**­­­Autograder User Manual**

1. **Autograder Configuration file**

This has two fields

[Autograder Setup]

grading\_root = \users\home\manujinda\grading

grading\_master = assignments

grading\_root

This is the path to the root directory where all the grading related stuff are stored within. In the directory directed to by this path, autograder looks for pre-specified directories and files to carry on its work.

grading\_master

This is the directory where all the assignment / project related stuff are stored. Each assignment / project has a sub directory within this directory. Such a sub directory contains the solutions, test input and their expected outputs etc. for that assignment / project.

Based on the user’s preferences it is suggested using either “assignments” or “projects” as the name of the grading\_master directory.

A blank autograder configuration file can be generated with the command:

$ python Autograder.py **gencfg** <configuration file name>

It creates the configuration file in the current directory where the command is run from and before generating the configuration file it checks whether there is a file with the same name in the current directory. If that is the case, it notifies about it and does not proceed with creating the configuration file.

1. **Autograder initial setup**

Run the command:

$ python Autograder.py **setup** <path for the autograder.cfg configuration file>

If the configuration file is in the current directory the command is run from, you do not need to provide the full path for the file.

This creates the directory structure required by the autograder at the grading\_root provided in the autograder configuration file. If the directory pointed to by grading\_root already exists, this notifies about that and takes no further action. For this to work, the directory pointed to by grading\_root should not exist and in that case this command creates that directory and the necessary autograder directory structure under the grading\_root. Then this copies the configuration file and a cascading style sheet that is used in student grading log files into the grading\_root directory. Further this renames the configuration file as autograder.cfg.

This creates a blank example assignment / project as a starting point within the directory specified by grading\_master which can be taken as an example for creating future projects.

The autograder generated directory structure is as follows:



grading\_root



grading



students



assignments

autograder.cfg

students.csv

+\_1\_assignment\_1.cfg



assignment\_1

style.css



+\_5\_logs

assignments directory – This is the grading\_master directory. The name of this directory is as specified in the autograder configuration file.

students directory – Each student repository is cloned in here. Each student has a unique directory in this.

students.csv – This stores all the student details. This file is stored in students directory.

grading directory – A copy of each student submission is created in this directory. So each student has a unique directory in this. All the provided files are also copied from respective grading master directory into student submission directories. Compiling of student submissions is done with this set of copied files.

+\_5\_logs directory – This is where grading logs and gradebooks will be stored. This directory will be empty until grading is started.

+\_1\_assignment\_1.cfg – Each assignment / project has a configuration file that describes the assignment / project. Each assignment / project is comprised of a set of problems.

style.css – This is a cascading style sheet used to style student log files.

1. **Student details**

Populate the students.csv file with the student details. Student repository URI’s should be properly recorded in this file for the system to clone / pull them and proceed with the grading.

Must contain columns with these headings:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No | UO ID | Duck ID | Last Name | First Name | Email | Dir Name | Repo |

The number is the index used by the Autograder to identify students. Each student should have a unique integer number. Further, the rows of students should be sorted in ascending order according to the number for Autograder to perform properly.

1. **Creating a new assignment / project**

$ python Autograder.py newasmnt <path the grading root directory> <assignment / project name>

This first checks whether a directory with the provided assignment / project name already exists in the grading\_master directory (e.g. assignments). If that is the case, it will be notified to the user and no further action will be taken. Otherwise, this creates a directory resembling the assignment / project name within the grading\_master directory and within that a skeleton assignment configuration file: +\_1\_<assignment / project name>.cfg and a blank directory +\_5\_logs will be created.

Assignment / project configuration file has the format:

Let’s assume that the assignment name is assignment\_2

[assignment\_2]

asnmt\_no =

name =

duedate =

problem\_ids =

asnmt\_no – Assignment / project number. This is just to identify the assignment / project Provide a suitable number for this assignment.

name – A descriptive name for the assignment / project. Again just to make the assignment / project better identifiable.

duedate – The due date for the assignment / project. The format should be mm/dd/yyy. (do we need to add the due time as well?)

problem\_ids – This is the most important parameter of the assignment configuration file. This describes the problem id’s and types for this assignment. Each problem id must be an integer. This is a space separated list of <problem id>:<problem type> pairs. Problem id and the problem type is separated by a colon (:).

Acceptable problem types:

prog – A programming problem. The student is expected to produce complete programs that can made to run and test with sample inputs. The grading happens by comparing the student output with reference output.

code – A coding problem. The student only produces code chunks but not complete programs.

ans – Short written answer type questions. Grading is done by comparing the answer with predefined answer keywords.

mcq – Multiple choice question type questions. Student submits the correct answer choice. To have more freedom, student can either submit the correct answer selected among the choices or just the correct answer choice number / letter. Grading is done by one-to-one mapping between the student answer and the correct answer choices.

example:

problem\_ids = 1:prog 2:code 3:prog 4:mcq

Note: For the moment, Autograder only supports programming type problems in languages C and C++.

1. **Generating problem configuration file**

After populating the assignment / project configuration file, a skeleton of a configuration file for the problems can be generated by:

$ python Autograder.py genprob <path the grading root directory> <assignment / project name>

This creates a problem configuration file within the assignment / project directory for this project with the file name: +\_2\_<assignment project name>\_problems.cfg

This file has a separate section for each problem described in the assignment / project configuration file. The section name for each problem is:

[<assignment name>\_problem\_<problem id>]

Contents of each section slightly varies for different types of problems.

For a programming type problem:

[assignment\_2\_problem\_1]

prob\_no =

name =

prob\_type = prog

prob\_desc =

files\_provided =

files\_submitted =

inp\_outps =

language =

command\_line\_options =

student\_make\_file =

make\_target =

timeout =

depends\_on =

marks =

Out of these the important ones for the current implementation are:

prob\_no – Problem number. This is set according to what you specified under problem\_ids in the assignment configuration file. You should not change this.

prob\_type – Problem type. This is set according to what you specified under problem\_ids in the assignment configuration file. You should not change this.

files\_provided – These are the names of the files provided to the student at the start of the assignment / project. This is a space separated list of file names. Student is not supposed to edit these files and student does not submit these files back. At the time of grading, these files will be copied from the assignment / project master directory in the grading\_master directory to each student grading directory.

files\_submitted – These are the names of the files student has to submit as part of this problem. To account for naming errors made by students each file name is followed by a colon separated sequence of aliases. This lists the most probable naming errors so that the Autograder can automatically can search for in the case of a file with the proper name is missing. This is just a space separated list of such sequences.

For example this could be:

files\_submitted = main.cpp:mian.cpp:m.cpp triangle.cpp:trig.cpp

inp\_outps – Describes the nature of inputs and outputs. This is a list of space separated triples. Each triple is colon (:) separated and has the format <input number>:<input nature>:<output nature>

Input number is a unique identifier for each input.

Input nature describes the size of the input. It can be either “short” or “long”. If the input nature is short, input is described within the input configuration file itself. If it is long, the input is described in a separate input file.

Output nature describes where the output is produced. This can have values “stdout”, “file” or “both”. stdout means that the output is produced only on the standard output. file means the program only produces files as output. both means the program produce some output in standard out while some in files.

make\_target – The final executable name produced for this problem. This is the program name that will be used to run and test against sample input.

timeout – This is a timeout value to kill the student program if it takes too much time on any sample input provided. For example, the student program might go in an infinite loop. If this value is -1, that means the student program is given time until it terminates naturally.

marks – This describes the grading rubric. This can specify marks to be allocated to each stage of evaluation. This is a space separated list of <marks category>:<mark> pairs. The marks categories are:

compile – The student code successfully compiles

compwarn – The student code compiles without warnings

link – The student code links successfully

linkwarn – Links without warnings

memleak – The student code runs without memory leaks. This part is not yet implemented.

Matching % - A value between 0 – 100 which is the percentage the student output must match with the reference output to earn the specified amount of marks. Each programming problem is run against all the grading input and the average of all matching percentages is calculated and checked against these values to determine the grade for a problem.

For example:

marks = compile:10 compwarn:5 link:15 linkwarn:5 30:20 50:40 80:50 90:65

1. **Generating files**

This generates all the blank files that is needed for this assignment / project. Run the command:

$ python Autograder.py genfiles <path the grading root directory> <assignment / project name>

Files related to programs are created in the assignment / project master directory for this assignment / project.

Provided files – Files provided to the student at the start of the assignment / project.

Submitted files – Files submitted by the student when submitting the assignment / project.

This further creates a directory +\_4\_in\_out within the assignment / project master directory. Within that directory this creates the input configuration file: +\_3\_<assignment name>\_inputs.cfg. For each input specified under “inp\_outps” in the problem configuration file, there is a section [<assignment / problem name>\_problem\_<problem number>\_input\_<input number>] in this file. Contents of sections slightly varies based on the nature of the inputs and outputs.

short:stdout long:file

[assignment\_2\_problem\_1\_input\_1]

nature = short

cmd\_line\_input =

input =

output = stdout

marks =

[assignment\_2\_problem\_1\_input\_1]

nature = long

cmd\_line\_input =

input\_file =

output = file

out\_file =

marks =

nature – Nature of the input as specified in the problem configuration file. You should not change this.

cmd\_line\_input – Command line inputs. Specify that actual command line inputs.

input – This is present only if the nature of the input is short. The actual input other than command line inputs provided to the program are specified here.

input\_file – This is present only if the nature of the input is long. Instead of specifying the actual input, this provides a path to a file where the actual input is specified. You should not change this file path. The file to include the actual input is also generated in the +\_4\_in\_out folder. These files have the file name format: io\_<assignment / problem name>\_problem\_<problem number>\_<input number>\_input.txt. Find that file and include the input to be used to test student programs here.

output\_file – This is present only if the you specified the nature of the output as file or both in the problem configuration file. Specify a sequence of space separated output file names the program is supposed to produce for this set of inputs.

marks – Not being used at present. The idea is the have a space separated list of <output matching %>:marks granted for this set of inputs. This gives finer grain grading rubric where each input set can have different weights on the final grade. This would make defining the grading rubric complex.

After these files are generated, the sample answer programs need to be written. A make file too must be created.

1. **Compiling the sample solution**

Run the command:

$ python Autograder.py compile <path the grading root directory> <assignment / project name>

Make sure that everything compiles properly.

1. **Linking the sample solution**

Run the command:

$ python Autograder.py link <path the grading root directory> <assignment / project name>

Make sure that executable programs are created successfully.

1. **Generate reference output using the sample solution**

Run the command:

$ python Autograder.py genout <path the grading root directory> <assignment / project name>

Make sure that executable programs are created successfully.

1. **Setting up an assignment / project**
   1. Create a directory for the assignment / project within the assignments / projects directory.

(e.g. .../grading\_root/assignments/assignment\_3)

* 1. Create the assignment / project configuration file and populate it to describe the assignment / project. This file must have the file name +\_1\_<assignment / project sub directory name>.cfg

(e.g. .../grading\_root/assignments/assignment\_3/+\_1\_assignment\_3.cfg)

[assignment\_3]

asnmt\_no = 3

name = hello world

duedate = 6/28/2016

problem\_ids = 1:prog 2:code 3:ans 4:mcq

asnmt\_no – Assignment / project number. Just to identify the assignment / project.

name – Name of the assignment / project. Again just to make the assignment / project better identifiable

dudate – Due date of the assignment / project. The format should be mm/dd/yyyy. (Do we need to add the due time as well?)

problem\_ids – IDs of problems that are part of this assignment / project. A problem type for each problem identifier should be specified separated by a colon (‘:’) for each problem. Valid problem types are as follows:

prog – A programming problem. The student is supposed to submit complete code. The code is compiled along with provided code and then tested against the test input. Verification is done by comparing the output provided by student submission with the desired output.

code – Student does not submit a complete program but some program segment. Student submission will not compile as it is. Grading is done by comparing student submitted code segments with the desired code segments provided to the autograder. As an extension, student code segments could be plugged into some code stubs, then compiled and tested just like a ‘prog’ type submission.

ans – Student submits a short written answer. Grading is done by comparing the answer with predefined answer keywords.

mcq – Multipole choice type question. Student submits the correct answer choice. To have more freedom, student can either submit the correct answer selected among the choices or just the correct answer choice number / letter. Grading is done by one-to-one mapping between the student answer and the correct answer choices.

* 1. Run the command

$ python Autograder.py genprob <path to assignment / project configuration file>

This creates a skeleton problem configuration file.

(e.g. .../grading\_root/assignments/assignment\_3/+\_1\_assignment\_3\_problems.cfg)

* 1. Populate the problem configuration file to describe the problems that are part of the assignment / project and their respective grading criteria.
  2. Run the command

$ python Autograder.py genfiles <path to problem configuration file>

This generates all the files described in all the problems that come under the particular assignment / project. All these files are stored in the directory created for the assignment / project.