1 Cell Wrapper

The original plan for this working week was to implement a version of the point reactor kinetics model which used actual cross section values instead of a given multiplication factor. Towards this end the CELL.C code was re-investigated.

The idea was that the cell code should generate some of the constant terms to be used in the point reactor kinetics model. The first step was to build a simple LabWindows interface for the CELL.C code. As luck would have it, this step consumed all of the working time for the week. The cell code requires four main input parameters, each is the name of a file. The main purpose of the interface is to organize this and to impose a few logical restrictions. A single item .cel file can be constructed by making a mixture. The Custom Mixture panel prompts the user with a list of materials taken from the materials file.

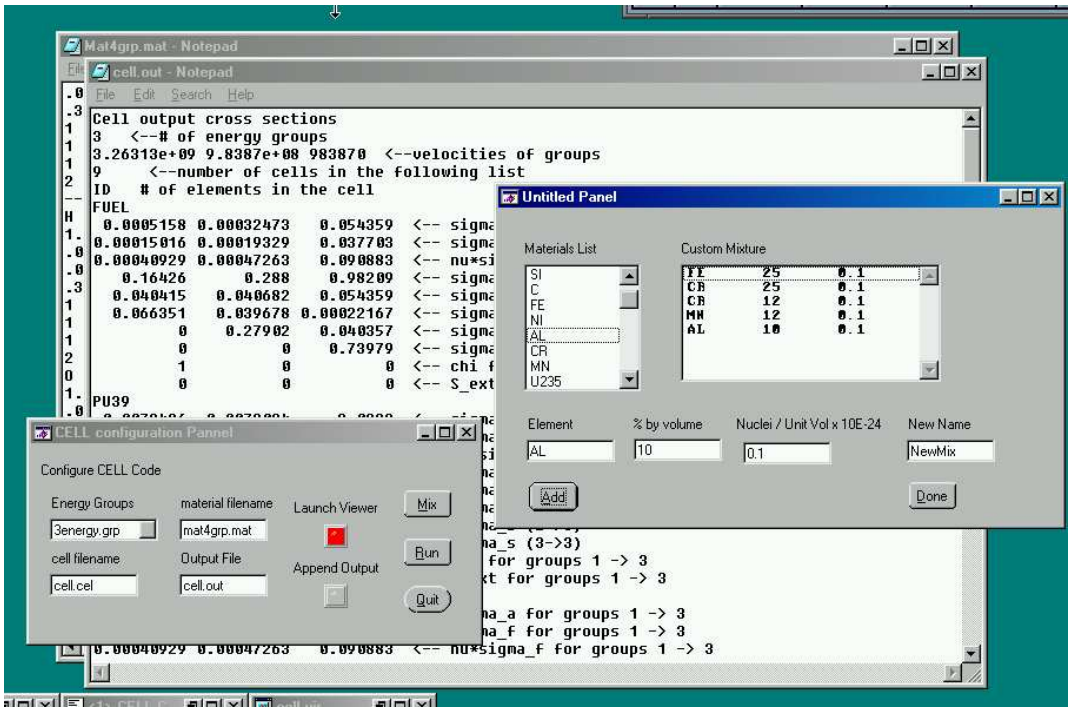

Figure 1: LabWindows interface to CELL.C

The custom mixture is built up one item at a time using the add button. Items can be

removed from the mixture by double clicking on them (this function is buggy at the moment). After a mixture design is complete, the user can click on done and a cell configuration file with a single item is generated.

The main menu has several file handling related options. Files can be viewed by double clicking on their names which opens them inside notepad. The output file can be appended to each time the CELL code is run, and the output code can optionally be viewed each time the output code is run. Notepad can be used to edit the files and save them and then re-run the program.

2 Modifications to classical CELL

2.1 Functional Modifications

Simply put, no functionality was added. I found the cell program somewhat confusing the first few times I used it, and I think some kind of built in file manager would be helpful to anyone trying to use this program. Somehow the idea of what the cell program is doing perhaps needs to be changed, I didn't change the general mode of input- i.e. the file, although ultimately file input style programs are really only useful in a batch mode.

The custom mixture window assists the user by enforcing a file format, and this theme could be worked on. The custom mixture window doesn't do any error checking, it for example does not check to see that ratios sum up to 100%, nor does it check for sanity on the other fields.

2.2 Stylistic Modifications

Many of the variables within the file were renamed using conventions that were discussed in the previous meeting. In particular this program uses 15 global variables, some of which are refereed to in each function. Every function within the cell file was renamed with the prefix cell_, since at 1200 lines the total program has grown large enough that it ought to be split into several files.

2.3 Bug Fixes in Original code

Several minor bugs were identified in the original version of CELL.C. The LabWindows runtime environment was quite good at pointing out miscoded pointers, which although were functionally

legal in the original code, were incorrectly typed, and could have potentially led to problems later on. Some file handling errors existed, one function opens the same file twice, and then doesn't close another file. All of these errors were easily corrected. The following segment of code is taken from the top of the prep() function and is incorrect and it is not clear how it should read:

```
 for (p=0;p<G_newnumber;p++){
        if (p==G_newnumber) energy_avg = G_newgroups[p].energy / 2.0;
        energy_avg = (G_newgroups[p].energy + G_newgroups[p+1].energy)/2.0;
       v[p] = 220000.0*pow(energy_avg/0.025, 0.5);
        fprintf (fpout,"%g ",v[p]);
}
```

There are two problems with this snippet:

1. Since the loop goes from 0..G_newnumber-1, p cannot equal G_newnumber, so the first condition cannot be met, and even if the first condition is met- it does nothing since energy_avg is redefined on the next line.

2. The array G_newgroups[] has size=G_newnumber. In C array indecies start at 0 and go to n-1. Since the loop counts up to G_newnumber-1, and then adds one to this, an illegal reference is always made by G_newgroups[p+1].energy when p=G_newgroups-1.

The comments do not explain what this portion of the code is doing, so I wasn't sure how to fix it. I guessed that the following snippet might have been the original intention, although this isn't clear.

```
for (p = 0; p < G_newnumber; p++) {
            if (p == G_newnumber - 1) {
                energy_avg = G_newgroups[p].energy / 2.0;
            } else {
                energy_avg =
                   (G_newgroups[p].energy + G_newgroups[p + 1].energy) / 2.0;
            }
                v[p] = 220000.0 * pow(energy_avg / 0.025, 0.5);
                fprintf(fpout, "%g ", v[p]);
            }
```

3 Comment

Unfortunately working on the CELL.C file was only a necessary warm up for extracting a few meaningful constants to use in other demonstration programs. I like the idea of building wrappers for old difficult programs, certainly computing students today are not comfortable with command line programs since DOS has faded from use in the last 5 years. We really need to define a list of tractable projects, since we only have a few couple of existing codes which can be repackaged like CELL.C.

LabWindows impressed me with its error detection facilities, usually pointers which drift off arrays by one are not detected. I compiled CELL.C under Unix about 2 years ago, and I didn't detect these bugs then. LabWindows however is not perhaps as easy to use as it first appears. I spent a great deal of time this week looking up callback functions in the help pages, and found myself struggling and cursing with the interface building tools. Better manuals would help, I understand they can be purchased, but the LabWindows complier and tools are fairly complex, and this should not be underestimated when handing this tool to new students.