a table, like so:



The problem involves counting the number of squares formed by these matches. The arrangement above, for example, forms two squares. Input consists of a sequence of arrangements. Each arrangement starts with a line containing two integers representing the number of rows and columns in the arrangement, ( $r$  and  $c$ ,  $1 \leq r, c \leq 20$ ), followed by  $2r+1$  lines representing the arrangement. Odd numbered rows will consist of  $c$  characters where each character is either a hyphen ('-'), representing a horizontal match or an asterisk ('\*'), representing a space. Even numbered rows will consist of  $c+1$  characters where each character is either a bar ('|'), representing a vertical match, or an asterisk ('\*'), representing a space. Input will be terminated by a line containing two zeroes (0 0).

Output will consist of one line for each arrangement giving the number of squares that can be found in the arrangement followed by a space and the word 'squares'.

### Sample Input

```
2 3
--*
|||*
*-*
|*||
---
2 2
--
|||
--
|||
--
0 0
```

### Sample output

2 squares  
5 squares

## Problem J - Checking Causality

30 Points

The concept of causality is important when monitoring distributed systems. Consider the event of one computer sending a message and the event of that message being received at another computer.

Causality tells us that the send event must occur before the receive event. If there were a global clock to timestamp all of the events that occur in a distributed system, then the timestamp of a message receive event would always be greater than the timestamp of a message send event.

However, distributed systems do not have such a global clock, all they have is a collection of idiosyncratic clocks, one in each computer, all ticking at different speeds. All we can say for certain is that each clock never stops and that its (apparent) granularity is fine enough that any sequence of events on a given computer will be given a series of timestamps that is strictly monotonically increasing.

Thus it is perfectly feasible to have a message that left A at time 50 arrive at B at time 40 and for the reply to leave B at 45 and arrive at A at 51, where all times are given according to the relevant computer's clock.

Given send and receive timestamps from a collection of messages exchanged by a group of computers, it is possible to use the concept of causality to make some checks that will detect faulty clocks. Continuing the example above, assume that a second message is sent from B at time 45 on B's clock (i.e. after the arrival of the first message). If A receives this before 51 (on its clock) then at least one clock is faulty, because causality has been breached (apparently), since we know that all clocks increase by at least 1 tick between successive events on the same computer. It is also possible to detect causality violations in longer chains, e.g., A to B, B to C, C to A.

Write a program to process sets of message timestamp data, and for each set say whether causality is violated, i.e. whether there is a cycle of messages, starting and ending at the same computer, where one or more messages has apparently travelled back in time.

Input consists of a number of timestamp sets. Each timestamp set starts with the number of messages in the set ( $n$ ), followed by  $n$  lines of timestamp data, where each line contains: sending computer, send time, receiving computer, and receive time, all separated by whitespace. Times are according to the relevant computer, computers are identified by a single upper case letter and times are integers. The sequence of timestamp sets will be terminated by a line containing a single zero (0).

Output consists of a single line for each timestamp set. This line contains the word 'OK' if there is no causality violation or the word 'Bad' if there is such a violation.

### Sample input

```
2
A 50 B 40
B 45 A 49
2
A 50 B 40
```

B 45 A 51

2

A 50 B 40

B 39 A 49

0

### **Sample output**

Bad

OK

OK

## Problem K - Olympic Ranking

30 Points

Every year since 1896, the nations of the world have competed in the Olympic Games for glory, medals and (particularly in recent years) media coverage. Every day eager fans will scan the medal table to see where their team is placed, which, particularly for teams from smaller countries, usually means badly. Part of the problem is that the medal table is sorted strictly in the order: number of gold medals, number of silvers, number of bronzes. This means that a team with 6 silvers and 10 bronzes but only 1 gold is placed well behind a team with 2 golds and nothing else. The situation could be alleviated by allocating a value to each medal type, calculating a total score for each team by multiplying the numbers of each medal type they have won by the appropriate value and then ranking the teams on that. This then raises the problem of determining the values, with the obvious proviso that a gold has to be worth more than a silver which, in turn, has to worth more than a bronze. To keep numbers manageable (remember we are dealing with the general population, not Computer Science students), values must be in the range 1 to 99.

For this problem you will be given a medal table, that is, a list of countries together with the numbers of medals they have won, from which your program is to determine, for each country in the table, what allocation of values to medal types will give that country the best possible placing, where the placing of a country is the number of rivals with a larger score, plus 1.

Input will consist of several scenarios (Games). Each will start with the name of the city where the Games were held, followed by an unsorted sequence of lines containing the name of a country followed by 3 integers representing the number of gold, silver, and bronze medals won by that country, and terminated by a line containing a single '#'. There will be no more than 100 teams in any one Games and no team will win more than 1000 medals. The sequence of Games will be terminated by a line consisting of only a single '#'. Output (for each scenario) will consist of a header line followed by a line for each team in the input in the order they appeared in the input. The header line will consist of 'Olympic Games in ' followed by the name of the relevant city. Each team line will consist of the name of the team followed by 4 integers, separated by single spaces, giving the medal values that results in their highest placing, and that placing. Remember that the values must be such that  $\text{gold} > \text{silver} > \text{bronze}$  and that they must be in the range 1 to 99. If there are several allocations of values that produce the same best placing, then choose the one that produces the smallest 6 digit number  $ggssbb$ , where  $gg$ ,  $ss$ , and  $bb$  are the relevant values (including leading zeroes if necessary). Leave a single blank line between scenarios (Games).

### Sample Input

```
Rome
UnitedStates 100 200 300
China 102 198 297
NewZealand 1 2 3
#
Auckland
```

```
UnitedStates 100 200 300
China 102 198 301
NewZealand 1 2 3
```

```
#
#
```

### **Sample Output**

```
Olympic Games in Rome
UnitedStates 3 2 1 1
China 4 2 1 1
NewZealand 3 2 1 3
Olympic Games in Auckland
UnitedStates 3 2 1 2
China 3 2 1 1
NewZealand 3 2 1 3
```

## Problem L - Bus Timetable

30 Points

A typical bus timetable has a column of stops down the left hand side, followed by a series of columns specifying a particular service and labelled with the number of a bus route. Each entry in the table, i.e. each intersection of the row for a given stop and a column for a given service, will be either blank or contain the time that that service is scheduled to leave that stop. As you can imagine, producing these timetables by hand is very difficult and error prone, so this is where you come in. Write a program to produce a timetable, given details of the various routes and services.

Input will consist of a number of scenarios. Each scenario will start with the title of the scenario, followed by a number of lines, one for each route in the scenario. The sequence of scenarios will be terminated by a line consisting of a single '#'.

The line for a route will start with the time that the first service for that route leaves the depot (hours and minutes) followed by the interval between services (minutes). This will be followed by a series of pairs of integers representing travel times and bus stops. The list of routes will be terminated by a line containing two zeroes (0 0) and will never contain details of more than 10 routes. Note that routes are numbered implicitly, starting from 1, bus stops are also numbered

from 1 and are always visited in order (apart from the return to the depot), there will never be more than 99 stops, and that the depot is effectively bus stop 0 (thus the last bus stop number in the list will always be 0). Note that no buses leave before 6:00 am and all services terminate (i.e. are back in the depot) by midnight, thus you should not generate a service that violates these constraints.

Output will consist of one timetable per scenario. Each timetable will start with the name of the scenario on a line by itself, followed by a heading line detailing the services (sorted in order of departure time from the depot, see example) followed by as many lines as there are bus stops mentioned in the input. Each line starts with the number of the stop (two columns) followed by an entry for each service. Each entry is 6 columns wide and starts with a '|' followed either by 5 spaces (if that service does not stop there) or a departure time in the form hh:mm, using a 24 hour clock (i.e., including leading zeroes). The line is terminated by a '|'. Note that times increase monotonically downwards, but not necessarily across. Leave a blank line between scenarios.

### Sample Input

```
MadeUpAsIWentAlong
7 0 240 20 1 10 2 20 3 10 0
9 0 240 20 1 10 2 20 3 10 0
0 0
#
```

### Sample Output

```
MadeUpAsIWentAlong
| 1| 2| 1| 2| 1| 2| 1| 2| 1|
1|07:20|09:20|11:20|13:20|15:20|17:20|19:20|21:20
|23:20|
```

2 | 07:30 | 09:30 | 11:30 | 13:30 | 15:30 | 17:30 | 19:30 | 21:30  
| 23:30 |  
3 | 07:50 | 09:50 | 11:50 | 13:50 | 15:50 | 17:50 | 19:50 | 21:50  
| 23:50 |

## Problem M - Conflicting Strings

100 points

In 2417 archaeologists discovered a large collection of 20th century text documents of vital historical importance. Although there were many duplicated documents it was soon evident that, as well as the damage due to time making much of the text illegible, there were also some disagreements between them. However, it was noticed that groups of texts could be made consistent, i.e.

consistency between texts could be achieved by leaving out some (small) number of texts. For example, the texts:

```
ap***  
ab*le  
app*e  
*p**e
```

(where \* denotes an illegible character) can be made consistent by removing just the second text.

Input will consist of a sequence of sets of texts. Each set will begin with a line specifying the number of texts in the set, and the maximum number of texts which can be removed. This will be followed by the individual texts, one per line. Each text consists of at least one and no more than 250 characters, either lower case letters or asterisks. All the texts in a set will be the same length and there will be no more than 10,000 texts in a set. The sequence of sets is terminated by a line containing two zeros (0 0).

Output for each set consists of a line containing one of the words 'Yes' or 'No' depending on whether or not the set can be made consistent by removing at most the specified number of texts.

### Sample input

```
4 1  
ap***  
ab*le  
app*e  
*pple  
3 1  
a  
b  
c  
4 2  
fred  
ferd  
derf  
frd*  
0 0
```

### Sample output

```
Yes  
No  
No
```

## Problem N

100 Points

This problem has been withdrawn for technical reasons

## Problem O - Adventure Game

100 points

Part of an adventure game requires the adventurer to make her way through a magical maze of numbered rooms. Each room has a set of numbered doors that lead to the indicated rooms. Additionally, each room might contain either a leprechaun or a troll.

Whenever the adventurer visits a room containing a leprechaun, the leprechaun will ensure that she leaves with at least a certain number of gold pieces (GP) in her sack. That is, if she arrives with fewer GP than the prescribed amount, then the leprechaun will “top her up” to that amount; however, if she arrives with the prescribed amount or more, then her supply of GP will remain unchanged.

Each time the adventurer visits a room containing a troll, the troll will demand a toll of a certain number of gold pieces which must be paid before continuing. The adventurer begins with 0 GP and the problem is to determine whether the adventurer can make her way from room 1 to the highest numbered room. If the highest numbered room contains a troll, the adventurer must also be able to pay the toll on arrival.

Input will consist of a sequence of mazes. Each maze will begin with a line containing an integer specifying the number of rooms in the maze ( $n$ ,  $1 \leq n \leq 1000$ ). This will be followed by  $n$  lines specifying the rooms in ascending order. Each line will specify the contents of the room (empty (E), a leprechaun (L) or a troll (T)), followed by a number specifying the number of GP that the leprechaun will top up to, or that the troll will require from the adventurer. (For empty rooms this amount is 0.) The rest of the line consists of a list of 1 or more numbers in the range  $1..n$  and terminated by a zero (0), representing the numbers on the doors of that room. The sequence of mazes is terminated by a line containing a single zero (0).

Output will consist of a sequence of lines, one for each maze — either “Yes” or “No” indicating whether it is possible to reach the final room from the first one.

### Sample input

```
3
E 0 2 0
L 10 3 0
T 15 1 2 0
4
E 0 2 3 0
L 201 2 3 0
L 10 4 0
T 15 2 3 1 0
0
```

### Sample output

```
No
Yes
```

## Problem P - I'm a Frayed Knot

100 points

On the table in front of you lie a bunch of pieces of coloured thread. All the threads are different colours and the ends have been arranged in a single line, for example red, blue, green, green, blue, red. Note that each colour appears exactly twice, once for each end of that thread. You've been asked to tie the threads into a single large loop by successively tying together ends of some pair

of adjacent threads. In the above example you could start by tying end 1 to end 2 or end 2 to end 3 but not end 3 to end 4 since that would make a green loop. Likewise, if your first tie was red to blue, then your second tie could not join the remaining red and blue.

Finding the job a little boring, you decide instead to count the number of ways in which you could perform the task, i.e. the number of sequences of ties that you could do. For instance, suppose that the initial pattern was `rrbb`, then there is only one allowed sequence of ties (join the middle two, then join the two remaining). On the other hand if it were `rbrb`, then there are three allowed

sequences (join any consecutive pair to begin with, then join the remaining two).

Likewise you can see that if the initial pattern were `rgrbgb`, then four of your initial ties lead to a subsequent pattern of the general form `abab`, while one leads to `abba`. Thus there are  $4 \times 3 + 1 \times 2 = 14$  ways to complete the ties in this case.

Input will consist of a sequence of colour patterns represented by words of length at most 22 consisting of lower case letters and will be terminated by a line containing the single character '#'.

Output will be a sequence of lines, one for each line in the input, each containing a the number of ways to tie off the pattern in the corresponding input line. If an input line is invalid (because it does not contain exactly two occurrences of each of its characters) the output for that line should be 0.

### Sample input

```
rrbb
rbrb
rgrbgb
gbg
#
```

### Sample output

```
1
3
14
0
```

## Appendix 2: Sample SPLAT programme

\*\*\* SPLAT ASSEMBLER Version 2.3. 18/11/88 11:50:49

SPLAT Source File : Example 1

LINE	LABEL	OP-CODE	OPERAND	COMMENTS
1	*	1	2	
2	*2	7		
3	*LABL	OP_CODE	OPERAN	<- COMMENTS ->
4		PRINT		"PLEASE ENTER WIDTH (CM) "
5		IN		
6		STORE	WIDTH	
7		PRINT		"PLEASE ENTER DEPTH (CM) "
8		IN		
9		STORE	DEPTH	
10		PRINT		"PLEASE ENTER HEIGHT (CM) "
11		IN		
12		MULT	DEPTH	
13		MULT	WIDTH	
14		PRINT		" THE VOLUME OF THE BOX IS "
15		OUT		
16		PRINT		" CUBIC CM"
17		HALT		

There were 0 errors found

PLEASE ENTER WIDTH (CM)?	3
PLEASE ENTER DEPTH (CM) ?	4
PLEASE ENTER HEIGHT (CM)?	5
THE VOLUME OF THE BOX IS	60 CUBIC CM

No. of instructions executed = 13

## Appendix 3: Sample YWPC problem set

AUCKLAND INSTITUTE OF TECHNOLOGY

FACULTY OF COMMERCE

### YOUNG WOMEN'S PROGRAMMING CONTEST

28 September 1997

The following questions may be attempted in any order, but ONLY beginners may attempt questions 1 to 4. Beginners may also attempt questions from the advanced section.

Hand in printouts for marking as soon as they are completed. Name the files in which your results are stored by the question number e.g. Q1A for question 1 part (a). Each answer MUST be clearly identified. with your team name and the question number.

Use the test data given for the program run that you hand in. If there are two or more sets of test data for one section of a question, both runs MUST be handed in together. If the markers are doubtful about your program, the} will ask you to re-run the program with other data. Failure to do so will mean no marks for that question.

Good  
Luck!!

#### **(A) Beginners Section (Tier 1 contestants do NOT attempt this section)**

##### **1. Scoring at New Netball:**

**Basic Mark = 3**

(a) In the game of New Netball, one goal earns three points. Write a program which will accept the number of goals achieved and will calculate and display the number of points earned by those goals.

For example:

Enter goals? 3 Number of points = 9
--

Test Data: 3, 10, 0.

**Bonus Mark = 3**

(b) Modify the program in (a) so that it will reject the number of goals if it is less than zero F or example:

Enter goals? -1 Error! Number of goals must be zero or greater!
--

Test Data: 1, 0, -1, -7.

**Bonus Mark = 3**

(c) Modify the program in (b) so that if an error is detected, then the program will ask for that entry again - for example:

```
Enter goals? -5
Error! Number of goals must be zero or greater!
```

```
Enter goals? 5
Number of points = 15
```

or,

```
Enter goals? -5
Error! Number of goals must be zero or greater!
```

```
Enter goals? -1
Error! Number of goals must be zero or greater!
```

```
Enter goals? 5
Number of points = 15
```

Test Data:

```
0;
-5, -1, 5;
-10, 6;
7.
```

**Bonus Mark = 3**

- (d) Modify the program in (c) so that:
- (i) a heading is displayed at the beginning of the program, and
  - (ii) the program will continue to ask for the number of goals until an entry of -99 (minus ninety-nine) is made.
  - (iii) print out the message "End of Points Program" just before the program halts.

For example:

```
New Netball: Points for Goals
Enter goals? 3 Number of points =          9

Enter goals? -5
Error! Number of goals must be zero or greater!

Enter goals? -1
Error! Number of goals must be zero or greater!

Enter goals? 5
Number of points =          15

Enter goals? 0
Number of points =          0

Enter goals? 8
Number of points =          24

Enter goals? 4
Number of points =          12

Enter goals? -99

End of Points Program
```

Test Data: Use data in the sample screen for (d).

**Bonus Mark =**

**5**

(e) Modify (d) so that another prompt is displayed which asks for the number of penalty goals scored. Each penalty goal receives two points. The number of penalty goals must be zero or greater. If the number of penalty goals is incorrect, the program must start from the beginning again and prompt for the number of goals. Display the points achieved by the penalty goals as shown in the following example, plus display the total points (the sum of the points for the goals plus the points for the penalty goals):

New Netball: Points for Goals

```
Enter goals? 12
Number of points =      36
Enter penalty goals? 1
Number of penalty points =  2
Total points =      38
```

```
Enter goals?  -5
Error! Number of goals must be zero or greater!
```

```
Enter goals? -1
Error! Number of goals must be zero or greater!
```

```
Enter goals? 4
Number of points =      12
Enter penalty goals? -1
Error! Number of penalty goals must be zero or greater!
```

```
Enter goals?  4
Number of points =      12
Enter penalty goals? 3
Number of penalty points =  6
Total points =      18
```

```
Enter goals?      -99
End of Points Program
```

**2. Series:****Basic Mark = 3**

- (a) Write a program to list each number from 0 to 12 with its square and cube as follows:

Number	Square	Cube
0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	243
8	64	512
9	81	729
10	100	1000
11	121	1331
12	144	1728

**Bonus Mark = 3**

- (b) Modify your answer to part (a) to:
- print a heading message "Number, Square and Cube Program" with a line underneath;
  - accept a starting number and a finishing number for the range to be printed out.

For example:

Number, Square and Cube Program		
Enter start number: ?	7	
Enter end number: ?	13	
Number	Square	Cube
7	49	343
8	64	512
9	81	729
10	100	1000
11	121	1331
12	144	1728
12	169	2197

Test Data: Start = 2, End = 13.

**Bonus Mark = 4**

- (c) Modify your answer to (b) so that the start number line is printed and the next number is the start number plus 1. Each number will increase by the previous increase plus 1, so that the third number will be the second number plus 2 and the fourth number will be the third number plus 3. For example:

Number, Square and Cube Program
---------------------------------

-----		
Enter start number: ?		-7
Enter end number: ?		12
Number	Square	Cube
-----		
-7	49	-343
-6	36	-216
-4	16	-64
-1	1	-1
3	9	27
8	64	512

Test Data: -7,12  
-1, 14

**3. Invoicing Amounts Greater than the Cost of Postage                      Basic Mark = 3**

(a) Computer invoicing programs sometimes produce invoices for an amount which is less than or equal to the price of postage. Write a program to input a postage cost, then input an invoice amount. If the postage amount is greater than or equal to the invoice amount, print "Do not send bill". If the postage amount is less than the invoice amount, print "Send bill". For example:

Postage Comparison Program ----- Enter cost of postage in cents? 120 Enter invoice amount in cents? 1050  Send bill
--

and .....

<u>Postage Comparison Program</u> Enter cost of postage in cents? 130 Enter invoice amount in cents? 105  Do not send bill
--

See over page for test data.

Test Data:  
150, 150;  
110, 588;  
160, 120.

**Bonus Mark = 3**

(b) Modify part (a) so that the cost of postage is checked to ensure that it is greater than zero.

```
Postage Comparison Program
-----
Enter cost of postage in cents? 0
Error! Cost of postage must be greater than zero!
```

Test Data:  
0;  
-2;  
180,100;  
125,450.

**Bonus Mark = 2**

(c) Modify part (b) so that if the cost of postage is less than or equal to zero, then the prompt for the postage cost will reappear:

```
Postage Comparison Program
-----
-----
Enter cost of postage in cents? 0

Error! Cost of postage must be greater than zero!

Enter cost of postage in cents? 120
Enter invoice amount in cents? 1050

Send bill
```

**Bonus Mark = 3**

(d) Modify part (c) so that the cost of postage is entered once, at the start, and a number of invoice amounts are entered and compared. Input is to end when an invoice amount of -99 is entered.

For example (see over page):

Postage Comparison Program

-----  
Enter cost of postage in cents?        -90

Error! Cost of postage must be greater than zero!

Enter cost of postage in cents?        120

Enter invoice amount in cents?        1050

Send bill

Enter invoice amount in cents?        1290

Send bill

Enter invoice amount in cents?        105

Do not send bill

Enter invoice amount in cents?        120

Do not send bill

Enter invoice amount in cents? -99

End of Program

Test Data: use data shown in example.

**4. Sorting the Heights of Basketballers into Sequence**

**Basic Mark = 6**

(a) Write a program which asks for the heights, in centimetres, of three basketballers to be input in any sequence, then prints them in ascending order.

Heights of Basketballers

Enter first height?    1600

Enter second height? 1605

Enter third height?    1650

Sorted heights:        1600            1605            1650

Test Data:

same as in example

1720, 1685, 1690;

1695, 1775, 1650.

**Bonus Mark = 2**

(b) Only players 1.6 metres and over will be accepted to play basketball. Modify part (a) so that if a height below 1600 centimetres is entered, it is rejected and the user is prompted to enter another height.

Test Data:

1830, 1700, 1599, 1690;

1600, 1400, 1600, 1745;

1550, 1600, 1590, 1650, 1580, 1600.

**Bonus Mark = 2**

(c) Write a program which asks for the three basketballers' heights to be input and prints them in descending order. Heights of under 1.6 metres are to be rejected as in part (b).

Test Data: Use the same test data as for 4(b).

**See over page for Part B ....**

## (B) Experienced Programmer's Section

Tier 2 Beginners Section contestants may also attempt questions from this section.

Tier I Experienced contestants must **ONLY** attempt these questions.

### 5. Gross Profit on Lamb burgers

Basic Mark = 8

(a) A politician once promoted lamb burgers. A small stall was set up in North Harbour Stadium to sell them to the event audience. Write a program to (i) input the sale price for a lamb burger for the event in cents; (ii) input the quantity sales of lamb burgers for an event, using a -99 to indicate the end of input of sales for that event, then print out the total quantity sold and the total sales value in cents. See example:

Lamb burger Sales

-----  
Enter today's price for lamb burgers in cents? 95

Enter quantity sold? 4  
Price in cents = 380

Enter quantity sold? 2  
Price in cents = 190

Enter quantity sold? 4  
Price in cents = 380

Enter quantity sold? 1  
Price in cents = 95

Enter quantity sold? -1  
Price in cents = -95

Enter quantity sold? 2  
Price in cents = 190

Enter quantity sold? -99

Total quantity sold = 12  
Total value in cents = 1140

End of Event

Test Data:

use the test data in the above example.

Note that the negative value may be input to cancel a previous sale.

Also use the following data for a separate run:

Price of 85 cents

Quantity sales: 5, 3, -1, 2, 6, -99.

**Bonus Mark =**

5

(b) Modify part (a) to accept the prices and sales for several events. When all events have been entered, the total quantity sold for all events, and the total sales value for all events must be displayed. The number for the event will start at one (1) and will increase by one (1) for each event. For example:

Lamb burger Sales	
-----	
Event 1	
Enter today's price for lamb burgers in cents? 95	
Enter quantity sold? 4	
Price in cents =	380
Enter quantity sold? 2	
Price in cents =	190
Enter quantity sold? 2	
Price in cents =	190
Enter quantity sold? -99	
Total quantity sold at event 1 is	8
Total value in cents = 760	
End of Event	
Enter I for another event, or 0 for finish. Enter choice? 1	
Event 2	
Enter today's price for lamb burgers in cents? 90	
Enter quantity sold? 10	
Price in cents =	900
Enter quantity sold ? 2	
Price in cents =	180
Enter quantity sold? 4	
Price in cents =	360
Enter quantity sold? 2	
Price in cents =	180
Enter quantity sold? -99	
Total quantity sold at event 2 is	18
Total value in cents = 1620	
End of Event	
Enter I for another event, or 0 for finish. Enter choice? 0	
Total quantity sold at all events = 26	
Total value in cents for all events = 2380	

End of Program

Test data: Use the data in the example.

**Bonus Mark = 5**

(c) Modify part (b) so that the *total value in cents for all events* is displayed in dollars, instead of cents.

Test data: : use test data for previous part (b).

**Bonus Mark = 5**

(d) Modify (c) so that, for each event, the cost price for a lamb burger is entered prior to the "Enter today's price for lamb burgers in cents?" prompt. At the end of all input and display of totals, display the gross profit (total sales value - cost) as shown following:

Lamb burger Sales

```
-----
Event      1
Enter cost price of lamb burger? 85
Enter today's price for lamb burgers in cents? 95

.....
.....
Enter quantity sold? -99

Total quantity sold at event  1 is   8
Total value in cents =       760

Event      2
Enter cost price of lamb burger? 80
Enter today's price for lamb burgers in cents? 90

.....
.....
Enter quantity sold? -99

Total quantity sold at event  2 is  18
Total value in cents =       1620

End of Event
Enter 1 for another event, or 0 for finish.
Enter choice? 0
Total quantity sold at all events =           26
Total cost of all sales at all events =        2120
Total value in cents for all events =        2380
Total value =           23 dollars and         80 cents
Gross Profit =           2 dollars and         60 cents
```

End of Program

Test Data:     Event 1: Cost = 75, price = 100; quantities = 2,4,3,5.  
                  Event 2: Cost = 60, price = 90; quantities = 1,3, -2, 7,3.  
                  Event 3: Cost = 70, price = 90; quantities = 5, 10.

**Bonus Mark = 3**

(e) Modify part (d) to also print out at the end of the run, the gross profit percentage [(gross profit \* 100)/cost].

Test Data:     same as for part (d).

## **6. Average Time for Computer Failures.**

**Basic Mark = 3**

(a) Input 10 values representing the number of minutes between computer failures. Display the total time (that is the sum of the times entered) in minutes.

Test Data: 120, 65, 150, 470, 60, 440, 5, 260, 390, 12.

**Bonus Mark = 3**

(b) Modify part (a) to display the average time, in minutes, between computer failures.

Use the same test data as for part (a).

**Bonus Mark = 3**

(c) Modify part (b) to display the total time in hours and minutes. Use same test data as for (a).

**Bonus Mark = 1**

(d) Modify part (c) to display the average time in hours and minutes. Use same test data as for (a).

**Bonus Mark =**

**6**

(e) Modify part (a) to display a horizontal graph. Each entered time between computer failures is represented by a line on the graph where an asterisk indicates one (1) hour. The entered times must be rounded to the nearest whole hour. 30 minutes and above will round to one hour, 29 minutes and less will round to no hours. See example for the test data:

Time between computer failures in hours

```
-----  
Enter time between failures      1:?   120  
Enter time between failures      2:?   65  
Enter time between failures      3:?   150  
Enter time between failures      4:?   470  
Enter time between failures      5:?   60  
Enter time between failures      6:?   440  
Enter time between failures      7:?    5  
Enter time between failures      8:?   260  
Enter time between failures      9:?   390  
Enter time between failures     10:?   12
```

Graph of Hours Between Failures

```
-----  
**  
*  
***  
*****  
*  
*****  
  
****  
*****
```

Average time between computer failures = 3 hours 2 minutes

Test data: use same test data as for part (a).

**7. Matrix**

**Basic Mark = 5**

A matrix is a rectangular array of numbers. The formula for multiplying a 3 x 1 matrix by a 1 x 3 matrix is:

```
(a1)
(a2) multiplied by (b1 b2 b3)
(a3)
```

Is

```
(a1xb1 a1xb2 a1xb3)
(a2xb1 a2xb2 a2xb3)
(a3xb1 a3xb2 a3xb3)
```

Write a program to input one set of 3 (three) numbers for a 3 x 1 matrix, and one set of 3 (three) numbers for a 1 x 3 matrix and print out the result of multiplying them together.

For example:

$$\begin{array}{r} (1) \\ (2) \times \begin{pmatrix} 3 & 1 & 0 \end{pmatrix} = \\ (4) \end{array} \begin{array}{r} \begin{pmatrix} 3 & 1 & 0 \\ 6 & 2 & 0 \\ 12 & 4 & 0 \end{pmatrix} \end{array}$$

SPLAT will not print the curved brackets. The answer must be formatted to print each row of the answer on a separate line.

Test Data:

First matrix 3, 5, 1;  
Second matrix 1,0,4.

## 8. Geometrical

**Basic Mark = 5**

(a) Write a program that will accept three integer lengths. The program must decide whether those three lengths can form a triangle, and if they can not do so, to print out the message "Cannot form triangle!". If the three integer lengths can form a triangle, the program must test whether the triangle so formed is scalene (all sides of different lengths), isosceles or equilateral. An appropriate message must be printed out to describe the type of triangle which may be formed.

Test data:     3 4 5  
                  3 7 8  
                  5 3 9  
                  6 6 6  
                  11 6 11  
                  2 9 6  
                  9 4 3  
                  8 8 6

**Bonus Mark = 3**

(b) Modify your answer for (a) so that it will continue to prompt for three sides of triangle and test them, until a side of negative value is entered.

Test data: use the same test data as for part (a).

**Bonus Mark = 5**

(c) Modify your answer for part (b) so that the program will also check for whether the three lengths represent a right angled triangle. An extra message "Triangle is right angled." should be printed when a triangle is found to be right angled.

See over page for test data.

Test Data: 1 5 11  
 3 3 3  
 8 9 9  
 5 12 13  
 22 11 10  
 6 9 2  
 3 4 5  
 9 5 3  
 8 8 6

**9. Date Calculation for Payment of Rates**

**Basic Mark = 5**

(a) Write a program to input the date a rates bill was received and calculate and display the date which is 30 days in the future. This will be the date deadline for receiving discount on payment of the rates bill. Assume that there will be no dates input which are earlier than 1992. For example:

Payment Date for Rates		
Enter day rates bill received?	20	
Enter month rates bill received?	12	
Enter year rates bill received?	1993	
Deadline for Payment of Rates = 18/ 1/ 1994		

Test Data:

(i)	1	1	1993
(ii)	24	4	1993
(iii)	10	2	1992
(iv)	2	12	1994

**Bonus Mark = 3**

(b) Modify 9(a) so that the number of days until the discount period expires is also input: For example:

Payment Date for Rates		
Enter number of days until discount expires? 20		
Enter day rates bill received?	20	
Enter month rates bill received?	12	
Enter year rates bill received?	1993	
Deadline for Payment of Rates = 8/ 1 / 1994		



**12.Mathematical****Basic Mark =10**

(a) Write a program which asks for a number (up to 20) to be input, and prints out that number as a product of primes as in the example below:

Example:  $12 = 2 \times 2 \times 3$

(Hints: Any factor of a number less than or equal to 20 which is not a multiple of 2 or 3 will be prime. Also, no number up to 20 will have more than 4 prime factors.)

Test Data: 20

**Bonus Mark = 12**

(b) As for part (a) but able to handle any number up to 100.

Test Data: 98

**GOOD LUCK!!!**

## **Author Profiles**

The authors were the initial champions of the New Zealand Programming Contest (Raewyn Boersen) and the Young Women's Programming Contest (Margot Phillipps).

Raewyn started work when commercial IT was just beginning in New Zealand (1970). She worked in programming and Systems Analysis until approached to teach in 1977. Raewyn started the New Zealand Programming Contest in 1986 and was heavily involved with it until 1994 when it was handed to Dr. Brian Cox (University of Otago) and she then focussed entirely on the ACM Contest. She is currently the South Pacific Regional Director managing 9 sites in Australia and New Zealand. The contest is run simultaneously and covers 3 time zones. She coached for many years and had a number of teams from various universities. The teams she coached had 2 wins at the world finals, with multiple other teams being placed in the top 10.

Raewyn completed an MBA in 1992. She moved out of academia into the e-commerce area, retired briefly and then returned to academia where she runs an Applied Business qualification.

Margot began her working career as an Analyst/Programmer for 8 years before joining AIT in 1985 where she remained for 15 years. After beginning the Young Women's Programming Competition, she became part-time on maternity leave but continued to be involved as a judge in the contest. In 2000, she worked at in another tertiary institute for 3 years before moving to Lynfield College (a high school) where she is currently a teacher of Mathematics and Information Systems/Programming.