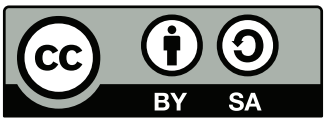
Guide for Advanced Algorithms for Australia and New Zealand Algorithmics & Computing League Competition.

07-Aug–2012



Prepared By: Darran Kartaschew

Document Version: 1.0

Last Updated on: 07-Aug–2012

Contents

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 competion must be in printed form only[1].

Scores are awarded for completed challenges (typically 1 point), and the time elasped 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 to 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 farily 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 constestants 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 Warsow, 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

[1] The 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:

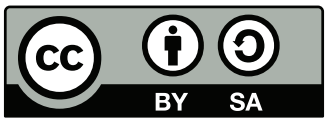
1. Basic Source Templates that cover the basic frameworks needed for challenge submissions.
2. Basic Algorithms and techniques.
3. Advanced Algorithms.

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



- **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 PC^2

The primary tool that allow students to submit their challenge entries to be judged in the PC^2 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: PC^2 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.

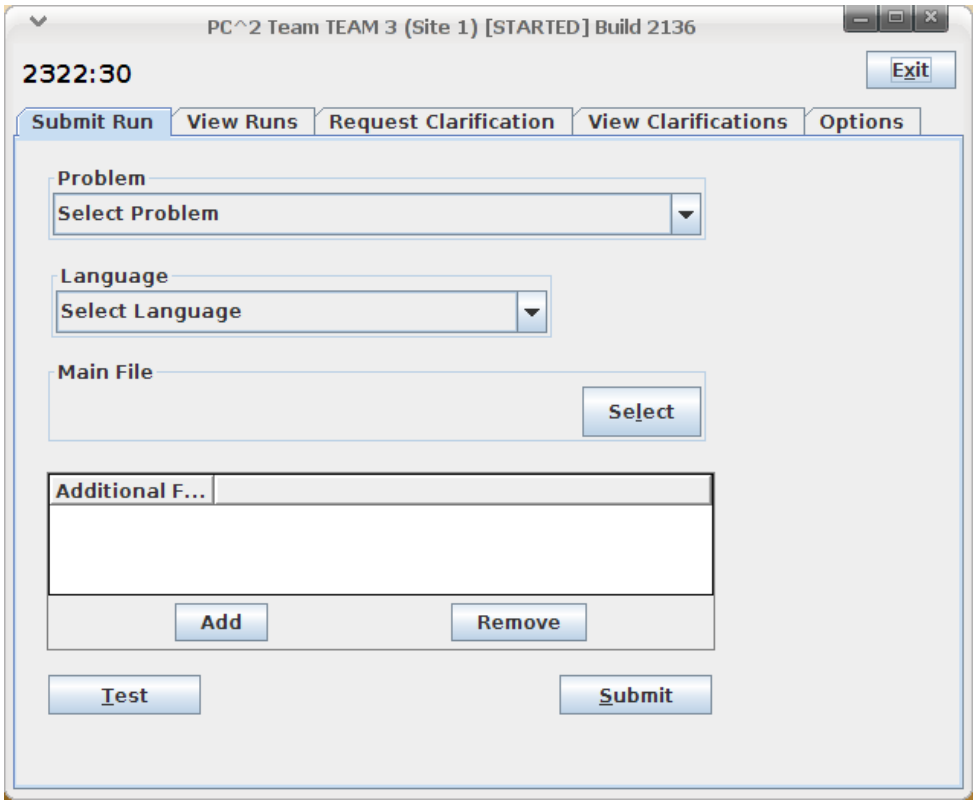


Figure 2.2: PC^2 Client Submit Run Tab

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.

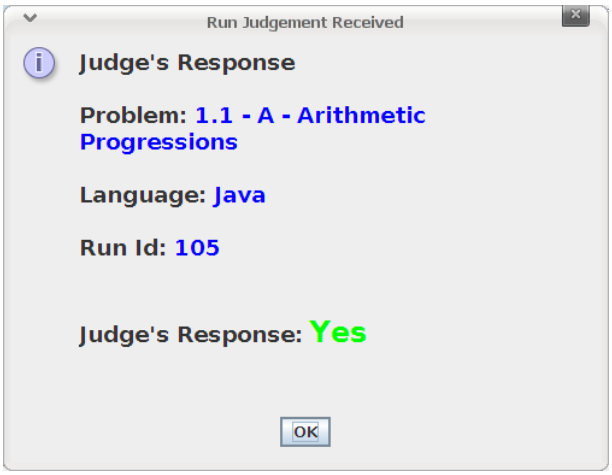


Figure 2.3: Sample Judge's Response

2.1.2 Submit Test

Before your 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 fille will be `<challengename>_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, simple start at this point and following the remaining steps.
7. The in `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.

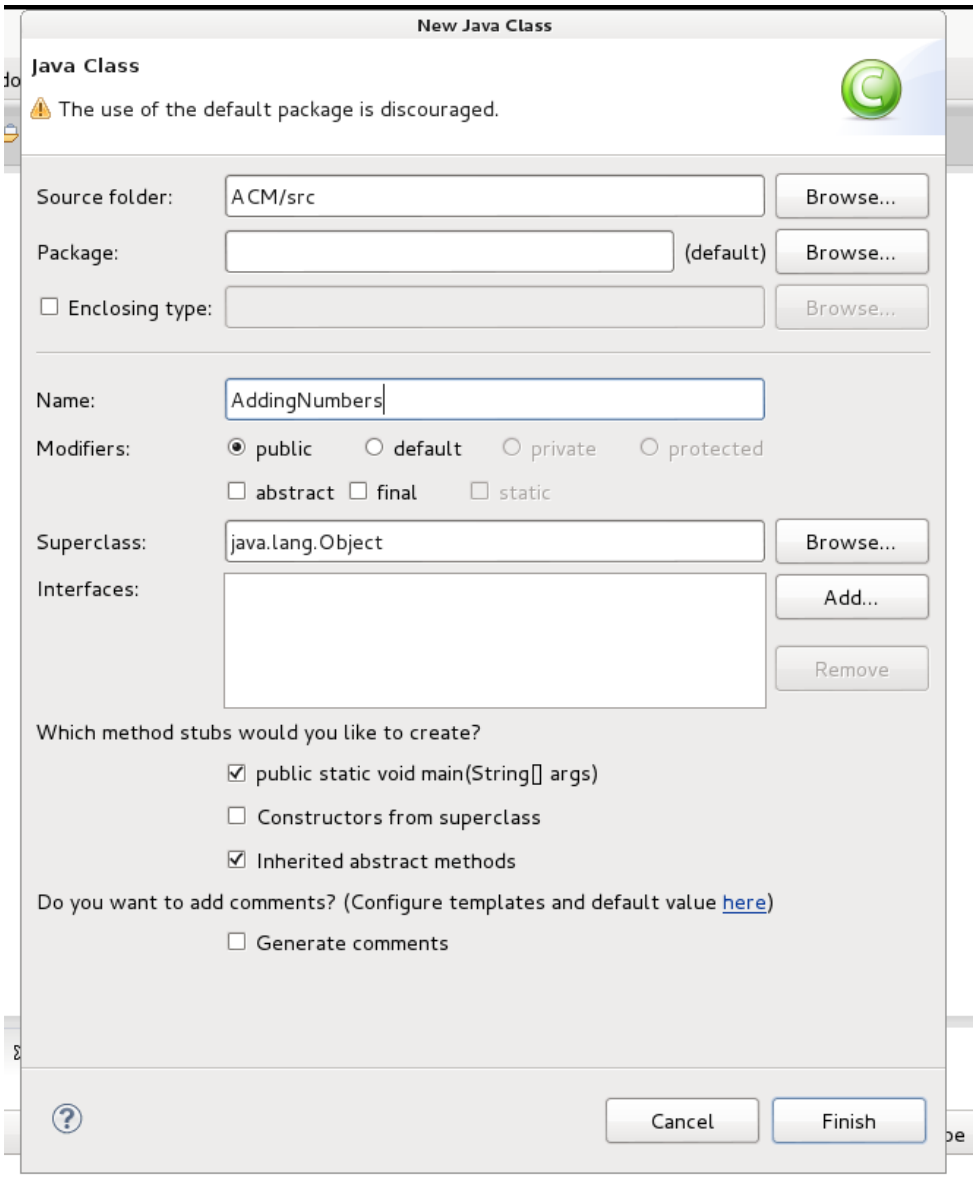


Figure 2.4: Eclipse - New Java Class

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 PC^2 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.

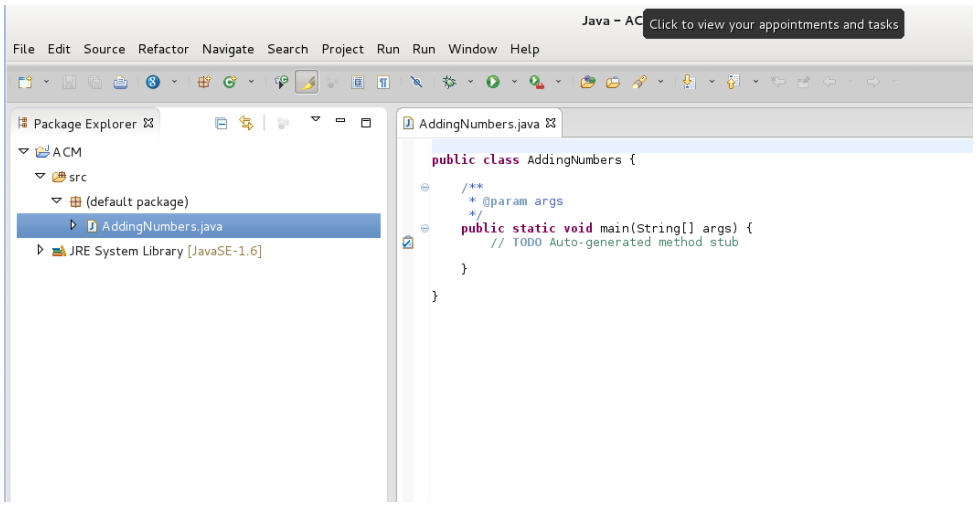


Figure 2.5: Eclipse - New Java Application

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 langauges, 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)

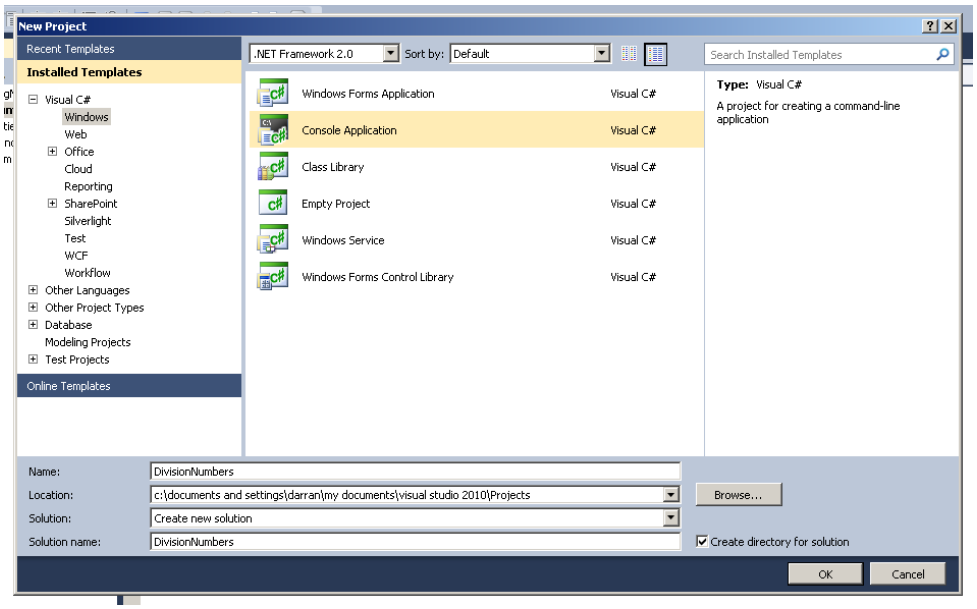


Figure 2.6: Visual Studio C# New Project

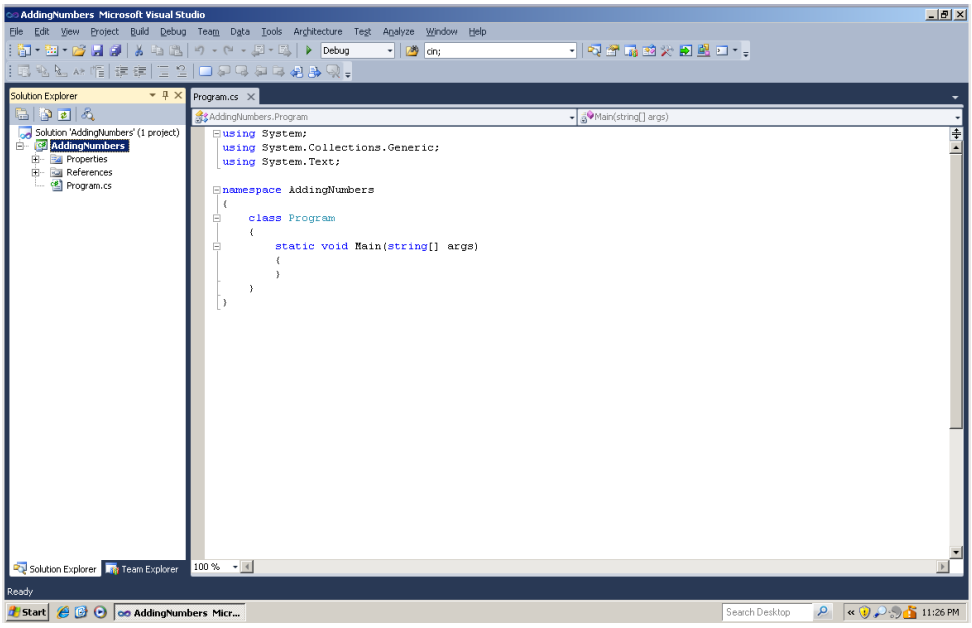


Figure 2.7: Visual Studio C# Program.cs

When testing your submission with PC^2 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, enusre 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.

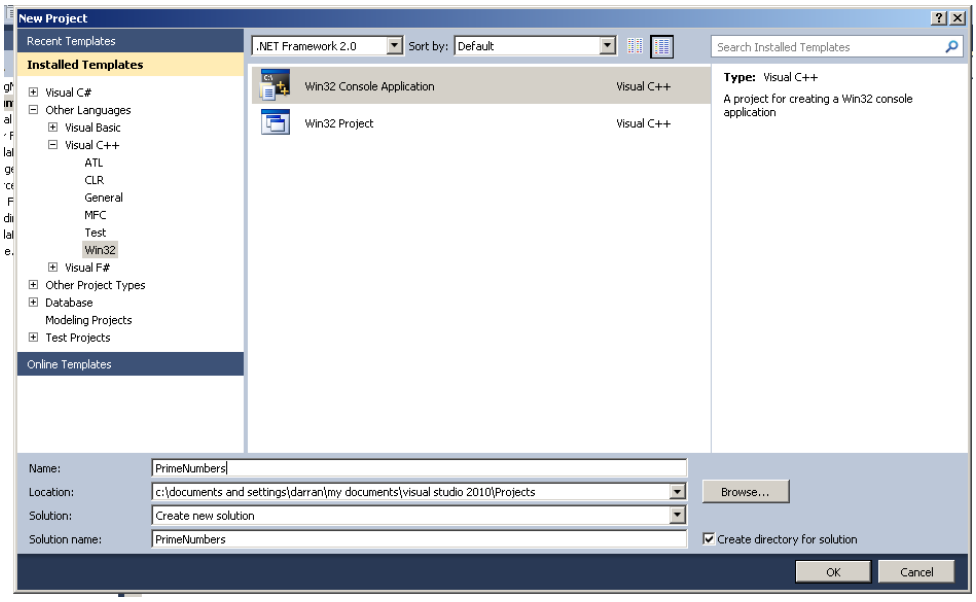


Figure 2.8: Visual Studio C++ New Project

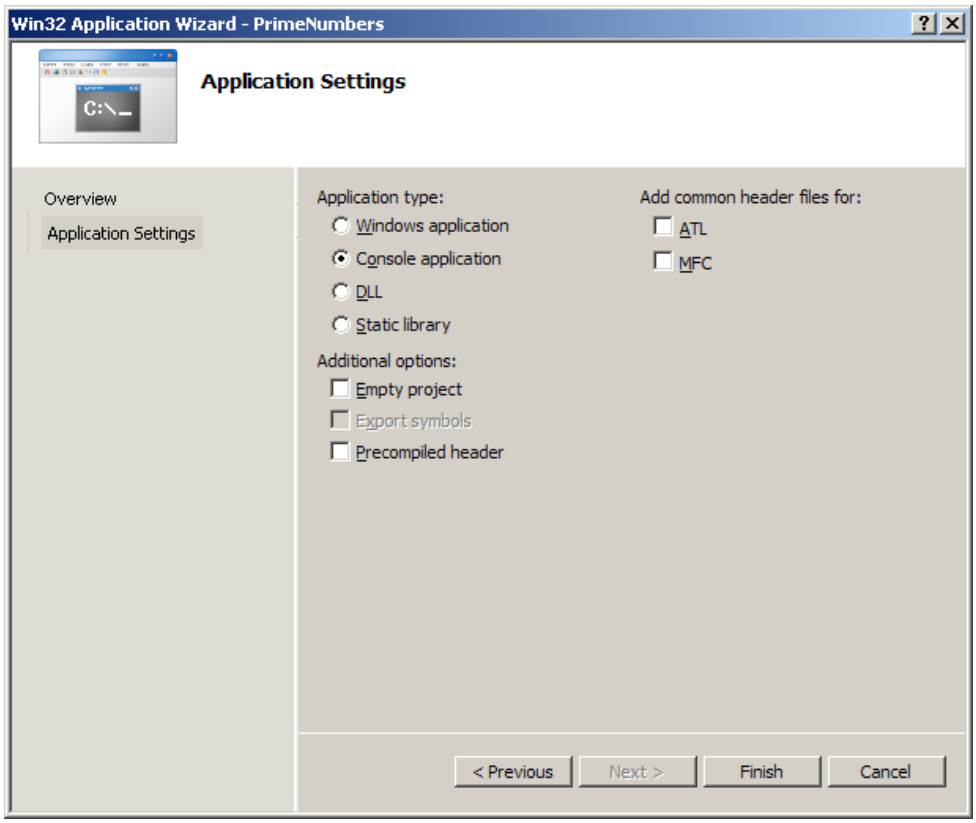


Figure 2.9: Visual Studio C++ Project Options

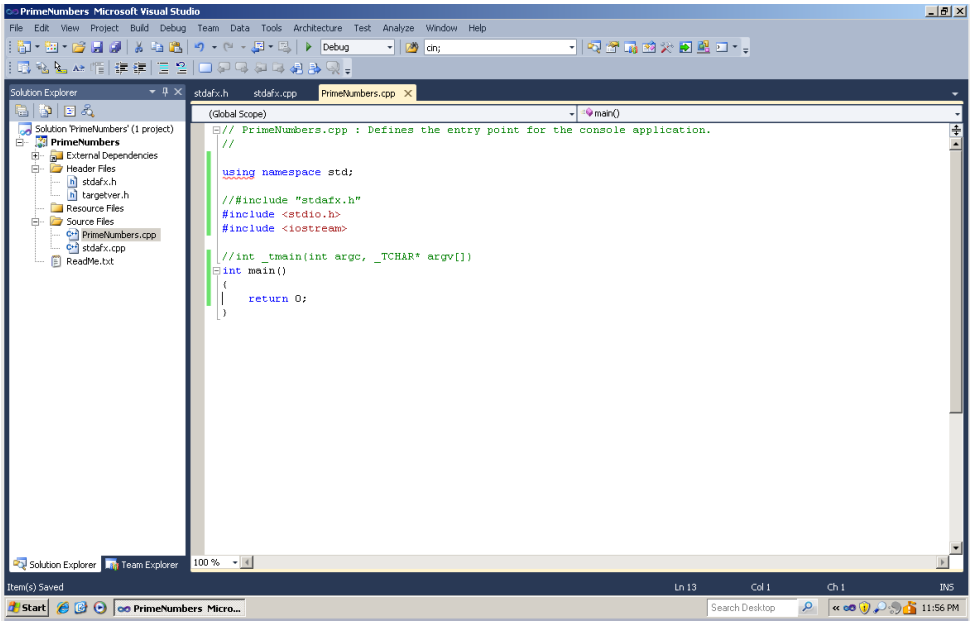


Figure 2.10: Visual Studio C++ Program.cpp

When testing your submission with PC^2 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 exectuable 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($\log_2$n) This algorithm will have a worst-case runtime of $\log_2$n 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(n$\log_2$n) This algorithm will have a worst case runtime in the order of n$\log_2$n with n elements to be processed.

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

O(n^3) This algorithm will have a worst case runtime in the order of n^3 with n elements to be processed.

O(2^n) This algorithm will have a worst case runtime in the order of 2^n 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
n$\log_2$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 exectution time needed to complete sections of code. All modern languages or software libraries contatins functions to determine exectution times[2].

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.

[2]Modern 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()` method[3]. 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.

[3]Accuracy 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 results[4] 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.

[4] Java 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 utlised by direct access.
2. Libreal use of function pointers and jump tables in C++.
3. Dispite strong OOP principles with each programming language, these are often ignored for more simple data structures and items like inheritence 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# submissions[5].
6. Ignoring typical design patterns, unless they provide direct and significant benefit in utlising an algorithm to complete a challenge.
7. Error checking is kept to a minimum, mainly designed around corner cases for algorithms to handle rather than handling bad and malformed input.

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.

[5] While 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

All source code submissions are to consist of a single source code file, as previously mentioned. This section aims to provide simple templates that can be utilised to create your submissions. It will also cover some of the basic console functions available with each langauge.

4.1 Input / Output

4.1.1 C#

The .NET Framework unfortunately has rather cumbersome support for handling console input and output. The `System.Console`[6] class provides methods for dealing with the console. The three main methods that are typically used are:

1. `Console.ReadLine();`
2. `Console.Write();`
3. `Console.WriteLine();`

4.1.1.1 Input The primary function for input from the Console is the `Console.ReadLine()` method which as the name indicates, reads a single line from the console and returns a `string`.

In order to extract information from the string, it is needed to split the string based on a delimiter (typically a space), then attempt to convert each part into the desired type. Figure 4.1 shows how to read a group of 3 integers (per line) from the console, until a three 0's (zeroes) are entered. The numbers for each line is added, and the sum is written back to the console.

4.1.1.2 Output The primary functions for output to the Console are the `Console.Write()` and `Console.WriteLine()` methods. These two differ only by the latter terminating the line with a `carriage-return`, while the former does not.

One item to note, that a single `Console.Write()` method may only take up to 5 parameters, the first being a string, and the other 4 being items to be inserted into the string. Item placement within the string is denoted by a number with `{}` brackets. (See the last line in Figure 4.1 for an example). The item placement parameter, may also take a second argument, being the type to have the item converted to, or displayed as. Common types include:

```
(C) Currency: . . . . . {0:C}
(D) Decimal:. . . . . {0:D}
(E) Scientific:. . . . . {0:E}
(F) Fixed point: . . . . . {0:F}
(G) General:. . . . . {0:G}
(P) Percent:. . . . . {0:P}
(X) Hexadecimal: . . . . . {0:X}
```

By default, console output is buffered, and only written periodically as determined by underlying system settings. To flush the output to console immediately, the `Console.Out.Flush()` method can be utilised.

[6] http://msdn.microsoft.com/en-us/library/system.console.aspx

```
using System;
namespace AddNumbers {
  class Program {
    static void Main(string[] args) {

      // Define our numbers to read.
      int[] numbers;
      int index;
      int sum;
      string line;
      string[] linesplit;

      // Keep reading the input from the console until we have nothing left.
      while ((line = Console.ReadLine()) != null) {

        // Reset array indices and sum of numbers
        index = 0;
        sum = 0;

        // Split the line read, and create a new int array to hold our value.
        linesplit = line.Split(' ');
        numbers = new int[linesplit.Length];

        // Attempt to convert the individual parts to int's
        foreach (string element in linesplit) {
          try {
            numbers[index] = Convert.ToInt32(element);
          }
          catch {
            numbers[index] = 0;
          }
          index++;
        }

        // Test for exit condition.
        if ((linesplit.Length == 3) && (numbers[0] == 0)
              && (numbers[1] == 0) && (numbers[2] == 0)) {
          break;
        }

        // Sum our number and output the sum to Console.
        foreach (int number in numbers) {
          sum += number;
        }
        Console.WriteLine("{0}", sum);
      }
    }
  }
}
```

Figure 4.1: C# Input Template

4.1.2 C++

Due to the environment in which C++ was originally developed, C++ has very strong capabilities for handling both console input and output. C++ offers two methods when working with the console:

1. iostreams
2. C standard library functions.

While the two methods can be intermixed, it is recommended that programmers utilised a single method for their application[7]. For the purposes of this guide, I'll only explain the `iostreams` method as it is often seen as easy to use of the two methods.

4.1.2.1 Input Input from the console is handled by the `std::cin` stream, and has the ability to take multiple types of inputs in a single line or function call. (This is possible due to operator overloading in C++). Figure 4.2 shows the same application written in C++, as shown in Figure 4.1.

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {
    int a, b, c;

    do {
        // Get input of 3 integers and store in a, b and c.
        cin >> a >> b >> c;

        // Test exit condition, and exit if true.
        if(a == 0 && b == 0 && c == 0)
            break;

        // Output the sum.
        cout << (a + b + c) << endl;
    } while(true);
   return 0;
}
```

Figure 4.2: C++ input and output example

Since the `cin` and `cout` streams operate on single variables, an alternate method is required to read a complete line in one function call. This method is `getline ( istream& is, string& str );`, where `is` is the character stream, and `str` is the string to place the input into. As example of `getline()` is in Figure 4.3.

```
#include <iostream>
#include <string>
using namespace std;

int main () {
  string str;
  cout << "Please␣enter␣full␣name:␣";
  getline (cin,str);
  cout << "Thank␣you,␣" << str << ".\n";
  return 0;
}
```

Figure 4.3: C++ getline() example

Once the line has been fetched with the `getline()` method, you are free to use any of the other string functions to extract information from the string.

[7]Mixing the two methods is possible, on provision that all input and output streams are emtpy when switching between either method. This is due to the buffering that each method utilises during Console IO operations.

4.1.2.2 Output The primary method of output is via the `cout` iostream, as shown in Figure 4.2. The format is simply:

```
cout << {object} << {object} << {object} << " string " << ..  << endl;
```

where each {object} represents any C++ primitive type or any C++ object.

There are two methods to generating a newline character, utilising C++. You may either:

1. Output string `"\n"`, or
2. Output `std::endl` .

The difference between the two methods, is that "`std::endl`" will flush the output buffer to console, where "`\n`" will not.

To control the precision of floating point numbers, you can use the `setprecision(`x`)` method as part of the output sequence. eg: `cout << setprecision(4) << (double)1.23456788 << endl;` will output `1.235` to the console.

4.1.2.3 iostreams vs printf The other method to perform console output is the C function `printf()`. `printf()` offers the same features as the `cout` iostream, and may be used when very fine control over output is required especially with floating point numbers.

The general format of the printf() function is:

```
printf(const char *str, ...);
```

`str` is a formatted string, that may contain 0 or more place holders for additional arguments. Placeholders in the formatted string are simply filled in order of additional arguments as specified in the function call, and the additional arguments must be of the same type as specified by the placeholder.

Formats for placeholders include:

`%d` - decimal
`%du` - decimal unsigned
`%f` - floating point
`%s` - string (char*)
`%c` - character
`%x` - hexidecimal number
`%l` - long
`%lu` - long unsigned

Additional fields may be added to the place holders to specify field width and/or precisions. For example:

`%.5f` - will display a floating point number to 5 decimal places.
`%5s` - will consume at exactly 5 character spaces for a string.

There are a few special reserved characters for the printf(), some of these include:

`\n` - carriage return.
`\t` - tab character.

There are some myths surrounding performance differences between `cout` and `printf()`. Through personal testing I've found few performance differences between each method, however this may be enterely compiler and platform dependent, so won't comment too greatly on the matter. So it's best to use the one that is suited to your problem at the time, as it's unlikely that any performance differences that exist will effect your challenge submission greatly.

4.1.3 Java

Similar to C++, Java has a very capable set of support functions for handling console input and output. These are mainly archived through the `System.in` and `System.out` classes used in conjuction with the `Java.util.Scanner` class provided with the default Java libraries.

4.1.3.1 Input Java historically has had a large number of different methods for handling console input, with each new version of Java providing a more streamlined method of handling these functions.

The current prefered method for console input in Java is to use the `Java.util.Scanner` class tied with the `System.in` object to extract the required information from the console. An example of the `Scanner` class can be found in Figure 4.4 which solves the same problem as shown in Figure 4.1.

```
import java.util.Scanner;
public class AddNumbers {
   public static void main(String[] args) {

      Scanner in = new Scanner(System.in);

      // get first line and check for end of test cases
      String line = in.nextLine();

      // Continue until exit condition
      while (!line.equals("0␣0␣0")) {

         // extract three ints
         Scanner sc = new Scanner(line);
         int a = sc.nextInt();
         int b = sc.nextInt();
         int c = sc.nextInt();

         System.out.printf("%d%n", (a + b + c));

         // get next line
         line = in.nextLine();
      }
   }
}
```

Figure 4.4: Java Input and Output

4.1.3.2 Output Output is easiest handled via the `System.out.printf()` method, as it offers a good match between flexiblity and performance. The format for the method call is the same as the C++ `printf()` function as described in Section 4.1.2.3. An example of the method call is also in Figure 4.4.

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