Assignment 2 (GA for Knapsack) CS472 DUE: Sat Oct 3, 2020 at 5pm PT

REWARD: 150 points

1 The Problem

The Knapsack Problem is a constraint problem. Imagine you have a knapsack (a backpack) which can hold a maximum of *maxwt* kilograms. You have a pile of k items. Each item has a weight in kilograms. You can envision the problem defined by a list of weights:

$$\vec{wt} = [wt_0,\ wt_1,\ wt_2,\ \ldots wt_{k-1}]$$

A solution is a corresponding list of Boolean values:

$$\vec{b} = [b_0,\ b_1,\ b_2,\ \ldots b_{k-1}] \ \text{ where } \ b_i \in \{0, 1\}$$

indicating which of the k items you will put in the pack. The constraint imposed by the knapsack is that the sum of the weights of the chosen items must not exceed the maximum capacity of the pack:

$$load = \sum_{i=0}^{k-1} wt_i b_i \leq maxwt$$

The knapsack problem is that given $\vec{wt}$ find the $\vec{b}$ that maximizes *load* under the constraint that $load \leq maxwt$.

Write a C++ program to solve the Knapsack Problem using a **generational GA**. The program will be called **knapsack**. It will take four (4) command line arguments:

1. the population size
2. the maximum number of generations
3. tournament size
4. the mutation probability per locus p_m is given, where mutations will at each locus will have probability of p_m/k. So if the parameter is 1.0 then the mutation probability at each locus is $1/k$.
5. the xover probability

The program will then read from stdin a specific problem in this order:

1. the *maxwt*
2. the number of weights in $\vec{wt}$ (k the problem description)
3. the list of weights $\vec{wt}$

An example invocation:

```
knapsack 100 5000 3 1.0 0.2 < testcase01.txt
```

where `testcase01.txt` contains the data from stdin. An example problem file might be:

```
107
11
2.0  3.0  5.0  7.0  11.0  13.0  17.0  19.0 23.0 29.0 31.0
```

Program has these requirements:

- it is a generational GA.
- use elitism of two individuals.
- selection will be tournament selection.
- xover will be uniform xover (sometimes just denoted ux) and is performed with probability given by the xover probability
- xover will produce one child
- mutate will flip each b_i with probability p_m/k.
- the program will allow $\vec{b}$ with weight exceeding capacity, $maxwt$, but it will punish the fitness of those genes by the overage as follows:

$$load = \sum_{i=0}^{k-1} wt_i b_i$$

$$fitness = \begin{cases} \text{if } load > maxwt & \text{then } load - (2 * (load - maxwt)) \\ otherwise & \text{then } load \end{cases}$$

- the program will stop when the maxgenerations have been run or when the fitness is within 99.995% of maxwt.

Note that selection can load up the second population with individuals from the first population and operators like mutate and xover can then modify the individuals in place. Just one possible way to manage the memory.

The output of your program will be the final population. `X`'s indicate the item was selected. The number is the fitness. If the program stops because it has found the answer it should also print that it was found in the exact format seen below. Note the location of spaces to separate even the colon. Your output may be read by another program so keep to the format.

```
.....X...XX..X...X.. 499.960000
.....X...XX..X...X.. 499.960000
.....X.X..X..X...X.. 482.210000
..........XX.X...X.. 459.690000
.....X...XX....X.X.. 482.310000
.......X..X.XX....XX 367.640000
........X.X..X...X.. 482.250000
..X..X....X..X...X.. 441.230000
.....X..XXX..X..XX.X 17.310000
.....X..XXXX.X...X.. 212.500000
......XX..X..X..XX.. 287.930000
XX...X....X.XX...X.. 181.820000
.....X....XXXXX..X.. 105.080000
..X......XXX.X...X.. 377.260000
.....XX...X..X....XX 499.990000
.........XX......X.. 316.800000
.....X...XX..X...X.. 499.960000
.....X...XX..X...X.. 499.960000
.X...X...XX..X...XX. 292.900000
X....X...X...X...X.. 403.780000
FOUND Gen 2676 : .....XX...X..X....XX 499.990000
```

If it was not found it should print the last population in the format as above but use the FAILED message:

```
FAILED Gen 5001
```

2 Turning in Your Programs

Please tar up your code as a set of files and not a directory of a set of files. You tar should include a makefile to build the code. My scripts will automatically explode your tar file into a fresh directory and then execute your code with various parameters to see that it is working.

Homework will be submitted as an uncompressed tar file to the homework submission page linked from the main class page. You can submit as many times as you like. The LAST file you submit BEFORE the deadline will be the one graded. There are no late papers. For all submissions you will receive email giving you some automated feedback on the unpacking and compiling and running of code and possibly some other things that can be autotested. I will read the results of the runs and the reports you submit.

Have fun.