COMP 200: Elements of Computer Science
Fall 2004
Supplemental Material

Moving Beyond Linear Forms of Expression

While traditional programming is done in languages that are similar to our pseudocode (C, C++, Java, Python, Perl, …), other programming paradigms have a significant role in the real world.

Spreadsheets

In 1978 or 1979, Dan Bricklin had an insight that revolutionized the practice of both business and programming. He observed that many of the activities involved in the financial end of business — from the balance sheet of a major corporation to the checkbook of a single household — were expressed in a tabular form. Bricklin, along with his partner Bob Frankston wrote a program that allowed users to build tabular worksheets on the personal computers of their day — machines that were small, slow, and clumsy by modern standards. Their program, VisiCalc, introduced the idea of a spreadsheet and contained most of the features found in a modern spreadsheet such as Excel (exceptions include multiple linked worksheets in a single workspace & many of the library functions).

The idea behind VisiCalc was simple. The user has a table of cells. Each cell is defined with a constant or a formula. Formulas consist of arithmetic expressions over constant values, operators, and references to cell values. Self-referential formulas are not allowed.

The VisiCalc model lacks control structures — no conditional execution, no iteration, no recursion. Still, it proved to be a powerful paradigm that let ordinary users create programs of stunning sophistication. VisiCalc, and its successors such as Lotus-1-2-3 and Excel, harnessed the growing power of commodity microprocessors to a range of tasks that arise in business and made the "*personal*" computer an essential tool of business. In some sense, Bricklin is responsible for the success of the personal computer industry. VisiCalc was the "*killer application*" for personal computers.

Graphical Programming Environments

Another important paradigm that moves away from programming languages expressed as a linear string of characters is *graphical programming.* The idea behind these programming environments is simple — tasks are represented as objects on the screen that consume inputs and produce outputs. The user "programs" by selecting objects and connecting them.

The idea of graphical programming has been around since the 1950s, when programmers were taught to express programs as *flowcharts*. Many minor systems have been built over the years, in both academia and industry.

In the 1990s, commercial exploitation of this idea took a serious turn with the introduction of LabView, a graphical programming environment built by National Instruments of Austin, TX. LabView began life as a tool for controlling instruments. It has grown into a more general programming environment. A recent Rice MS thesis demonstrated how to connect the graphical language of LabView to the lambda calculus, a standard tool used by programming language theorists to reason about the meaning of programs and the expressiveness of programming languages.

(The lambda calculus is intimately related to Scheme, the programming language that we will us in some of our assignments this semester.)