Alternative Programming Languages

TAKE YOUR PICK!

- Parasol
- Perl
- Sather
- Modula-3
- Oberon
- Bob
- Tcl

J Stevens’ Quincy C Interpreter

GCC

Sather

Modula-3

Tcl

Quincy

Bob

Winter 1994
Raise Your Programming

The Alternative Programming Languages CD-ROM from Dr. Dobb's Journal is your resource for the latest in cutting-edge programming environments. Gain access to the most innovative, solution-providing languages and tools available today.

**DELVE INTO THE FUTURE**

Examine the next generation of languages. Write the next generation of applications. This CD-ROM is the tool you need to get up to speed on...

- Distributed Computing with languages that support parallel computing and networks.
- Embeddable Languages for raising the power of your applications.
- Increased Productivity utilizing cleaner, more efficient code.
- Future Languages that break the chains of today's languages.

**APPLY TOMORROW'S LANGUAGES TODAY**

The Alternative Programming Languages CD-ROM provides languages and tools you can put to work immediately on your development projects:

**Object-Oriented Development.** Tired of C++? Look no further than Sather, Dylan or Bob and get re-energized.

**Distributed Computing.** When building distributed systems, languages like Parasol and M0 make your life easier.

**Multiple Platforms.** Check out Perl and Python, plus a wide range of compilers, including the GNU compile suite.

**Text Processing and Document Formatting.** Lout and Ghostscript let you edit and produce complex documents with ease.

**Prototyping.** Developing applications for distributed systems is one thing, prototyping connected distributed systems is another. Glish makes it all happen.

**Neural Networks.** Neudl, a neural net description language, takes the pain out of building complex neural networks.

**ONLY $44.95**

**60-DAY MONEY-BACK GUARANTEE!**
Features

All languages are robust, imaginatively designed, break new technical ground, and come with source code for compilers or interpreters. They may not be widely known yet, but they will be! Full descriptions and specifications are provided.

Meet the programming challenges of tomorrow—TODAY!

Challenge your imagination. With the right tools there are no limits to what you can do! And that's why the Alternative Programming Languages CD-ROM is a must have for any serious programmer. Order now!

ALL THESE LANGUAGES, COMPILERS, AND INTERPRETERS—AND MORE!

Smalltalk: A pure object-oriented language. This implementation of Little Smalltalk runs under UNIX, DOS, and Windows.

Sather: An object-oriented language which places more emphasis on efficiency and less on correct software construction.

Tcl: A tool control language for applications that run under DOS, OS/2, and UNIX.

Perl: This is becoming the general-use scripting language for use on UNIX.

Glish: An interpreted language and shell designed to make loosely coupled distributed systems easy to prototype.

MO: An interpreted language for distributed systems.


ReXX: A powerful scripting language for writing utilities and off-the-cuff programs.

Lout: A text processing and document formatting language with a compact and efficient implementation.

GNU: Compiler suite which includes C, C++, and Obj-C.

Ghostscript: An implementation of Postscript.

Dylan: Object-oriented, this is the Mindy bytecode interpreted implementation of the Interim-Dylan spec.

Duel: An embedded debugging language that couples software construction. with gdb.

Bob: A small object-oriented language particularly useful for embedding into applications.


Python: An object-oriented scripting language similar to Perl.

Oberon: A language which simplifies Modula-2 by adding extensions for object-oriented programming.

Parasol: A language and OS designed for distributed systems.

S-Lang: A text-oriented editor designed for use with the JED editor (included).

Quincy: Al Steven's interpreter. Runs under DOS.

For details about what's on the CD-ROM, see page 41 in this issue.

FREE CALL
800-727-3662
24 hours a day, 7 days a week.

MAIL ORDERS
Programming Languages CD-ROM
P.O. Box 7046
San Francisco, CA 94120 USA

INTERNATIONAL
415-655-4190
Or use Fax/Mail

E-MAIL
sbarnes@mfi.com

Only $44.95. Take $5 off the regular $49.95 price if you've already purchased this Dr. Dobb's Sourcebook of Alternative Programming Languages. Your cost is only $44.95! Plus shipping: $2.00 USA/Canada, $12.50 all other countries. Calif. residents add sales tax $3.70.
FEATURES

The Parasol Programming Language 6
by Robert Jervis
Parasol, short for "Parallel Systems Object Language," is a development environment influenced by C and Smalltalk, although its design also reflects C++, CLU, Aigol, and Turbo Pascal.

The Perl Programming Language 12
by Oliver Sharp
Perl, a language designed to handle a variety of network system-administration tasks, makes manipulating the socket protocol easier still. Oliver shows how to write Perl scripts that communicate across networks of UNIX machines.

The Sather Programming Language 18
by Stephen M. Omobundro
Sather is a language that's simple, interactive, and nonproprietary. It sports parameterized classes, object-oriented display, statically checked strong typing, multiple inheritance, garbage collection, and more.

The Modula-3 Programming Language 24
by Sam Harbison
Feature-for-feature, Modula-3 is on a par with Ada and C++, but avoids complexity by simplifying individual features like inheritance and generics. Furthermore, Modula-3 is less of a moving target because it already has features only now being added to those other languages.

Bob: A Tiny Object-Oriented Language 32
by David Betz
This tiny C-like object-oriented language, developed by the creator of XLisp, XScheme, and other well-known public-domain languages, is a powerful extension language.

The Tcl Programming Language 40
by John K. Ousterbouh
Tcl (pronounced "tickle") is a command language John designed to be a powerful and flexible "glue" language for assembling software components.

Quincy: The Architecture of a C Interpreter 48
by Al Stevens
Quincy, a C interpreter with a front end based on AI's D-Flat windowing system, is fast, small, and efficient.

The Dylan Programming Language 54
by Tamme D. Bowen and Kelly M. Hall
Originally based on Scheme, Dylan is an object-oriented, dynamic language designed to replace existing static languages for the development of large software systems.

The Oberon Programming Language 60
by Josef Tempi
Oberon, a general-purpose, object-oriented programming language that evolved from Pascal and Modula-2, has been implemented for DOS, Windows, Amiga, Mac, and UNIX.

Editorial 5
by Jonathan Erickson

PROGRAMMER'S SERVICES

As a service to our readers, all source code is available on a single disk and online. To order the disk, send $14.95 (California residents add sales tax) to Dr. Dobb's Journal, 411 Borel Ave., San Mateo, CA 94402, call 415-655-4321 x5701, or use your credit card to order by fax, 415-358-9749. Specify issue number and disk format. Code is also available through the DDJ Forum on CompuServe (type GO DDJ) or via anonymous FTP from site ftp.mv.com (192.80.84.1) in the /pub/ddj directory.
The Sather Programming Language

Sather is an object-oriented language which aims to be simple, efficient, interactive, safe, and non-proprietary. One way of placing it in the "space of languages" is to say that it aims to be as efficient as C, C++, or Fortran, as elegant and safe as Eiffel or CLU; and as supportive of interactive programming and higher-order functions as Common Lisp, Scheme, or Smalltalk.

Sather has parameterized classes, object-oriented dispatch, statically checked strong typing, separate implementation and type inheritance, multiple inheritance, garbage collection, iteration abstraction, higher-order routines and iterators, exception handling, constructors for arbitrary data structures and assertions, preconditions, postconditions, and class invariants. This article describes a few of these features. The development environment integrates an interpreter, a debugger, and a compiler. Sather programs can be compiled into portable C code and can efficiently link with C object files. Sather has a very unrestricted license which allows its use in proprietary projects but encourages contribution to the public library.

Stephen does research on learning and computer vision, as well as developing Sather at the International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704. He wrote the three-dimensional graphics for Mathematica and was a codeigniter of Star-Lisp for the Connection Machine. He can be contacted at om@icsi.berkeley.edu.

Stephen M. Omohundro

The original 0.2 version of the Sather compiler and tools was made available in June 1991. This article describes version 1.0. By the time you read this, the combined 1.0 compiler/interpreter/debugger should be available on ftp.icsi.berkeley.edu, and the newsgroup comp.lang.sather should be activated for discussion.

Code Reuse

The primary benefit object-oriented languages promise is code reuse. Sather programs consist of collections of modules called "classes" which encapsulate well-defined abstractions. If the abstractions are chosen carefully, they can be used over and over in a variety of different situations.

An obvious benefit of reuse is that less new code needs to be written. Equally important is the fact that reusable code is usually better written, more reliable, and easier to debug because programmers are willing to put more care and thought into writing and debugging code which will be used in many projects. In a good object-oriented environment, programming should feel like plugging together prefabricated components. Most bugs occur in the 10 percent or so of newly written code, not in the 90 percent of well-tested library classes. This usually leads to simpler debugging and greater reliability.

Why don't traditional subroutine libraries give the same benefits? Subroutine libraries make it easy for newly written code to make calls on existing code but not for existing code to make calls on new code. Consider a visualization package that displays data on a certain kind of display by calling display-interface routines. Later, the decision is made that the package should work with a new kind of display. In traditional languages, there's no simple way to...
Parameterized Classes
Listing One (page 22) shows a class which implements a stack abstraction. We want stacks of characters, strings, polygons, and so on, but we don't want to write new versions for each type of element. STACK{CHAR} is a parameterized class in which the parameter T specifies the stack-element type. When the class is used, the type parameter is specified:

For example, the class FOO in Listing Two shows a class which implements a stack abstraction. We want stacks of characters, strings, polygons, and so on, but we don't want to write new versions for each type of element. STACK{CHAR} is a parameterized class in which the parameter T specifies the stack-element type. When the class is used, the type parameter is specified:

The Sather solution to this is based on abstract types. The Sather libraries include the abstract class $POLYGON$, which defines the abstract interface that all polygons must provide. It also includes the descendant class POLYGON, which implements generic polygons. The add_vertex routine defined in POLYGON but does not make sense for triangles, squares, and so on. In languages which do not separate abstract types from particular implementations, you must either make all descendants implement routines that don't make sense for them, or leave out functionality in parent classes.

The Sather type system is a major factor in the computational efficiency, clarity, and safety of Sather programs.

Object-Oriented Dispatch
Listing Two (page 22) shows an example of object-oriented dispatch. The class $POLYGON$ is an "abstract" class, which means it represents a set of possible object types called its "descendants" (in this case, TRIANGLE and SQUARE). Abstract classes define abstract interfaces which must be implemented by all their descendants. Listing Two only shows the single routine number_of_vertices:INT, which returns the number of vertices of a polygon. TRIANGLE's implementation returns the value 3, and SQUARE's returns 4.

Routine dispatch in the interface of an abstract type may be called on variables declared by that type. The actual code that's called, however, is determined at run time by the type of the object which is held by the variable. The class FOO2 defines a routine with a local variable of type STACK{POLYGON}. Both TRIANGLE and SQUARE objects can be pushed onto stacks of this type. The call s.pop might return either a triangle or a square. The call s.pop.number_of_vertices calls either the number_of_vertices routine defined by TRIANGLE and returns 3, or the number_of_vertices routine defined by SQUARE and returns 4.

Separate Implementation and Type Inheritance
In most object-oriented languages, inheritance defines the subtype relation
These so-called "dangling pointers" are still being referenced, a later access may find the memory in an inconsistent state. It is not freed even though there are no references to it, are also hard to find. Programs with this bug use more and more memory until they crash. Sather combines flexibility with high efficiency.

uses a garbage collector which tracks down unused objects and reclains the space automatically. To further enhance performance, the Sather libraries generate far less garbage than is typical in languages such as Smalltalk or Lisp.

Interactive, Interpreted Programming
Sather combines the flexibility of an interactive, interpreted environment with very high-efficiency compiled code. During development, the well-tested library classes are typically run compiled, while the new experimental code is run interpreted. The interpreter also allows immediate access to all the built-in algorithms and data structures for experimentation. Listing Three (page 22) is an example of an interactive Sather session.

Iteration Abstraction
Most code is involved with some form of iteration. In loop constructs of traditional languages, iteration variables must be explicitly initialized, incremented, and tested. This code is notoriously tricky and is subject to "fencepost errors." Traditional iteration constructs require the internal implementation details of data structures like hash tables to be exposed when iterating over their elements.

Sather allows you to cleanly encapsulate iteration using constructs called "iters" (Murer, Omohundro, and Syperski, 1993) that are like routines, except their names end in an exclamation point (!), their bodies may contain yield and quit statements, and they may only be called within loops. The Sather loop construct is simply: loop...end. When an iter yields, it returns control to the loop. When it is called in the next iteration of the loop, execution begins at the statement following the yield. When an iter quits, it terminates the loop in which it appears. All classes define the iters until!, while! and break! to implement more traditional looping constructs. The integer class defines a variety of useful iters, including upto! (INT!INT), downto! (INT!INT), and step! (num!step!INT!INT). Listing Four (page 22) shows how upto! is used to output digits from 1 to 9.

Container classes, such as arrays or hash tables, define an iter else! (T) to yield the contained elements and an iter called set_els! (T) to insert new elements. Listing Four shows how to set the elements of an array to successive integers and then how to double them. Notice that this loop doesn't have to explicitly test indexes against the size of the array.

The trees classes have iters to yield their elements according to the "pre," "post," and "in" orderings. The graph classes have iters to yield the vertices according to depth-first and breadth-first search orderings.

The Implementation
The first version of the Sather compiler was written in Sather by Chu-Cheow Lim and has been operational for several years. It compiles into C code and has been ported to a wide variety of machines. It is a fairly large program with about 30,000 lines of code in 183 classes (this compiles into about 70,000 lines of C code).

Lim and Stokke extensively studied the performance of the compiler on both MIPS and SPARC architectures. Because the compiler uses C as an intermediate language, the quality of the executable code depends on the match of the C-code templates used by the Sather compiler to the optimizations employed by the C compiler. Compiled Sather code runs within 10 percent of the performance of handwritten C code on the MIPS machine and is essentially as fast as handwritten C code on the SPARC architectures. On a series of benchmark tests (towers of Hanoi, 8 queens, and the like) Sather performed slightly better than C++ and several times better than Eiffel. The new compiler performs extensive automatic inlining and so provides more opportunities for optimization than typical handwritten C code.

The Libraries
The Sather libraries currently contain several hundred classes, and new ones are continually being written. Eventually, we hope to have efficient, well-written classes in every area of computer science. The libraries are covered by an unrestricted license which encourages the sharing of software and inventive of authors, without prohibiting use in proprietary and commercial projects. Currently there are classes for basic data structures, numerical algorithms, geometric algorithms, graphics, grammar manipulation, image processing, statistics, user interfaces, and connectionist simulations.

pSather
Sather is also being extended to support parallel programming. An ini-
Conclusion

I've described some of the fundamental design issues underlying Sather 1.0. The language is quite young, but we are excited by its prospects. The user community is growing, and new class development has become an international, cooperative effort. We invite you join in its development!

Acknowledgments

Sather has adopted ideas from a number of other languages. Its primary debt is to Eiffel, designed by Bertrand Meyer, but it has also been influenced by C, C++, CLOS, ML, Modula-3, Oberon, Python, Common C, Pascal, Smalltalk, and S𝐠. Many people have contributed to the development and design of Sather. The contributions of Jeff Bilmes, Ari Huttunen, Jerry Feldman, Chu-Cheow Lim, Stephan Murer, Heinz Schmidt, David Stotumire, and Clemens Szyperski were particularly relevant to the issues discussed in this article.

References

ICSI Technical reports are available via anonymous ftp from ftp.icsi.berkeley.edu.


