

Computers in Chemical Education Newsletter

Fall 1991

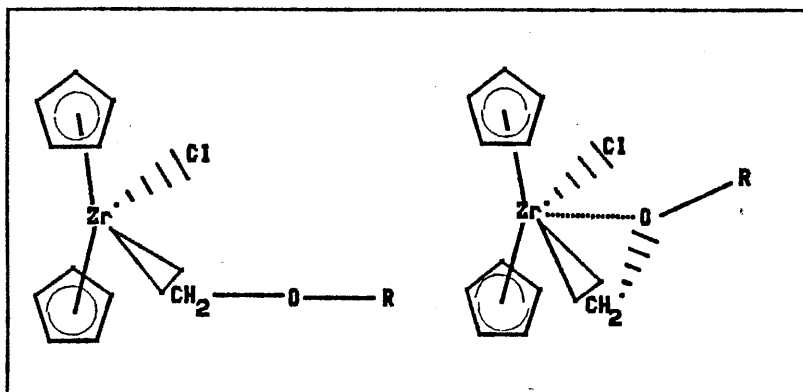
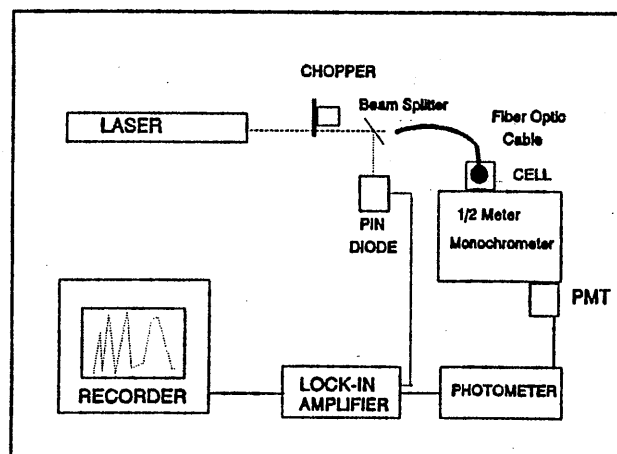


Figure 1 Monohapto vs. Dihapto coordination



$$q^{Rot} = \sum_J (2J+1) e^{-\beta E(J)} = \int_0^{\infty} (2J+1) e^{-\beta h c B J(J+1)} dJ$$



Very Large Large small fine Extra Large

Editor Brian J Pankuch, Department of Chemistry, Union County College, Cranford, NJ 07016

Submissions: General articles should be sent to editor Brian Pankuch at the above address. We would appreciate both 1) printed copy (hardcopy) and 2) a readable file on a 3 1/2 inch Macintosh or IBM compatible diskette. We have fewer problems with 3 1/2 inch diskettes.

We can read the files from Microsoft Word or Works, for other word processors please send text or ASCII code files if possible. Hardcopy is fine if that is most convenient.

ALL NEW AND RENEWAL SUBSCRIPTIONS : PLEASE SEND REMITTANCE TO
Alfred J. Lata, CCCE Chairman, Dept. of Chemistry, 2010 Malott Hall, University of Kansas,
Lawrence, KS 66045.

RATES: USA 1 year \$2.50, two years \$4.50: **Other countries** 1 yr \$5, two yr \$9. Please make a check or money order payable in US funds to Computers in Chemical Education Newsletter.

Managing Editor Kathy Stone Bailey, The Paideia School, 1509 Ponce de Leon Ave., Atlanta, GA 30307

Consulting Editor Donald Rosenthal, Department of Chemistry, Clarkson University, Potsdam, NY 13676. Send meeting notices, etc. to Don.

Contributing Editors: Send your contribution on specific areas to the appropriate editor.

For hardware Queries and Replies. Jim Beatty, IBM and compatibles Chemistry Dept., Ripon College, Ripon, WI 54971 . OR Tom Oyster, Apple Products, Computer Center, Box 248, Ripon College, Ripon, WI 54971.

For software Queries and Replies. Ken Loach, Department of Chemistry, SUNY College, Plattsburgh, NY 12901. BitNet address: Loachkw@snyplaba.

For Book Reviews. Harry Pence, Department of Chemistry, SUNY-Oneonta, Oneonta, NY 13820.

COVER: Sample graphics output from PC Update by Jim Beatty. Please see last page for more samples.

The newsletter is done using a MacIIX, Aldus PageMaker 3.01, and is printed using a LaserWriterII.

Editor: Using 'expert' software to improve student problem solving ability.

I've been using 5 chemistry programs for the Macintosh, which are very helpful with our freshman chemistry courses. During the first week of class I take each lecture group (usually under 35 students) down to our Mac lab which has 18 or more Macintoshes - all running the Mac tour diskette. This diskette comes with every Mac and is a basic introduction to using the Mac and the mouse. The first 15 minutes of the tour show the techniques needed to use the chemistry programs - mostly using the mouse to move the cursor on the screen and clicking on buttons. I pair students who have had experience using a mouse with students who have not. As expected some students finish the tour faster and are individually helped getting into the chemistry programs which are on a shared hard disk. A total of 25-35 minutes suffices to explain how to use the Mac, the mouse, and the chemistry programs.

In the 2 years that the programs have been available students have tried using the programs a number of ways. The most successful is when 2-3 students each using separate Mac's next to each other, help each other and compete for the best score.

The 5 programs cover topics that are spread through the first semester of any introductory chemistry course. The major topics are: 1) scientific notation and rounding; (An easy program to start students on.)

2) conversions between English and metric units of volume, mass, length; 3) gramsA \rightarrow molesA, gramsA \rightarrow molesA \rightarrow molesB \rightarrow gramsB, in many combinations and with chemical equations;

4) molecules, moles, molarity, and limiting reactants;

5) gas laws- Boyles' Law, Charles' Law, Combined Gas Law, Ideal Gas Law, with density and molecular mass.

The first 2 programs are easier to use and provide a gradual learning curve for those not familiar with using computers or chemistry. Students learn to analyze a problem and pick out the pertinent information needed to solve it. They use problem solving techniques (dimensional analysis or unit factor method) that are widely applicable to all sorts of fundamental problems in engineering and science. They learn to use important conversions between many sets of units, and to set up problems in a consistent, understandable manner like an expert. Many are very pleased to find that they enjoy setting up and solving chemistry problems - when they minimize errors and concomitant frustration.

Advantages of using a miniexpert system that makes up and solves its own problems:

1) An almost unlimited number of problems are available. The system makes up its own problems using random number generators. (A random number generator is just a small program which will generate different or random numbers of a certain size.) As a result there is an almost unlimited number of problems.

2) The format is the very familiar multiple choice, a question and four answers. The four answers are in buttons. Students only need to learn how to click on buttons to make their choice (much simpler than learning to use many of the more complicated parts of Copy and Paste).

3) For weaker students a multilevel tutorial is available to provide specific help in each step of the setup and solution.

The system tracks where a user is in the problem and what error they have made. When an error is made the system immediately provides specific help for the particular step in the specific problem the student is solving. This minimizes the frustration of making an error in the beginning of the problem and not

realizing the error until many steps later when the answer doesn't agree with the one in the book.

4) Better students can review many types of questions at many levels of difficulty. The most common student errors are included in the four answers provided. The student automatically gets an explanation of the mistake they made. By clicking on each wrong answer, the user gets an explanation of why each one is incorrect. The student can see the most common mistakes and avoid repeating them.

5) At every step students are provided with 4 answers or choices. The setup and solution of problems requires constant interactive choice from students. If their choice is wrong help is automatically provided. This help is very specific to the particular step in the particular problem.

6) General problem solving help and program help are always available to the student which reduces student frustration.

7) Using the programs cuts the time students spend randomly putting numbers through a calculator and thinking a problem has been solved when you magically agree with the answer in the book. The emphasis in each type of problem is to set up each problem using the same general dimensional analysis.

Comparison of test results showed that students who used the programs did significantly better than those who did not on the type of question covered by the programs. CHE 005 is a noncredit course taken by students who have never had chemistry, had it many years ago, or who do poorly on our placement test. CHE 111-112 is for science and engineering majors; CHE 105, CHE 113-114 for health students. (Questions similar to those covered by the programs, Mac type questions, represented 30-60% of each test)

resented 30-60% of each test)

Class	Total Possible Score	Average Score Students using Mac	Average Score without Mac
005	60	50.0	27.5
111	30	24.5	12.5
105	60	50.0	26.4
113	40	30.6	25.6

Table1.

Results shown in Table 1 fell into several categories:

1) Students who spend more than an hour (average about 6 hours) with the programs get virtually perfect scores on type questions, covered by the programs, and a wide range of scores on the complete test.

2) Students who spent less than an hour with the programs usually did slightly better than non-users but not nearly as well as those who spent over an hour.

3) Excellent students who didn't use the program and did well on all the questions. The same

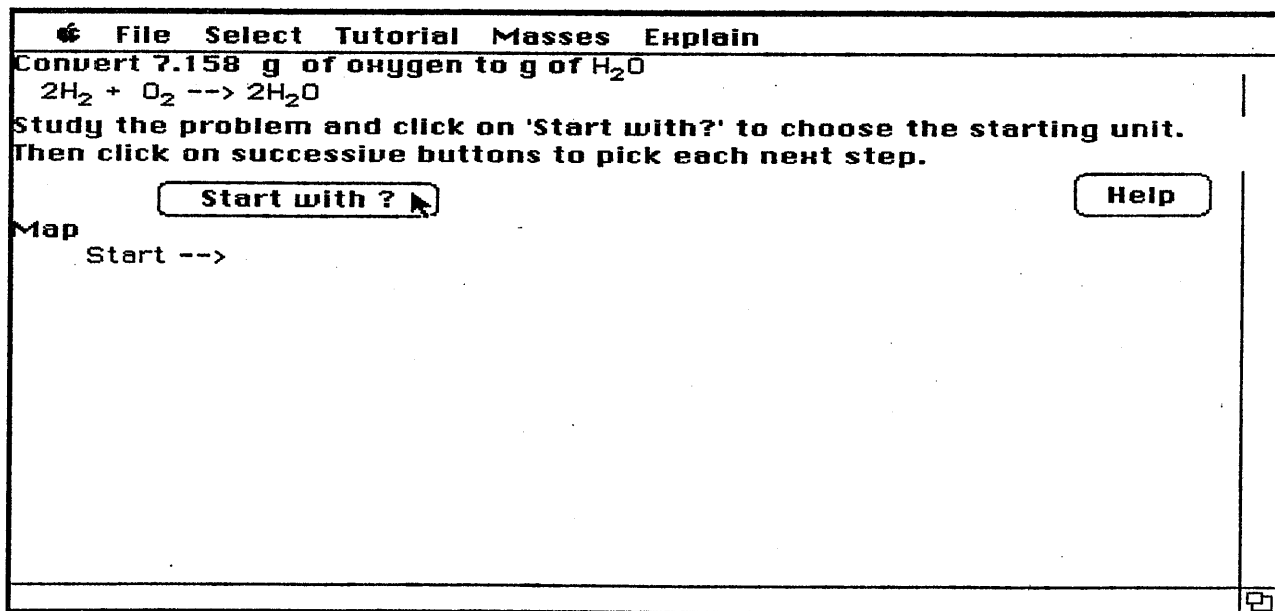
material is covered in the text and lecture. It would seem that someone who is bright and can get the information through the usual channels would not need extra help from the programs. Note the CHE 113 class, which was the best group of students I've had in a decade, did not show as impressive performance gain as the other more average classes. A number of CHE 113 students told me that they were able to get the material from my lectures and their textbook.

4) Weaker students who didn't use the program failed the test.

I was impressed that over half the class would spend 6 or more hours using a program. Students were asked (on an unsigned questionnaire) what advice they would give a friend taking the course if the friend wanted to do well in the course. Ninety three percent of the students stated that using the programs would be the biggest help in learning the course material. The different levels of help and problem difficulty evidently give students a positive learning experience.

The programs proved even more effective than I had expected when I designed and wrote them.

Preliminary testing with high school students show the programs to be useful at this level. The following windows give you an example of the tutorial a student can choose. Every problem in each program has its own specific help available.



Window 1. First step in the tutorial. After reading the problem click on the 'Start with?' button. Clicking on the **Help** button provides help on how to use the program.

File Select Tutorial Masses Explain
 Convert 7.158 g of oxygen to g of H₂O
 $2\text{H}_2 + \text{O}_2 \rightarrow 2\text{H}_2\text{O}$
 As each step in the problem map below is highlighted, choose the correct units for that step.

Map
 7.158 g O₂ --> **????-->** ?--> ?--> = g H₂O

Not quite. You can only take steps—such as g oxygen→mol oxygen, mol oxygen→mol water, mol water→g water. Be careful of the order, then check if you have g or mol of the correct substance.

a) mol oxygen
 b) mol water
 c) g water
 d) g oxygen

Window 2 shows what happens when the wrong answer is picked. The prompt explains the steps that can be taken. At this point other possibilities can be tried. Similar help is available for each incorrect answer in each step of the problem.

File Edit View Special
 Convert 7.158 g of oxygen to g of H₂O
 $2\text{H}_2 + \text{O}_2 \rightarrow 2\text{H}_2\text{O}$
 As each step in the problem map below is highlighted, choose the correct units for that step.


Map
 7.158 g O₂ --> **????-->** ?--> ?--> = g H₂O

Good. 4 new units appear to your left. Pick the unit for the black highlighted part of the above map. g toward g. 3 steps to go to get to g.

Look above to your problem map starting with g and going to g of water. Each ' ? - - > ' is a step you will fill in.

a) mol oxygen
 b) mol water
 c) g water
 d) g oxygen

Window 3. This time the correct answer has been chosen. Click on the next correct units to fill the black highlighted part of the map.

 **File Select Tutorial Masses Explain**

Convert 7.158 g of oxygen to g of H₂O

$2\text{H}_2 + \text{O}_2 \rightarrow 2\text{H}_2\text{O}$


As each step in the problem map below is highlighted, choose the correct units for that step.

Map

7.158 g O₂ --> mol O₂ --> mol H₂O --> ? --> = g H₂O

Your map is finished. Click on the last unit, g, you are looking for to go to the next step.

Window 4 shows the problem map having been successfully completed. The user is about to click on the final unit-g water.

 **File Select Tutorial Masses Explain**

Look at the highlighted part of the setup. Click on the Next setup button to get more info about each step in the setup.

Map

7.158 g O₂ --> mol O₂ --> mol H₂O --> g H₂O = g H₂O

Click on 'Next setup' for info on each step of the setup below.

7.158 g O₂ $\times \frac{\text{mol O}_2}{\text{g O}_2}$ $\times \frac{\text{mol H}_2\text{O}}{\text{mol O}_2}$ $\times \frac{\text{g H}_2\text{O}}{\text{mol H}_2\text{O}}$ = g of H₂O

Click on 'Conversions' for more information about these steps.

7.158 g O₂ $\times \frac{1 \text{ mol O}_2}{31.99 \text{ g O}_2}$ $\times \frac{2 \text{ mol H}_2\text{O}}{1 \text{ mol O}_2}$ $\times \frac{18.01 \text{ g H}_2\text{O}}{1 \text{ mol H}_2\text{O}}$ = g of H₂O
Ans 8.058 x 10⁰

Window 5. The problem map and setup with conversion factors are shown. The user is about to click on the 'Next setup' button to get more hints on setting up the problem.

File Select Tutorial Masses Explain

a) mol O₂ / g H₂O
 b) mol O₂ / mol H₂O
 c) mol O₂ / mol O₂
 d) mol O₂ / g O₂

Look below. The units in the numerator (top) of the setup are exactly the same as your map.
 The units in each denominator are the same as the units in the numerator to the left. They cancel each other. Match the number of g or mol and formula.

7.158 g O₂ --> mol O₂ --> mol H₂O --> g H₂O = g H₂O

SETUP

7.158 g O₂ × $\frac{\text{mol O}_2}{\text{g O}_2}$ × $\frac{\text{mol H}_2\text{O}}{\text{mol O}_2}$ × $\frac{\text{g H}_2\text{O}}{\text{mol H}_2\text{O}}$ = g of H₂O

7.158 g O₂ × $\frac{1 \text{ mol O}_2}{31.99 \text{ g O}_2}$ × $\frac{2 \text{ mol H}_2\text{O}}{1 \text{ mol O}_2}$ × $\frac{18.01 \text{ g H}_2\text{O}}{1 \text{ mol H}_2\text{O}}$ = g of H₂O
 Ans 8.058 × 100

Window 6. Choose the factor that belongs in the highlighted box. The user is picking from among 4 possibilities for the factor that fits in the highlighted portion. Match the correct units given in the highlighted portion to the correct units in a button. In this case to button d). The box in the upper right explains why. Specific help is provided if an error is made.

File Select Tutorial Masses Explain

a) g H₂O / mol O₂
 b) g H₂O / g H₂O
 c) g H₂O / g O₂
 d) g H₂O / mol H₂O

OK! For the highlighted position convert from 'mol' to 'g'. The numerators in the buttons are 'g water'-what we are looking for.
 We need the denominator to be 'mol water' to cancel 'mol water' to the left. We use g/mol to convert from 'mol' to 'g'.

7.158 g O₂ --> mol O₂ --> mol H₂O --> g H₂O = g H₂O

SETUP

7.158 g O₂ × $\frac{\text{mol O}_2}{\text{g O}_2}$ × $\frac{\text{mol H}_2\text{O}}{\text{mol O}_2}$ × $\frac{\text{g H}_2\text{O}}{\text{mol H}_2\text{O}}$ = g of H₂O

7.158 g O₂ × $\frac{1 \text{ mol O}_2}{31.99 \text{ g O}_2}$ × $\frac{2 \text{ mol H}_2\text{O}}{1 \text{ mol O}_2}$ × $\frac{18.01 \text{ g H}_2\text{O}}{1 \text{ mol H}_2\text{O}}$ = g of H₂O
 Ans 8.058 × 100

Window 7. Choosing the last step.

File Select Tutorial Masses Explain

You're finished with the setup. You can click on the 'Conversions' button in the bottom window for more info about putting the conversions in.

7.158 g O₂ --> mol O₂ --> mol H₂O --> g H₂O = g H₂O

SETUP

7.158 g O₂ × $\frac{\text{mol O}_2}{\text{g O}_2}$ × $\frac{\text{mol H}_2\text{O}}{\text{mol O}_2}$ × $\frac{\text{g H}_2\text{O}}{\text{mol H}_2\text{O}}$ = g of H₂O

Conversions

7.158 g O₂ × $\frac{1 \text{ mol O}_2}{31.99 \text{ g O}_2}$ × $\frac{2 \text{ mol H}_2\text{O}}{1 \text{ mol O}_2}$ × $\frac{18.01 \text{ g H}_2\text{O}}{1 \text{ mol H}_2\text{O}}$ = g of H₂O

Ans 8.058 x 100

Window 8. The user has looked at each step and has successfully set up this problem. Next click on **'Conversions'** to check the numbers in each conversion.


File Select Tutorial Masses Explain

Click on the correct conversion factor needed to do problem.

a) 1 mol O₂ / 31.99 mol O₂

b) 1 mol O₂ / 31.99 mol H₂O

c) 1 mol O₂ / 31.99 g H₂O

d) 1 mol O₂ / 31.99 g O₂ 

Select the button with correct units for the highlighted conversion in your setup below. Use the correct units, mol oxygen / g oxygen, to select the conversion 1 mol oxygen / 31.99 g oxygen for step 1. In the denominators match the number of g or mol and the specific substance.

7.158 g O₂ × $\frac{\text{mol O}_2}{\text{g O}_2}$ × $\frac{\text{mol H}_2\text{O}}{\text{mol O}_2}$ × $\frac{\text{g H}_2\text{O}}{\text{mol H}_2\text{O}}$ = g of H₂O

SETUP with CONVERSIONS

7.158 g O₂ × $\frac{1 \text{ mol O}_2}{31.99 \text{ g O}_2}$ × $\frac{2 \text{ mol H}_2\text{O}}{1 \text{ mol O}_2}$ × $\frac{18.01 \text{ g H}_2\text{O}}{1 \text{ mol H}_2\text{O}}$ = g of H₂O

Ans 8.058 x 100

Window 9. Next the numbers that go with each conversion can be checked. Match the highlighted portion with the correct button. An incorrect button is being clicked on.

The previous windows show a few of the steps available in the tutorial part of the programs. The student always selects the next step from four choices. The process is very rapid. As you can see correct or incorrect choices result in specific help for the problem being done. Students are nudged toward mapping each step of the problem before they begin its solution. Next using the map unit conversions from one set of units to another is done. Finally the numbers for the conversions are entered. The highlighted boxes lead the student thru step by step with choice and help available at each step. The student has complete choice on how much detail they want.

As you may have gathered I'm quite pleased with the improvement in performance of my students brought about by using the programs. Let me remind you that I teach at a community college and our average student, now 28 years old, may be different from yours. If you have students who want to do well and are willing to put in some time these programs work. In particular they allow the student to concentrate on the logic chemists use to solve problems, and to work up gradually to more difficult problems.

Technical information: The student programs have been tested on Mac cix, Mac Iix, MacII, SE PLUS, and an Appletalk network. Apple system software 6.03 or greater is recommended. No IBM or Apple II versions are available. The usual price is \$35/program. Mention you heard about them in this Newsletter and the price is \$25/program or \$95 for all five. Handling and shipping is \$4.50 for one or all 5. Send a check (no PO orders please) to EdExpert, RR 1 Box 86, Califon, NJ 07830.

PC UPDATE The IBM PC 10th Birthday or

When are you going to get a Mac, Jim? (They will have to sneak one in my PC)
by Jim Beatty, Ripon College

Topics Discussed Will Include

Hardware: IBM Compatibles Printers
 Modems Plotters
 Switches Projection Systems
 Plotter Cartridges

Software: Wordprocessing

 Data Bases
 Spreadsheets
 Presentation Graphics Programs
 Molecular Graphics and Modeling
 Miscellaneous Chemistry Programs

In this article I reflect back on the ten year history of the IBM PC and its compatibles in the Chemistry Department at Ripon College. Despite the fact that the Macintosh interface is easier to use and graphic displays are superior in many aspects, we chose IBM compatible PC's for a variety of reasons including: 1. Our graduates found that the dominant machine was the IBM compatible. 2. IBM compatibles were much less expensive. 3. Scientific instruments frequently came with an IBM compatible and almost never with a Mac. However, the gap is closing between the two computer types and perhaps in a few years the question will be mute. A plate of mostly graphic output is included with this article to illustrate typical uses.

Hardware : Today, the Commodore Pet's and Apple II's are retired and the first Mac, two 386 PC's, and a HP LaserJet III are on order. Presently, our 8086 machines are being replaced with 386 machines.

The 386 machines are configured with 2 megabytes of RAM, VGA, a co-processor, and a 40-60 meg hard disk. The co-processor is needed to speed up graphics which are increasingly important. Two years ago a 40 meg disk was viewed as lifetime storage. Forty meg is now the minimum because programs and files are getting larger. A 10 meg program is not uncommon. RAM requirements have increased in like manner to disk storage for similar reasons of program file size.

The Laser printer does virtually all printing and most plotting is done using the Laser printer. The plotter is used when multi-color presentation materials are desired. It is nice to have a plotter, but a plotter is not essential. Unfortunately, our FTIR will not plot to a Laser printer. We purchased a Plotter in a Cartridge from Pacific Data Products for our Laser printer to enable our FTIR to plot quality spectra on the Laser printer. This is an excellent emulation package and enables us to plot IR spectra about

5 times faster than with a plotter.

A variety of fonts and font sizes are desirable in text and for such things as signs and labels. A font cartridge such as **25 In One** by Pacific Data Products is a fine solution for the number of fonts available, but not in the sizes. **Bitstream FaceLift**, a scalable fonts program, gives scalable fonts in virtually any size. It is a little slower, but the results are worth it. The best of both worlds looks as if it would be a new cartridge by Pacific Data Products, **The Library**, 51 different fonts scalable to virtually any size, or its PostScript emulation cartridge. The combination of a font cartridge with fixed fonts and scalable fonts provide an inexpensive alternative to PostScript printing.

Communication with library data banks is provided via a 9600 baud modem. Two lines go from PC's to the central college academic computer. Electronic mail is being installed this fall to faculty computers. An RS-232 automatic switching system provides trouble free access to the Laser printer. A **Sayett Data Show Projection Unit** and a transparency projector are available, but for most applications, either transparencies are used or students are taken to the computers.

Software: Wordperfect 5.1 is a significant improvement over previous versions. Pull down menus make it easy to select desired features and facilitate its use. Symbols a printer does not have are created and sent to the printer. An equation mode gives the user the ability to form complex equations with ease. Publication quality tables are easy to form. Almost any computer graphic may be imported into a document. A special publication program is not needed with WordPerfect. Quality reports, newsletters, and laboratory manuals can be created with ease. **Wordperfect 2.2 for the Mac** facilitated the moving of

complete documents with graphics, bold, italics, footnote, etc., in either direction without losses.

SideKick and **SideKick Plus** have given us problems. WordPerfect Office is our answer for a Menu, File Manager, Appointment Calendar, Notebook, etc. program. You can move from program to program with ease or move parts of files between programs through a clipboard.

We are using **PlanPerfect**, WordPerfect's spreadsheet, for most calculations and data plotting. It is similar to **Lotus**, **QuatroPro**, or **SuperCalc 5**. PlanPerfect is slower. Symbols and fonts may be included easily in tables. Tables and graphs are easy to export to WordPerfect. QuatroPro and SuperCalc spreadsheets were easy to move into PlanPerfect. The Main Advantages of PlanPerfect for students and faculty are that the pull down menus, mouse support, and keystrokes are very similar to WordPerfect. We are not actively using a data base program, but we are using the data base capabilities of PlanPerfect to keep track of our chemical inventory. DataPerfect is a possibility and a data base program will be needed when we go to bar coding of chemicals and supplies.

DrawPerfect, a presentation graphics program, is used to prepare transparencies, slides, charts and graphs, and other drawings. It comes with over 500 clip-art drawings and another 1000 drawings are available. It has also been used to prepare eye catching signs, drawing instrument schematics, and to run computer graphic shows.

Molecular Drawing/Graphics Programs: We are using three programs, **Alchemy II**, **WIMP**, and **ChemDraft II**, for molecular drawings and molecular graphics. All create graphic structures that can be imported into WordPerfect with ease.

Alchemy II is the easiest to use and is a powerful molecular modeling program enabling one to build structures on the screen. Energy minimization (MN2) is included. Space filled, ball and stick, orthogonal, and stereo views are available. It is a neat package to experiment with your ideas on the molecular structure of compounds. The day after it was installed, our analytical chemist was working with his research student on the structures of a number of biochemical molecules.

ChemDraft II will create chemical structures in two and three dimensions and provides high quality hard copy. Text and chemical systems can be added with ease. It is easy to use and less expensive than Alchemy II. A MNDO and MN2 package is available for it. The program is less friendly than Alchemy II.

WIMP, the Wisconsin Interactive Molecular Program, which is available from Aldrich, is a powerful program for drawing quality chemical structures, reaction schemes, and chemical equipment diagrams. WIMP is not as easy to use; however, the current **Microsoft Windows** version may help simplify its application.

Each program provides a unique set of complex tasks. I look forward to the day when one molecular graphics/drawing package will have the features of the three programs and more.

Parting Shots: Our students are as adept as the faculty in wordprocessing, importing experimental data into programs, and using spreadsheets and graphics programs. Little formal instruction is given. At the end of early physical chemistry experiments, I help them enter their data in prepared spreadsheets and polish

output. One to three hours are spent in class instruction each year. As they wish to learn more, fellow students or faculty give them one on one help.

The protection of the Copyrights of the owners of the programs you use is an important responsibility of faculty members. We serve as important role models for our students on ethics and values. In the department we avoid this temptation to violate copyrights by purchasing site licenses, and special programs single license programs are found only on a single computer. For example, our three molecular graphics programs are on three different machines. This saves hard disk room and installation and upkeep time.

Computer viruses have not been bothersome yet on IBM machines on campus. They abound on the College's Macintoshes. It could be luck, our students, the IBM world, or the chemical fumes. Backing up of file is up to the individual, but we hope to have a complete back-up system in place by this year.

Hardware repairs have been few with the exception of two Sweet Pea plotters which were "fried" when live RS-232 cables were connected to live plotters. They were under service contract and were replaced promptly. Our HP plotter has had no such problem. Our computer center and local computer dealer have taken care of all minor repairs promptly with colorful noise and grumbling.

The Future: We plan to replace all of our older computers with new and faster machines. Windows and Excel for Windows are being evaluated. A fibre optics campus network is on the drawing board. Our dreams include: computer bar coding for our chemicals and supplies to keep better track of them, a color laser printer to enrich

our documents, a CD-ROM unit to be able to use some of the new educational materials available on CD-ROM, and a graphics super micro computer workstation for powerful molecular graphics programs.

BOOK REVIEW COLUMN

Reader's responses to this column are always cordially welcomed. If you have some special area of computer expertise that you would like to share with your colleagues, why not write and volunteer to review a book in your field of interest? It's always interesting to hear from the readers, so send your suggestions or opinions to Dr. Harry E. Pence, Book Review Editor, Department of Chemistry, SUNY-Oneonta, Oneonta, NY 13820.

THE SUPERCOMPUTER ERA

by Sidney Karin
and Norris Parker Smith
Harcourt Brace Jovanovich, Publishers, Boston, MA
1987, 313 pages, hardbound,
\$19.95

Reviewed by Harry E. Pence

Supercomputers are the fastest and most powerful computing systems available, and they are playing an increasingly important role in many fields of science and engineering. In view of this fact, it's surprising to note that support for supercomputing in this country has been rather erratic. During the 1970s Federal sponsorship of these systems seemed to languish, and more recently one of the major U.S. supercomputer manufacturers, ETA Systems, a subsidiary of the Control Data Corp., has ceased operations, leaving the market to Cray, IBM, and their Japanese competitors.

On the positive side, the National Science Foundation Super-

computing Initiative, begun in 1985, has played a vital role in supporting the creation of both new supercomputing centers and improved communications networks (i.e. NSFnet) to increase connectivity to these

centers. In addition, more pharmaceutical and chemical companies are purchasing supercomputers. These developments are making this technology much more readily accessible for both industrial and academic

use. Supercomputing is no longer limited to a handful of elite academic institutions.

This availability is broadening opportunities for actually using supercomputers and also creating a need for more discussion of the capabilities of these devices at all levels of higher education. As the authors point out, "Not enough people in the U.S. know enough about supercomputers." Improved accessibility is not enough; the goal must be to greatly expand the number of faculty and students who can make use of this accessibility.

This book is a survey that focuses primarily upon the historical development and current applications of supercomputing. Much of the book deals with the way that supercomputers are used in various academic disciplines. Surprisingly enough, even though chemists are major users of this technology, chemical applications are discussed only briefly. A more extensive discussion of topics like computational chemistry, molecular modeling, and numerical simulation would have been especially welcome, since these methods are becoming increasingly routine in industrial and academic laboratories.

The book is intended to make the reader more aware of supercomputers rather than to serve as a technical primer on how to use one. The relative neglect of chemical applications is unfortunate.

nate, but otherwise the authors are reasonably successful at achieving this goal. The writing style is informal, with enough interesting and humorous asides to make the book quite readable. Chemists who wish only a general overview of supercomputers may find this book to be useful.

LABORATORY LOTUS

A Complete Guide to Instrument Interfacing

by Louis M. Mezei

Prentice Hall, Englewood Cliffs, NJ
317 pages, hardbound \$36.00,
1989

Reviewed by Harry E. Pence

Lotus 1-2-3 ubiquitous business spreadsheet, is already widely used in chemistry laboratories to organize large data sets and to do simple, repetitive calculations. Mezei's book describes how this software can not only do calculations and make graphs but also collect data from and control laboratory instrumentation.

Much of the book is devoted to explaining how to use either

Lotus 1-2-3 or Symphony to interface instruments having RS-232 communications. The presentation is extremely systematic and clear. The author provides brief summaries at the beginning and end of most chapters and also includes a running outline to make the discussion clearer. Sample programs are developed one section at a time to make these examples easier to understand. This modular approach also helps to simplify the explanation and minimize the chances for error.

The instruments chosen to serve as examples include the ACROSYSTEMS ACRO-900 interface, the Perkin-Elmer LS-2B Filter Fluorimeter, the Mettler AE163 analytical balance with optional data interface, and the Bio-Tek El-309 Microplate Reader, but for readers who don't have access to these specific instruments, the author

explains how these lessons can be applied more generally to other instrumentation. The discussion is not limited to data collection and treatment, but also covers how Lotus can set up the instrument, take the sample, and accumulate the data.

The presentation covers both RS-232 as well as IEEE-488 communications. For older instruments that lack a modern communications port, instructions are provided on how to access the data through a chart recorder output, an analog voltage output, or a digital display. Although considerations of cost and convenience cause the author to express a preference for the basic spreadsheet programs, he also discusses how to use Lotus Measure[®]R for instrument interfacing.

The final chapter includes a general summary and some excellent general advice about developing a communications program for a laboratory instrument. This section offers some especially valuable tips on programming and troubleshooting. Appendices are provided that cover RS-232 cables and pinout specifications and also review the most useful Lotus functions and commands.

Among the profusion of books on Lotus it is delightful to find one that is specifically aimed at chemists. It is even more pleasant to be able to report that the discussion is so systematic and lucid. This book should be both an excellent resource for those who are already dedicated Lotus users as well as an effective argument to convince new users that this software can play a valuable role in their laboratories.

PERSONAL COMPUTERS FOR SCIENTISTS

by Glenn I. Ouchi

American Chemical Society, Washington, DC, 1987
300 pages, paperbound, \$22.95,
hardbound \$34.95

Reviewed by Harry E. Pence

Glenn Ouchi is a well-known advocate of laboratory uses of microcomputers, who has written articles for many widely-read computer journals, taught ACS short courses, and edited a newsletter on computing in the chemistry laboratory. He is well qualified to fulfill the promise that the book will "help you select and use a personal computer (PC) to solve problems in your work."

The initial section of the book covers the fundamental hardware and operating systems characteristic of microcomputers. The author does not presume very much prior computing knowledge on the part of the reader. The clear and concise explanations are accompanied by many excellent illustrations and line drawings. First-time computer users should find this book to be both attractive and readable.

The core of the book is the section on applications software. Ouchi emphasizes that when choosing a computer the essential consideration should be to identify the critical problems which are to be solved and then to select a machine that will run the software that is best able to accomplish those jobs. Advanced hardware capabilities, such as high performance or extended memory, are worthless unless the software needed for the tasks at hand will run on the computer. As many users have sadly learned, purchasing a computer based on the assurance that the required software will be available "real soon now" will usually lead only to frustration and wasted time.

The author discusses the "big three" types of software: word processors, spreadsheets, and data bases, but also includes applications of major interest for scientists, such as graphics and statistics programs. Much of the software that he describes is widely used for business applications, but he provides good examples of how those commercial packages are equally applicable to scientific projects. The realistic, sample experiments are excellent for showing

how each piece of software can be used, but if there were more emphasis on the types of jobs that each type of software could do, the book would become less dated as new releases of these packages have become available.

The final sections of the book deal with communications and interfacing. This material reviews some of the key topics fundamental to data communications, interfacing, and using electronic data bases. The treatment is rather brief, but not unreasonable so for this level. Each chapter includes a short list of references for further reading, and a glossary of commonly-used computer terms is provided.

This is a clear, well-written summary of the basic knowledge needed to use a microcomputer for scientific projects. Scientific computing is developing so rapidly that some of the material already seems somewhat dated, but there is also a great deal of sound advice that is independent of the most recent software releases. Well-written, introductory books of this type make an important contribution, especially for faculty or students who have had little previous experience with computers.

*Professor of Chemistry

SUNY-Oneonta

Oneonta, NY 13820

* * * * *

END

INFORMAL NOTES ON PROGRAMMING-LANGUAGES (AND SOME UTILITIES)

K. W. Loach, Chemistry Dept.,
SUNY Plattsburgh.

LOACHKW@SNYPLAVA.BITNET
or (518)564-4116

This has been written on the basis of personal experience, talks with other users and general reading. The dis-

cussion is limited to high-level languages (i.e. languages intended to be intelligible to humans) and emphasizes systems suitable for chemists and available on personal computers. The sources and citations are not comprehensive. My opinions expressed here are avowedly influenced by personal taste, and by the following sources:

General Sources of Information:

PC Magazine, PC Tech Journal
BytePC Week, Info World, Digital
NewsDigital Review, Unix Review,
DEC Professional, C Users Journal,
Dr. Dobbs Journal, J. Chem. Edu-
cationTrends in Analyt. Chem.

Sources of Software:

Programmer'sShop, Campus
Technology, Public Brand
Software, Programmer's Connec-
tion, Micro Warehouse, Austin
Codeworks, Programmer's Paradise
ACS Software, Campus Technol-
ogy, C User's Group, Computer
Solutions Free Software Founda-
tion

(addresses and phone-numbers of
these sources can be obtained from
many of the General Sources above,
or from the author.)

WHAT IS A LANGUAGE?

What is a 'language', in the general sense? It is any system that allows the storage and execution of autonomous high-level computer instructions, and which allows the sequence of those instructions to be controlled by data comparisons. 'Storage' and 'autonomy' are essential to the definition, because they serve to distinguish languages from interactive command systems (which execute commands as they are entered by a human operator). In an interactive system, the operator makes decisions between commands, and the commands themselves lose autonomy and cease to be a 'program'.

The 'classical' languages are mostly familiar: Fortran, Cobol, Basic, Snobol, Pascal, Ada, C, Lisp, and APL; there are also the less familiar 'neoclassical' languages, e.g. Smalltalk, Forth, Prolog, and Awk. I call these all 'classical' (in spite of their very wide differences) because they are all instantly recognizable as languages. In contrast, a great deal of programming is now done with 'utilities' that are not thought of as 'languages'. I have heard my colleagues deny that they use programming languages, and yet the systems that they use (word processors, spreadsheets, equation solvers, shells, file managers, editors, etc.) all commonly have programming language capabilities.

There has been a strong convergence. While 'utilities' have been increasingly acquiring language capabilities, 'languages' have been adding utility-like features to become 'development systems' or 'programming environments'. Meanwhile, environments have become increasingly rich, with features such as object-oriented programming (OOP), hypertext, the graphical user interface (GUI), hot-linkage (dynamic data exchange or DDE), interoperability, open-systems and the definition of language standards.

I will discuss classical languages first, and then the environment issues, and then utilities with strong language features.

PART ONE: 'CLASSICAL' LANGUAGES

Fortran: For many chemists, Fortran is the programming language. It was one of the earliest of high-level languages, and is still widely used by scientists and engineers. In the seventies, when 'structured programming' became the fashion,

Fortran was berated for its 'pass-by-address' subroutine arguments, its use of the 'goto' and its encouragement of unstructured 'spaghetti' coding. Lisp enthusiasts criticized its lack of recursion. On the other hand, Fortran did get some things right. Its input/output was graceful but powerful, and served as the model for many later languages, and it allowed independent compilation of subroutines and the consequent creation of large subroutine libraries. It went through several revisions (Fortran II, IV or 66, and 77). Each revision improved the language's power and transferability. For a long time, Fortran was strictly a 'main-frame/batch' language, but there are now several very good PC compilers. It has also become the standard language for ultra-fast computers, because it lends itself well to vectorization and parallelism.

In the 80's, it was revised again to define Fortran90. This is not yet easily available (I have seen one mention of a prototype F90 compiler, from England). It looks as though it will be a transferable, powerful and useful language, with well-structured controls (they finally put in a while-loop, after a lot of nagging from users), strong vector/array handling, greatly improved record- and data-structuring, and recursion. I gave up using Fortran about ten years ago, but Fortran90 might be worth going back to.

PF77 (Lahey), F77L (Lahey), Fortran 5.1 (Microsoft), RM/Fortran, (Ryan McFarland), Fortran77 (Absoft), Object-oriented Fortran, (Absoft) Watfor-77 (Watcom). Byte, Sept. 1991, p.147. W. H. Press et al. Numerical Recipes in Fortran (Cambridge U. Press, 1986)

Cobol: ?Cobol? Yes, Cobol. It's generally dismissed by chemists as a wordy, computationally primitive business language, and rarely considered as a scientific implementa-

tion language. But it is worth a look. One of the earliest of the classical languages, it is available on more computers, and has more support than any other single language. It was standardized very early, and its I/O and file handling are very powerful, and very machine independent. Modern Cobol-85 is computationally efficient, and can do exact-decimal arithmetic. The labor of writing in Cobol is considerably eased by the wide availability of Cobol-oriented CASE (Computer Assisted Software Engineering) tools. If you want to write software that is highly transferable, then Cobol is worth considering.

RM/Cobol-85 (Liant), Personal Cobol (Micro Focus), Cobol-85 (Microsoft), Cobol/2 (Micro Focus)

Basic: Basic became well known for several reasons: It was designed for an interactive environment; it had good text-handling capabilities; beginners found it easy to use; finally, it was the only high-level language available on most early micro-computers. Basic was very popular in the seventies, but began to be overtaken by newer languages.

Basic had disadvantages: it was interpreted and slow; it was unstandardized, with a wilderness of dialects; it had poor readability, with required line-numbers, no indentation, and ugly comments (REM REM REM REM ...); it was unstructured, with far too many goto's. Worst of all, Basic had a global data-environment, with subroutines lacking argument-passing and local variables. This global structure made it easy to write small programs, but difficult to write or modify large programs or to create reusable subroutine libraries. By the eighties, Basic appeared to be on the way to being pigeonholed as a 'hobbyist' language of little use to serious programmers.

Then it was reborn. Compilers became available. Microsoft produced a structured Basic. Powerful librar-

ies extended Basic to data-base sorting and searching, efficient matrix arithmetic, and graphics. Recently, Microsoft announced plans to standardize on Basic as a graphics-interface and systems-programming language (Visual Basic; WordBasic), embedded in all future Microsoft products.

If you think of Basic as a trivial language, or have given it up for another language, take a good look at the new Basic generation; you may be pleasantly surprised.

QuickBasic (Microsoft), Basic Compiler (Microsoft), Visual Basic (Microsoft), Realizer (Within Technologies).

PC Magazine, Oct. 31, 1989, 187; Oct. 15, 1991, p.393. PC Magazine, Sept. 10, 1991, p.81; Oct. 15, 1991, p.42. PC Week, July 31, 1989, p.84; Sept. 16, 1991, p.101. InfoWorld, Aug. 19, 1991, p.59. PC Tech Journal, Feb. 1989, 7 (2) 90. Byte, Oct., 1991, p.221. A. F. Corley et al. Computational Methods in the Chemical Sciences, (Wiley, 1989)

Snobol: this is one of the great 'classical' languages. It reached a de facto standardization by the mid-sixties, and has changed very little since. It is well known as a string and text processing language. Unfortunately, its other capabilities are not as well known. It is recursive. It has very powerful data-structure and control capabilities, including pointers, lists, and directly-indexed arrays and tables. The data-structures can be non-homogeneous (i.e. can contain data items of mixed types) and dynamic (i.e. can alter their storage-size and data-types freely during execution). It is one of the few languages (along with Lisp and Prolog) that are readily capable of self-processing and self-generation (i.e. a Snobol program can generate and immediately execute new Snobol code), making it suitable for arti-

ficial intelligence or automata applications. Snobol is unstructured, but the Rebus preprocessor allows structured controls to be added. Snobol is interpreted, but the Spitbol dialect is compiled. I believe that Snobol has been neglected by chemists. Chemical names are strings; molecular structures can be modelled as crosslinked lists or connection tables.

Snobol4+ (Catspaw), Spitbol (Catspaw), Rebus (Catspaw), R. E. Griswold et al., A Snobol4 Primer R. E. Griswold et al., Snobol4 Programming Language (2nd. ed.) J. F. Gimpel, Algorithms in Snobol4 (Catspaw).

Pascal: Pascal was designed by Wirth, in the late 60's. It is one of the Algol (block-structured) family. Wirth intended it as an instructional language, to discourage unstructured Fortran/Cobol/Basic spaghetti and preach the gospel of well-structured programming (no goto's). In addition to good structure, Pascal also requires that every structure be declared explicitly, in sufficient detail to discourage structural inconsistencies. The procedures and functions are nested, and argument-passing is by value. Pointer-based (or indirect-reference) data-structures are well supported. Pascal was widely adopted as a teaching language in the late 70's to early 80's. At the same time, microcomputers became widely available, especially the Apple II and the IBM-PC. Two compilers (UCSD-Pascal and Borland Turbo Pascal) encouraged the use of Pascal because they were cheap, and because they were the earliest readily-available 'programming environments' (i.e. a compiler, linker and execution system, integrated with an editor, to coordinate program development and debugging). Pascal was standardized in the late 80's and so made more transferable. Object-oriented Pascal has appeared. Pascal has become very popular in academia as a

utility-implementation language, and is the common alternative to C or C++.

The choice between the two languages depends on the taste of the programmer. Pascal is 'cleaner' and more elegant than C. It is easier to learn. Its deeply nested, hierarchical modules encourage consistency of structure. On the other hand, C is more powerful and better suited than Pascal to the building of libraries of reusable software tools. I think it is fair to say that Pascal is more popular with occasional programmers, and C with full-time professional programmers. (From recent reading, it seems that in Europe, the popular alternative to Pascal is Modula; in this view, C is an American aberration.)

Pascal 4.0 (Microsoft)
QuickPascal 1.0 (Microsoft)
Turbo Pascal 6.0 (Borland)
Turbo Pascal for Windows (Borland)

PC Week July 3, 1989, p.57; April 16, 1990, p.126 InfoWorld April 30, 1990, p.59; Feb. 18, 1991, p.65. PC Magazine Feb. 27, 1990, p.263; April 24, 1990, p. 46. W. H. Press et al. Numerical Recipes in Pascal (Cambridge U. Press, 1986)

Ada: After some twenty years of experience of Cobol, Fortran, PL/1, etc., it became apparent the classical languages were inadequate for the creation of large software systems. The classical languages had too many dialectical variations, the connections between independently-compiled units could not be established reliably, and the data-structures and data-operations were too sensitive to trivial details of implementation. The creation of Ada was driven by the needs of the Department of Defence, which wanted a language suitable for the creation of large software systems with a high degree of reliability.

Ada is a block-structured language

of the Algol family, modelled on Pascal, with a very high level definition of data-structure and programming module structure. It stresses the use of data-objects. An object is a data-structure tightly linked with the operations that can be performed on the structure. The details of implementation can be hidden from the user, so that a programmer need only consider the properties of the data, without concern for code details. Higher-level objects can be built up from lower-level objects, again without concern for code details.

The DOD went further in its requirements. Ada was provided with a large set of test programs, to provide bench-marks of compiler and execution performance. No Ada compiler could be certified as acceptable unless it met the standards defined by the test-suite. Also, subsets, dialects and variants of Ada were forbidden, so that compiler differences could be discouraged.

The long process of definition and testing slowed Ada's availability. Most of the early Ada literature emphasized the mass, power and sophistication of the language. As a result, Ada has always been forbiddingly impressive. But this is changing. PC Ada compilers are now available, at moderate prices. Introductory-level textbooks are now appearing. (My college Computer Science department has just dropped Pascal in favor of Ada as the introductory-level language of instruction.) Simple Ada programs resemble Pascal, and are no more formidable, but Ada offers greater power than Pascal. Ada is a very promising language for chemists who would like to build a library of co-operating and interacting chemical and instrumental utilities.

Janus/Ada 2.1 (R. R. Software)
IntegrAda 4.2 (Aetech)
HALO Ada (Media Cybernetics)
FirstAda 4.4 (Alsys)
AdaVantage (Meridian Software)

C, C++: C's remote ancestor was Algol, by way of BCPL, with some Fortran influence. It was developed in the 70's by Kernighan and Ritchie, along with the Unix operating system. At first, the language and system were nearly synonymous, but they have since diverged. The intent was to create a language suitable for systems programming, with the power of assembly language but with high-level structure and control. Many languages (e.g. Cobol, Pascal) are restrictive; they limit the operations allowed to the programmer in order to prevent or discourage programming mistakes and dangerous practices. C was deliberately designed to be permissive, so as to give the programmer great power of action and control. Early C had relatively poor type-checking, but type-checking was tightened up, and inconsistencies discouraged, with the ANSI standardization.

C is readily extensible; The very appearance of the language can be altered and features can be added using macros. The input/output and other functions were deliberately specified as add-on libraries and not as part of the fundamental language definition. C has become very popular as an implementation language for other languages, and for micro-computer utilities. It became even more popular with the addition of C-tools and C-programming environments. In the early 80's, an ANSI standard was defined, making it more consistent and transferable. C was extended and redefined by Stroustrup and data-objects were adopted (from Smalltalk and Ada) to create C++, now the most popular and widely available object-oriented programming (OOP) language.

C has disadvantages. Its power can lead to powerful or obscure errors, difficult to debug. Like APL, C tends to encourage obscure or perverse programming. Its permissive nature requires a corresponding personal

discipline on the part of the programmer.

Turbo C 2.0 (Borland)
Power C (Mix)
C 6.0 (Microsoft)
QuickC (Microsoft)
Lattice C 6.0 (Lattice)
TopSpeed C (Jensen)
C++ 3.0 (Zortech)
C++ (Oregon)
C++ (Comeau)
QuickC for Windows (Microsoft)
C/386 (Watcom)
NDP C++ 386/486 (Microway)
Objective-C (Stepstone)
Green Hills C++ (Oasys)
CSL C-Libraries (Eigenware)
Interface Library 2.0 (Zinc)
C libraries (Greenleaf)
C++ libraries (Rogue Wave)
C/Base/Library (IMSL)

InfoWorld, April 30, 1990, p.59; May 14, 1990, p.5. InfoWorld, April 8, 1991, p.55; Sept. 9, 1991, p.25 PC Magazine, Sep. 13, 1988, 115. C Users J., Mar. 1987, p.33; Apr. 1989, p.51; July 1989, p.9. W. H. Press et al. Numerical Recipes in C (Cambridge U. Press, 1988)

Lisp: Lisp is perhaps the oldest language still in widespread use. It was created in the 50's by McCarthy. From the start, it has been known as a highly specialized language, for the processing of lists. It acquired the forbidding reputation of being arcane, inefficient, difficult, and unsuitable for numeric or text processing. These things were somewhat true of the earliest Lisp interpreters, but are true no longer. Modern Lisp compilers are capable of high-speed data-processing, and Lisp programming environments provide Lisp-specific editors and other effective programming tools. Lisp was also fragmented into a great many dialects, but now there is a well standardized and widely available Common Lisp, which is highly transferable across machines.

The chief advantage of programming in Lisp is its power. It is probably less constricting than any other general purpose programming language. All Lisp programs are themselves strict lists, so Lisp is capable of self-generation and self-execution of its own code (like Snobol and Prolog). Lisp is particularly suitable for artificial-intelligence and robotics applications, but is no longer confined to use in these areas. It is increasingly being used as a software implementation language. Almost all chemical data can be represented as list structures; molecular structures can be modelled as connectivity lists (graphs) of atoms or functional groups.

There is no question that Common Lisp should now be used for all serious Lisp programming. However, beginners should consider starting out with Scheme, a Lisp dialect from MIT that is particularly suitable for learning. Not only is the Scheme interpreter cheap and simple to use, but it is supported by a superb text (Abelson/Sussman/Sussman) that will ease the task of learning Lisp. The transition from Scheme to Common Lisp programming should be easy. Even if you have no intention of ever using Lisp for serious programming, it is still well worth learning, because it will transform your view of programming. Lisp is truly one of the fundamental languages.

mu-Lisp 87 (Soft Warehouse)
Common Lisp (Lucid)
PC Scheme (Texas Instruments)
Star Sapphire Common Lisp (Sapiens)

Dr. Dobbs Journal May 1987, p.132; June 1987, p.116
H. Abelson, G. J. Sussman, J. Sussman, Structure and Interpretation of Computer Programs (MIT Press, 1989)
G. L. Steele Jr., Common Lisp - The Language (Digital Press, 1984)

APL: In the 70's, this was on the way to establishing itself as a great

language. Created by a mathematician, it has a very compact, symbolic notation. It was particularly designed for the handling of vectors, arrays and matrices. APL created an enthusiastic following and, for a while, seemed to be the possible successor to Fortran as the principal numeric language. But the bottom seems to have fallen out of the APL bucket. It is very good for numeric processing, and can even handle string/text processing fairly well. However, it does not seem to be suited to handling pointer and record structures. I see fewer and fewer mentions of APL in recent years. It may be dying, a victim of its highly specialized notation (requiring a special keyboard), its com-

pactness (leading to poor readability) and its narrow focus on vectors and matrices. It may well survive as a niche-language, for numeric array processing, but I think it is unlikely to remain part of the main-stream of computing languages.

[END OF PART ONE: TO BE CONTINUED]

COMMENTS FROM THE EDITOR:

Finally another issue. It appears to be getting harder for people to find time to write articles.

We can't promise when the next issue will come out since we depend on volunteers like yourself to contribute articles. We would be delighted to publish more articles by readers. When we get enough we will again put out a newsletter.

I'd like to add to professor Loach's comments on C and Pascal. I find that may 'older' programmers, full time and otherwise, use Pascal rather than C. The reasons are many though the possibility of making serious hard to find errors in C, and the fact that you can do anything in Pascal you can do in C are probably the most important. I find younger programmers prefer C and C++.

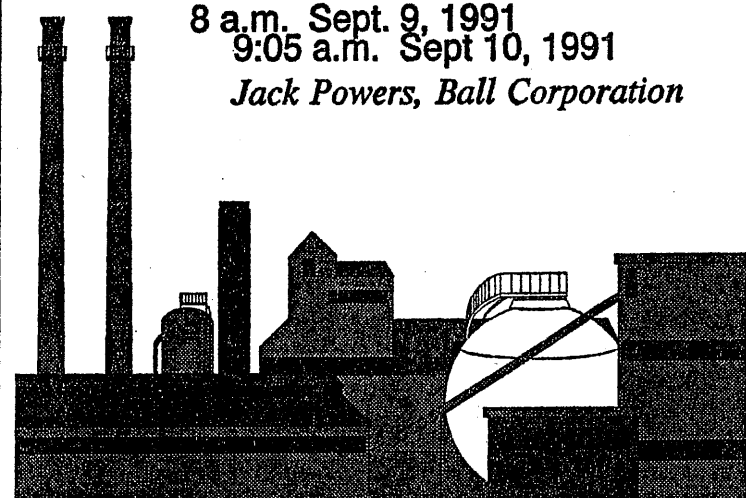
Below are more samples from Jim Beatty.

Ripon College Chemistry Seminars

8 a.m. Sept. 9, 1991

9:05 a.m. Sept 10, 1991

Jack Powers, Ball Corporation



X-RAY DIFFRACTION DATA SHEET				11/28/90		Michael De Frank		
Ripon College, Ripon, WI								
Left	Right	Factor		Identification		12 me & 50 kW		
48.100	537			Substance		1 hour exposure		
178.800	245	0.500	1.541	System:	fcc	NE Camera		
Centers			1.544	Density =				
112.450	202	180.025	1.542		Ave. a zero =	4.079		
Powder Pattern Lines								
Line	Relative	X in	X - L.C.	Angle	d_{hkl}	M ²	a_c	Miller
#	Intensity	mm	mm, 2θ	$^\circ$				Indices
1	80	150.750	36.300	19.147	2.350	3	4.071	110
2	30	156.950	44.500	22.247	2.036	4	4.072	200
3	50	178.800	64.350	32.171	1.448	8	4.066	211
4	100	190.250	77.800	36.895	1.228	11	4.072	220
5	20	194.650	82.200	41.004	1.173	12	4.063	310
6	10	210.400	97.950	48.068	1.022	16	4.068	222
7	00	223.200	110.750	55.367	0.937	19	4.084	321
8	60	227.650	115.100	57.542	0.914	20	4.066	400
9	60	247.750	135.300	67.641	0.833	24	4.060	411,330