

FOLHA DE APROVAÇÃO

Marco Túlio de Oliveira Valente

Projeto e Implementação de uma Linguagem
Orientada por Objetos para o Desenvolvimento
Sistemático de Programas

Projeto e Implementação de Uma Linguagem Orientada por
Objetos para o Desenvolvimento Sistemático de Programas

Dissertação apresentada ao Departamento
de Ciência da Computação do Instituto de
Ciências Exatas da Universidade Federal de
Minas Gerais, como requisito parcial para a
obtenção do grau de Mestre em Ciência da
Computação.

Belo Horizonte

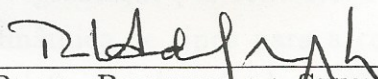
Fevereiro de 1996

FOLHA DE APROVAÇÃO

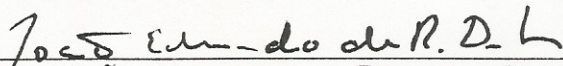
Projeto e Implementação de uma Linguagem Orientada por Objetos para o Desenvolvimento Sistemático de Programas

MARCO TÚLIO DE OLIVEIRA
VALENTE

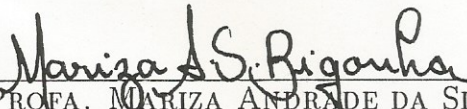
Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:



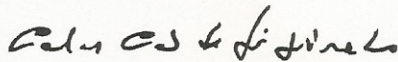
PROF. ROBERTO DA SILVA BIGONHA - Orientador
DCC - ICEX - UFMG



PROF. JOÃO EDUARDO DE REZENDE DANTAS
DCC - ICEX - UFMG



PROFA. MARIZA ANDRADE DA SILVA BIGONHA
DCC - ICEX - UFMG



PROF. CARLOS CAMARÃO DE FIGUEIREDO
DCC - ICEX - UFMG

Belo Horizonte, 27 de dezembro de 1995.

Resumo

O presente trabalho apresenta uma extensão de C orientada por objetos, chamada *lta*, em cujo projeto procurou-se conciliar clareza com poder de expressão. A fim de favorecer a produção de *software* com alto grau de reusabilidade, *lta* introduz conceitos como os de ocultamento de informação, herança e polimorfismo. Para implementação de tipos abstratos de dados, existe em *lta* o conceito de classes, que inclusive podem ser genéricas ou abstratas. Incentiva-se também a produção de *software* correto através de um estilo de programação por contrato. A linguagem propõe ainda uma solução baseada em verificação dinâmica de tipos para a controvérsia sobre o uso de covariância ou contravariância na redefinição de métodos em subclasses.

A fim de testar e validar tais recursos propostos por *lta*, foi implementado um compilador para a linguagem.

Abstract

This work shows an object oriented extension of C, named *lta*, whose design alies simplicity with power of expression. To support the production of *software* with high degree of reusability, *lta* adds concepts like information hiding, inheritance and polymorphism. To implement abstract data types, there is the concept of class, which in *lta* can inclusively be generic or abstract. The production of correct software is stimulated by means of the concept of programming by contract. The language still proposes a solution based in dynamic type verification to the problem about the use of covariance or contravariance in the redefinition of methods in subclasses.

In order to test and validate such resources proposed by *lta*, a compiler for the language was implemented.

Agradecimentos

Ao professor Bigonha, na esperança de que parte de sua competência na área de linguagens tenham sido refletidos nesse trabalho.

Aos professores Dantas e Cássio e à professora Maria, membros da banca de dissertação, pela leitura crítica de meu texto e pelas modificações sugeridas.

Ao professor Leoni, pela oportunidade de trabalhar próximo a ele.

A TELA MUI, ex-primeira da análise de sistemas Tália Silva, pelo apoio à conclusão desta dissertação.

Aos colegas de curso, Alexandre, Mark, Raul e Claudiney, pela amizade e companheirismo.

A meus avós,
Luquico e Fiinha,
Vadico e Nila.

Agradecimentos

Ao professor Bigonha, na esperança de que parte de sua competência e entusiasmo pela área de linguagens tenham sido refletidos nesse trabalho.

Aos professores Dantas e Camarão e à professora Mariza, membros da banca examinadora da dissertação, pela leitura criteriosa de seu texto e pelas modificações sugeridas.

Ao professor Leacir, pela oportunidade de trabalhos prévios na área.

À TELEMIG, na pessoa do analista de sistemas Túlio Silva, pelo apoio à conclusão dessa dissertação.

Aos colegas de curso, Alexandre, Mark, Raul e Claudiney, pela amizade e companheirismo.

Sumário

1	Introdução	1
1.1	Motivação	1
1.2	Objetivo da Dissertação	2
1.3	Organização da Dissertação	3
2	Orientação por Objetos	5
2.1	Programação Orientada por Objetos	5
2.2	Linguagens Orientadas por Objetos	6
2.2.1	C++	6
2.2.2	Eiffel	9
2.2.3	Oberon-2	11
3	A Definição de Ita	12
3.1	Introdução	12
3.2	Estrutura Léxica	12
3.2.1	Comentários	12
3.2.2	Palavras Chave	12
3.2.3	Constantes	13
3.3	Tipo Conjunto	13
3.4	Classes	13
3.4.1	Instanciação de Classes	14
3.4.2	Classes <i>Friends</i>	15
3.4.3	Classes Genéricas	16
3.5	Semântica de Referência	17
3.5.1	Operações Pré-definidas	17
3.5.2	Passagem de Parâmetros	18
3.6	Funções Polivalentes	18
3.7	Funções Polissêmicas	19
3.7.1	Polissemia por Número e Tipo dos Parâmetros	19
3.7.2	Polissemia por Estado	20
3.8	Asserções	21
3.8.1	Tratamento de Exceção	22
3.9	Herança	24
3.9.1	Redefinição de Membros	24
3.10	Polimorfismo	26
3.10.1	Polissemia por Forma	26
3.10.2	Verificação Dinâmica de Tipos	26

3.11	Classes Abstratas	27
3.12	Inicialização	27
3.13	Estrutura de Programa	28
4	O Projeto de Ita	29
4.1	Tipos	29
4.1.1	Classes	29
4.1.2	Conjuntos	31
4.2	Generalidade	31
4.3	Programação por Contrato	32
4.3.1	Tratamento de Exceções	34
4.4	Herança	34
4.5	Polimorfismo, Polissemia e Polivalência	35
4.5.1	Polimorfismo	36
4.5.2	Polissemia e Polivalência	36
4.6	Redefinição de Métodos	39
4.6.1	Covariância x Contravariância	39
4.6.2	A Regra da Contravariância Guardada	41
4.7	Classes Abstratas	42
4.8	Estrutura de Programa	42
4.9	Considerações Finais	43
5	Um Estudo de Caso em Ita	44
5.1	Arranjos	44
5.2	Listas	46
5.2.1	Listas de Tamanho Fixo	47
5.2.2	Listas Encadeadas	48
5.2.3	Listas Duplamente Encadeadas	51
5.3	Considerações Finais	53
6	O Compilador de Ita	54
6.1	Analizador Léxico	54
6.2	Analizador Sintático	55
6.3	Tabela de Símbolos	56
6.4	Geração de Código	58
6.4.1	Geração de Código para Classes	58
6.4.2	Chamada Dinâmica de Funções	59
6.4.3	Teste-de-Tipo e Guarda-de-Tipo	59
6.4.4	Operações com Semântica de Valor	62
6.4.5	Geração de Código para Classes Genéricas	63
6.4.6	Geração de Código para Asserções	65
6.4.7	Geração de Código para Conjuntos	65
6.5	Considerações Finais	67
7	Conclusões	68
7.1	Avaliação da Linguagem	68
7.2	Trabalhos Futuros	70

Bibliografia

72

A A Gramática de Ita

75

A.1 Definições Externas	75
A.2 Declarações	76
A.3 Classes	79
A.4 Comandos	80
A.5 Expressões	81
A.6 Constantes	83

Introdução

1.1 Motivação

A década de 80, a era de linguagens de programação foi marcada pelo surgimento das chamadas linguagens de programação orientadas por objetos (LOO). Tais linguagens tinham como objetivo proporcionar um salto de qualidade no processo de desenvolvimento de software, substituindo principalmente a reutilização de código. Reutilização é um dos aspectos da grande flexibilidade de programação, pois possibilita que o custo de um componente de software seja amortizado em projetos seguintes. Para reforçar esse desejo, constata-se que parte da atividade de programar consiste essencialmente em adaptar algumas rotinas comuns, como funções, estruturas de dados, algoritmos, buffers, etc. A LOO pretende então fornecer meios para que o programador possa reaproveitar, possivelmente modificando ou estendendo, rotinas já existentes.

Atualmente já estão disponíveis uma série de LOO, como Smalltalk, C++, Objective C, Object Pascal, Oberon-2, Eiffel, Sather, dentre outras. Dentre essas linguagens, a que chamou mais atenção de mercado foi, sem dúvida, C++, talvez por ser uma extensão da linguagem C. Apesar de não ter um número, um dos fabricantes de compiladores C++, a Borland, garante já ter vendido mais de um milhão de cópias de seu produto. No entanto, existem fortes críticas sobre C++, principalmente quanto ao seu extensivo número de recursos, alguns deles de difícil entendimento, e por isso mesmo, pouco usados. Por isso, alguns temem que a linguagem possa transformar-se em uma espécie de "PL/I dos anos 80".

Por outro lado, ainda na década de 80, surgiram outras linguagens como Eiffel e Oberon-2, mostrando que orientar por objetos não implica necessariamente em linguagens de difícil domínio e com baixa curva de aprendizado. A definição BNF de Oberon-2, por exemplo, ocupa apenas uma página e não por isso possui poder de expressão menor que suas concorrentes. Ressalte-se que essa tendência à simplicidade está presente também em outras áreas da computação, como a de microprocessadores, com os chamados chips RISC (Computadores com Conjunto de Instruções Reduzido). O próprio criador de Eiffel cunhou o termo *RISC* brevior para se referir à sua linguagem.