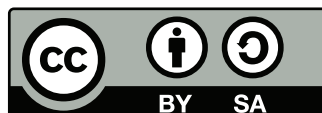


Guide for Advanced Algorithms for Australia and New Zealand Algorithmics & Computing League Competition.

06-Aug-2012



This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Australia License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/au/>.

Prepared By: Darran Kartaschew

Document Version: 1.0

Last Updated on: 06-Aug-2012

Contents

1	Introduction.....	1
1.1	About the Competition	1
1.1.1	ANZAC 2012.....	1
1.1.2	ACM-ICPC.....	1
1.2	About this Guide	1
1.2.1	License.....	2
2	Supported Competition Environments.....	4
2.1	PC ^ 2.....	4
2.1.1	Submit Run.....	5
2.1.2	Submit Test.....	6
2.2	Software Languages	6
2.2.1	Java	6
2.2.2	C++	7
2.2.3	C#	7
2.3	IDEs.....	7
2.3.1	Eclipse	7
2.3.2	Visual Studio.....	9
2.4	Submission Guidelines	13
3	Performance	14
3.1	BigO Notation	14
3.2	Measuring Performance.....	14
3.2.1	C#	14
3.2.2	C++	15
3.2.3	Java	15
3.2.4	Integers vs Floating Point.....	15
3.3	Implementation and Modern Software Engineering Practices.....	17
4	Basic Source Templates.....	18
4.1	Input / Output.....	18
4.1.1	C#	18
4.1.2	C++	18
4.1.3	Java	18
5	Basic Algorithms.....	19
5.1	Basic String Handling.....	19
5.2	Searching	19
5.3	Sorting	19
5.4	Array Handling.....	19
5.4.1	Array Rotation	19
5.4.2	Array Mirroring and Flipping.....	19
6	Advanced Algorithms.....	20
6.1	Simple Maths	20
6.1.1	Greatest common divisor	20
6.1.2	Sieve of Eratosthenes (prime number generation).....	20
6.2	String based algorithms.....	20

1 Introduction

1.1 About the Competition

The programming contests held in Universities across Australia and New Zealand, are part of the Australia and New Zealand Algorithmics & Computing League Competition and is used in conjunction with the ACM-ICPC competition. These competitions are aimed at challenging students in completing a set number of problems within the allocated time slot (typically 5 hours), with the winners in each location given some prestige.

In recent years teams from not only Universities taken part, but teams from TAFE and other educational institutions have taken part in the competition. Additionally teams outside of Australia and New Zealand such as those from the Phillipines have also taken part.

1.1.1 ANZAC 2012

The ANZAC 2012 competition takes place in 5 to 6 rounds each year and are sponsored by a local University and associated Faculty member. Typically, a single round will run for 5 hours (starting at midday for East Coast Australia), and at least 6 problems will be presented for completion by students.

All challenges require some form of problem solving skills or techniques and do require at least a basic understanding of different algorithms in order to complete the challenges, let alone to be competitive in the competition.

In order to compete within the competition it is recommended that 3 students form a team to work together on solving the challenges. Each team is only given 1 computer to work on, and all reference material brought into the competition must be in printed form only¹.

Scores are awarded for completed challenges (typically 1 point), and the time elapsed from the start of the competition to accepted submission of the challenge is also noted. If a submitted challenge fails, then a 20 minute time penalty is added to the teams total time value.

As a minimum each contest will allow either C/C++ and Java, however additional programming languages may also be included. Typically C# has been allowed in recent years, due to the popularity of the language, especially as it is taught fairly early in a students undergraduate degree.

Overall, the competition is designed to be challenging, fun and also students to advance within their field of study. It is also a great way to network amongst other equally capable students within the programming field.

1.1.2 ACM-ICPC

The ACM-ICPC competition is an International level competition sponsored by IBM, ACM and Upsilon Pi Epsilon, and contestants who make the world finals are often sort after by industry for later employment, as well as bringing notoriety and prestige to the University or College to which the contestants originate from. The regional component of the competition is typically held as the last ANZAC competition, as both competitions share the same tools, resources and rules.

The top two teams from each region (and in the case of Australia and New Zealand, the top team from Australia and top team from New Zealand), attend the International competition held annually in late March/early April in an overseas location. The 2012 ACM-ICPC Finals consisting of teams from all over the world was held in Warsaw, Poland.

1.2 About this Guide

This guide is designed to give students some background knowledge of the environments utilised within the competition, as well as information on various algorithms needed to solve problems. The included

¹The printed material requirement is to ensure that no copying of existing source code is allowed, only transcription of source code from written form

algorithms are by no means exhaustive, however represent the bulk of the algorithms that will be useful in completion of challenges.

This guide book is split into multiple parts:

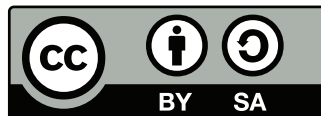
1. Basic Source Templates that cover the basic frameworks needed for challenge submissions.
2. Basic Algorithms and techniques.
3. Advanced Algorithms.

All algorithms described will include:

1. A short statement on the algorithm and the intended uses, as well as other possible uses.
2. The pseudocode for the algorithm.
3. An actual implementation in at least 1 programming language. This will typically be in the form of a function or method call.
4. An example challenge that requires the use of the algorithm.
5. An example solution to the challenge.

Throughout the guide there will be notes on performance aspects of each algorithm, as well as helpful utility functions to make better use of the algorithm implementations. One example will be a function to convert an Adjancy List into an Adjancy Matrix used for different graph based algorithms.

1.2.1 License



This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Australia License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/au/>.

This means you are free:

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

- **Attribution** — You must give the original author credit.
- **Share Alike** — If you alter, transform, or build upon this work, you may distribute the resulting work only under a licence identical to this one.

With the understanding that:

- **Waiver** — Any of the above conditions can be waived if you get permission from the copyright holder.
- **Public Domain** — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

- **Other Rights** — In no way are any of the following rights affected by the license:
 - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
 - The author's moral rights;
 - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- **Notice** — For any reuse or distribution, you must make clear to others the licence terms of this work.

Original Author endorsed waivers:

- The original author however allows use of source code snippets, that is, source code written in the languages of C++, C# or Java contained with this guide for any purpose, without attribution. This waiver does not extend to the text, nor other materials contained within the guide.

2 Supported Competition Environments

The guide will focus on Java being developed in Eclipse, and C# being developed in Visual Studio. However there will be examples in C++ when appropriate. Most other IDEs have similar options, when used for development, debugging and/or profiling.

2.1 PC²

The primary tool that allow students to submit their challenge entries to be judged in the PC² Software Suite. The application itself is developed by California State University, Sacramento for the purposes of programming competitions and has been adopted by both the Australia and New Zealand Algorithmics & Computing League (ANZACL) and ACM for their respective competitions.

An example of the Login Interface is shown in Figure 2.1.



Figure 2.1: PC² Login Screen

Once logged into the system, the following options are typically available:

Submit Run Allows you to submit a challenge entry to be judged, or alternatively to test your entry against some supplied sample data.

View Runs Allows you to view a history of submissions made to the judges.

Request Clarification Allows you to request a clarification from the judges about one of the challenges.

View Clarifications Allows you to see the responses to your requests for clarifications.

Options Allows you to access various options that control the clients operation. However this tab, only allows you to view the operational log of the client.

Most of the operations on the various areas are self explanatory, so won't be covered in detail. The main screen that competitors will utilise is the **Submit Run** tab as shown in Figure 2.2.

This screen has two main modes of operation, allow a competitor to test their submission against some sample input, or submit their source code to be judged. Both have similar operations, except the test has one additional step.

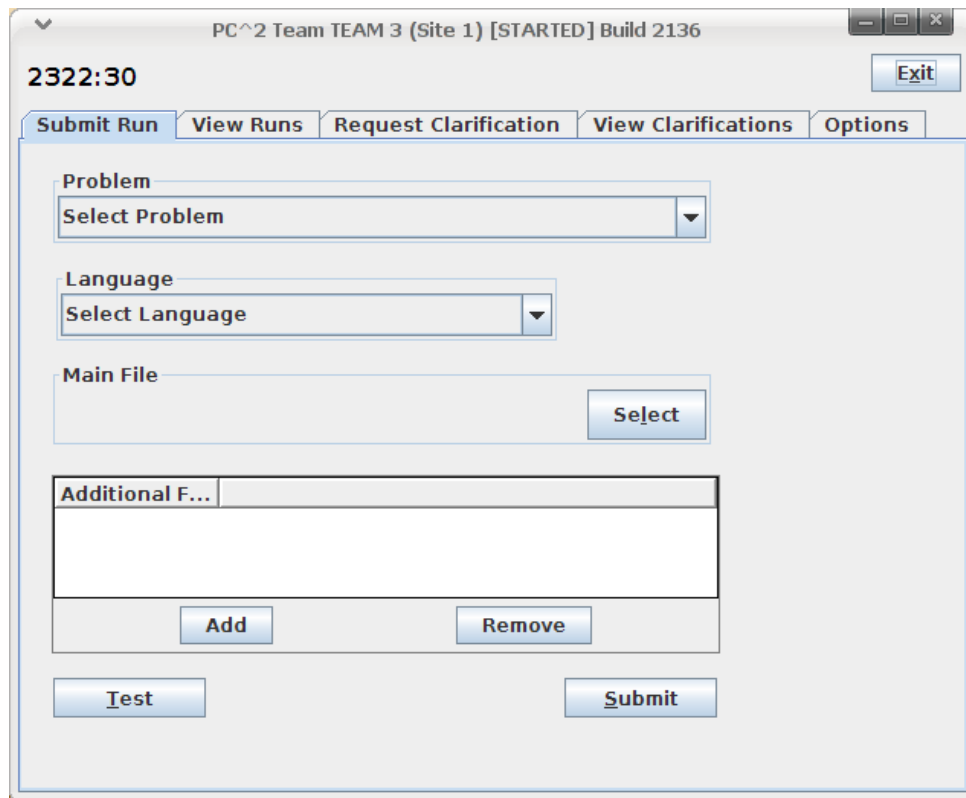


Figure 2.2: PC ^ 2 Client Submit Run Tab

2.1.1 Submit Run

To submit a run for judging, perform the following steps:

1. From the Problem dropdown list select the challenge that you are attempting.
2. From the Language dropdown list select the programming language in the submission is written in.
3. Use the Select button to select the source code file for the submission. (Note: A single Source Code file is required, do not attempt to submit data files or executable files).
4. Use the Add Button to select any additional files needed to complete your submission. (Note: This is rarely needed).
5. Click on Submit, and Yes to confirm to have your submission judged.
6. You will receive a confirmation dialog confirming that your entry has been submitted.

Once you entry has been judged you will receive one of the following confirmations:

- Yes - Your submission was successful in passing all tests. Congratulations, you have been awarded one point.
- No - Your submission failed one or more tests.
- Time Overrun - Your submission took more time that allowed for the challenge.

An example Judge's Response Dialog is shown in Figure 2.3.

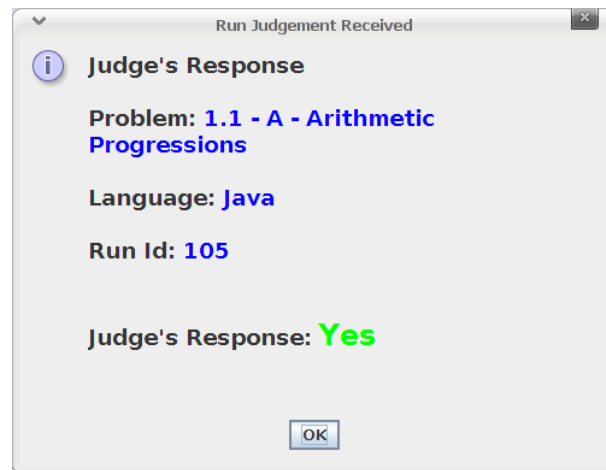


Figure 2.3: Sample Judge's Response

2.1.2 Submit Test

Before you submit your solution to be judged it is **highly recommended** that you perform a test run on your submission first, due to possible differences between the environment you utilised for development and the environment in which your submission will be run on the judges machine.

To test your submission first, perform the following steps:

1. Ensure that your source code file and the sample data files are in the same folder/directory on your system.
2. From the Problem dropdown list select the challenge that you are attempting.
3. From the Language dropdown list select the programming language in the submission is written in.
4. Use the Select button to select the source code file for the submission. (Note: A single Source Code file is required, do not attempt to submit data files or executable files).
5. Use the Add Button to select any additional files needed to complete your submission. (Note: This is rarely needed).
6. Click on "Test".
7. Select the appropriate sample input file in the open dialog box. (Typically the sample input file will be <challenge>_sample_in.txt).
8. Wait for the output dialog and compare to the expected output.
9. If you are happy with your submission, then submit your solution for judging, by clicking on "Submit".

2.2 Software Languages

Currently the competition support the following software development languages with some variations between regional areas: Java, C++ and C#.

2.2.1 Java

Java is compiled utilising the Oracle Java 6SE JRE implementation, however future competitions may migrate to Java 7SE as Java 7 becomes more popular. (This guide will target the Oracle Java 6 SE JRE).

By default the competition will utilise the 32bit JRE, however this may vary as needed between each region. Additionally the Java compiler and JVM are run using default settings only.

2.2.2 C++

C++ (and by extension C) is compiled with an POSIX compatible compiler, typically being mingw on Windows. mingw utilises the GNU GCC compiler suite, and offers a near complete POSIX environment including the C++ STL.

It should be noted, that in some instances the Microsoft Visual Studio C++ compiler has been used within the competition, so it is best to check with the local staff supporting the competition which compiler will be utilised.

Irrespective of the C++ compiler and/or environment, it should be noted that the default compiler settings are utilised through the competition, so features including optimisation flags or 64bit operation are not enabled.

This guide will target a 100% pure POSIX environment.

2.2.3 C#

C# will typically be compiled by Microsoft Visual Studio 2010 with the .NET 2.0 Framework. However there may be variations to this, so it is best to check with the local staff supporting the competition which compiler and/or .NET framework will be utilised.

2.3 IDEs

At the moment there are no official supported IDEs utilised by the competition, however the majority of contestants utilise either Eclipse and/or Visual Studio.

Other IDEs or Editors commonly utilised by competitors include NetBeans (Java, C++), Code::Blocks (C++) and Notepad++ (Java, C#, C++).

2.3.1 Eclipse

Eclipse may be utilised to develop either Java applications or C++ applications (on provision the appropriate eclipse plugins for C++ are installed, and a compatible C++ compiler such as mingw is also installed).

There is no special configuration for Eclipse to be utilised within the competition. As all competition entries operate within a command line only interface there is no requirement for any GUI builder plugins to be present.

To utilise Eclipse for Java development, perform the following steps:

1. Start Eclipse, and switch to a Workspace that is empty, or has been designated for use for competition. (Use File -> Switch Workspace to move).
2. In the File menu, select New.
3. In the New dialog box, select Java Project. Click on Next.
4. Enter any name for the project name. Leave all other settings as default.
5. Click on Finish. This will create a basic project that can be used for developing submissions.
6. In the Package Explorer pane, right click on the 'src' package, and select New -> Java Class. This will be the first submission that you will work on. When developing further submissions, simply start at this point and following the remaining steps.
7. In the New Java Class dialog, enter in the challenge name in the Name: field, and click on "public static void main(String[] args)" to select this option. Leave all other options as default, and click on Finish. (See Figure 2.5).

8. The new submission Java file will open in the file pane. (See Figure 2.5)
9. You are now reading to develop you submission.

All debugging facilities may be utilised within Eclipse with no restrictions.

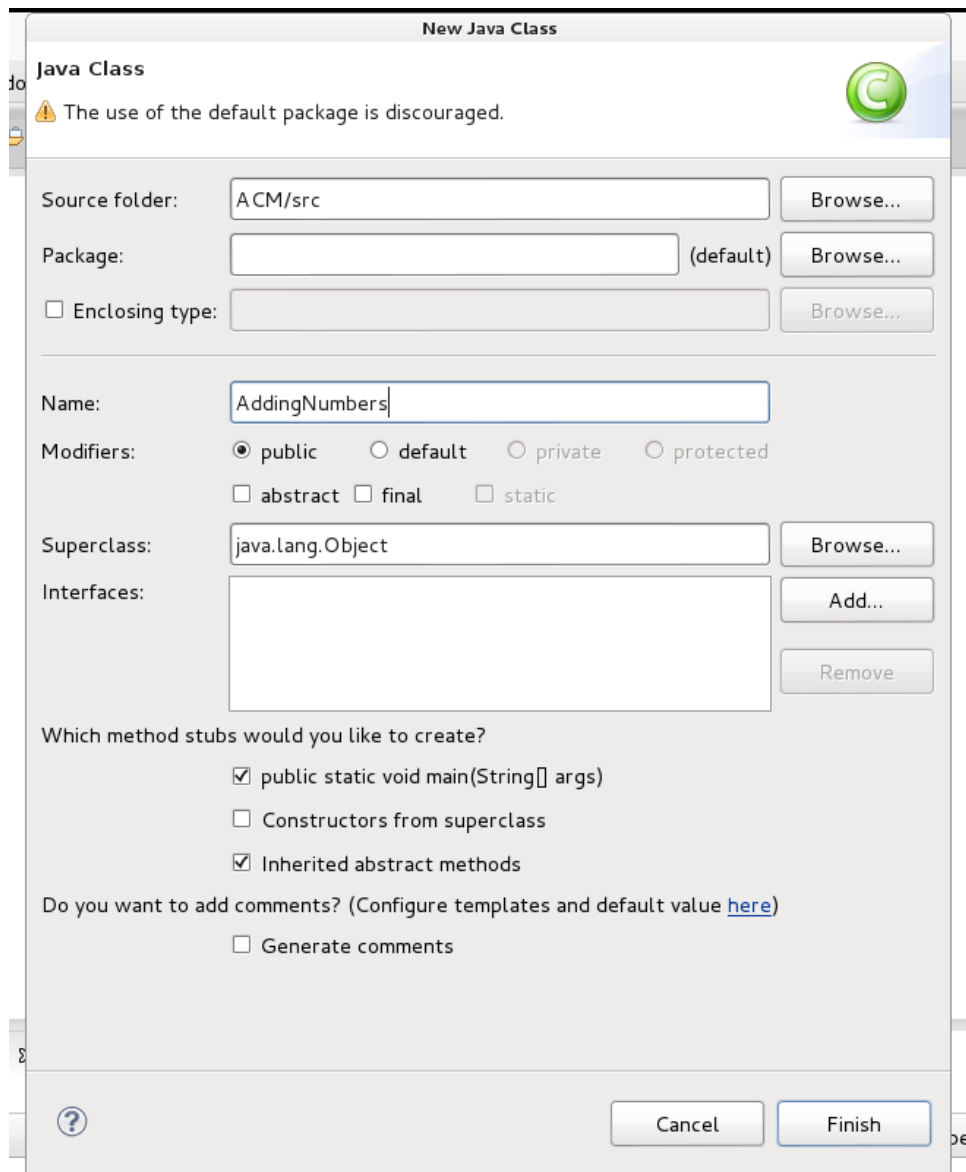


Figure 2.4: Eclipse - New Java Class

One item to note with the file structure of Eclipse and essentially all Java application development, you will need to note the exact location of the source files so are able to find them later in order to submit your solutions.

Using the example in Figure 2.4, note the “Source Folder” location, this is the location that your submissions will be located in, in this case ACM/src/AddingNumbers.java.

When testing your submission with PC² you will be required to either:

1. Copy your source code file to the same location as sample input files, or
2. Copy the sample input files into the same location as the source code file.

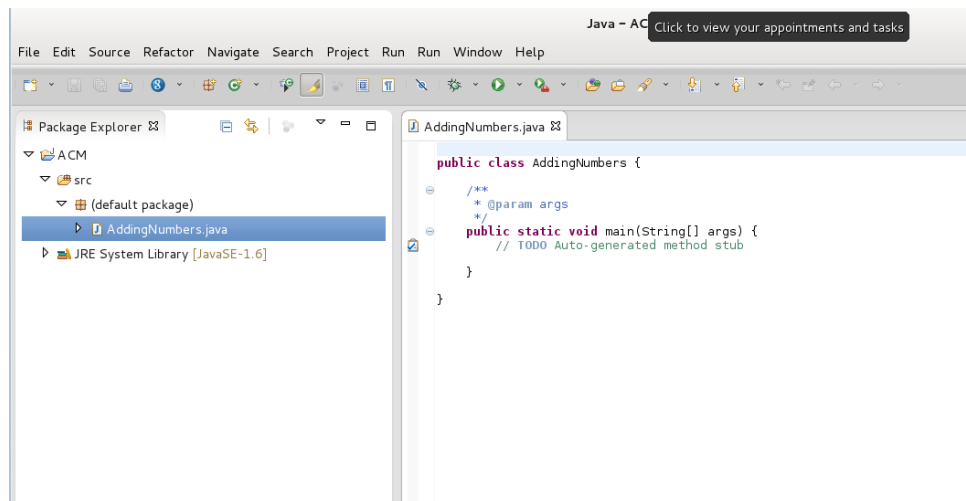


Figure 2.5: Eclipse - New Java Application

2.3.2 Visual Studio

Visual Studio is capable of working with a number of programming languages, including C# and C++. This guide will cover working with both languages, noting the differences between the two.

2.3.2.1 C# There is no special configuration for Visual Studio to be utilised within the competition. As all competition entries operate within a command line only interface there is no requirement for any GUI builder plugins to be present.

To utilise Visual Studio for C# development, perform the following steps:

1. Start Visual Studio.
2. If you receive a "Select Development Language" dialog, select Microsoft C#.
3. Select "New Project" from the Start Page, or alternatively from the File menu.
4. In the "New Project" dialog, ensure that:
 - (a) Visual C# > Windows is selected in the Installed Templates pane.
 - (b) Console Application - Visual C# is selected in the Application Type pane.
 - (c) .NET Framework 2.0 is selected in the .NET Framework dropdown.
 - (d) Enter the name of the challenge in the Name: field.
 - (e) Note the location in which the project is being created.
 - (f) Click on OK to build the project. (See Figure 2.6)
5. The new Program.cs file will be opened and displayed in the File Pane. (See Figure 2.7)

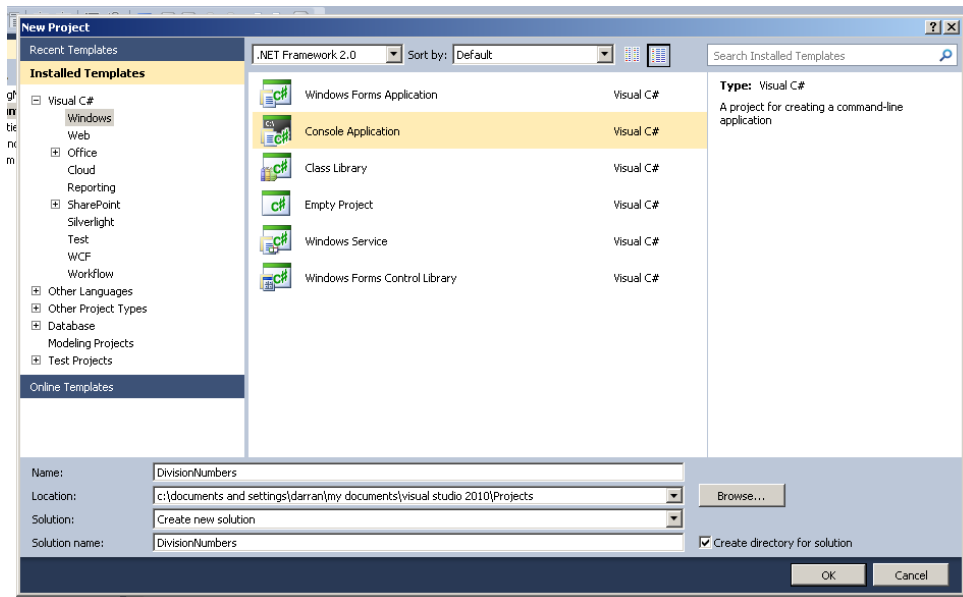


Figure 2.6: Visual Studio C# New Project

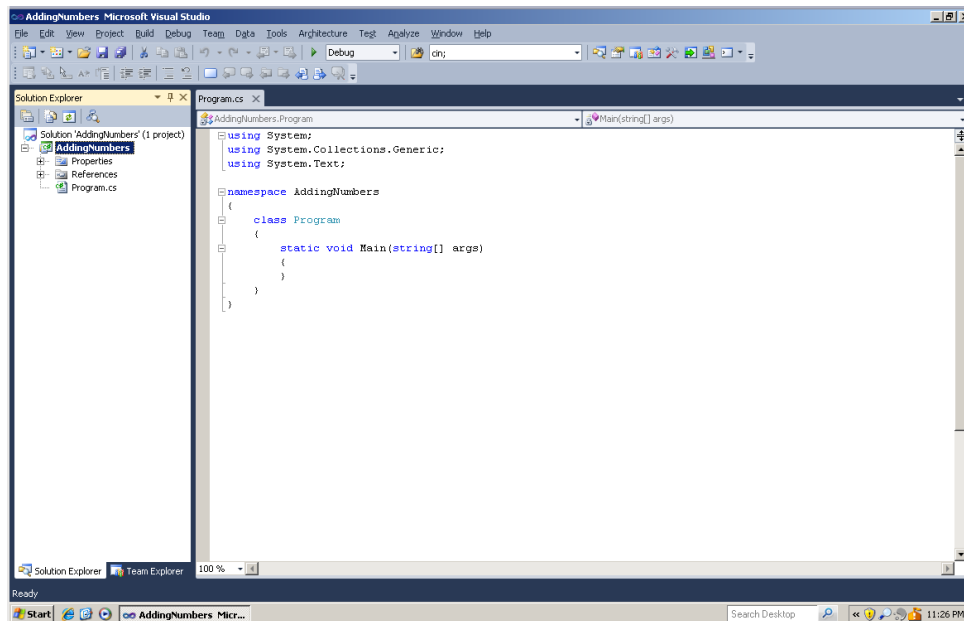


Figure 2.7: Visual Studio C# Program.cs

When testing your submission with PC² you will be required to either:

1. Copy your source code file to the same location as sample input files, or
2. Copy the sample input files into the same location as the source code file.

2.3.2.2 C++ To utilise Visual Studio for C++ development, perform the following steps:

1. Start Visual Studio.
2. If you receive a "Select Development Language" dialog, select Microsoft C++.
3. Select "New Project" from the Start Page, or alternatively from the File menu.

4. In the “New Project” dialog, ensure that:
 - (a) Visual C++ > Win32 is selected in the Installed Templates pane.
 - (b) Win32 Console Application - Visual C++ is selected in the Application Type pane.
 - (c) .NET Framework 2.0 is selected in the .NET Framework dropdown.
 - (d) Enter the name of the challenge in the Name: field.
 - (e) Note the location in which the project is being created.
 - (f) Click on OK to build the project. (See Figure 2.8)
5. The Win32 Application Wizard will run. Select Next on the Wizard Welcome screen.
6. On the Applications Settings dialog, ensure that “Console Application” is checked, and “Precompiled Headers” is unchecked. (See Figure 2.9). Click on Finish to build the project.
7. The new <Application>.cpp file will be opened and displayed in the File Pane. (See Figure 2.10)
8. The following changes are recommended to the main source file to ensure maximum platform compatibility:
 - (a) Add “using namespace std;” before any included headers.
 - (b) Remove or comment out the “#include "stdafx.h"” line
 - (c) Add “#include <stdio.h>” and “#include <iostream>” to ensure POSIX compatibility.
 - (d) Change `int _tmain(int argc, _TCHAR* argv[])` to `int main()`
9. You are now ready to develop your submissions.

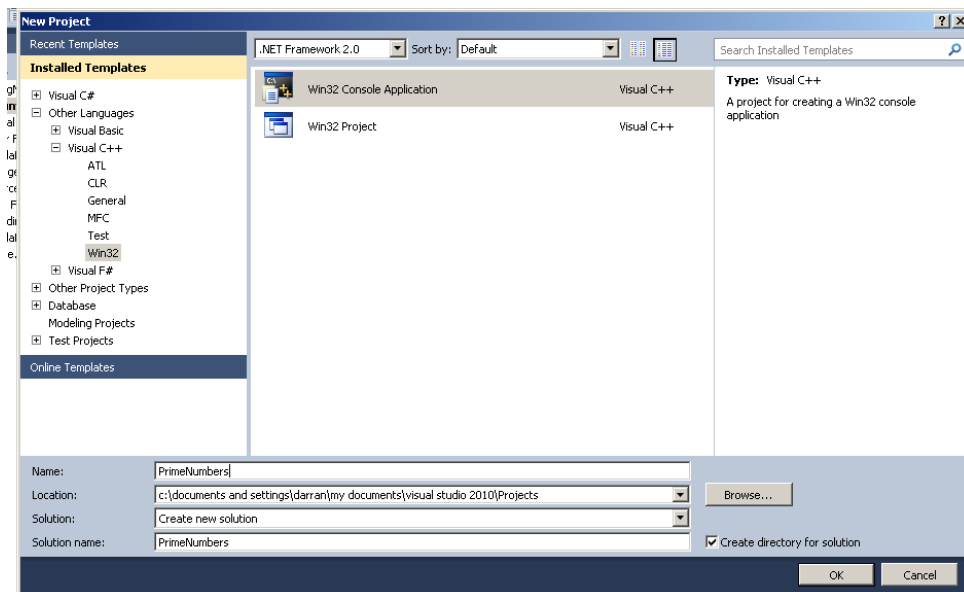


Figure 2.8: Visual Studio C++ New Project

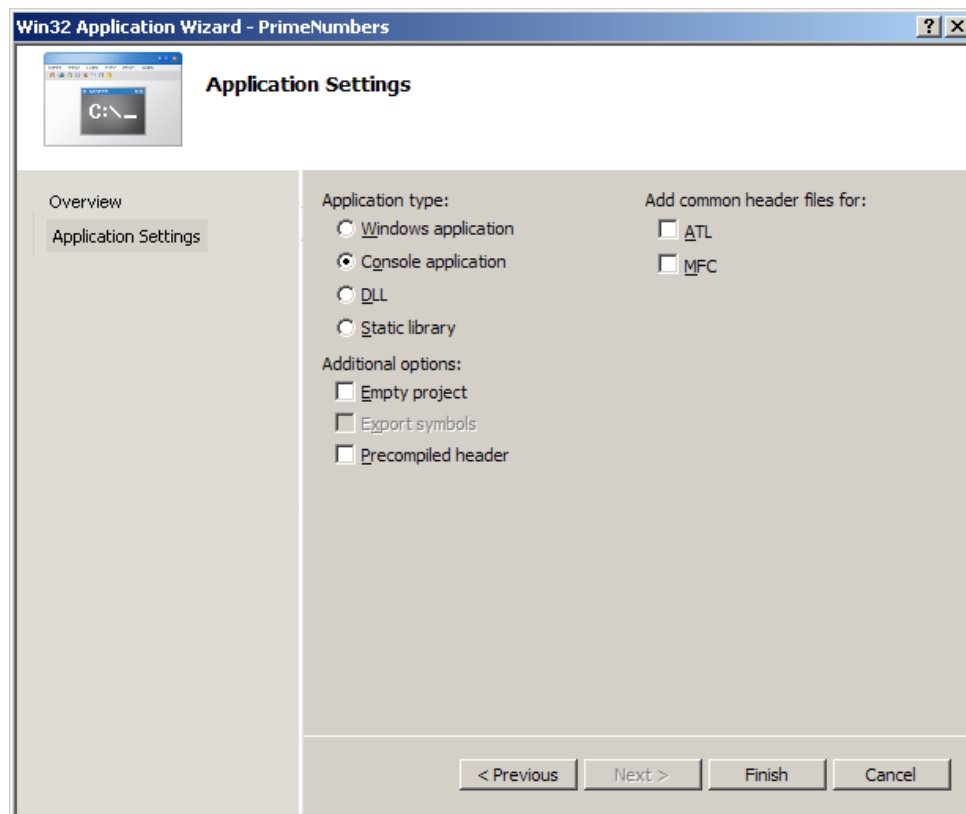


Figure 2.9: Visual Studio C++ Project Options

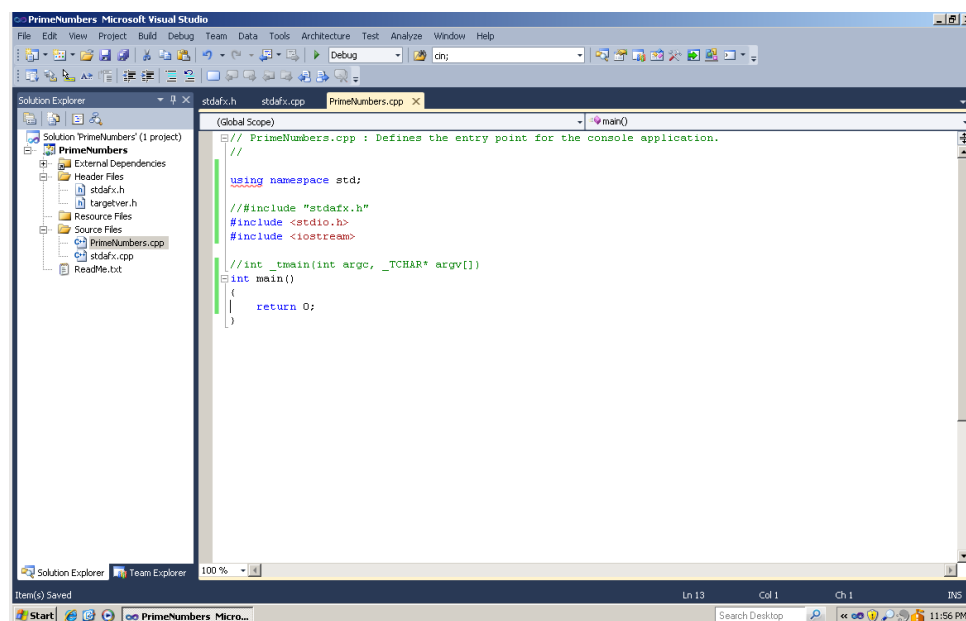


Figure 2.10: Visual Studio C++ Program.cpp

When testing your submission with PC² you will be required to either:

1. Copy your source code file to the same location as sample input files, or
2. Copy the sample input files into the same location as the source code file.

2.4 Submission Guidelines

The primary requirement of any submission, is that all source code required for the submission is located in a single text file.

To test your submission, the judges machine will compile your source code to an executable or class file in the case of Java, then execute it. All input for the application is feed in via `stdin` (or using standard console input), and all application responses and feedback should be returned via `stdout` (or using standard console output). In effect the judges machine will run:

```
$      submission.exe < challenge1_input.txt > challenge1_output.txt
```

Any output the application produces is saved to a file, and this file is then compared to a known correct answer file. Any variations from between the applications output and the answer file will result in a **No** response. If both the output of the application and answer file match, then the judges machine will return a **Yes** response.

3 Performance

3.1 BigO Notation

Big O notation is used in Computer Science to describe the performance or complexity of an algorithm. Big O specifically describes the **worst-case** scenario, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

Typically, algorithms will be assigned the following functions:

O(1) This algorithm operates in the same time irrespective of the number of elements to be processed. (The ideal algorithm).

O(log₂n) This algorithm will have a worst-case runtime of log₂n with n elements to be processed.

O(n) This algorithm will have a worst case runtime in the order of the number of elements to be processed.

O(nlog₂n) This algorithm will have a worst case runtime in the order of nlog₂n with n elements to be processed.

O(n²) This algorithm will have a worst case runtime in the order of n² with n elements to be processed.

O(n³) This algorithm will have a worst case runtime in the order of n³ with n elements to be processed.

O(2ⁿ) This algorithm will have a worst case runtime in the order of 2ⁿ with n elements to be processed.

As an example, suppose the each operation can be done in 1 microsecond, and we have 256 elements to be processed. The respective worst case runtimes for each function is shown in Table 1.

Function	Time
log ₂ n	8 microseconds
n	256 microseconds
nlog ₂ n	2 milliseconds
n ²	65 milliseconds
n ³	17 seconds
2 ⁿ	3.7×10 ⁶⁴ centuries

Table 1: Big O runtimes

The information within this section is a very brief introduction to BigO notation, is included to help you choose between two different algorithms to complete a task based on the Order function given in the algorithm description. Much research has been completed on algorithm analysis, with many text books and reference books being authored on this one subject of Computer Science.

3.2 Measuring Performance

While information described with algorithms can be useful to gain an understanding of the algorithm complexity, it can be helpful to be able to measure the actual execution time needed to complete sections of code. All modern languages or software libraries contains functions to determine execution times².

3.2.1 C#

The .NET Framework provides a Stopwatch class that is capable of being used to measure execution times. Figure 3.1 shows an example of the syntax.

²Modern IDEs such as Oracle Solaris Studio 12.3 include profiling tools to determine 'hot-spots' within applications and also be able to automatically record execution times of individual functions/methods to later analysis.


```
Stopwatch st = new Stopwatch(); st.Start();  
// code to be timed goes here  
st.Stop();  
// time in milliseconds  
long elapsed = (long) st.ElapsedTicks * 1000000 / Stopwatch.Frequency;  
Console.WriteLine("time={0}", elapsed);
```

Figure 3.1: Timing - C#

3.2.2 C++

The C standard library includes time specific functions in `<time.h>` on most systems. The primary function is the `clock()` function that returns the number of 'clicks' since the application started execution. A macro `CLOCKS_PER_SEC` is used to determine the ratio between clicks and seconds. Figure 3.2 shows an example of the syntax.

```
#include <time.h>  
#include <stdio.h>  
clock_t start = clock();  
// code to be timed goes here  
clock_t end = clock();  
// time in clicks  
long elapsed = (long)(end - start)/CLOCKS_PER_SEC;  
printf("time=%d\n", elapsed);
```

Figure 3.2: Timing - C++

3.2.3 Java

The Java System library provides multiple timers with varying accuracy. Since Java 5, nanosecond timers has been available via the `System.nanoTime()` method³. Figure 3.3 shows an example of the syntax.

```
long start = System.nanoTime();  
// code to be timed goes here  
long end = System.nanoTime();  
// time in nanoseconds  
long elapsed = (end - start)/1000000;  
System.out.printf("time=%d\n", elapsed);
```

Figure 3.3: Timing - Java

3.2.4 Integers vs Floating Point

Modern CPUs have integrated high-performance floating-point execution units, however it should be noted that the choice of Integer, Long's and Floating Point number will have an impact of performance of your solution. The code snippet in Figure 3.4 demonstrates the performance differences of using 'int', 'long' and 'double' for a simple add, multiple and divide sequence.

³Accuracy of the `System.nanoTime()` method is reliant on the JVM version and underlying Operating System. However most modern operating systems do provide some form on nanosecond timer.

```
public class NumberTypeTesting {
    static final long ITERATIONS = 1000000000;
    public static void main(String[] args) {
        final double DOUBLE_PRIME = 73;
        final int INT_PRIME = 73;
        final long LONG_PRIME = 73;
        long start;
        long end;
        long count;
        double valueDoubleA = 1.00;
        double valueDoubleB = Math.PI; // pi = 3.142
        double valueDoubleC = Math.E; // e = 2.718
        int valueIntA = 1;
        int valueIntB = 10;
        int valueIntC = 31;
        long valueLongA = 1;
        long valueLongB = 31;
        long valueLongC = 33;

        // Integers
        count = ITERATIONS;
        start = System.nanoTime();
        while (count-- != 0) {
            valueIntA += valueIntC * valueIntB / INT_PRIME;
        }
        end = System.nanoTime();
        System.out.printf("Integer time = %d msec\n", (end - start) / 1000000);

        // Long
        count = ITERATIONS;
        start = System.nanoTime();
        while (count-- != 0) {
            valueLongA += valueLongC * valueLongB / LONG_PRIME;
        }
        end = System.nanoTime();
        System.out.printf("Long time = %d msec\n", (end - start) / 1000000);

        // Double
        count = ITERATIONS;
        start = System.nanoTime();
        while (count-- != 0) {
            valueDoubleA += valueDoubleC * valueDoubleB / DOUBLE_PRIME;
        }
        end = System.nanoTime();
        System.out.printf("Double time = %d msec\n", (end - start) / 1000000);
    }
}
```

Figure 3.4: Source code for timing test

The results⁴ for the above test in Figure 3.4 are:

```
Integer time = 1131 msec
Long time = 2673 msec
Double time = 1298 msec
```

While there is a minor performance drop for using floating point numbers, using 64bit longs yields over double the execution time. This may easily be fixed by utilising an environment that runs as 64bit code, but this is not guaranteed to be available during the competition.

⁴Java 6SE 32bit was used to generate the following results.

3.3 Implementation and Modern Software Engineering Practices

One of the aims of the competition is to develop efficient solutions to the challenges being presented. However often this also means not following modern software engineering practices and taking as many shortcuts as possible.

Some items that are typically seen (and encouraged) are:

1. Liberal use of global variables utilised by direct access.
2. Liberal use of function pointers and jump tables in C++.
3. Despite strong OOP principles with each programming language, these are often ignored for more simple data structures and items like inheritance and encapsulation are ignored.
4. Nested classes liberally use public variables allowing for direct access.
5. The “goto” statement being used in C++ and C# submissions⁵.
6. Ignoring typical design patterns, unless they provide direct and significant benefit in utilising an algorithm to complete a challenge.

As mentioned in Section 2.4, that all source code required for the submission is located in a single text file, also requires some creative uses of both local and anonymous classes.

⁵While most professional programmers avoid “goto” as it’s considered inherently evil, there are some instances where its use can save execution time and/or reduce code complexity.

4 Basic Source Templates

4.1 Input / Output

4.1.1 C#

4.1.2 C++

4.1.2.1 iostreams vs printf

4.1.3 Java

5 Basic Algorithms

5.1 Basic String Handling

5.2 Searching

5.3 Sorting

5.4 Array Handling

5.4.1 Array Rotation

5.4.2 Array Mirroring and Flipping

6 Advanced Algorithms

6.1 Simple Maths

6.1.1 Greatest common divisor

6.1.2 Sieve of Eratosthenes (prime number generation)

6.2 String based algorithms

substring search