CS 345 — Fall 2001 **Programming Assignment #1**
Programming Languages **Due November 21, 2001**

General Notes

Read these directions carefully.

Turn in a file called "README" that contains at least your full name and your login name.

You may use either Java or C++ to do this assignment. If you use Java, the class with a main() method must be in "University.java". If you use C++, include in your README file the command used to compile your code to the executable and make sure that your executable is called "university". Also, make absolutely sure that your code compiles and runs on the linux systems on campus, because if we cannot compile or run your program, your grade will suffer significantly.

You will need to turn in both a written and an electronic copy of your source code (do not turn in binaries). To turn in the electronic copy, put your source code and README file in a directory called "proj1" and then type "/p/bin/turnin -submit reeber cs345-lin proj1". Turn in the written copy in the usual way. Both are due by 5pm.

Feel free to discuss algorithms with each other and to look up algorithms in books. However, do not show each other code or look at anyone else's code. Copying code or down-loading code from the Internet is cheating and will lead to failure of the course.

One third of your grade on this assignment will be based on code readability, and two-thirds will be based on how well your program works.

Keep in mind that the next programming assignment will expand upon this one, so re-usability is important.

Problem Description

In this assignment you will write a program that simulates an ideal university, which has no students. In this university, each day every faculty member works on a research project and may generate one or more research projects. Your program will simulate, for each day, the projects that the faculty work on, the projects that the faculty generate, and the amount of money that the faculty earn for working on the various projects.

There are three types of faculty: wimpy Math professors, geeky Physics professors, and Computer Science professors. Each project is associated with one of the three departments. Additionally, projects can be academic or industrial. The academic projects are further divided between fundamental projects and crazy projects.

The input for this simulation will come from the standard input stream and will consist of the following:

$n\ f\ a\ b\ c\ A_m\ I_m$

$Name_0\ Department_0\ Crazy_0\ Fundamental_0\ Industrial_0$

$Name_1\ Department_1\ Crazy_1\ Fundamental_1\ Industrial_1$

$Name_2\ Department_2\ Crazy_2\ Fundamental_2\ Industrial_2$

...

$Name_{f-1}\ Department_{f-1}\ Crazy_{f-1}\ Fundamental_{f-1}\ Industrial_{f-1}$

Here, n is a positive integer that indicates the number of days to simulate. f is a positive integer consisting of the number of faculty members in the university. The next inputs are positive integers a b and c, which determine the following function

$$P(i) = ((a * i + b) mod\ c) + 1$$

which is used to determine the pay rate of research projects. If the ith project generated is academic, it pays $A_m * P(i)$. If the ith project generated is industrial, it pays $I_m * P(i)$. A_m and I_m are positive integers. Note that all projects pay at least one unit.

After these numbers have been given, each line describes one of the n faculty members. First the member's name is given, then his or her department is given by specifying a single character: either 'C', 'M', or 'P' corresponding to CS, Math, and Physics respectively. After that, three non-negative numbers are given. These numbers are further described below.

For this system, the following rules hold:

- The day numbers, professor numbers, and project numbers all start with 0 (e.g. days go from 0 to n-1).
- Each day begins with each professor picking a project to work on. Each project must have been generated on one of the previous days. If multiple available projects fit a professor's requirements, the professor picks the project that pays most (or the project with a lower id in the case of a tie in pay). If no projects fit the requirements, then a professor will work on what interests him or her without pay.
- If professor number j, works without pay, then he or she generates $Crazy_i$ crazy projects, $Fundamental_i$ fundamental projects, and $Industrial_i$ industrial projects. All of these projects are in his or her field of study.
- If professor number j is working on a project, then he or she generates the same number of projects that he or she would have generated without pay, unless otherwise specified.
- No project may be picked twice. The professors pick projects in the following order: on the k^{th} day the professor numbered k mod *f* picks first. The remaining professors precede in order, such that the jth professor to pick is numbered (k+j)mod *f*. Thus, if we have 4 professors, on the 3^{rd} day, the 3^{rd} professor picks first, the 0^{th} professor picks second, the 1^{st} professor picks third, and the 2^{nd} professor picks last.
- A Math professor may pick any project in the field of Math or CS.
- A Physics professor may pick any project in the field of Physics or Math.
- CS professors are smart, so a CS professor may pick any project.
- Anyone working on an industrial Physics or industrial Math project generates one more industrial CS project than he or she normally would.
- Anyone working on an industrial project generates no crazy projects.
- Anyone working on a crazy project generates no industrial projects.
- Anyone working on a fundamental physics or fundamental CS project generates one additional industrial math project than he or she normally would.
- The project number (used to generate pay) begins at 0 and increases by one for each project generated. After all the professors have picked their projects, each professor generates projects. The ordering of professors when generating projects is the same as the ordering of professors that was used for picked projects earlier that day. Furthermore, if a professor generates projects of multiple types, the professor generates crazy projects first, then fundamental projects, then industrial projects. In the case where a professor generates multiple projects of the same type, the CS projects of this type are generated first, then the math projects, then the physics projects.

Finally, the output of your program should look like this:

$Name_1$ $DollarsMade_1$ $IdeaDollars_1$

$Name_2$ $DollarsMade_2$ $IdeaDollars_2$

...

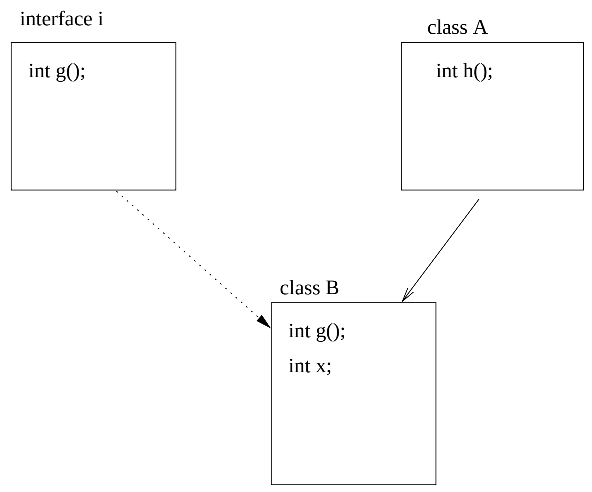
$Name_f$ $DollarsMade_f$ $IdeaDollars_f$

Here, there is one line for each professor, *output in the same order as they were input.* Along-side each name is the total amount of money that the professor has made (the sum of the payments of all the projects he or she worked on) and the amount of dollars that his or her ideas made for other professors (the sum of the payments of all the projects that he or she generated that were eventually worked on). Given this definition, the DollarsMade column always adds up to the same number as the IdeaDollars section adds up to.

1. Data structures will be needed for storing projects and for storing professors. Write classes suited to both these tasks using no data structures other than arrays, vectors (in Java), basic types (int, float,...), and pointers.

Since there may be a lot of projects at any given time, it is important that the data structure for storing projects performs insertion and deletion in no worse than O(log n) time, where n is the number of projects currently available (hint: implement a heap).

2. In addition to your program, turn in a picture of the class hierarchy that your program uses. This picture can be handwritten and does not need to be included with the electronically submission. Represent each class as a box with its public functions and variables inside. Represent inheritance with an arrow, and implement an interface with a dotted line (if you're using Java). For example, if *I* is an interface with function "int g()", *A* implements *I* and contains "int g()", and the integer x, and *B* is a subclass of *A*, which contains "int h()", then your hierarchy should look like:



Example

Imagine that in a file called in.txt, we store the following:

```
3 3 2 10 20 2 5
Lin C 2 0 1
Snyder P 0 1 0
Wiles M 0 0 0
```

Now we type in the command (assuming a Java implementation):

```
> java University < in.text
Lin 64 123
Snyder 39 77
Wiles 97 0
```

To understand this output, first note that the payment function is P(x)= [((2*x+10) mod 20) + 1]. This will get multiplied by 2 for academic projects and by 5 for industrial projects.

Below is the solution broken down into days. (Your programs will not need to output any of this, but it is shown below to illustrate how the simulation works.) The 0^{th} day is straightforward, since no projects are available.

```
Day 0:
Person#   Name        Working-On    Dollars-Earned
0         Lin         Nothing       0
1         Snyder      Nothing       0
2         Wiles       Nothing       0

Projects-Generated:
Project#  Generator   Proj-Dep   Type        Pays
0         Lin         C          Crazy       2[((2*0 + 10) mod 20)+1] = 22
1         Lin         C          Crazy       2[((2*1 + 10) mod 20)+1] = 26
2         Lin         C          Industrial  5[((2*2 + 10) mod 20)+1] = 75
3         Snyder      P          Fundamental 2[((2*3 + 10) mod 20)+1] = 34
```

The next day is more interesting. Snyder can't pick a CS project, so he is stuck with a lower-paying physics project. Wiles, takes the best project, leaving Lin stuck with the best of what's left. Since Snyder is then working on a Fundamental physics project, he generates a math project. Furthermore, since Lin is working on a Crazy project, he cannot generate an industrial project.

```
Day 1:
Person#   Name        Working-On    Dollars-Earned
1         Snyder      3             34
2         Wiles       2             75
0         Lin         1             26

Projects-Generated:
Project#  Generator   Proj-Dep   Type        Pays
4         Snyder      P          Fundamental 2[((2*4 + 10) mod 20)+1] = 38
5         Snyder      M          Industrial  2[((2*5 + 10) mod 20)+1] = 5
6         Lin         C          Crazy       2[((2*6 + 10) mod 20)+1] = 6
7         Lin         C          Crazy       2[((2*7 + 10) mod 20)+1] = 10
```

In the last day, Wiles cannot pick the highest paying project, since that project is in physics. Instead, he picks a project that was generated in day 0. Lin picks the highest-paying project, and Snyder is stuck with a low paying project. Also note, that Lin now generates four projects, since he is again able to generate his normal industrial CS project, and he also generates an industrial math project because he is working on an fundamental physics project.

```
Day 2:
Person#   Name        Working-On    Dollars-Earned
2         Wiles       0             22
0         Lin         4             38
1         Snyder      5             5

Projects-Generated
Project#  Generator   Proj-Dep   Type        Pays
8         Lin         C          Crazy       14
9         Lin         C          Crazy       18
10        Lin         C          Industrial  55
11        Lin         M          Industrial  65
12        Snyder      P          Fundamental 30
12        Snyder      M          Industrial  85
```

If you want another example try working out what happens if we change the file to the following.

```
3 3 3 1 10 2 5
Lin C 2 0 1
Snyder P 0 1 0
Wiles M 0 0 0
```

Now you should get:

```
> java University < in.text
Lin 30 70
Snyder 10 45
Wiles 75 0
```